# Andy Niu Help 1.0.0.0

▶工具软件

心得体会

架构和设计

程序员修炼

▶正则表达式

▶笔试面试

首页 模块 模块 这里列出了所有模块: [详情级别 1234] **►** Algorithm C ► CPP **Example ► Linux** Lua **MFC** ► MySQL **▶** Python Redis ▶Shell脚本 **► STL** ► TCP\_IP Unix环境高级编程 **Windows** ▶家庭 ▶工作有关

计算机原理 设计模式 ▶读书 软件工程和思想	视频音频	
▶读书	计算机原理	
	设计模式	
软件工程和思想	▶读书	
4/11 TT1 TT1 H (10) (10)	软件工程和思想	
进程_线程_同步_异步	进程_线程_同步_异步	

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 模块 | 变量

## 模块

常用算法

排序算法

数据结构\_树

数据结构\_链表

编程珠玑

## 变量

n皇后问题

二叉树求幂集

查找最大的k个数

牛顿迭代求平方根

## 变量说明

#### n皇后问题

```
/*
   这里的int* q 是一维数组,对于4皇后,就是q[4]
   按道理需要二维数组表示棋盘,这里做了简化。比如4皇后,
   使用一维数组,q[4]四个元素,表示4行,a[0]的值表示第0行,
   比如a[0]=2, 表示第0行, Queen放在第2列
*/
int* q;
//输出一个解
void PrintQueen_N(int n)
{
   static int count = 0; //输出解的同时,记录解的个数
   printf("第[%d]个解:",++count);
   for(int i=0;i<n;i++)</pre>
       printf("(%d,%d) ",i,q[i]);
   printf("\n");
   for(int i=0;i<n;i++) // 遍历行
   {
       for(int j=0;j<n;j++) // 遍历列
       {
          if(q[i]==j) // 如果行的取值等于当前列,说明这~
             printf("● ");
```

```
else
          {
              printf("o ");
          }
       printf("\n");
   }
}
/*
   检查在第row行的第col列,放置Queen是否合法
   如何检查?
   前面的row-1行已经放置了Queen,遍历这些Queen,检查Queen。
*/
bool IsValid(int row, int col)
{
   // 遍历每一行的Queen
   for(int i=0;i<row;++i)</pre>
   {
       /*
          检查Queen与当前位置是否冲突
          因为一行一行地放置Queen,肯定不在同一行了
          要检查是否在同一列或者在对角线上。
          同一列:q[i]==col 第i行Queen放的列等于当前列
          对角线: 左上到右下,行-行==列-列
                  左下到右上,行-行==-(列-列)
                  也就是abs(r1-r2)=abs(c1-c2)
       * /
       if(q[i]==col \mid | abs(i-row)==abs(q[i]-col)) /
       {
          return false;
       }
   return true;
}
```

```
// 在每一行放置Queen
void PlaceQueenInRow(int row,int n)
{
   if(row>=n) // 从0行开始,已经放到第n行,放置结束
   {
       PrintQueen_N(n);
   else // 在第row行放置Queen
    {
       // 对于第row行,遍历所有的列,逐个尝试
       for(int j=0;j<n;j++)</pre>
       {
           if(IsValid(row,j)) // 判断row的j列放置Queer
           {
               q[row] = j; //合法,标识第row的Queen
               PlaceQueenInRow(row+1,n); //在下一行方
           }
       }
   }
}
void Queen_N(int queenNum)
{
   q = new int[queenNum];
   PlaceQueenInRow(0, queenNum);
   delete[] q;
}
运行程序,得出n皇后解的个数:
1
       1
2
       0
3
       0
       2
4
5
       10
6
       4
7
       40
8
       92
```

```
9 352
10 724
```

#### 二叉树求幂集

```
// C++实现
void GetPowerSetByBTree(int i,const string& src,strir
    if(i>=src.size())
    {
        powerSet.push_back(oneEle);
    else
    {
        oneEle+=src[i];
        GetPowerSetByBTree(i+1, src, oneEle, powerSet);
        //oneEle=oneEle.substr(0,oneEle.size()-1);
        oneEle.erase(oneEle.size()-1);
        GetPowerSetByBTree(i+1, src, oneEle, powerSet);
    }
}
void GetPowerSet(const string& src, vector<string>& pc
{
    string oneEle;
    GetPowerSetByBTree(0, src, oneEle, powerSet);
}
## Python实现
def GetPowerSet(i,aTuple,oneEle):
    if i >= len(aTuple):
        print oneEle
    else:
        oneEle+=str(aTuple[i])
        GetPowerSet(i+1, aTuple, oneEle)
```

```
oneEle=oneEle[0:len(oneEle)-1]
   GetPowerSet(i+1, aTuple, oneEle)

aTuple=(1,2,3)
oneEle=''
GetPowerSet(0, aTuple, oneEle)
```

#### 查找最大的k个数

```
// 使用选择排序
def FindKMax_BySelect(aList,k):
   for i in range(0,k):
       maxValue_Index=i
       for j in range(i,len(aList),1):
           if(aList[maxValue_Index]<aList[j]):</pre>
               maxValue_Index=j
       if(maxValue_Index!=i):
           aList[maxValue_Index], aList[i]=aList[i], a
// 使用冒泡排序
def FindKMax_ByBubble(aList,k):
   for i in range(0,k):
       for j in range(i,len(aList)-i-1,1):
           if(aList[j]>aList[j+1]):
               aList[j],aList[j+1] = aList[j+1],aLis
#查找最大的K个数,正常的冒泡排序,从小到大,最大的K个数沉到最下
#反向排序,只需要修改if(aList[j]<aList[j+1])即可
#但是 要想最大的K个数在前面,需要从后向前,把最大的数沉到前面
def FindKMax_ByBubble_2(aList,k):
   for i in range(0,k):
       for j in range(len(aList)-2,i-1,-1):
           if(aList[j]<aList[j+1]):</pre>
               aList[j],aList[j+1] = aList[j+1],aLis
1 1 1
```

```
#上面算法的时间复杂度是0(n*k),有没有更好的算法呢?
#思考快速排序,从大到小,每次把列表分成两段,前面一段都大于当前
#对快速排序变形,记录分割后的下标,
#如果下标Index大于K,说明,这一次分割,前面一段是Index个最大的
#如果下标Index小于K,说明,这一次分割,前面一段是Index个最大的
#分割成两端,并且返回下标,这里是闭区间
def Partition(aList,left,right):
   i =left
   j =right
   target=aList[i]
   while(i<j):</pre>
       while(i<j and aList[j]<target):</pre>
          j=j-1
       if(i<j):</pre>
          aList[i]=aList[j]
          i=i+1
       while(i<j and aList[i]>target):
          i=i+1
       if(i<j):
          aList[j]=aList[i]
          j=j-1
   aList[i]=target
   return i
def FindKMax_ByQuickSort(aList,left,right,k):
   #每次分割后下标
   index=0
   #相当于这一次的分割,处于第几个位置,取决于当前分割的left
   n=0
   if(left<right):</pre>
       index=Partition(aList,left,right)
       n=index-left+1
```

```
# 分割的刚刚好
      if(n==k):
          return index
      # 前面一段分割的大数太多了,继续分割前一段,去掉小的数
      elif(n>k):
          return FindKMax_ByQuickSort(aList,left,ir
      # 前面一段分割的大数太少了,不够k个,在后面一段中,再
      else:
          return FindKMax ByQuickSort(aList,index+1
# 统计比x大的元素个数
def FindNumBigThanX(aList,x):
   num=0
   for i in range(len(aList)):
      if (aList[i]>x):
          num+=1
   return num
1 1 1
查找出最大的k个数,我只有找到一个数x,在集合中比x大的个数是k就
注意:这里不要求集合中一定存在元素x,举例来说,集合5,3,200,1,
我们知道第3大的数是5,但是我们不需要找出5,找出56,也就是x=56
那么问题来了,如何确定x呢?
思路是我假定x为(max+min)/2,遍历集合,统计比x大的个数。
如果个数>k, 说明我假定的x太小了,
如果个数<k, 说明我假定的x太大了.
这是典型的牛顿迭代问题。
def FindKthMax(aList,k):
   min=0
   max = 1000
   x=(min+max)/2
   num=FindNumBigThanX(aList,x)
   while(num != k-1):
      if(num>k-1): # x假定的太小了
          min=x
```

```
else:
             # x假定的太大了
          max=x
       x=(min+max)/2
       num=FindNumBigThanX(aList,x)
   return x
# 上面的方法求出的x可能不在集合中,只是保证在集合中比x大的个数:
# 如何保证求出的x在集合中呢?
# 上面循环结束的条件是 num != k-1, 也就是比x大的个数满足条件!
# 这里应该修改为 把x限定为很小的一个范围内,对于整数范围是1,5
def FindKthMax_2(aList,k):
   min=0
   max = 1000
   while(max-min>1):
       x=(min+max)/2
       num=FindNumBigThanX(aList,x)
       if(num>k-1): # x假定的太小了
          min=x
       else:
                 # x假定的太大了
          max=x
   return x
def FindKMax_ByKth(aList,k):
   # 找出最大的k个数,先找出第k+1大的数
   x=FindKthMax_2(aList,k+1)
   for i in range(len(aList)):
       if aList[i]>x:
          print aList[i]
```

使用最小堆,思路是建立一个k个元素的最小堆,堆顶是最小值,遍历集如果比堆顶大,说明应该踢出堆顶元素,新的元素进入,重新调整堆。举个例子,公司调整干部序列,目前的干部序列最差的排在前面,遍历员比最差的干部强,踢出最差的干部,当前员工进入干部序列,并重新调整在最前面。

```
def HeapAdjust(aList,curIndex,size):
    lChildIndex=2*curIndex+1
    rChildIndex=2*curIndex+2
    if(lChildIndex<size):</pre>
        if(rChildIndex<size):</pre>
             if(aList[lChildIndex]<aList[rChildIndex])</pre>
                 if(aList[lChildIndex]<aList[curIndex]</pre>
                      aList[lChildIndex],aList[curIndex
                      HeapAdjust(aList, 1ChildIndex, size
             else:
                 if(aList[rChildIndex]<aList[curIndex]</pre>
                      aList[rChildIndex],aList[curInde>
                      HeapAdjust(aList,rChildIndex,size
        else:
              if(aList[lChildIndex]<aList[curIndex]):</pre>
                      aList[lChildIndex],aList[curIndex
                      HeapAdjust(aList, 1ChildIndex, size
def FindKMax_ByHeap(aList,k,bList):
    for i in range(k):
        bList.append(aList[i])
    start=len(bList)/2-1
    for i in range(start, -1, -1):
        HeapAdjust(bList,i,len(bList))
    for i in range(k,len(aList),1):
        if(aList[i]>bList[0]):
             bList[0]=aList[i]
             HeapAdjust(bList, 0, len(bList))
```

#### 牛顿迭代求平方根

```
double GetSqrt(int n)
{
```

```
double left = 0;
double right = n;
double mid = (left+right)/2;

// |x|>0.1 等价于 x>0.1||x<-0.1
while(0.00001< mid*mid-n || mid*mid-n < -0.000001
{
    if(mid*mid > n)
    {
       right = mid;
    }
    else
    {
       left = mid;
    }
    mid = (left+right)/2;
}
return mid;
}
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 变量 常用算法 Algorithm

## 重变

### base64

找出增删改的记录

## 变量说明

#### base64

```
string Base64_Encode(const string& plainText)
{
    string cipherText;
    Base64_Encode(cipherText, (unsigned char *) plair
    return cipherText;
}
// 特别注意,需要考虑解码后的string内部包含取值为0的字符
string Base64_Decode(const string& cipherText)
{
    unsigned char* buf = new unsigned char[cipherText
    memset(buf, 0, cipherText.size());
    int size = 0;
    Base64_Decode(cipherText, buf, size);
    string plainText;
    plainText.resize(size);
    memcpy(&plainText[0], buf, size);
    delete buf;
    buf = NULL;
    return plainText;
}
void Base64_Encode(string& outStr, const unsigned cha
    //编码表
```

```
//编码的时候a[0]=A
const char EncodeTable[]="ABCDEFGHIJKLMNOPQRSTUVV
unsigned char a1;
unsigned char a2;
unsigned char a3;
int groupSize = inSize/3;
for(int i=0; i<groupSize; i++)</pre>
{
    a1 = *inData++;
    a2 = *inData++;
    a3 = *inData++;
    outStr+= EncodeTable[a1 >> 2];
    outStr+= EncodeTable[((a1 << 4) | (a2>> 4)) &
    outStr+= EncodeTable[((a2 << 2) | (a3 >> 6))
    outStr+= EncodeTable[a3 & 0x3F];
}
//对剩余数据进行编码
int mod=inSize % 3;
if(mod==1)
{
    a1 = *inData++;
    outStr+= EncodeTable[(a1 & 0xFC) >> 2];
    outStr+= EncodeTable[((a1 \& 0x03) << 4)];
    outStr+= "==";
}
else if(mod==2)
    a1 = *inData++;
    a2 = *inData++;
    outStr+= EncodeTable[(a1 & 0xFC) >> 2];
    outStr+= EncodeTable[((a1 & 0x03) << 4) | ((\epsilon
    outStr+= EncodeTable[((a2 & 0x0F) << 2)];</pre>
    outStr+= "=";
}
```

```
}
void Base64_Decode(const string& inStr, unsigned char
{
   unsigned char* inData = (unsigned char*)(inStr.c
   int inSize = inStr.size();
   outSize = 0;
   //解码表
   //解码的时候a[A]=0,也就是a[65]=0
   const char DecodeTable[] =
   {
       62, // '+'
       0, 0, 0,
       63, // '/'
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, // '6
       0, 0, 0, 0, 0, 0, 0,
       0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
       13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 2
       0, 0, 0, 0, 0, 0,
       26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 3
       39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 5
   };
   int nValue;
   int i = 0;
   while (i < inSize)</pre>
   {
       if (*inData != '\r' && *inData!='\n')
       {
           nValue = DecodeTable[*inData++] << 18;
           nValue += DecodeTable[*inData++] << 12;
           outData[outSize++]=(nValue & 0x00FF0000)
           if (*inData != '=')
```

```
nValue += DecodeTable[*inData++] << 6</pre>
                 outData[outSize++]=(nValue & 0x0000FF
                 if (*inData != '=')
                 {
                     nValue += DecodeTable[*inData++];
                     outData[outSize++]=nValue & 0x000
                 }
                 else
                 {
                     //printf("Meet 4Th =");
                 }
            }
            else
                 //printf("Meet 3Th =");
            i += 4;
        }
        else// 回车换行,跳过
        {
             inData++;
            i++;
        }
    }
}
```

### 找出增删改的记录

```
/*
算法找出增删改的记录,考虑老的集合为a,b,c,新的集合为b,c,
*/
void NiuAlgorithm::GetOptVec(vector<Student>& oldVec,
{
    // 遍历oldVec,newVec找不到是删除,找到了是修改
    for(vector<Student>::iterator iter = oldVec.begin
```

```
{
        vector<Student>::iterator itq = find(newVec.k
        if(itq == newVec.end())
        {
            Student st = *iter;
            st._Opt = Opt_Del;
            optVec.push_back(st);
        }
        else
        {
            Student st = *itq;
            st.\_Opt = Opt\_Mod;
            optVec.push_back(st);
        }
    }
    // 遍历newVec, oldVec找不到是增加
    for(vector<Student>::iterator iter = newVec.begir
    {
        vector<Student>::iterator itq = find(oldVec.t
        if(itq == oldVec.end())
        {
            Student st = *iter;
            st._{opt} = Opt_{Add};
            optVec.push_back(st);
        }
    }
}
```

参见

# Andy Niu Help 1.0.0.0

首页 模块 排**序算法** Algorithm

# 重变

	冒泡排序	
	插入排序	
	希尔排序	
	选择排序	
	14 > <del>4</del> 44 6 <del>6</del>	
	快速排序	
	## HP ID	
	堆排序	
	们	
	归并排序	

## 变量说明

#### 冒泡排序

```
1、冒泡排序,外部循环,执行趟数,第一趟,对集合相邻两个比较,大
   第二趟,从头部开始,尾部减1,找出次大的数。
2、时间复杂度:
   平均情况【n*n】
   最好情况【n*n】
   最坏情况【n*n】
4、辅助空间:
5、稳定的排序算法
template <typename T>
void BubbleSort(std::vector<T>& vec)
{
   for(int i=0;i<vec.size();++i)</pre>
   {
       for(int j=0;j<vec.size()-i-1;++j)</pre>
       {
           if(vec[j] > vec[j+1])
           {
              Swap(vec[j], vec[j+1]);
       }
   }
}
```

- 1、上面的冒泡排序存在问题,比如已经有序的情况,012345,4
- 2、可以优化,如果这一趟没有执行swap,可以断定第二趟也不会执行s

```
template <typename T>
void BubbleSort_2(std::vector<T>& vec)
{
    bool isNeedContinue = true; // 初始化需要进行下一趟
    for(int i=0;i<vec.size()&&isNeedContinue;++i)</pre>
    {
        isNeedContinue = false; // 假定不需要下一趟
        for(int j=0;j<vec.size()-i-1;++j)</pre>
        {
            if(vec[j] > vec[j+1])
            {
                Swap(vec[j], vec[j+1]);
                isNeedContinue = true; // 存在交换,还
        }
    }
}
```

#### 堆排序

// 堆化的长度

```
1、堆排序分为两步:构建堆和去掉头部,重新调整堆。
2、堆排序的思路:堆是逻辑上的完全二叉树,通过下标确定父子关系,堆化需要知道当前下标和集合的长度,对当前下标和子节点堆化。
3、构建堆,从第一个非叶子结点,开始,向前递进。去除头部,重新调整堆,当前下标是0,长度是原来的长度减1。
4、时间复杂度:平均情况【nlogn】最好情况【nlogn】最好情况【nlogn】
```

```
void HeapAdjust(vector<int>& vec, int curIndex,int le
{
    while(curIndex<len)</pre>
    {
        int leftChildIndex = 2*curIndex+1;
        int rightChildIndex = 2*curIndex+2;
        if(leftChildIndex < len) //左孩子存在
        {
            if(rightChildIndex < len) //右孩子存在
            {
                // 两个孩子中找到一个大的
                if(vec[leftChildIndex] >= vec[rightChildIndex]
                    // 和左孩子比较
                    if(vec[curIndex] >= vec[leftChilc
                    {
                        break;
                    else // 把左孩子提拔上来
                        swap(vec[curIndex], vec[leftCl
                        curIndex = leftChildIndex;
                    }
                }
                else
                {
                    // 和右孩子比较
                    if(vec[curIndex] >= vec[rightChi]
                    {
                        break;
                    else // 把右孩子提拔上来
                    {
                        swap(vec[curIndex], vec[right(
                        curIndex = rightChildIndex;
                    }
```

```
}
           else
           {
               // 和左孩子比较
               if(vec[curIndex] >= vec[leftChildInde
                   break;
               else // 把左孩子提拔上来
                   swap(vec[curIndex], vec[leftChild]
                   curIndex = leftChildIndex;
               }
           }
       }
       else
       {
           break;
       }
   }
}
void HeapSort(vector<int>& vec)
{
   // 从最后一个非叶子节点,向前,堆化
   int start = vec.size()/2 -1; // 最后一个非叶子结点
   for(int i = start; i >= 0; --i)
    {
       HeapAdjust(vec,i,vec.size());
   }
   // 头和尾交换,从头到尾部前一个堆化
   for(int i =0;i< vec.size()-1;++i) // 最后一个元素不
    {
       swap(vec[0], vec[vec.size()-i-1]);
       HeapAdjust(vec, 0, vec.size()-i-1);
```

```
}
```

#### 希尔排序

```
1、对于插入排序,如果已经基本有序,算法效率很高。
2、所谓基本有序,也就是对于整个序列,小的数在前面,中间的数在中
  如何使它基本有序呢?分组,跳跃式的分组。考虑9个元素,分成3
  a[0],a[3],a[6]
  a[1], a[4], a[7]
  a[2],a[5],a[8]
  每一组插入排序,就使得基本有序。然后缩小间隔,直到为1
3、也就是说,希尔排序有三层循环,外层循环缩小间隔,因为是插入排
  中间循环从每个分组的第二个元素开始,遍历a[3],a[4]...,内
  挖前面的数,填后面的坑。
  虽然希尔排序有三层循环,但是效率高。外层循环缩小,下降很快
  interval不是常数,会变化。希尔排序的时间复杂度 小于 n*n
4、时间复杂度:
  平均情况【nlogn到n*n】
  最好情况【n^1.3 】n的1.3次方
  最坏情况【n*n
5、辅助空间:
  1
6、不稳定的排序算法。
void ShellSort(vector<int>& vec)
{
```

```
void ShellSort(vector<int>& vec)
{
    int interval = vec.size();
    do
    {
        interval = interval/3+1;

        for(int i = interval;i<vec.size();++i) // 从年
        {
            int curIndex = i; //记录当前下标
            int target = vec[i];//记录当前元素,挖数,留
            // 没有走到分组的第一个元素,并且目标小
```

#### 归并排序

```
1、归并排序的思路:使用递归,调用MSort, MSort内部使用一个临时
2、时间复杂度:
   平均情况【nlogn】
   最好情况【nlogn】
   最坏情况【nlogn】
3、辅助空间:
4、稳定的排序算法。
// 对于 aVec, 范围[first,mid)和范围[mid,last)都是有序的,
void Merge(vector<int>& aVec, vector<int>& bVec, int fi
{
   int i=first; // 前一段的开始下标
   int j=mid; // 后一段的开始下标
   int k=first; // 目标集合的开始下标
   while(i<mid && j<last) // 两段都没有走到结尾
   {
      if(aVec[i]<aVec[j]) // 前面的小
       {
          bVec[k++] = aVec[i++]; // 从前一段取元素
      else
```

```
bVec[k++] = aVec[j++]; // 从后一段取元素
       }
   }
   // while退出后,只有一段还有剩余元素,哪个段的最大值大,
   // 复制前一段的剩余元素
   while(i<mid)</pre>
   {
       bVec[k++] = aVec[i++];
   }
   // 复制后一段的剩余元素
   while(j<last)</pre>
   {
       bVec[k++] = aVec[j++];
   }
}
// vecRef和vec是同一个集合的引用,把vec的[first,last]归并引
void MSort(vector<int>& srcVec, vector<int>& dstVec, ir
{
   if(first == last-1)
   {
       dstVec[first]=srcVec[first];
   }
   else
   {
       int mid = (first+last)/2;
       // 栈上分配一个临时的集合
       vector<int> tmpVec;
       tmpVec.resize(dstVec.size());
       // vec的[first,mid) 归并到vecCpy的[first,mid)
       MSort(srcVec, tmpVec, first, mid);
```

```
// vec的[mid,last) 归并到vecCpy的[mid,last)
       MSort(srcVec, tmpVec, mid, last);
       // vecCpy的[first,mid)和[mid,last)有序,进行Mer
       Merge(tmpVec, dstVec, first, mid, last);
   }
}
void MergeSort(vector<int>& vec)
{
    MSort(vec, vec, 0, vec.size());
}
1、使用递归,导致空间上的浪费,每次都要一个临时对象。
2、递归的思路是先自上而下,再自下而上。
3、能不能直接自下而上,相邻合并,再相邻合并。
void MergePass(vector<int>& srcVec, vector<int>& dstVe
{
    int i = 0;
    int len = dstVec.size();
   while(i< len-2*interval+1)</pre>
    {
       Merge(srcVec, dstVec, i, i+interval, i+2*interval
       i = i+2*interval;
    }
    if(i<len-interval) // 最后剩余2个序列,长度不同
    {
       Merge(srcVec, dstVec, i, i+interval, dstVec.size(
    else // 最后剩余1个序列
    {
       for(int j = i; j < len; ++j)
        {
           dstVec[j] = srcVec[j];
    }
```

```
void MergeSort_2(vector<int>& vec)
{
  vector<int> tmpVec;
  tmpVec.resize(vec.size());

int i = 1;
  while(i<vec.size())
  {
    MergePass(vec,tmpVec,i); // 把vec相邻,间隔为1的
    i*=2;
    MergePass(tmpVec,vec,i); // 把tmpVec相邻,间隔;
    i*=2; // 为下一趟做准备
  }
}
</pre>
```

#### 快速排序

- 1、快速排序,调用递归方法。因为是折半递归,效率并不差。如果是线
- 2、快速排序思路:递归调用,前后相遇,结束递归。没有相遇,先从前一个循环里,后面挖个数填前面的坑,然后前面挖个数,填后面的相遇之后,跳出循环,填上遗留的坑,对前后两块递归调用。
- 3、时间复杂度:

```
平均情况【nlogn】
```

最好情况【nlogn】,每次都刚好分成均匀的两块

最坏情况【n\*n 】,每次都分成极端不均衡的两块,1和n-1个

4、辅助空间:

递归造成的栈空间使用,递归的深度最好情况 logn,最坏情况n,

5、比较和交换是跳跃式的,因此是不稳定的排序算法。

```
template <typename T>
void QSort(std::vector<T>& vec,int left,int right)
{
  int i = left; // 左边
```

```
int j = right-1;// 右边
   if(i >= j) // 已经相遇,结束递归
   {
       return;
   }
   T target(vec[i]); // 从前面挖个数,留下一个坑
   while(i<j) // 前后夹击,直到相遇
   {
       while(i<j && vec[j]>target) // 从后向前,找出一
       {
           --j;
       if(i<j) // 没有相遇
       {
           vec[i++] = vec[j]; //从后面挖个数,填上前面的
       }
       while(i<j && vec[i]<target) // 从前向后,找出一
       {
           ++i;
       }
       if(i<i) // 没有相遇
       {
           vec[j--]=vec[i]; //从前面挖个数,填上后面的坑
       }
   }
   vec[i] = target; // 填上遗留的坑
   QSort(vec, left, i);
   QSort(vec, i+1, right);
}
template <typename T>
```

```
void QuickSort(std::vector<T>& vec)
{
    QSort(vec,0,vec.size());
}
```

#### 插入排序

- 1、插入排序,前面的元素有序,当前元素进行插入。外部循环,记录当对于前面有序的子序列,从后向前,找出比它小的,比它大,从前最后填上遗留的坑。
- 2、时间复杂度:

```
平均情况【n*n】
```

最好情况【n 】,已经有序,每次都不需要从前面挖数,填后面的最坏情况【n\*n】,倒序排列,每次都要从前面挖数,填后面的坑

3、辅助空间:

1

4、稳定的排序算法。

#### 选择排序

```
1、选择排序,外部循环,假定当前下标就是最小值的下标,内部循环从
2、内部循环退出,和假定不一致,进行交换。
3、时间复杂度:
   平均情况【n*n】
   最好情况【n*n】
   最坏情况【n*n】
   因为,假定当前下标的元素就是最小值,必须遍历内部循环才确定。
4、辅助空间:
5、稳定的排序算法。
template <typename T>
void SelectSort(std::vector<T>& vec)
{
   for(int i =0;i<vec.size();++i)</pre>
   {
       int minIndex = i; // 假定当前下标就是最小值的下标
       for(int j =i+1;j<vec.size();++j)</pre>
          if(vec[minIndex]>vec[j])
          {
              minIndex = j; // 找出最小值的下标
       }
       if(minIndex != i) // 和假定不一致,交换元素
       {
          Swap(vec[minIndex], vec[i]);
       }
   }
}
```

# 模块

二叉树的遍历

各种树

# 重变

二叉树镜像

### 变量说明

#### 二叉树镜像

```
void MirrorTree(TreeNode* node)
{
    if(node == NULL)
    {
        return;
    }
    if(node->_Left != NULL)
        if(node->_Right != NULL) // 有两个孩子
        {
            TreeNode* tmp = node->_Left;
            node->_Left = node->_Right;
            node->_Right = tmp;
        else // 只有左孩子
        {
            node->_Right = node->_Left;
            node->_Left = NULL;
    }
    else
    {
        if(node->_Right != NULL) // 只有右孩子
            node->_Left = node->_Right;
```

```
node->_Right = NULL;
}
else // 没有孩子
{
    // do nothing
}

MirrorTree(node->_Left);
MirrorTree(node->_Right);
}
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

首页 模块

二叉树的遍历
Algorithm » 数据结构\_树

### 变量

前序(递归)

前序(利用栈向左走到底,去掉栈顶,从右节点开始)

前序(利用栈和链表,入栈先右后左,链表每次从栈顶取元素)

中序(递归)

中序(利用栈向左走到底,去掉栈顶,从右节点开始)

后序(递归)

后序(利用栈A和辅助栈B,每次栈A向右走到底)

后序(利用栈A和辅助栈B,栈A先左后右入栈)

## 变量说明

中序(利用栈向左走到底,去掉栈顶,从右节点开始)

```
void inOrder_2(Node* root)
{
    stack<Node*> stk;
    Node* cur = root;
   while(stk.size() >0 || cur != NULL)
    {
        while(cur != NULL)
        {
            stk.push(cur);
            cur = cur->left;
        }
        // 与前序的区别是:打印元素的时机不同
        if(stk.size() >0)
        {
            cur = stk.top();
            printf("%d-", cur->value);
            stk.pop();
            cur = cur->right;
        }
    }
}
```

参见

#### 中序(递归)

```
void inOrder(Node* root)
{
    if(root != NULL)
    {
        inOrder(root->left);
        printf("%d-",root->value);
        inOrder(root->right);
    }
}
```

参见

### 前序(利用栈向左走到底,去掉栈顶,从右节点开始)

```
void preOrder_2(Node* root)
{
    stack<Node*> stk;
    Node* cur = root;
    while(stk.size() >0 || cur != NULL)
    {
        while(cur != NULL)
        {
            printf("%d-", cur->value);
            stk.push(cur);
            cur = cur->left;
        }
        if(stk.size() >0)
        {
            cur = stk.top();
            stk.pop();
            cur = cur->right;
        }
```

```
}
}
```

参见

#### 前序(利用栈和链表,入栈先右后左,链表每次从栈顶取元素)

```
void preOrder_3(Node* root,list<Node*>& nodeList)
{
    stack<Node*> stk;
    Node* cur = root;
    stk.push(cur);
    while(stk.size() >0)
    {
        cur = stk.top();
        nodeList.push_back(cur);
        stk.pop();
        if(cur->right != NULL)
        {
            stk.push(cur->right);
        }
        if(cur->left != NULL)
        {
            stk.push(cur->left);
        }
    }
}
```

参见

#### 前序(递归)

```
void preOrder(Node* root)
```

```
if(root != NULL)
{
    printf("%d-",root->value);
    preOrder(root->left);
    preOrder(root->right);
}
```

参见

### 后序(利用栈A和辅助栈B,栈A先左后右入栈)

```
void postOrder_3(Node* root, stack<Node*>& nodeStk)
{
    stack<Node*> stk;
    Node* cur = root;
    stk.push(cur);
    while(stk.size() >0)
    {
        cur = stk.top();
        nodeStk.push(cur);
        stk.pop();
        if(cur->left != NULL)
        {
            stk.push(cur->left);
        }
        if(cur->right != NULL)
            stk.push(cur->right);
        }
    }
}
```

#### 后序(利用栈A和辅助栈B,每次栈A向右走到底)

```
// 利用栈A和辅助栈B,每次栈A向右走到底,加入栈B,去除栈A的栈顶
// 栈B的顺序中 右 左, 出栈刚好左 右 中
void postOrder_2(Node* root, stack<Node*>& postStk)
{
   stack<Node*> stk;
   Node* cur = root;
   while(stk.size() >0 || cur != NULL)
    {
       while(cur != NULL)
       {
           stk.push(cur);
           postStk.push(cur);
           cur = cur->right;
       }
       if(stk.size() >0)
       {
           cur = stk.top();
           stk.pop();
           cur = cur->left;
       }
   }
}
```

参见

#### 后序(递归)

```
void postOrder(Node* root)
{
   if(root != NULL)
```

```
{
    postOrder(root->left);
    postOrder(root->right);
    printf("%d-",root->value);
}
```

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

首页 模块 变量 各种树 Algorithm » 数据结构\_树

# 重变

完全二叉树

### 详细描述

### 变量说明

#### 完全二叉树

- 1、满二叉树:每一层都是满的
- 2、完全二叉树:除了最后一层,其它层都是满的,如果最后一层不满,
- 3、拥有孩子节点个数成为当前节点的度,完全二叉树的性质:n0 = n2
- 4、n0:度为0的节点个数,n1,n2同理,n是节点总数。n1 = 0或者1 考虑一个深度为3的满二叉树,n0为4,n2为3,使用归纳法,最后 从右边叶子节点再去掉一个,n2也减少1,满足条件。
- 5、已知完全二叉树的总结点个数为 699,求叶子节点个数。 当然可以使用笨的办法,假设最后一层为x,则 2^(n-1)-1+x = 节点数为2^(n-2)-(x-1)/2+1,二者相加。 使用完全二叉树的性质:n0 = n2+1,则n0+n1+n2=n,也就是2 进一步推算,n0=(n-1)/2+1,也就是y/x + y%x==0?0,1 等价
- 6、n1取值为0或者1,而 2n0-1+n1=n,2n0 = n+1-n1,得出n1 与

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

黄页 模块 数据结构\_链表 Algorithm

# 模块

单向链表

# 详细描述

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

首页 模块 变量 单向链表 Algorithm » 数据结构\_链表

# 重变

删除节点

### 变量说明

#### 删除节点

```
struct ListNode
{
   int _Value;
ListNode* _PNext;
   ListNode():_Value(0),_PNext(NULL)
   {
   }
   ListNode(int value, ListNode* pNext):_Value(value)
   {
   }
};
/*
   1、头结点需要使用指针的指针,因为:如果直接使用指针,并且删
       被调用方法内,头结点变化了,而调用者的头结点没有变化
   2、注意有效性检查
   3、注意删除头结点
*/
void DeleteNode(ListNode** pPHead,ListNode* delNode)
{
   // 有效性检查
   if(pPHead == NULL || delNode==NULL)
   {
       printf("param error");
```

```
return;
   }
   ListNode* curNode = *pPHead;
   if(curNode == delNode) //删除头结点,特殊处理一下
   {
       *pPHead = delNode-> PNext;
       delete delNode;
       delNode = NULL;
       return;
   }
   while(curNode!=NULL)
   {
       if(curNode-> PNext == delNode)
       {
          ListNode* nextNode = delNode->_PNext;
          curNode->_PNext = nextNode;
          delete delNode;
          delNode = NULL;
          break; // 找到之后, break
       curNode = curNode->_PNext;
   }
}
/*
   1、DeleteNode遍历节点,找到目标,然后删除,时间为O(n)
   2、有没有更好的办法呢?
       思路是:找到待删除节点的下一个节点,捏到一起,
       下一个节点覆盖待删除节点,然后删除下一个节点
   3、上面的思路,有一种情况需要特殊对待。
       删除的是尾节点,没有下一个节点,不能拿下一个节点覆盖
   4、注意:这种思路有一个假设条件,需要调用者保证:那就是链表
* /
void DeleteNode_2(ListNode** pPHead,ListNode* delNode
{
```

```
// 有效性检查
   if(*pPHead == NULL || delNode==NULL)
    {
        printf("param error");
        return;
   }
   ListNode* curNode = *pPHead;
   if(curNode == delNode) //删除头结点,特殊处理一下
   {
        *pPHead = delNode->_PNext;
       delete delNode;
       delNode = NULL;
       return;
   }
   // 链表至少包含两个节点
   if(delNode->_PNext == NULL) // 删除尾节点
   {
       while(curNode-> PNext !=NULL)
        {
           if(curNode->_PNext->_PNext == NULL)
            {
               delete delNode;
               curNode->_PNext = NULL;
               break; // 找到之后, break
           curNode = curNode->_PNext;
        return;
   }
   ListNode* nextNode = delNode->_PNext;
   *delNode = *nextNode;
   delete nextNode;
}
```

```
void PrintListNode(ListNode* pFirst)
{
    while(pFirst!=NULL)
    {
        printf("%d ",pFirst->_Value);
        pFirst=pFirst->_PNext;
    }
    printf("\n");
}
void Test()
{
    ListNode* n6 = new ListNode(6, NULL);
    ListNode* n5 = new ListNode(5,n6);
    ListNode* n4 = new ListNode(4,n5);
    ListNode* n3 = new ListNode(3,n4);
    ListNode* n2 = new ListNode(2, n3);
    ListNode* n1 = new ListNode(1, n2);
    PrintListNode(n1);
    DeleteNode 2(&n1,n6);
    PrintListNode(n1);
    ListNode* curNode = n1;
    ListNode* next=NULL;
    while(curNode!=NULL)
    {
        next=curNode-> PNext;
        delete curNode;
        curNode=next;
    curNode=NULL;
}
```

首页 模块 模块 模块 | 变量 编程珠玑 Algorithm

# 模块

随机数

## 变量

一亿整数排序

全排列

变位词

循环移位

过滤重复字符并排序

### 变量说明

#### 一亿整数排序

- 1、考虑下面的需求,对文件中的1亿个32位整数排序,这些整数不重复,使用new int[1亿],从文件中读取所有的整数,放入数组,排序,怎么解决这个问题?
- 2、使用位向量表示整数,也就是第n个bit,表示整数n,bit的取值为是而上面的方法,整数需要4个字节,32bit表示。因此,位向量只用注意:如果重复次数不超过3,使用两个bit表示一个整数。也就是这两个bit的取值表示出现次数。
- 3、为了简化问题,考虑30个整数,使用位向量排序。 new出32bit的内存,也就是4个字节。从文件中遍历读取整数,在 遍历位向量,输出bit值为1的位置。
- 4、位向量也就是,第n个bit表示整数n,bit的取值为整数n出现的次数
- 5、考虑,如果文件中的数字更多,比如说有40亿,怎么处理? 多趟处理,第一趟遍历40亿整数,过滤出大于等于0,小于1亿[0,1 第二趟遍历40亿整数,过滤出大于等于1亿,小于2亿[1亿,2亿),

#### 头文件

```
#ifndef PEARLS_H_
#define PEARLS_H_
typedef unsigned char uint_8;

namespace Pearls
{
    void SetBitValue(uint_8* start,int index,int valuint GetBitValue(uint_8* start,int index);
}
```

#endif

#### 源代码

```
#include "pearls.h"
namespace Pearls
{
   void SetBitValue(uint_8* start,int index,int value)
    {
       int bytePos = index/8; // 第几个字节
       int bitPos = index%8; // 字节内的第几个bit
       start+=bytePos;
       if(value == 0)
       {
           (*start)&=(~(1<<bitPos)); // 把某一个bit置。
       else if(value == 1)
       {
           (*start)|=(1<<bitPos); // 把某一个bit置成
       }
   }
   int GetBitValue(uint_8* start,int index)
   {
       int bytePos = index/8; // 第几个字节
       int bitPos = index%8; // 字节内的第几个bit
       // 做个副本,转化为unsigned,再移位
       uint_8 ch = *(start+=bytePos);
       return ((ch)&=(1<<bitPos))>>bitPos;
   }
}
```

```
#include "pearls.h"
#include <stdio.h>
#include <string.h>
#include <fstream>
void ReadFileAndSetBitVec(uint_8* bitVec)
{
    FILE* pf = fopen("./ints.txt","r");
    char readBuf[4] = \{0\};
    char* pBuf = readBuf;
    while(feof(pf) == false)
    {
        if((*pBuf++ = fgetc(pf)) == ' ')
        {
            int value = atoi(readBuf);
            Pearls::SetBitValue(bitVec, value, 1);
            memset(readBuf, 0, 4);
            pBuf = readBuf;
        }
    }
}
void PrintBitVec(uint_8* bitVec)
{
    for(int i = 0; i < 32; ++i)
    {
        int dd = Pearls::GetBitValue(bitVec,i);
        if(Pearls::GetBitValue(bitVec,i) == 1)
        {
            printf("%d ",i);
        }
    }
}
int main(int argc, char* argv[])
{
```

```
uint_8* bitVec = new uint_8[4];
memset(bitVec,0,4);

ReadFileAndSetBitVec(bitVec);

PrintBitVec(bitVec);

getchar();
return 0;
}
```

#### 全排列

- 1、考虑字符串1234,输出全排列,如下:1234,1243,1324,1342,1
- 2、思路一, 初始左边为"", 右边为1234, 递归结束条件是右边只有一个
- 3、思路二,全排列,最小是1234,最大是4321,从1234开始,找出下 一直找到4321结束。

```
void PrintTotalOrder(const string& str)
{
    //PrintOrder("",str);
    PrintOrder_2(str);
}

void PrintOrder(const string& prefix,const string& ot
{
    if (other.size()==1)
    {
        //如果后缀只剩一个字符,输出当前顺序,递归结束
        printf("%s%s\n",prefix.c_str(),other.c_str())
    }
    else
    {
        //如果后缀长度大于1,每次把后缀中的一个字符添加到前缀
        for (int i=0;i<other.size();i++)
        {
        // **The control of the control of
```

```
PrintOrder(prefix+other.substr(i,1),strir
       }
   }
}
void PrintOrder_2(const string& str)
   string cpy = str;
   do
    {
       printf("%s\n",cpy.c_str());
   }while(Next(cpy));
}
bool Next(string& str)
{
   int len = str.size();
   // 从后向前,找到第一个位置,这个位置的元素小于下一个位置的
   int pos = len-2; // 假定是倒数第二个元素
   while(pos>=0 && str[pos]>str[pos+1]) // 不满足条件
   {
       --pos;
   }
   if(pos==-1) // 位置不存在,已经是逆序排列,取值最大
   {
       return false;
   else
   {
       // 从后面找出一个比当前下标大的元,这个backpos必定有
       int backPos = len-1;
       while(backPos>0 && str[backPos] < str[pos])</pre>
       {
           --backPos;
       }
```

```
Swap(str[pos], str[backPos]);
        ReverseString(str,pos+1,str.size());
        return true;
    }
}
// 对string的left和right区间反转
void ReverseString(string& str,int left,int right)
{
    if(left>right)
    {
        Swap(left, right);
    if(right>str.size())
    {
        return;
    }
    int mid = (left+right)/2;
    for(int i = left;i<mid;++i)</pre>
    {
        Swap(str[i],str[--right]); // left下标加1,rig
    }
}
int main(int argc, char* argv[])
{
    Pearls::PrintTotalOrder("1234");
    getchar();
    return 0;
}
```

#### 变位词

1、考虑下面的需求,给出包含单词的文档,找出其中的变位词。所谓的

- 2、最笨的办法是, dog不动, 对god, cat进行排列组合, 看看是否存在
- 3、上面的办法效率太差,很长的单词,排列组合很多,cpu开销太大。
- 4、变位词的特点是,大家所包含的字符相同,只不过排列组合不同。包标识符按字符顺序排列好,互为变位词,也就是说标识符相同。因此较标识符,相同就是变位词。
- 5、使用map, key是标识符(可能是一个无效的单词), value是vect
- 6、更广泛的用处,比如使用搜狗输入法输入 ty,就会提示 统一,同意 统一,同意 , 同义求标识符,放入map的key, value保存这些词。

```
string GetIdentifier(const string& word)
    char buf[64] = \{0\};
    sprintf(buf,word.c_str());
    gsort(buf, strlen(buf), sizeof(char), CharCmp);
    return buf;
}
int CharCmp(const void* a,const void* b)
{
    return *(char*)a - *(char*)b;
}
bool IsSeparator(char ch)
{
    return (ch == ' ' || ch == ',' || ch == '.'||
}
void GetWord(vector<string>& wordVec,char* buf)
{
    buf[strlen(buf)-1] = '\0'; //去除分隔符
    if(strlen(buf) > 0)
    {
        wordVec.push_back(buf);
    }
}
```

```
void ReadWordFromFile(vector<string>& wordVec)
    {
        FILE* pf = fopen("./words.txt","r");
        char readBuf[64] = \{0\};
        char* pBuf = readBuf;
        while(feof(pf) == false)
        {
            *pBuf = fgetc(pf);
            if(IsSeparator(*pBuf))
            {
                GetWord(wordVec, readBuf);
                memset(readBuf, 0, 64);
                pBuf = readBuf;
            }
            else
            {
                ++pBuf;
        }
int main(int argc, char* argv[])
{
    vector<string> wordVec;
    Pearls::ReadWordFromFile(wordVec);
    map<string, vector<string> > wordMap;
    for(vector<string>::iterator iter = wordVec.begir
        iter != wordVec.end();++iter)
    {
        wordMap[Pearls::GetIdentifier(*iter)].push_ba
    }
    getchar();
    return 0;
}
```

- 1、循环移位,对于abcdefgh,向左循环移动3位,成为defghabc,结
- 2、一场考虑,字符指针为NULL,对于8个字符,左移11位等价于左移3
- 3、方法一:从头部取出一个字符,后面的依次左移,取出的字符放入尾
- 4、方法一的cpu开销大,方法二是:分配n个字节的缓冲区,从头部取出字符放入尾部。
- 5、方法二的内存开销大,需要多分配n字节字符。方法三,利用循环左利后面的字符反转,然后整个字符串反转。如下:abcdefgh-->cba

```
void LoopMove(char* pc,int moveSize)
    if(pc == NULL)
    {
        return;
    int len = strlen(pc);
    if(len <= 1)
    {
        return;
    }
    moveSize = moveSize%len;
    // 每次左移一位,循环n次
    //for(int i=0;i<moveSize;++i)</pre>
    //{
    // char first = pc[0];
    // for(int j=0;j<len-1;++j)</pre>
    //
    //
            pc[j]=pc[j+1];
    // }
    //
        pc[len-1]=first;
    //}
    // 一次左移n位
    //char* buf = new char[moveSize];
```

```
//memset(buf, 0, moveSize);
    //memcpy(buf,pc,moveSize);
    //for(int j=0;j<len-moveSize;++j)</pre>
    //{
    //
        pc[j]=pc[j+moveSize];
    //}
    //memcpy(pc+len-moveSize, buf, moveSize);
    // 反转三次
    Reverse(pc,pc+moveSize);
    Reverse(pc+moveSize,pc+len);
    Reverse(pc, pc+len);
}
void Reverse(char* first,char* last)
{
    int len = last-first;
    for(int i=0;i<len/2;++i)</pre>
    {
        char tmp = first[i];
        first[i] = first[len-i-1];
        first[len-i-1]=tmp;
    }
}
{
    char buf[] = "abcdefgh";
    Pearls::LoopMove(buf,3);
    Pearls::LoopMove(buf,3);
    Pearls::LoopMove(buf, 4);
    Pearls::LoopMove(buf,81);
    getchar();
```

```
return 0;
}
```

#### 过滤重复字符并排序

```
void GetCharCountByMap(const string& str)
{
    map<char,int> chMap;
    for(int i = 0;i<str.size();++i)</pre>
    {
        chMap[str[i]]++;
    }
    for(map<char,int>::iterator iter = chMap.begin();
        iter != chMap.end(); ++iter)
    {
        printf("[%c:%d]\n",iter->first,iter->second);
    }
}
void GetCharCountByHashTable(const string& str)
{
    // 最多512个字符,下标对应字符,下标取值对应出现的次数
    int hashTable[512]={0};
    for(int i = 0; i < str.size(); ++i)
    {
        hashTable[str[i]]++;
    }
    for(int i=0; i<512; ++i)
    {
        if(hashTable[i]>0) // 出现过
        {
            printf("[%c:%d]\n",i,hashTable[i]);
```

```
}
}
int main(int argc, char*argv[])
{
    string str = "acbbdkdehfghgefahb";
    Pearls::GetCharCountByMap(str);
    Pearls::GetCharCountByHashTable(str);
    return 0;
}
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 **『造机数 Algorithm** » 编程珠玑

# 重变

产生随机数

取样问题

### 变量说明

#### 产生随机数

- 1、产生随机数[0,5),也就是说得到数字0,1,2,3,4的概率都是1/5,根据这个方法,构造方法Random\_5\_By\_Random\_7,怎么办?
- 2、这是一个大范围的随机数,像小范围映射,解决办法很简单,就是对 Random\_7产生[0,7)每个数字的概率都是1/7,如果直接对5取模 是1/7,这显然不满足随机数的要求。
- 3、怎么解决上面的问题? 原因是:大范围向小范围映射,由于不能整除,导致部分数字的概定 比如Random\_5\_By\_Random\_7,如果Random\_7产生的随机数>= 满足要求。
- 4、那这样产生[0,5)每个数字的概率是多少呢?是1/7吗?

错,还是1/5,为什么?从数学角度来看,以0为例,概率为:

第1次产生0:1/7

第2次产生0:2/7\*(1/7) //前面1次产生的是5,6,再 第3次产生0:(2/7)^2\*(1/7) //前面2次产生的是5,6,再

. . . .

总概率 1/7+2/7\*(1/7)+(2/7)^2\*(1/7)... = 1/7\*((2/7) 等比数列求和,设y=1+n+n^2+n^3+....

则ny=n+n^2+n^3+n^4.... 相减,得到

(1-n)y=1-n/m,其中n小于1,m为无穷大,则y = 1/(1-n),n; 产生0的概率为 1/7\*(7/5) = 1/5

- 5、那接着思考, Random\_7\_By\_Random\_5, 该怎么实现呢? 先考虑Random\_10\_By\_Random\_5, 当然不能直接映射。如果直接容易想到的办法是 2\*Random\_5, 这个也不对, 这样的话, 1的概率那么Random\_5+Random\_5, 不对, 0的概率是1/5\*(1/5)=1/25而我们期望的是[1,10)每个数字的概率都是1/10
- 6、怎么解决上面的问题?

必须是从大范围向小范围映射值,同时裁剪掉不满足条件的数字,连续产生两次,范围就是5\*5=25,对7取余,去掉21,22,23,24

```
int Random_5()
{
    static bool isFirstCall = true;
    if(isFirstCall)
    {
        srand(time(NULL));
        isFirstCall = false;
    return rand()%5;
}
int Random_7()
    static bool isFirstCall = true;
    if(isFirstCall)
    {
        srand(time(NULL));
        isFirstCall = false;
    return rand()%7;
}
int Random_5_By_Random_7()
    int a = Random_7();
    while(a > = 5)
    {
        a=Random_7();
    return a;
}
// 想象一个二维数组,第一次产生行数,第二次产生行的第几个元素。
int Random_7_By_Random_5()
```

```
int row = Random_5();
int col = Random_5();
while(row*5+col >= 21)
{
    row = Random_5();
    col = Random_5();
}
return (row*5+col)%7;
}
```

#### 取样问题

```
1、从[0,7)中随机产生两个数,解决办法是调用两次 Random_7,共和
2、考虑,从[0,7)中随机选择两个不重复的数字,这与上面的情况完全
   每个数字被选中的概率是2/7
3、怎么办?
   容易想到的办法是,遍历[0,7),每次产生一个随机数,随机数对
   表示当前数字被选中。如下:
   int Select 2 From 7(vector<int>& intVec)
   {
      intVec.clear();
      for(int i=0;i<7;++i)
         if(Random_7()%7 <2)
         {
            intVec.push_back(i);
      return 0;
4、这个做法有问题,我们要保证intVec的size为2,上面的做法会导致
   因为上面遍历的时候,把每个数字被选的事件当成了互不关联的独。
   考虑,1被选的情况,跟0是否被选有关系:
   ⊙被选 [2/7], 1被选的概率为1/6
   0没有被选[5/7],1被选的概率为2/6
   1被选的总概率为 2/7*(1/6)+5/7*(2/6)=2/7
```

5、思路是:对于0,是从7个里面选,对于1,是从6个里面选,对于2是遍历过程中,分母是一直减小的。每次选中一个,分子减少1

```
int Select_2_From_7(vector<int>& intVec)
    static bool isFirstCall = true;
    if(isFirstCall)
    {
        srand(time(NULL));
        isFirstCall = false;
    }
    intVec.clear();
    int total = 7;
    int select = 2;
    for(int i=0;i<total;++i)</pre>
    {
        if(rand()%(total-i) <select)</pre>
         {
             intVec.push_back(i);
             --select;
        }
    return 0;
}
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

fread的注意事项

宏

获取文件大小的方法

printf打印格式

sprintf不会清空之前的数据

c语言常用的头文件

非标准C库函数的实现

### 详细描述

### 变量说明

### c语言常用的头文件

```
1、<stdio.h> 标准输入输出库
```

2、<stdlib.h> 标准工具库

#### fread的注意事项

1、函数原型 size\_t fread ( void \*buffer, size\_t size, 参 数

buffer

用于接收数据的内存地址

size

要读的每个数据项的字节数,单位是字节

count

要读count个数据项,每个数据项size个字节.

stream

输入流

- 2、特别注意:size和count,不是表示每次读取多少和读取多少次。 而是只读取一次,但是指定读取每个数据项的大小,以及读取多少 因为只读取一次,fread(buf,1,1024,pFile)和fread(buf,1
- 3、另外一点特别注意:int readed = fread(buf,size,count,p 必定满足条件 readed<=count,因此readed不能表示实际读取
- 4、考虑下面的测试用例

```
#include <stdio.h>
#include <string.h>
```

#include <time.h>

```
int main(int argc,char* argv)
{
    int bufSize = 64;
   char* buf = new char[bufSize];
   memset(buf, 0, bufSize);
   FILE* pFile = fopen("aaa.txt","r");
   printf("[%d] Start Read\n",(int)time(NULL));
   while(feof(pFile) == false)
    {
        int readed = fread(buf,1,5,pFile);
       printf("readed[%d]\n", readed);
    }
    printf("[%d] Stop Read\n",(int)time(NULL));
   fclose(pFile);
   getchar();
    return 0;
假定内容是8个字节, int readed = fread(buf, 1, 5, pFile)
[1479693243] Start Read
readed[5]
readed[3]
[1479693243] Stop
最后一次只能读取3个数据项
int readed = fread(buf,5,1,pFile); 打印结果是
[1479693338] Start Read
readed[1]
readed[0]
[1479693338] Stop Read
最后一次只能读取0个数据项,因为一个数据项是5个字节,3个字章
int readed = fread(buf, 3, 3, pFile); 打印结果是
[1479693519] Start Read
```

readed[2]

[1479693519] Stop Read

只够读取一次,去读3个数据项,不够3个,因此实际上只读取了2个 5、因此, int readed = fread(buf, size, count, pFile); rea 但是, 当数据项大小为1的时候, 比如fread(buf, 1, 1024, pFil 因为, fread(buf, 1, 1024, pFile)和fread(buf, 1024, 1, pF 使用int readed = fread(buf, 1, 1024, pFile),用readed 6、对于fwrite, 道理和fread类似。

# printf打印格式

1 打印格式加下:

р

n

%

m

エノ ココトレコロ エハンH 」	•	
字符	对应数据类型   含》	ሂ
d/i	int	接受整数值并将它表示
0	unsigned int	无符号8进制整数(不输
u	unsigned int	无符号10进制整数
x/X	unsigned int	无符号16进制整数,x
f(lf)	float(double)	单精度浮点数用f,双精
e/E	double	科学计数法表示的数 ,
g/G	double	有效位数,如:%8g表:
С	char	字符型。可以把输入的
s/S	char*/wchar_t*	字符串。输出字符串中

- 2、int和小于int的整数都使用%d
- 3、%c和%d的区别是:对于整数65,%c打印出来【A】,%d打印出来【€
- 4、%ld是指4个字节整数,%lld是指8个字节整数

void \*

int \*

无输入

无

5、怎么理解printf的打印格式?

打印格式其实就是,告诉计算器把后面的参数当成什么类型来解释。

以16进制形式输出指针

到此字符之前为止,一

打印errno值对应的出

不进行转换,输出字符'%

- 6、因此这会存在不匹配的问题,比如参数a的真实类型是int,前面使用 这会存在严重的问题,首先a打印出来不正确,其次,1f是8个字节
- 7、整数不能当成浮点数解释,一方面是它们的表示方式不同,另一方面
- 8、对于整数, int也不能使用%lld, 因为字节个数不同。

但是有一个特殊, int以及比int小的整数(char、short)都可 可以认为,中间过渡了一下,有个临时变量,强转为int,来解释。

- 9、注意%d和%u的区别,对于最大的int值 0xfffffffff 当成%d打印是 -1,当成%u打印是4294967295(42亿)
- 10、关于%f和%lf的问题

C++是非常微妙的语言,%f和%lf对于printf()和scanf()的效果事实上,对于printf(),无论是%f还是%lf,效果都是一样的。因为遇到float,printf()会将float类型自动提升到double,实际上,对于printf()的%lf,虽然很多编译器接受,但严格地设而对于scanf(),由于接受的是指针,并没有类型提升的说法,所

### sprintf不会清空之前的数据

```
1、测试代码,如下:
    #include <stdio.h>
    int main()
    {
             char tmp[6]={0};
             sprintf(tmp, "%s", "abcde");
             sprintf(tmp, "%s", "123");
             return 0;
2、测试
    [root@localhost niu5]# g++ -g -o main main.cpp
    [root@localhost niu5]# gdb main
    GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-23.
    Copyright (C) 2009 Free Software Foundation, Inc.
    License GPLv3+: GNU GPL version 3 or later <a href="http:">http:</a>
    This is free software: you are free to change and
    There is NO WARRANTY, to the extent permitted by
    and "show warranty" for details.
    This GDB was configured as "i386-redhat-linux-gnu
    For bug reporting instructions, please see:
    <a href="http://www.gnu.org/software/gdb/bugs/>...">http://www.gnu.org/software/gdb/bugs/>...</a>
    Reading symbols from /home/niu5/main...done.
    (gdb) b 5
    Breakpoint 1 at 0x8048405: file main.cpp, line 5.
    (qdb) info b
```

```
Disp Enb Address What
Num
        Type
                        keep y 0x08048405 in maj
        breakpoint
1
(gdb) r
Starting program: /home/niu5/main
Breakpoint 1, main () at main.cpp:5
                char tmp[6]=\{0\};
(gdb) n
                sprintf(tmp, "%s", "abcde");
6
(gdb) p tmp
$1 = "\000\000\000\000\000"
(qdb) n
                sprintf(tmp, "%s", "123");
7
(gdb) p tmp
$2 = "abcde"
(gdb) p &tmp
$3 = (char (*)[6]) 0xbfffea2e
(qdb) p *(char*)0xbfffea2e
$4 = 97 'a'
(gdb) p *(char*)0xbfffea2f
$5 = 98 'b'
(gdb) p *(char*)0xbfffea30
$6 = 99 'c'
(gdb) p *(char*)0xbfffea31
$7 = 100 'd'
(gdb) p *(char*)0xbfffea32
$8 = 101 'e'
(gdb) p *(char*)0xbfffea33
$9 = 0 '\000'
(gdb) n
8
                return 0;
(gdb) p tmp
$10 = "123\000e"
(gdb) p *(char*)0xbfffea2e
$11 = 49 '1'
(gdb) p *(char*)0xbfffea2f
$12 = 50 '2'
```

```
(gdb) p *(char*)0xbfffea30
   $13 = 51 '3'
   (gdb) p *(char*)0xbfffea31
   $14 = 0 ' \000'
   (gdb) p *(char*)0xbfffea32
   $15 = 101 'e'
   (gdb) p *(char*)0xbfffea33
   $16 = 0 '\000'
   (gdb)
3、需要说明的是:
   a、加上-g才能使用gdb调试
   b、p *(char*)0xbfffea2e 表示把地址0xbfffea2e强转为cl
4、可以看到, sprintf并没有清空之前的内容。
   这里没有问题,是因为对于文本字符串("abcde"或者"123")末。
   当成字符串来处理,后面的e看不到了。
5、在windows下测试,使用内存查看,也是同样的结果。
```

#### 宏

```
1、例子如下:
   #define PRINT STR(s) printf("%s",s.c str())
   string str = "abcd";
   PRINT STR(str);
   结果如下:abcd
2、现在我期望打印出, str=abcd, 容易想到的解决办法是:
   #define PRINT_STR(s) printf("s" "=" "%s", s.c_str(
   打印出来的结果是 s=abcd, 不是我们所期望的, 想一下, 为什么
   在这里,编译器不认为"s"中s就是前面的s,不进行替换。如果进
3、怎么解决上面的问题?使用另一种方式,也就是#(字符串替换,前/
   #define PRINT_STR(s) printf(#s "=" "%s", s.c_str()
   可以认为,对于#s,编译器对s进行替换,并且在s前后加上双引号
4、考虑下面的情况,
   int token8 = 102;
   PRINT TOKEN(8);
   期望打印出 token8, 容易想到的解决办法是:
   #define PRINT_TOKEN(d) printf("%d",tokend)
```

- 这样是明显错误的,编译器认为tokend是一个整体,不可能只去替5、怎么解决这个问题?要进行替换,必须把d隔离出来,而一旦隔离出多这就要使用##(宏连接符),可以认为## 进行分割,分割后替换 #define PRINT\_TOKEN(d) printf("%d",token##d)
- 6、一个#字符串替换,两个#宏连接符

### 获取文件大小的方法

```
#include <sys/stat.h>
#include <io.h>
#include <fcntl.h>
    GetFieSize(const string& filePath)
int
{
    // linux下的fpos_t是个结构体,不兼容
    FILE* pFile1 = fopen(filePath.c_str(), "r");
    fseek(pFile1, 0, SEEK_END);
    fpos_t fPos = 0;
    fgetpos(pFile1, &fPos);
    fclose(pFile1);
    printf("Method1[%d]",
        fPos);
    FILE* pFile2 = fopen(filePath.c_str(), "r");
    fseek(pFile2, 0, SEEK_END);
    int size = ftell(pFile2);
    printf("Method2[%d]",
        size);
    int pFile3 = open(filePath.c_str(),0_RDONLY);
    int length = filelength(pFile3);
    printf("Method3[%d]",
        length);
    struct stat fileInfo;
    stat(filePath.c_str(),&fileInfo);
    printf("Method4[%d]",
        fileInfo.st_size);
```

```
return fileInfo.st_size;
}
```

#### 非标准C库函数的实现

```
1、下面的这些函数,不是标准的C库函数,在windows下VC有实现,li
   // WINDOWS中有,LINUX中没有的库函数
   #ifndef WIN32
   char*
          strupr(char *string);
   char* strlwr(char *string);
           stricmp(const char *string1, const char '
   int
          strrev(char *string);
   char*
   char*
           itoa(int value, char *string, int radix);
   #endif
2、实现如下:
   #ifndef WIN32
   char* strupr(char *string)
    {
       for (char *p = string; p < string + strlen(st
       {
           if (islower(*p))
           {
               *p = toupper(*p);
           }
       }
       return string;
   }
   char*
           strlwr(char *string)
   {
       for (char *p = string; p < string + strlen(st
       {
           if (isupper(*p))
               *p = tolower(*p);
```

```
}
    return string;
}
        stricmp(const char *string1, const char '
int
{
    int strLen1 = strlen(string1);
    int strLen2 = strlen(string2);
    char *szString1 = new char[strLen1 + 1];
    char *szString2 = new char[strLen2 + 1];
    strcpy(szString1, string1);
    strcpy(szString2, string2);
    strupr(szString1);
    strupr(szString2);
    int nRet = strcmp(szString1, szString2);
    delete[] szString1;
    delete[] szString2;
    return nRet;
}
char* strrev(char *string)
{
    if (!string)
        return NULL;
    int i = strlen(string);
    int t = !(i\%2)? 1 : 0;
    for(int j = i-1, k = 0; j > (i/2 - t); j--)
```

```
char ch = string[j];
        string[j] = string[k];
        string[k++] = ch;
    }
    return string;
}
char* itoa(int value, char *string, int radix)
{
    int rem = 0;
    int pos = 0;
    char ch = '!';
    do
    {
        rem = value % radix;
        value /= radix;
        if (16 == radix)
        {
            if(rem >= 10 && rem <= 15)
            {
                switch(rem)
                case 10:
                    ch = 'a';
                    break;
                case 11:
                     ch = 'b';
                    break;
                case 12:
                     ch = 'c';
                    break;
                case 13:
                     ch ='d';
                    break;
                case 14:
                     ch = 'e';
```

```
break;
                 case 15:
                     ch ='f';
                     break;
                 }
            }
        if('!' == ch)
        {
            string[pos++] = (char)(rem + 0x30);
        else
            string[pos++] = ch;
    }while(value != 0);
    string[pos] = '\0';
    return strrev(string);
}
#endif
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页	模块	
СРР		模块   变量

# 模块

CPP工具	
CPP常用方法	
CPP常见错误	
CPP常见问题	
CPP文件操作	
2112111	
CPP沉思录	
011777003	
ModernCPP	
深度探索CPP对象模型	
/A/文]不分UFF/A]	

## 变量

编码规范

代码换行

静态库与动态库相互调用

### 详细描述

### 变量说明

#### 代码换行

1、字符串太长,换行显示,怎么办?

2、使用反斜杠,如下:

string str = "abcd\
 1234";

注意:反斜杠后面不准有任何字符。下一行开头的制表符不包含在

3、使用双引号,如下:

string str = "abcd"
"1234";

注意:两个字符串会自动拼接,两个字符串之间可以有任意多个空间

NULL:没有值

空字符:"",不包含任何字符

空白字符:有空格字符,制表符,换行符

### 参见

### 编码规范

- 1、不要使用using namespace std; 因为这会名称空间污染。
- 2、接口要暴露,就写在头文件,接口不暴露,就写在cpp文件,并且定

### 参见

### 静态库与动态库相互调用

测试场景, Test, lib1, lib2, dll1, dll2, 分为下面四种情况:

1、Test->lib1->lib2

lib1编译自己的代码,对lib2的部分,只需要lib2的头文件,对 生成Test连接的时候,把lib1的代码实现包含进来,再递归,把] 运行Test的时候,不再需要lib1和lib2。

也就是说,lib2不合并到lib1中,等到exe的时候,一起合并到e

2、Test->lib1->dll2

lib1编译自己的代码,对dll2的部分,只需要dll2的头文件,对 生成Test连接的时候,把lib1的代码实现包含进来,但是,dll2 运行Test的时候,不需要lib1,但是需要dll2【不需要dll的li 也就是说,dll2不合并到lib1中,等到exe的时候,把lib1合并

3、Test->dll1->lib2

dll1编译自己的代码,对lib2的部分,需要lib2的头文件和实现 生成Test连接的时候,dll1中的代码不包含进来,也不再需要lib 【可以这样测试,生成lib2,生成dll1,生成test,删除lib2, 可以成功,说明生成test.exe,连接的时候,根本不需要lib2, 也就是说,静态库会被连接到动态库或者exe中,但是不会连接到过 运行Test的时候,需要dll1,但是不需要lib2

也就是说,lib2合并到dll1中,等到exe的时候,不需要合并dl]

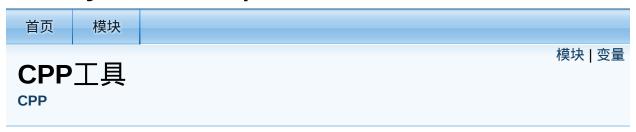
4、Test->dll1->dll2

dll1编译自己的代码,对dll2的部分,需要dll2的头文件和lib 生成Test连接的时候,Test需要dll1的lib,dll1需要dll2的] 运行Test的时候,需要dll1和dll2【不需要他们的lib】

也就是说,dll2不合并到dll1中,等到exe的时候,二者都不需要

参见

# Andy Niu Help 1.0.0.0



# 模块

### Rapidxml

## 重变

C 跨平台定时器

我的日志

控制台进度条

### 详细描述

### 变量说明

### 我的日志

#### 头文件

```
#ifndef NIU_LOG_H_
#define NIU_LOG_H_
#ifdef DLL_FILE
#define DLL_API __declspec(dllexport)
#else
#define DLL_API __declspec(dllimport)
#endif
#include <stdarg.h>
#include <string>
#include <windows.h>
using namespace std;
#define LOG DEBUG
                    1
#define LOG INFO
                    2
#define LOG_WARN
                    4
#define LOG ERROR
                    8
#define LOG FATAL
                    16
#define OUTPUT STD
                        1
#define OUTPUT_CONSOLE
```

```
class DLL_API NiuLog
{
public:
    NiuLog();
public:
    void Debug(const char* file,const char* func,int
    void Info (const char* file, const char* func, int
    void Warn (const char* file, const char* func, int
    void Error(const char* file,const char* func,int
    void Fatal(const char* file,const char* func,int
    void SetLevel(int level);
    void GetLevel(int& level);
    void SetOutput(int output);
    void GetOutput(int& output);
    inline void SetConsoleColor(WORD colorValue);
private:
    inline string getNowTimeAsString();
private:
    int _level;
    int _output;
HANDLE _consoleHandle;
};
#define LogDebug(log,format,...) do\
{\
    log.Debug(__FILE__, __FUNCTION__, __LINE__, format
}while(0)
#define LogInfo(log,format,...) do\
{\
```

```
log.Info(__FILE__, __FUNCTION__, __LINE__, format,
}while(0)

#define LogWarn(log, format, ...) do\
{\     log.Warn(__FILE__, __FUNCTION__, __LINE__, format,
}while(0)

#define LogError(log, format, ...) do\
{\     log.Error(__FILE__, __FUNCTION__, __LINE__, format,
}while(0)

#define LogFatal(log, format, ...) do\
{\     log.Fatal(__FILE__, __FUNCTION__, __LINE__, format,
}while(0)

#endif
```

### 源代码

```
#include "niu_log.h"

#include <stdio.h>
#include <time.h>

#define LOG(x) do\
{\
    if((_output & (OUTPUT_STD+OUTPUT_CONSOLE)) == 0)\
    {\
        return;\
    }\
    if((_level & LOG_##x) == 0)\
    {\
        return;\
```

```
char buf[2048] = \{0\};\
    sprintf(buf,"[" #x " %s %s:%s:%d] ",\
        this->getNowTimeAsString().c_str(),\
        file,\
        func, \
        lineNum);\
    char* p = buf+strlen(buf);\
    va_list ap;\
    va_start(ap, format);\
    p += _vsnprintf(p, sizeof(buf) - 1, format, ap);\
    va end(ap);\
    int len = strlen(buf);\
    buf[len] = '\r';\
    buf[len+1] = '\n';\
    if(( output & OUTPUT STD) >0)\
    {\
        printf(buf);\
    }\
    if(( output & OUTPUT CONSOLE) >0)\
    {\
        ::OutputDebugStringA(buf);\
    }\
} while (0)
NiuLog::NiuLog()
{
    _consoleHandle = GetStdHandle(STD_OUTPUT_HANDLE);
}
void NiuLog::Debug(const char* file,const char* func,
{
    SetConsoleColor(FOREGROUND_GREEN);
    LOG(DEBUG);
}
```

```
void NiuLog::Info(const char* file,const char* func,i
{
    SetConsoleColor(FOREGROUND_RED|FOREGROUND_GREEN|F
    LOG(INFO);
}
void NiuLog::Warn(const char* file,const char* func,i
{
    SetConsoleColor(FOREGROUND_RED|FOREGROUND_GREEN|F
    LOG(WARN);
}
void NiuLog::Error(const char* file,const char* func,
{
    SetConsoleColor(FOREGROUND RED|FOREGROUND INTENS]
    LOG(ERROR);
}
void NiuLog::Fatal(const char* file,const char* func,
{
    SetConsoleColor(FOREGROUND BLUE|FOREGROUND INTENS
    LOG(FATAL);
}
void NiuLog::SetLevel(int level)
{
    _level = level;
}
void NiuLog::SetOutput(int output)
{
    _output = output;
}
string NiuLog::getNowTimeAsString()
{
    char buf[64] = \{0\};
```

```
time_t now = time(NULL);
    //方法1:
    /*tm* ptm = localtime(&now);
    sprintf(buf, "%04d-%02d-%02d %02d:%02d:%02d",
        ptm->tm_year+1900,
        ptm->tm_mon+1,
        ptm->tm_mday,
        ptm->tm_hour,
        ptm->tm_min,
        ptm->tm_sec);*/
    //方法2:
    strftime(buf, 64, "%Y-%m-%d %H:%M:%S",localtime(&
    return buf;
}
void NiuLog::SetConsoleColor(WORD colorValue)
{
    if ( consoleHandle != INVALID HANDLE VALUE)
    {
        SetConsoleTextAttribute(_consoleHandle, color
    }
}
```

## 测试

```
#include "niu_log.h"
#include <string>

using namespace std;

int main(int argc, char* argv[])
{
    NiuLog log;
    log.SetLevel(LOG_DEBUG|LOG_INFO|LOG_WARN|LOG_ERR(log.SetOutput(OUTPUT_STD|OUTPUT_CONSOLE);
```

```
LogDebug(log, "This is Debug");
LogInfo (log, "This is Info");
LogWarn (log, "This is Warn");
LogError(log, "This is Error");
LogFatal(log, "This is Fatal");

getchar();
return 0;
}
```

#### 控制台进度条

```
int len = 50;
string str ="";
for(int i=0; i<len;++i)</pre>
{
    str+="|";
log.SetConsoleColor(FOREGROUND_RED|FOREGROUND_GREEN|F
printf("start...\n");
printf("%s\r", str.c_str()); // 打印竖线, 然后回到行首
str ="";
for(int i=0; i<len;++i)</pre>
    str+=">";
log.SetConsoleColor(FOREGROUND_GREEN|FOREGROUND_INTEN
for (int i = 0; i < len; ++i)
{
    putchar(str[i]);
    Sleep(200);
}
putchar('\n');
printf("end\n");
```

## C跨平台定时器

## 1、头文件

```
#ifndef TIMER_H_
    #define TIMER_H_
    typedef void (*TimerTick)(void* param);
    struct TimerInfo
    {
        TimerTick _TimerTick;
       void* _Param;
       int
                 _Start;
        int
                  _Interval;
        TimerInfo(TimerTick timerTick, void* param, int
        {
            _TimerTick = timerTick;
            _Param = param;
            _Start = start;
            _Interval = interval;
        }
    };
    class Timer
    public:
       Timer(TimerTick timerTick, void* param, int sta
#endif
```

## 2、实现

```
#include "timer.h"
#ifdef WIN32
```

```
#include <windows.h>
#else
#include <pthread.h>
#include <unistd.h>
#endif
void NiuSleep(int ms)
{
    #ifdef WIN32
        ::Sleep(ms);
    #else
        ::usleep(ms*1000);
    #endif
}
#ifdef WIN32
DWORD WINAPI ThreadRun(PVOID param)
#else
void* ThreadRun(void* param)
#endif
{
    TimerInfo* timerInfo = (TimerInfo*)param;
    NiuSleep(timerInfo->_Start);
    while(true)
    {
        timerInfo->_TimerTick(timerInfo->_Param);
        NiuSleep(timerInfo->_Interval);
    }
    return 0;
}
Timer::Timer(TimerTick timerTick, void* param, int
    TimerInfo* timerInfo = new TimerInfo(timerTic
```

```
#ifdef WIN32
HANDLE hThread = CreateThread(NULL,0,ThreadRu
CloseHandle(hThread);
#else
pthread_t hThread;
pthread_create(&hThread,NULL,ThreadRun,(void')
#endif
}
```

#### 3、测试代码

```
#include "timer.h"
#include <stdio.h>

void tick(void* param)
{
    printf("haha\n");
}

int main(int argc,char* argv[])
{
    Timer timer(tick,NULL,5000,1000);
    getchar();
    return 0;
}
```

## 4、编译测试

```
[root@localhost Timer]# 11
total 12
-rw-r--r-- 1 root root 181 Jan 7 17:17 main.cpp
-rw-r--r-- 1 root root 823 Jan 7 17:15 timer.cpp
-rw-r--r-- 1 root root 441 Jan 7 16:47 timer.h
[root@localhost Timer]# g++ -c timer.cpp
```

```
In file included from timer.cpp:1:
timer.h:28:7: warning: no newline at end of file
timer.cpp:48:2: warning: no newline at end of fil
[root@localhost Timer]# g++ -c main.cpp
In file included from main.cpp:1:
timer.h:28:7: warning: no newline at end of file
main.cpp:16:2: warning: no newline at end of file
[root@localhost Timer]# g++ -o main main.o timer.
[root@localhost Timer]# 11
total 28
-rwxr-xr-x 1 root root 6485 Jan
                                 7 17:23 main
-rw-r--r-- 1 root root 181 Jan 7 17:17 main.cpg
-rw-r--r-- 1 root root 1308 Jan 7 17:23 main.o
-rw-r--r-- 1 root root
                        823 Jan 7 17:15 timer.cr
-rw-r--r-- 1 root root
                        441 Jan 7 16:47 timer.h
-rw-r--r-- 1 root root 1856 Jan
                                 7 17:23 timer.o
[root@localhost Timer]# ./main
haha
haha
haha
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 变量

节点加入另一个xml

修改节点的值

解析xml会修改内存的值

打包成string

map的打包和解析

# 变量说明

## map的打包和解析

```
#include "rapidxml/rapidxml.hpp"
#include "rapidxml/rapidxml_utils.hpp"
#include "rapidxml/rapidxml_print.hpp"
#include <stdio.h>
#include <string>
#include <map>
using namespace std;
using namespace rapidxml;
bool ParseXml2Map(map<string, string>& outMap, const st
{
    try
    {
        xml document<char> doc;
        doc.parse<0>(doc.allocate_string(inXml.c_str(
        xml_node<char>* root = doc.first_node("root")
        for(xml_node<char>* node = root->first_node()
        {
            outMap[node->name()] = node->value();
        }
    catch(parse_error& e)
        printf("Parse Xml Err[%s] Where[%s]",
```

```
e.what(),
            e.where<char>());
        return false;
    return true;
}
bool PacketMap2Xml(string& outXml,const map<string,st
{
    try
    {
        xml document<char> doc;
        xml_node<char>* root = doc.allocate_node(node)
        doc.append_node(root);
        for(map<string, string>::const_iterator iter =
        {
            xml_node<char>* ele = doc.allocate_node(r
            root->append_node(ele);
        }
        print(back_inserter(outXml), doc, 0);
    catch (...)
        return false;
    return true;
}
```

## 修改节点的值

```
1、rapidxml修改节点的value,修改之后,序列化还是原来的值,具void TestRapidXml()
{
char* xmlContent = new char[1024];
sprintf(xmlContent,"<root><head>aaa</head><body >
```

```
xml_document<> xmlDoc;
    xmlDoc.parse<0>(xmlContent);
    xml node<>* body = xmlDoc.first node()->first noc
    body->value("ccc");
    xml_attribute <> * x = body -> first_attribute("x");
    x->value("20");
    string xmlStr = "";
    // xmlString为 <root><head>aaa</head><body x=\"20
    // 也就是说, attr的value可以修改成功, 而node的value还是
    rapidxml::print(std::back inserter(xmlStr),xmlDoc
    delete []xmlContent;
}
2、怎么解决上面的问题,笨办法,既然不能修改,我就添加一个新的,
void TestRapidXml()
{
    char* xmlContent = new char[1024];
    sprintf(xmlContent, "<root><head>aaa</head><body >
    xml_document<> xmlDoc;
    xmlDoc.parse<0>(xmlContent);
    xml_node<>* root = xmlDoc.first_node();
    xml node<>* body = root->first node("body");
    xml_node<>* newBody = xmlDoc.allocate_node(node_e
        xmlDoc.allocate_string("body"), xmlDoc.allocat
    // 插入一个新的body
    root->insert_node(body, newBody);
    // 复制老body的attr
    for(xml_attribute<>* attr = body->first_attribute
    {
```

```
xml_attribute<>* copy = xmlDoc.allocate_attri
            xmlDoc.allocate_string(xmlDoc.allocate_st
        newBody->append_attribute(copy);
    // 删除老的body
    root->remove_node(body);
    string xmlStr = "";
    // xmlString为 <root><head>aaa</head><body x=\"10
    rapidxml::print(std::back_inserter(xmlStr), xmlDoc
    delete []xmlContent;
}
3、还有一个办法,就是使用 xmlDoc.parse<parse_no_data_node
void TestRapidXml()
{
    char* xmlContent = new char[1024];
    sprintf(xmlContent, "<root><head>aaa</head><body >
    xml_document<> xmlDoc;
    //xmlDoc.parse<0>(xmlContent);
    xmlDoc.parse<parse_no_data_nodes>(xmlContent);
    xml_node<>* body = xmlDoc.first_node()->first_noc
    body->value("ccc");
    xml_attribute<>* x = body->first_attribute("x");
    x->value("20");
    string xmlStr = "";
    // xmlString为 <root><head>aaa</head><body x=\"20
    rapidxml::print(std::back_inserter(xmlStr),xmlDoc
    delete []xmlContent;
}
```

```
void TestRapidXml()
{
   char* xmlContent = new char[1024];
   sprintf(xmlContent, "<root><head>aaa</head><body>k
   xml_document<> xmlDoc;
   xmlDoc.parse<0>(xmlContent);
   // back_inserter 返回一个输出流,rapidxml的print方法
   string xmlStr = "";
   rapidxml::print(std::back_inserter(xmlStr),xmlDoc
   xmlStr = "";
   xml_node<>* root = xmlDoc.first_node();
   xml_node<>* node = root->first_node()->next_sibli
   internal::print_element_node(std::back_inserter()
   // 注意:要序列化之后,才能释放内存,因为rapidxml为了效率
   // 因此,在序列化的时候还要使用内存的值。
   delete []xmlContent;
}
```

## 节点加入另一个xml

```
void TestRapidXml()
{
    char* xmlContent = new char[1024];
    sprintf(xmlContent,"<root><head>aaa</head><body>k

    xml_document<> xmlDoc;
    xmlDoc.parse<0>(xmlContent);

    char* xmlContent2 = new char[1024];
    sprintf(xmlContent2,"<point x=\"1\" y =\"1\"/><point x=\"1\" y =\"1\" y =\"1\
```

```
xmlDoc2.parse<0>(xmlContent2);

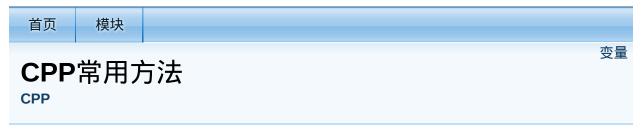
xml_node<>* root = xmlDoc.first_node();
for(xml_node<>* node = xmlDoc2.first_node();node!
{
    // 注意:这里使用xmlDoc,是大的xml,不要使用xmlDoc
    root->append_node(xmlDoc.clone_node(node));
}

string xmlStr = "";
rapidxml::print(std::back_inserter(xmlStr),xmlDoc
delete []xmlContent;
delete []xmlContent2;
}
```

### 解析xml会修改内存的值

- 1、使用rapidxml解析的时候,也就是 调用xmlDoc.parse<0>(xml 特别注意,rapidxml会修改内存的值,把右尖括号>修改为'\0' 因此特别注意,如果char指针指向的内容还要使用,需要做个备份
- 2、如果char指针指向的内容,不再使用,不需要做备份,rapidxml会这个时候不要忘记delete []释放资源。delete释放内存,跟内比如使用new分配内存8个字节,操作系统会多分配一些内存,额外这样在释放的时候,就知道应该释放多少内存。
- 3、有没有其他办法呢? 使用xmlDoc.parse<parse\_non\_destructive>(xmlContent

# Andy Niu Help 1.0.0.0



# 变量

<b>GetNowTimeAsString</b>
---------------------------

时间字符串与unix时间戳的转化

设置系统时间

int与枚举的转化

VS中string实现的小秘密

# 变量说明

### **GetNowTimeAsString**

```
// 字符串表示的当前时间
string NzbUtils::GetNowTimeAsString()
    char buf[64] = \{0\};
    time_t now = time(NULL);
    //方法1:
    /*tm* ptm = localtime(&now);
    sprintf(buf, "%04d-%02d-%02d %02d:%02d:%02d",
        ptm->tm_year+1900,
        ptm->tm_mon+1,
        ptm->tm_mday,
        ptm->tm_hour,
        ptm->tm_min,
        ptm->tm_sec);*/
    //方法2:
    strftime(buf, 64, "%Y-%m-%d %H:%M:%S",localtime(&
    return buf;
}
```

## int与枚举的转化

- 1、int范围比枚举大,从枚举到int一般没有问题,不会超出范围。
- 2、但是从int到枚举,强制转化,有可能会出现问题。 如果int是个有效的枚举值,没有问题。

## 如果int是个无效的枚举值,枚举取值为真实的int值。

```
enum Male
{
    Man,
    Woman,
};
int main(int argc, char* argv[])
{
    Male a = (Male)(0); // a取值为Man
    Male b = (Male)(2); // 无效的枚举值,b取值为真实的
}
```

## VS中string实现的小秘密

```
1、string的大小是32个字节。
2、当string的内容不超过16个字节的时候,string的内容就放在str
3、当string的内容超过16个字节(加上\0)的时候,string的内容;
4、测试代码如下:
    int main()
    {
        string aa;
        int d = sizeof(aa);
        string* ps = &aa;

        // 使用内存窗口,观察ps
        aa = "123456781234567"; // 16个字节(包含\0),
        aa = "1234567812345678"; // 17个字节(包含\0),
        return 0;
    }
```

## 时间字符串与unix时间戳的转化

```
// 由"2016-09-06 14:18:16" 得到 1473142696
int NzbUtils::GetUnixTimeStamp(const string& timeStr)
{
   char year[5] = {0};
```

```
char month[3] = \{0\};
    char day[3] = \{0\};
    char hour[3] = \{0\};
    char min[3] = \{0\};
    char sec[3] = \{0\};
    if(sscanf(timeStr.c_str(), "%4c-%2c-%2c %2c:%2c:%
    {
        return -1;
    }
    struct tm tmLocal = \{0\};
    tmLocal.tm_year = atoi(year) - 1900;
    tmLocal.tm mon = atoi(month) -1;
    tmLocal.tm_mday = atoi(day);
    tmLocal.tm_hour = atoi(hour);
    tmLocal.tm_min = atoi(min);
    tmLocal.tm_sec = atoi(sec);
    return (int)mktime(&tmLocal);
}
// 由1473142696 得到 "2016-09-06 14:18:16"
string NzbUtils::GetTimeStr(int unixTimeStamp)
{
    char buf[64] = \{0\};
    time_t now = (time_t)(unixTimeStamp);
    //方法1:
    //tm* ptm = localtime(&now);
    //sprintf(buf, "%04d-%02d-%02d %02d:%02d:%02d",
    // ptm->tm_year+1900,
    // ptm->tm_mon+1,
    // ptm->tm_mday,
    // ptm->tm_hour,
    // ptm->tm_min,
    // ptm->tm_sec);
```

```
//方法2:
strftime(buf, 64, "%Y-%m-%d %H:%M:%S",localtime(&return buf;
}

特别注意:这里的tm* ptm = localtime(&now); 年份与真实年份(这里偏移的1900与Unix纪元【1970-1-1 0:0:0】中的1970,是两个
```

#### 设置系统时间

```
1、c语言的time()方法,返回从Unix纪元【1970-1-1 0:0:0】至当
   注意:time()只能获取系统时间,不能设置系统时间。
2、使用time()方法获取系统时间,有两种用法:
   time_t aa = time(NULL);
   time_t bb;
   time(&bb);
   当然,下面的方法也可以获取系统时间,cc和dd的取值相同。
   time_t cc;
   time t dd = time(&cc);
3、那么如何设置系统时间呢?
   使用windows的库函数,如下:
   SYSTEMTIME sysTime;
   GetLocalTime(&sysTime);
   sysTime.wYear = 2015;
   sysTime.wMonth = 10;
   sysTime.wDay = 20;
   sysTime.wHour = 14;
   sysTime.wMinute = 38;
   sysTime.wSecond = 30;
   int ret = SetLocalTime(&sysTime);
   int errCode = GetLastError();
4、Linux下面可以使用 date -s命令设置时间。
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 变量

系统类型fd\_set重复定义 Stack overflow NULL指针初始化string 模拟CPP程序崩溃 无法从char指针转换为string引用 不是类或命名空间名称 error LNK2005 vector\_subscript\_out\_of\_range 无法打开文件vld\_lib 已经在LIBCMTD lib中定义 踩内存导致delete的时候崩溃 无法定位程序输入点 无法解析的外部符号 无法打开文件LIBC lib问题 VS调试报错:不是内部或外部命令,也不是可运行的程序或批处理文件 vector重复定义 野指针可能不会立即崩溃,会导致未定义行为

# 变量说明

#### error LNK2005

```
1、错误原因:链接的时候,发现重复定义。
2、比如:在a.cpp和b.cpp中都定义方法如下:
    int doSth(int a)
    {
       return a+1;
    }
3、编译报错,b.obj : error LNK2005: "int __cdecl doSth(链接的时候,发现b.obj中的doSth方法已经在a.obj中定义,重
```

## NULL指针初始化string

```
1、报错:
    terminate called after throwing an instance of 's
2、错误原因:对一个空指针进行操作。如下:
    char* p = NULL:
    string str(p); //运行时报错
```

参见

## Stack overflow

```
错误原因:栈上分配了太多的内存,栈的内存一般只有1M。
错误代码:
char tmp[1024*1024*10];
```

## vector\_subscript\_out\_of\_range

```
1、错误原因, vector下标越界。
2、代码如下:
   int main(int argc, char* argv[])
   {
      vector<int> intVec;
      intVec.push_back(1);
      int ff = intVec[1];
      getchar();
3、VS弹出错误窗口, Debug Assertion Failed!
   这时候,点击重试,触发一个断点,点击中断,在调用堆栈即可找!
   注意:VS弹出错误窗口,千万不能点击中止,否则的话,程序直接
4、另外有几个单词,需要注意一下:
             脚本
   script
   subscript 下标
          抄写员
   scribe
   subscribe 订阅
```

# vector重复定义

```
1、测试代码如下:
    a.h
    #ifndef A_H_
    #define A_H_
    #include "b.h"
    class A
    {
    public:
        int _Id;
        B _B;
};
```

```
#endif
   b.h
   #ifndef B H
   #define B H
   #include <vector>
   using namespace std;
   vector<int> aVec;
   class B
   public:
       int Id;
   };
   #endif
   也就是说,A包含B的头文件,而在B的头文件中定义了一个aVec;
   同时还有a.cpp、b.pp以及main.cpp
   a.cpp
   #include "a.h"
   b.cpp
   #include "b.h"
   main.cpp
   #include "a.h"
   int main(int argc, char* argv[])
   {
       A a;
       return 0;
2、编译报错如下:
   1>a.obj : error LNK2005: "class std::vector<int,c
   1>b.obj : error LNK2005: "class std::vector<int,c
   1>D:\Temp\NiuzbRepository\trunk\C++\CppTest\Debuc
3、错误原因, vs已经说的很清楚, 就是avec重复定义了。
   原因是:在头文件定义aVec,被多个编译单元包含,每个编译单元
   链接的时候,每个编译单元都是外部链接,这么多avec,不知道该
4、表面一看,上面报错的错误比较奇怪,一大堆信息,不容易看清楚,
```

```
我们换成一个int就很清楚了。如下:
b.h

#ifndef B_H_
#define B_H_
int hhh;
class B
{
public:
    int _Id;
};
#endif
报的错误如下:
1>a.obj : error LNK2005: "int hhh" (?hhh@@3HA) 已
1>b.obj : error LNK2005: "int hhh" (?hhh@@3HA) 已
1>D:\Temp\NiuzbRepository\trunk\C++\CppTest\Debug
```

VS调试报错:不是内部或外部命令,也不是可运行的程序或批处理文件

1、错误原因是什么?

我们运行一个程序,会引用其它的库文件或者调用其它的命令,必是如果没有告诉操作系统,对于引用的库文件,报错:无法启动此程对于调用的命令(也就是执行其它的程序),报错:不是内部或外部

2、怎么解决这个问题:

必须告诉操作系统,所依赖的库文件和命令在哪个地方。windows VS调试的时候,在工程属性-->调试-->环境,增加 path=../cc使用批处理运行,在开头增加 path=../common;%path%;

注意:在linux下使用 export LD\_LIBRARY\_PATH=../commo

3、在构建工程的时候,对于编译时引用的头文件和连接时引用的lib文 头文件的设置在:在工程属性-->C/C++-->常规-->附加包含目: lib文件的设置在:在工程属性-->链接器-->常规-->附加库目录

## 不是类或命名空间名称

- 1、今天遇到一个奇怪的问题,使用一个类,并且引用了对应的头文件, 报错error C2653: "ibpUtils": 不是类或命名空间名称。
- 2、仔细思考,为什么? include "xxx.h" 就是对xxx.h进行文本替换。这就存在问题,

谁会很傻地重复引用同一个头文件呢?如下:include "xxx.h"

include "xxx.h"

这种很傻瓜的错误当然可以避免,但是有一种情况,却很难避免, a.h 引用头文件 b.h

c.h 引用头文件 a.h和b.h,由于c.h无法知道a.h已经引用了b.include "a.h"

include "b.h"

编译器对include会递归地进行替换,也就是先替换a.h,对于a.这就导致了b.h的内容重复出现在c.h中。

3、怎么解决这个问题?

使用头文件保护符,如下:

#ifndef \_XXX\_H\_
#define \_XXX\_H\_

. . . . . .

#endif

这就保证了不会重复包含同一个头文件。

- 4、但是,有一点需要特别注意。那就是:头文件保护符不能有重复的宏
- 5、现在问题有思路了,既然明明包含了头文件,却找不到对应的类或命
- 6、解决办法:ibpUtils不存在,找到ibpUtils对应的头文件,搜索

## 参见

## 已经在LIBCMTD lib中定义

1、错误信息如下:

1>MSVCRTD.lib(MSVCR80D.dll) : error LNK2005: \_pri

2、错误原因:

使用不同的库,而这些库使用不同的运行时库,在这些运行时库中

3、解决办法:

使用相同的运行时库。在属性-->C/C++-->代码生成-->运行时库 libcmt对应多线程的静态库

msvcr80对应多线程的动态库

其中d表示debug版本。

4、还有一种解决办法:

忽略某一个特定的库,这种办法可能存在问题,不建议使用。

## 无法从char指针转换为string引用

```
1、代码如下:
    char tmp[64];
    sprintf(tmp,"Andy");

    string aa = tmp;
    string& bb = tmp;
    const string& cc = tmp;
```

- 2、其中string& bb = tmp; 报错 error C2440: "初始化": 无》
- 3、思考为什么?
- 5、string& bb = tmp; 左边是string的引用,期望接收string对也就是临时对象,但是这个临时对象是const对象,const对象不这就能解释为什么const string& cc = tmp; 是正确的。
- 6、那么为什么中间的临时对象是const对象呢? 假定中间的临时对象不是const对象, string& bb = tmp 是正期望会修改tmp的值,我们知道这是不可能的,因为修改的是临时》也就是说,中间的临时对象不是const对象,会导致与程序员期望为了避免这个问题,中间的临时对象是const对象。
- 7、注意:string aa = tmp; 存在隐式类型转化,直接调用对应的核

## 参见

## 无法定位程序输入点

1、错误原因:动态加载的时候无法找到VTDU\_A\_SetResMsgCallBac

## 无法打开文件LIBC\_lib问题

## 1、错误原因:

如果将用低版本的VC开发的项目,拿到高版本的VC开发环境上去编1>LINK: fatal error LNK1104: 无法打开文件"LIBC.lik

2、解决办法:链接时忽略此库,在此提供两种解决方案:

Project | Properties | Configuration Properties |

```
Ignore Specific Library: libc.lib
或
#pragma comment(linker, "/NODEFAULTLIB:libc.lib")
```

#### 无法打开文件vld lib

fatal error LNK1104: 无法打开文件vld.lib

- 1、今天发现一个很奇怪的问题,具体的场景是:
  cmu\_sdk(动态库)---> ibp\_utils(动态库)---> vld(过
- 2、现在修改ibp\_utils,不依赖vld,重新编译ibp\_utils和cmu\_sccmu\_sdk链接的时候报错:fatal error LNK1104: 无法打开了
- 3、错误原因是:cmu\_sdk还在试图去连接vld.lib
- 4、这是为什么呢?

思考,cmu\_sdk去连接的库,包含两部分:直接依赖的库和间接依间接依赖的库也就是,依赖一个静态库,而这个静态库又依赖其它

- 5、仔细查找,果然发现cmu\_sdk(动态库)--->new\_ibp\_protoco]
- 6、重新编译new\_ibp\_protocol静态库,再次编译cmu\_sdk,结果OK

### 无法解析的外部符号

- 1、错误原因:链接的时候,找不到方法的实现。
- 2、但是整个解决方案都搜索不到 SetModuleName
- 3、仔细分析,对于当前类的方法,只有声明,没有定义,如果没有调用
- 4、但是对于继承,比如 VruMaster继承了ibpMaster。 ibpMaster声明了虚方法,没有定义,即使没有调用方法,子类构
- 5、因此,可以断定VruMaster 引用的头文件有SetModuleName 这个

## 模拟CPP程序崩溃

- 1、在项目中经常遇到C++程序崩溃,这些崩溃都是有原因的,往往是因。
- 2、除0,如下:

```
int main(int argc, char* argv[])
{
    return 1/0;
}
编译:
```

```
[root@localhost dump]# g++ -o main main.cpp
   main.cpp: In function 'int main(int, char**)':
   main.cpp:3: warning: division by zero in '1 / 0'
   编译已经给出警告
   [root@localhost dump]# ./main
   Floating point exception
   没有生成core文件,原因是没有设置ulimit -c,ulimit -c表
   [root@localhost dump]# ulimit -c
   0
   [root@localhost dump]# ulimit -c 1000
   [root@localhost dump]# ./main
   Floating point exception (core dumped)
   [root@localhost dump]# 11
   total 112
   -rw----- 1 root root 196608 Jan 28 17:53 core.2
   -rwxr-xr-x 1 root root 4863 Jan 28 17:49 main
   -rwxrwxrwx 1 root root
                             50 Jan 28 17:46 main.c
   dump转存的时候,如果程序在内存的大小,超过-c的设置,就不到
   [root@localhost dump]# ulimit -c unlimited
3、上面已经是很短的崩溃程序,能不能更短呢?
   int main=0;
   崩溃原因:程序会试图按照调用方法的方式去执行main,这当然不
4、栈溢出(stack overflow)
   #include <stdio.h>
   int main(int argc, char* argv)
   {
       char pc[1024*1024*1024*2];
       getchar();
       return 0;
   编译报错,如下:
   [root@localhost dump]# g++ -o main main.cpp
   main.cpp: In function 'int main(int, char*)':
   main.cpp:4: error: overflow in constant expression
   main.cpp:4: error: size of array 'pc' is negative
   编译器很聪明,因为栈的大小一般为1M,这里是2M,直接编译报错
   #include <stdio.h>
```

```
int main(int argc, char* argv)
    {
        int size = 1024*1024*1024*2;
        char pc[size];
        getchar();
        return 0;
5、踩内存
    #include <string.h>
    int main(int argc, char* argv)
    {
        int a = 10;
        char pc[4] = \{0\};
        strcpy(pc, "aaaaaaaaaaaaaa");
        a = a+10;
        return 0;
    }
    或者
    #include <memory.h>
    int main(int argc, char* argv)
    {
        int a = 10;
        char pc[4] = \{0\};
        char* pc2 = "aaaaaaaaaaaaaaa";
        memcpy(pc,pc2,strlen(pc2));
        a = a+10;
        return 0;
    }
```

## 系统类型fd set重复定义

- 1、今天遇到一个很奇怪的问题,如下: error C2011: "fd\_set": "struct"类型重定义
- 2、发现是因为包含某个头文件导致的。调整一下头文件的包含顺序,发
- 3、include头文件,本质上就是文本替换(替换的过程中,会根据宏进不同的include顺序会导致不同的结果。
- 4、举例来说:

```
a.h
   #ifndef A_H_
   #define A_H_
   struct AAA
   {
   };
   #endif
   b.h
   #ifndef B_H_
   #define B H
   #ifndef A H
   struct AAA
   {
   };
   #endif
   #endif
5、考虑a.h与b.h不同的include顺序,a,b顺序没有问题,b,a导到
```

## 参见

## 踩内存导致delete的时候崩溃

```
1、今天遇到一个奇怪的问题,delete的时候程序崩溃。new出来的内存
2、思考为什么?
delete崩溃,说明要释放的堆内存已经损坏。损坏的原因有很多,
char* aaa = new char[4];
sprintf(aaa,"111111111");
delete []aaa;
```

## 野指针可能不会立即崩溃,会导致未定义行为

```
1、测试代码
#include <string>
```

```
using namespace std;
   class Person
   {
   public:
      int
           _Age;
      string _Name;
   };
   int main()
   {
      Person* p = new Person();
      p-> Age = 18;
      p -> Name = "Andy";
      delete p;
      p - > Age = 20;
      p-> Name = "Bill";
      return 0;
2、delete p; 之后,p是野指针,
   p->_Age = 20; 不会导致崩溃
   而p->_Name = "Bill"; 会导致崩溃,崩溃信息如下:
   NzbUtils.exe 中的 0x57b6d5a8 (msvcr80d.dll) 处未处
3、这是因为在VS的Debug模式, delete p之后, 内存信息都是 0x f
4、也就是说,对于野指针的操作,比如p->_Age = 20;不会导致崩溃
5、怎么解决这个问题?
   delete p之后,马上赋值为NULL,再对p操作,马上就崩溃。崩污
   NzbUtils.exe 中的 0x00411ce0 处未处理的异常: 0xC0000
6、以上是在VS测试的,在linux使用g++测试,对于野指针的操作,
   delete p;
   p->_Age = 20;
   p->_Name = "Bill";
   都不会崩溃
7、因此,野指针的问题非常严重,会导致不确定的行为。不知道什么时
   一定要避免野指针的问题,也就是delete之后,马上赋值为NULL
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

动态库彼此相互引用	
string内部包含取值为0的字符	
普通方法指针绑定成员方法	
隐式类型转化	
隐式类型转换隐式类型转换操作符操作符重载	
隐式类型转换临时对象	
写时拷贝	
new_A与new_A小括号	
VS2005各种文件	
对象池	
名称遮掩	
CPP内部连接	
成员方法的定义放在类定义内外的区别	
异常保护	
main之前执行什么	
调试断点不起作用	
目标文件是什么样	

崩溃产生core文件 条件编译 编译32位和64位差异有哪些 编译找不到头文件 时间2038年问题 CPP两个类相互引用引发的思考 一块内存覆盖到对象上的处理 查看内存的内容 Debug的时候String赋值失败 构造方法调用另一个构造方法 使用指针作为key存在的问题 禁止在栈上创建对象 linux捕获信号 程序退出的处理

## 变量说明

#### CPP两个类相互引用引发的思考

```
1、考虑下面的场景,示例代码如下:
    a.h
   #ifndef A_H_
   #define A_H_
   #include "b.h"
   class A
    public:
       int _Id;
         _B;
       В
   };
   #endif
   b.h
   #ifndef B H
   #define B H
   #include "a.h"
   class B
    public:
       int _Id;
       A _A;
    };
   #endif
   main.cpp
```

```
#include "a.h"
   int main(int argc, char* argv[])
   {
      A a;
      return 0;
2、编译报错:
   b.h(8): error C2146: 语法错误: 缺少";"(在标识符"_A
3、思考为什么?
4、先不考虑编译报错的问题,如果可以编译正常,会有什么问题?
   A类中包含了B类的实例,B类中又包含了A类的实例,我们知道每个
   那么A的实例a大小是多少?
   a的大小=4个字节(int大小)+ B实例的大小,而B实例的大小=4
   这样循环包含,也就是死循环,a的大小为无穷大,当然不可以。
5、为什么java、C#没有这样的问题?
   因为它们关联的对象都是引用。
6、注意:空类的实例大小为1个字节,这是编译器安插的,为什么要安排
   因为要区分空类的不同对象,如果空类对象不占用任何空间,怎么
   占用一个字节,也就是可以说,这个空类对象占用第几个字节。
7、现在考虑编译报错的原因,我们知道include是文本替换,替换的过
   那么我们把main.cpp转化为一个编译单元,对include进行文本。
   #include "a.h"
   int main(int argc, char* argv[])
   {
      A a;
      return 0;
   把"a.h"展开,转化为
   #include "b.h"
   class A
   {
   public:
      int _Id;
       _B;
      В
   };
   int main(int argc, char* argv[])
```

```
A a;
      return 0;
   把"b.h"展开,由于头文件保护符,a.h不再包含进来,转化为
   class B
   {
   public:
      int _Id;
      A _A;
   };
   class A
   public:
      int _Id;
      B _B;
   };
   int main(int argc, char* argv[])
   {
      A a;
      return 0;
   现在来分析,最后展开的编译单元,对于一个编译单元,CPP编译
   从上向下,遇到的名称(也就是使用的名称)必须在之前声明过。
   注意:可以没有定义,因为是在链接的时候,才去找对应的定义(
   因此,对于B中的 A _A;找不到名称A的声明。要引用的A在下ゴ
   所谓超前引用,就是超前引用下文才出现的名称。
8、这里需要注意的是:即使在开头增加A的声明,编译代码如下:
   class A;
   class B
   {
   public:
      int _Id;
      A A;
   };
   class A
```

```
public:
       int _Id;
       B _B;
   };
   int main(int argc, char* argv[])
   {
      A a;
       return 0;
   也会报错如下:
   main.cpp(58) : error C2079: "B::_A"使用未定义的 cla
   这是为什么?
   对于B类包含了A类的实例,必须提前知道A的定义。因为如果不知证
9、怎么解决上面的问题?
   不能循环包含对方的实例,可以包含对方的指针。方式有:
   A包含B的实例,B包含A的指针,反过来也行。
   A和B都包含对方的指针。
   示例如下:
   class A;
   class B
   {
   public:
      int _Id;
      A* _A;
   };
   class A
   public:
       int _Id;
       B _B;
   };
   int main(int argc, char* argv[])
   {
      A a;
       return 0;
```

```
这种情况下,也要提前声明class A; 这就是前置声明。
10、思考,为什么上面的方式就没有问题?
   对于A* __A; 编译器是知道分配多少内存的,因为任何类型的:
   注意:不同类型的指针,本质上没有什么区别,都是一个int值。
   重要的是:通过指针类型告诉编译器,把指针指向的这块内存当成。
11、接着思考这个问题,如下:
   niuzibin@ubuntu:~/work/test1/cpp1$ more a.h
   #ifndef A H
   #define A H
   #include "b.h"
   class A
   {
   public:
       int _Id;
       В
          _B;
       int GetId()
       {
          return _Id;
       }
   };
   #endif
   niuzibin@ubuntu:~/work/test1/cpp1$ more b.h
   #ifndef B H
   #define B_H_
   #include "a.h"
   class B
   {
   public:
       int _Id;
      A _A;
       int GetId()
       {
          return _Id;
       }
   };
```

```
#endif
   niuzibin@ubuntu:~/work/test1/cpp1$ more main.cpp
   #include "a.h"
   int main(int argc, char* argv[])
   {
      A a;
      return 0;
   In file included from a.h:3:0,
                 from main.cpp:1:
   b.h:8:5: error: 'A' does not name a type
      Α
        _A;
12、改为前置声明+关联指针的方式,修改b.h,如下:
   #ifndef B H
   #define B H
   class A;
   class B
   public:
      int _Id;
      A* _A;
      int GetId()
      {
          return _Id;
   };
   #endif
   构建正常
13、现在考虑,在B中调用A的方法,也就是前置声明的情况下调用方法
   #ifndef B H
   #define B H
   class A;
   class B
   public:
```

```
int _Id;
       A* _A;
       int GetId()
       {
           return _Id + _A->GetId();
   };
   #endif
14、报错如下:
   niuzibin@ubuntu:~/work/test1/cpp1$ q++ -o main ma
   In file included from a.h:3:0,
                   from main.cpp:1:
   b.h: In member function 'int B::GetId()':
   b.h:12:17: error: invalid use of incomplete type
   return _Id + _A->GetId();
   b.h:3:7: error: forward declaration of 'class A'
   class A;
   注意:error: invalid use of incomplete type 'class
   错误的原因是:对于前置声明,调用指针的方法。
15、怎么解决这个问题?
   方法一:b.h不再使用前置声明,a.h使用前置声明。如下:
       niuzibin@ubuntu:~/work/test1/cpp1$ more b.h
       #ifndef B H
       #define B H
       #include "a.h"
       class B
       public:
           int _Id;
           A* _A;
           int GetId()
           {
               return _Id + _A->GetId();
           }
```

```
#endif
    niuzibin@ubuntu:~/work/test1/cpp1$ more a.h
    #ifndef A H
    #define A H
    class B;
    class A
    {
    public:
        int _Id;
        В*
            В;
        int GetId()
            return _Id;
        }
   };
    #endif
方法二:方法的实现移到b.cpp,并且在b.cpp包含a.h,如下:
    niuzibin@ubuntu:~/work/test1/cpp1$ more b.h
    #ifndef B H
   #define B_H_
    class A;
    class B
    {
    public:
        int _Id;
       A* _A;
       int GetId();
    };
    #endif
    niuzibin@ubuntu:~/work/test1/cpp1$ more b.cpr
    #include "b.h"
    #include "a.h"
    int B::GetId()
    return _Id + _A->GetId();
```

```
}
niuzibin@ubuntu:~/work/test1/cpp1$ g++ -c b.c
niuzibin@ubuntu:~/work/test1/cpp1$ g++ -o mai
```

#### CPP内部连接

```
1、考虑下面的场景,如下:
    a.h
    int a = 100;
    void a1_print();
    void a2_print();
    a1.cpp
    #include "a.h"
    #include <stdio.h>
    void a1_print()
    {
        printf("before a1.cpp a[%d]\n",a);
        a = 101;
        printf("after a1.cpp a[%d]\n",a);
    }
    a2.cpp
    #include "a.h"
    #include <stdio.h>
    void a2_print()
    {
        printf("before a2.cpp a[%d]\n",a);
        a = 102;
        printf("after a2.cpp a[%d]\n",a);
    }
    main.cpp
    #include "a.h"
    #include <stdio.h>
    int main(int argc, char* argv[])
```

```
a1_print();
       a2_print();
       getchar();
       return 0;
2、编译报错,连接的时候对于a重复定义,怎么解决?
3、使用static修饰,使对于a的连接成为内部连接。这种情况下,每个
   static int a = 100;
4、对于方法存在同样的问题,如下:
   f.h
   int GetId()
    {
       static int a = 0;
       return ++a;
   void f1_print();
   void f2_print();
   f1.cpp
   #include "f.h"
   #include <stdio.h>
   void f1_print()
   {
       printf("before f1.cpp a[%d]\n", GetId());
       printf("after f1.cpp a[%d]\n", GetId());
   }
   f2.cpp
   #include "f.h"
   #include <stdio.h>
   void f2 print()
   {
       printf("before f2.cpp a[%d]\n", GetId());
       printf("after f2.cpp a[%d]\n",GetId());
   }
```

```
main.cpp
   #include "f.h"
   #include <stdio.h>
   int main(int argc, char* argv[])
   {
      f1_print();
      f2 print();
      getchar();
      return 0;
5、方法GetId()的作用是返回全局自增的Id,上面连接的时候出错, 引
   为了解决这个问题,使用static成为内部连接,但是这样出大问是
   方法有了两个副本,彼此独立,不能保证全局自增唯一的Id了。
   注意:如果是可重入的方法使用static内部连接没有问题。
6、怎么解决?
   不使用static内部链接,头文件只是声明方法,方法定义放入其中
7、考虑类,如下:
   类的静态字段是外部连接。
   类的成员方法(实例方法和静态方法都一样)取决于方法定义的位
   方法实现放在类定义外,就是外部连接。
```

#### 参见

#### Debug的时候String赋值失败

```
}
3、思考为什么不行?
直接修改aa.c_str(), aa的size等字段根本就没有变化, 还是0也就是说,直接修改aa.c_str(), aa根本意识不到。注意:Debug的时候只能看到字段和虚方法表,其他方法是看不到4、为什么要关联虚方法表?
为了要实现多态功能,根据真实类型决议方法的调用。考虑Dog继为Animal* pa = new Dog(); pa->Play(); 这句代码会编译为 pa->vptr[n] 在运行的时候, pa->vptr[n]就能根据真实类型决议到方法的调用
```

#### linux捕获信号

```
1、示例代码如下:
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include <unistd.h>
   #include <errno.h>
   #include <signal.h>
   void catch_signal(int sign)
   {
       printf("Catch Signal[%d]\n", sign);
       exit(0);
   }
   //建议使用封装之后的mysignal
   int mysignal(int sign, void (*func)(int))
   {
       struct sigaction act, oact;
       //传入回调函数
       act.sa_handler=func;
       //将act的属性sa mask设置一个初始值
       sigemptyset(&act.sa_mask);
       act.sa_flags=0;
```

```
return sigaction(sign,&act,&oact);
   }
   int main(int arg, char *args[])
    {
       mysignal(SIGINT, catch_signal);
       mysignal(SIGALRM, catch_signal);
       mysignal(SIGKILL, catch_signal);
       for(int i = 1; i <= 64; i++)
       {
           mysignal(i,catch signal);
       }
       getchar();
       return 0;
2、使用kill杀掉进程
    [root@localhost niu12]# kill -1 `pidof main`
    [root@localhost niu12]# kill -2 `pidof main`
   可以观察到捕获到信号,如下:
    [root@localhost niu13]# ./main
   Catch Signal[1]
    [root@localhost niu13]# ./main
   Catch Signal[2]
   注意:信号9和19不能捕获到,如下:
    [root@localhost niu12]# kill -9 `pidof main`
    [root@localhost niu13]# ./main
   Killed
    [root@localhost niu12]# kill -19 `pidof main`
    [root@localhost niu13]# ./main
    [2]
         Killed
                                 ./main
    [3]+ Stopped
                                 ./main
3、信号9和19不能捕获到,这也就是,为什么程序可以对Ctrl+C【信号
   而对干kill -9不能进行处理。
```

比如Redis服务,设置快照的保存策略,也就是满足某个条件,进 Ctrl+C关闭服务, Redis捕捉到信号, 刷新到文件, 然后退出, 数 但是对于kill -9,不能捕捉到信号,Redis服务直接退出,来不

4、如何查看各种信号,使用kill -1,如下:

[root@localhost niu12]# kill -1

- 1) SIGHUP 2) SIGINT 3) SIGQUIT 5) SIGTRAP 6) SIGABRT 7) SIGBUS 3 10) SIGUSR1 11) SIGSEGV 12 9) SIGKILL 13) SIGPIPE 14) SIGALRM 15) SIGTERM 1 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 21) SIGTTIN 22) SIGTTOU 23) SIGURG 2 25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 29) SIGIO 30) SIGPWR 31) SIGSYS 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 29) SIGIO 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 4
- 47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 5
- 51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 5
- 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7 5
- 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3
- 63) SIGRTMAX-1 64) SIGRTMAX

注意:从1到64,并不是连续的,中间有些信号值是不存在的,比\$ 5、信号处理的流程是:

- a、在用户态的控制流程中,由于中断,异常或者系统调用进入内核
- b、内核态检查到,对于信号有相关的注册方法,回调到用户态,抗
- c、在用户态执行完注册的方法,回到内核态

#### main之前执行什么

- 1、main之前,有一些准备工作要做,就是对非局部的静态对象进行初始
- 2、静态对象分为局部静态对象和非局部静态对象。局部静态对象就是方 非局部静态对象包括全局对象,命名空间内的对象,类的静态对象

#### new A与new A小括号

1、首先考虑A是基本类型的情况,如下: int\* p1 = new int; // p1没有初始化,为0xcdcdcdcd

```
int* p2 = new int(); // p2初始化为0
   int a1;  // a1没有初始化,为0xccccccccint a2 = int();  // a2初始化为0
   注意:int a1(); 不是定义a1为int对象,而是声明了一个方法。
2、我们知道有两种方式,X和X(),我们称之为X方式和X()方式。
   X方式是:默认初始化,调用default构造方法。X中的字段是否初
   用户定义的default构造方法。如果用户没有定义构造方法,调用
   特别注意:编译器自动生成的default构造方法,不对X的字段进行
   X()方式是:值初始化,调用default构造方法。首先尝试调用用。
   如果用户没有定义构造方法,调用编译器自动生成的default构造
   特别注意:编译器自动生成的default构造方法,不对字段进行初
3、现在考虑A是class或者struct,如下:
   new A 是默认初始化,调用用户定义的default构造方法,对A的
   调用编译器自动生成的default构造方法,没有对A的字段初始化。
   new A() 是值初始化,调用用户定义的default构造方法,对Afr
   会调用编译器自动生成的default构造方法,自动生成的default
   后续有一个动作,对字段进行值初始化。
   A a1;
   A a2 = A();
   上面的栈上对象和new A, new A()的调用过程一样。
4、特别注意:上面的规则只适合用户没有定义任何的default构造方法
   如果用户定义了A,或者A中字段的default构造方法,结果会是什
   class Cat
   public:
      int _Age;
      int _Height;
   };
   class Dog
   {
   public:
      int _Age;
      int Height;
   };
   class Person
```

```
public:
       int _Age;
       int _Height;
       Cat _Cat;
       Dog _Dog;
   };
   int main(int argc, char* argv[])
   {
       Person* p1 = new Person;
       Person* p2 = new Person();
       Person a1;
       Person a2 = Person();
   p1全部是0xcdcdcdcd
   p2全部是0
   a1全部是0xccccccc
   a2全部是0
5、如果Person增加用户定义的default构造方法,如下:
   public:
       Person():_Age(25),_Height(175)
   p1的_Age(25),_Height(175),_Cat、_Dog的字段是0xcdcd
   p2 和p1一样
   a1的_Age(25),_Height(175),_Cat、_Dog的字段是0xcccc
   a2 和a1一样
6、如果Cat增加用户定义的default构造方法,如下:
   public:
       Cat():_Age(2),_Height(23)
       {
       }
```

p1的\_Cat , \_Age(2), \_Height(23) , 其它的字段是0xcdcdcdcp2 和p1一样 a1的\_Cat , \_Age(2), \_Height(23) , 其它的字段是0xccccccca2 和a1一样

- 7、如果Dog增加用户定义的default构造方法,和6的结果一样。
- 8、通过5,6,7的测试,得出结论,如果用户定义了A,或者A中字段的d 也就是如果用户定义了Person的default构造方法,或者Cat、D new Person()中的字段,也就不存在额外的值初始化动作,和ne
- 9、为什么?

定义了部分的构造方法,也就是用户不期望额外的值初始化动作,这个时候,new Person()会自动变成new Person的形式。

10、特别注意:根据实际需要,确认是否要自定义default构造方法,对于上面的情况,只有在没有定义任何的default构造方法,并且满足这种条件的场景很少,千万不要依赖这种情况,而应该自定义。

#### 参见

#### string内部包含取值为0的字符

```
1、一般情况下, string内部不包含取值为0的字符, 因为在赋值的时候
2、但是存在下面的情况,如下:
   string s1 = "aaa&bbb";
   s1[3] = '\0';
   调试看到s1的取值为aaa,但是,实际上取值是aaa\0bbb,s1的
3、对于这样的string,进行base64编码解码需要特别注意:
   base64编码没有问题,因为base64本来就是对字节编码,取值为
   在解码的时候,解码出来的string,要考虑string内部包含取值
4、返回解码后的string如下:
   // 特别注意,需要考虑解码后的string内部包含取值为0的字符
   string KeyManager::DecSerialObject(const string &
   {
      unsigned char* buf = new unsigned char[cipher
      memset(buf, 0, cipherText.size());
      int size = 0:
      Base64_Decode(cipherText, buf, size);
```

```
string plainText;
plainText.resize(size);
memcpy(&plainText[0], buf, size);
delete buf;
buf = NULL;
return plainText;
}
5、重新实现base64的加密解密,增加输入string,输出string的方
```

## VS2005各种文件

```
pch: PreCompiled Header, 预编译头文件,比较大,由编译
1、
     在以后建立工程时不再重新编译这些代码,以便加快整个编译
   ilk:Incremental Linking,作用是增量链接,连接过程中生
2、
   obj:编译器对编译单元(一般是cpp文件)进行编译生成的目标。
3、
     编译器对编译单元编译,生成obj文件,对obj文件链接,生。
   pdb:Program DataBase,程序数据库文件,在建立工程时自i
4、
   idb:系统生成的中间状态文件,用于加快随后的编译速度。如果
5、
   lib:库文件,分为两种情况,对于静态库,lib包含实现。对于
6、
   d11: 动态库
7、
  exp:同lib类似,跟dll或者exe一起生成的文件,其中包含了函
8、
     LINK工具将使用EXP文件来创建动态链接库。
   sln:解决方案文件
9、
10、suo:解决方案,用户自定义的选项文件
11、ncb:用于智能感知,比如提示对应的方法和字段
12、vcproj: 工程文件
```

#### 一块内存覆盖到对象上的处理

```
1、考虑结构体,如下:
struct Dog
{
    int __Id;
    int __Age;
    int __Length;
    char* __Desc;
};
```

```
_Desc表达跟着一块内存,包含描述信息
2、设置如下:
   int main(int argc,char* argv[])
    {
       char* pc = new char[28];
       memset(pc, 0,28);
       Dog^* d = (Dog^*)pc;
       d -> Id = 1;
       d \rightarrow Age = 2;
       d \rightarrow Length = 28;
       sprintf(pc+12, "%s",
            "abcdefq");
        return 0;
3、也可以去地址,如下:
   sprintf((char*)(&(d->_Desc)), "%s",
        "abcdefq");
4、仔细思考,这里的_Desc取值根本没有意义,因为这块内存紧跟着_L
5、可以使用更好的解决办法,使用char __Desc[1];表示位置,因
    struct Dog
    {
       int _Id;
       int _Age;
       int _Length;
char _Desc[1];
                          //包含描述 16个字节
   };
   int main(int argc,char* argv[])
    {
       char* pc = new char[28];
       memset(pc, 0,28);
       Dog^* d = (Dog^*)pc;
       d \rightarrow Id = 1;
```

#### 使用指针作为key存在的问题

```
1、new出来一个对象,然后delete,紧接着再次new出来一个对象,两
   #include <stdio.h>
   int main()
   {
       int* pa = new int(1);
       printf("pa[%p:%d]\n", pa,*pa);
       delete pa;
       int* pb = new int(2);
       printf("pb[%p:%d]\n", pb,*pb);
       getchar();
       return 0;
   也就是说,释放的内存被系统回收,马上又重新使用。
   [niu_zibin@localhost testnew]$ g++ -o main main.c
   [niu zibin@localhost testnew]$ ./main
   pa[0x869b008:1]
   pb[0x869b008:2]
2、注意:只有在delete之后,紧接着new的时候,才会出现上面的情况
   如果中间包含了动态内存分配,不会出现上面的情况,如下:
   #include <stdio.h>
   int main()
   {
          int* pa = new int(1);
          printf("pa[%p:%d]\n", pa,*pa);
          delete pa;
```

```
int* tmp = new int(100);
          int* pb = new int(2);
           printf("pb[%p:%d]\n", pb,*pb);
           getchar();
           return 0;
   }
   [niu_zibin@localhost testnew]$ g++ -o main main.c
   [niu_zibin@localhost testnew]$ ./main
   pa[0x91ba008:1]
   pb[0x91ba018:2]
3、包含栈上的内存分配,两次new出来的指针值还是一样,如下:
   #include <stdio.h>
   int main()
   {
          int* pa = new int(1);
          printf("pa[%p:%d]\n", pa,*pa);
          delete pa;
           int tmp = 100;
          int* pb = new int(2);
           printf("pb[%p:%d]\n", pb,*pb);
          qetchar();
           return 0;
   [niu_zibin@localhost testnew]$ g++ -o main main.c
   [niu_zibin@localhost testnew]$ ./main
   pa[0x8149008:1]
   pb[0x8149008:2]
4、上面的测试结果,在windows下也是同样的。
   也就是说, new出来的内存, 马上delete, 紧接着再次new, 系统
   还会存在巧合的情况,之前释放的内存,又被重新使用。
   也就是说,程序中会存在两次new使用同一块动态内存(前一次的r
5、这会存在什么问题?
   使用指针作为map的key,考虑把pa保存到map,然后把pa释放,与
```

但是如果在delete pa之后,new出来pb并且保存到map中,以后这显然不是用户所期望的。

6、怎么解决?

解决办法很简单,使用自增Id作为map的key来进行管理。

#### 写时拷贝

1、写时拷贝(copy-on-write,简称COW),为了解决什么问题? 考虑下面的代码:

string aa = "abcd";

string bb = aa;

如果bb是深复制,也就是申请内存,并且复制内容。

语言中到处都是string的copy构造(copy赋值的情况一样),这

- 2、为了解决这个问题,使用浅复制,也就是bb和aa的指针相同一块内存但是这样导致两个新的问题:
  - a、因为aa和bb指向同一块内存,aa离开作用域,调用析构方法的同时,当aa和bb都离开作用域,必须释放内存,否则造成资流
  - b、如果只是读取,没有问题。考虑修改bb的值,比如 bb[1] = 但是因为aa和bb共享一块内存,导致aa的值也发生了变化,
- 3、为了解决上面的两个问题,需要使用引用计数和写时拷贝技术。 引用计数:在copy构造、copy赋值,和析构的时候,对引用计数<sup>1</sup> 写时拷贝:每次修改的时候,引用计数等于1,直接修改,反正没人
- 4、这里又存在一个问题,怎么判断是修改呢?

bb[1]有可能是读取,也有可能是写入,你可能会想到,提供两个const char& operator[](int index) const; // 针对char& operator[](int index); // 针对ned是,这还是存在问题,决定调用哪个方法,不是取决于用户的读证对于const string 只可能调用第一个方法,没有问题。

对于non-const string,无论是读取还是写入,都是调用第二个

5、怎么解决上面的问题?

有两个解决办法,如下:

- a、对于操作符[],不区分读取写入,只要引用计数大于1,存在共对于读取的情况,增加了开销。
- b、再搞出一个代理类CharProxy,对char封装,string的内容: CharProxy提供隐式类型转换操作符,转化为char,用于读 重载operator= 用于赋值操作。

读取操作肯定调用隐式类型转换操作符,而写入操作肯定调用

```
隐式类型转换操作符,直接返回对应的char
       operator= , 进行COW, 引用计数等于1, 直接修改, 反正注
6、特别注意,写时拷贝只是一种提高性能的技术,对于STL中string是
   也就是说,STL中的string的实现,有可能是写时拷贝,也可能是
   测试代码如下:
   #include <stdio.h>
   #include <string>
   using namespace std;
   int main(int argc, char* argv[])
   {
       string aa = "abcd";
       string bb = aa;
       printf("aa[%x], ", aa.c_str());
       printf("bb[%x]\n", bb.c_str());
       aa[0]='w';
       printf("aa[%x], ", aa.c_str());
       printf("bb[%x]\n", bb.c_str());
       char* pc= (char*)(&aa[1]);
       printf("pc[%x:%c]\n", pc, *pc);
       *pc = 'h';
       printf("pc[%x:%c]\n", pc, *pc);
       printf("aa[%x:%s], ", &(aa[0]), aa.c_str());
       printf("bb[%x:%s]\n", &(bb[0]), bb.c_str());
       getchar();
       return 0;
   在VS2005打印如下:
   aa[18ff04], bb[18fedc]
   aa[18ff04], bb[18fedc]
   pc[18ff05:b]
```

```
pc[18ff05:h]
   aa[18ff04:whcd], bb[18fedc:abcd]
   说明, VS2005中的string实现不是写时拷贝, 是深复制。
   在Redhat的Linux打印如下:
   [root@localhost string2]# ./main
   aa[97fc014], bb[97fc014]
   aa[97fc02c], bb[97fc014]
   pc[97fc02d:b]
   pc[97fc02d:h]
   aa[97fc02c:whcd], bb[97fc014:abcd]
   说明, Redhat的Linux中, string实现是写时拷贝。
7、对于Redhat的Linux中, string实现是写时拷贝。特别注意以下几
   a、对于aa.c str(), %x或者%d打印指针地址, %s打印内容。
   b、aa[0]='w'; 会导致写时拷贝。按道理只有写的时候,才应该
       也就是说, char c = aa[0], 不应该导致aa深复制, 但是
       原因是:对于[]操作,很难区分是读操作,还是写操作,因此
       当然,也是可以区分的,effective C++讲过,再增加一个
       也就是说,Linux的string为了实现的简单性,忽略了一部。
   c、引用计数大于1的时候,才会写时拷贝,等于1,没有必要再去?
   d、修改谁,谁才会去深复制。考虑下面的情况,
       #include <stdio.h>
       #include <string>
       using namespace std;
       void foo(char a, char b)
       {
              // do nothing
       }
       int main(int argc, char* argv[])
       {
              string aa = "abcd";
              string bb = aa;
              printf("aa[%x], ", aa.c_str());
printf("bb[%x]\n", bb.c_str());
```

```
foo(aa[0],bb[0]);
               printf("aa[%x], ", aa.c_str());
               printf("bb[%x]\n", bb.c_str());
               char* pc= (char*)(&aa[1]);
               printf("pc[%x:%c]\n", pc, *pc);
               *pc = 'h';
               printf("pc[%x:%c]\n", pc, *pc);
               printf("aa[%x:%s], ", &(aa[0]), aa.c_
               printf("bb[%x:%s]\n", &(bb[0]), bb.c_
               getchar();
               return 0;
       测试,打印如下:
        [root@localhost string2]# ./main
       aa[8835014], bb[8835014]
        aa[8835014], bb[883502c]
        pc[8835015:b]
        pc[8835015:h]
        aa[8835014:ahcd], bb[883502c:abcd]
       调用foo(aa[0],bb[0]); 按道理应该是aa深复制,为什么
8、写时拷贝带来什么问题?
   考虑下面的场景,代码如下:
   #include <stdio.h>
   #include <string>
   using namespace std;
   int main(int argc, char* argv[])
    {
           string aa = "abcd";
           string bb = aa;
           printf("aa[%x:%s], ", aa.c_str(), aa.c_st
           printf("bb[%x:%s]\n", bb.c_str(), bb.c_st
```

```
char* pc = (char*)bb.c_str();
           pc[1] = 'H';
           printf("aa[%x:%s], ", aa.c_str(), aa.c_st
           printf("bb[%x:%s]\n", bb.c_str(), bb.c_st
           getchar();
           return 0;
   打印结果,如下:
    [root@localhost string2]# g++ -o main main.cpp
    [root@localhost string2]# ./main
   aa[9258014:abcd], bb[9258014:abcd]
   aa[9258014:aHcd], bb[9258014:aHcd]
   char* pc = (char*)bb.c_str(); 进行了强制转化,没有触
   这种场景,还是显而易见的。
9、更隐蔽的情况,比如说,把bb传给一个方法,无论是传引用,或者传
   #include <stdio.h>
   #include <string>
   using namespace std;
   void foo(const string& str)
   {
           char* pc = (char*)str.c_str();
           pc[1] = 'E';
   }
   int main(int argc, char* argv[])
   {
           string aa = "abcd";
           string bb = aa;
           printf("aa[%x:%s], ", aa.c_str(), aa.c_st
           printf("bb[%x:%s]\n", bb.c_str(), bb.c_st
           foo(bb);
           printf("aa[%x:%s], ", aa.c_str(), aa.c_st
           printf("bb[%x:%s]\n", bb.c_str(), bb.c_st
```

```
getchar();
        return 0;
或者
#include <stdio.h>
#include <string>
using namespace std;
void foo(string str)
{
        char* pc = (char*)str.c_str();
        pc[1] = 'Q';
}
int main(int argc, char* argv[])
{
        string aa = "abcd";
        string bb = aa;
        printf("aa[%x:%s], ", aa.c_str(), aa.c_st
        printf("bb[%x:%s]\n", bb.c_str(), bb.c_st
        foo(bb);
        printf("aa[%x:%s], ", aa.c_str(), aa.c_st
        printf("bb[%x:%s]\n", bb.c_str(), bb.c_st
        getchar();
        return 0;
还有更隐蔽的情况,动态加载库,如下:
test.cpp
#include <string>
#ifdef __cplusplus
extern "C"
#endif
```

```
void SetName(std::string name)
{
        printf("name[%x:%s]\n", name.c_str(), name.c_str()
        char* pc = (char*)name.c_str();
        pc[2] = 'M';
#ifdef __cplusplus
};
#endif
main.cpp
#include <string>
#include <dlfcn.h>
using namespace std;
typedef void (*Func) (string str);
int main(int argc, char* argv[])
{
        void* pHandle = dlopen("./libtest.so", RT
        void* error = dlerror();
        printf("Error[%s]\n", error);
        Func pFunc = (Func)dlsym(pHandle, "SetName
        error = dlerror();
        printf("Error[%s]\n", error);
        string aa = "Andy";
        printf("aa[%x:%s]\n", aa.c_str(), aa.c_st
        string bb = aa;
        pFunc(bb);
        printf("aa[%x:%s]\n", aa.c_str(), aa.c_st
        return 0;
运行,如下:
[root@localhost dll2]# q++ -fPIC -shared -o libte
[root@localhost dll2]# g++ -o main main.cpp -ldl
```

```
[root@localhost dll2]# ./main
Error[(null)]
Error[(null)]
aa[90843dc:Andy]
name[90843dc:Andy]
aa[90843dc:AnMy]
10、怎么解决?
抑制写时拷贝,使用深复制。如下:
string aa = "Andy";
string bb(aa.c_str()); 或者 string bb = aa.c_str(
```

#### 动态库彼此相互引用

```
1、考虑下面的情况, main-->person<-->utils
   可执行文件main引用动态库person,动态库person和动态库uti
2、一般情况下,我们是在生成so文件的时候,每一步都进行链接依赖的
   那么这里就存在问题了, person和utils之间相互依赖, 导致了列
3、怎么解决这个问题?
   正常的顺序如下:
   先生成utils,不链接person
   再生成person,链接utils
   最后生成main,链接person
4、示例代码如下:
   niuzibin@ubuntu:~/work/test1/dll/utils$ more util
   #ifndef UTILS H
   #define UTILS H
   #include <string>
   #include "../person/person.h"
   using namespace std;
   class Utils
          public:
                 string GetKey(const Person& p);
   };
   #endif
```

```
niuzibin@ubuntu:~/work/test1/dll/utils$ more util
#include "utils.h"
string Utils::GetKey(const Person& p)
{
        return "123-" + p.GetKey();
}
niuzibin@ubuntu:~/work/test1/dll/person$ more per
#ifndef PERSON H
#define PERSON H
#include <string>
using namespace std;
class Person
{
        public:
                Person(const string& name);
                string GetName() const;
                string GetKey() const;
                string m_name;
};
#endif
niuzibin@ubuntu:~/work/test1/dll/person$ more per
#include "person.h"
#include "../utils/utils.h"
Person::Person(const string& name)
{
        m_name = name;
}
string Person::GetName() const
{
        Utils ut;
        return ut.GetKey(*this) + m_name;
string Person::GetKey() const
```

```
{
            return "abc-";
    }
    niuzibin@ubuntu:~/work/test1/dll$ more main.cpp
    #include "person/person.h"
    #include <stdio.h>
    int main(int argc,char* argv[])
    {
            Person p("Andy");
            printf("Person[%s]\n", p.GetName().c_str(
            return 0;
5、构建运行如下:
    niuzibin@ubuntu:~/work/test1/dll/utils$ g++ -fPI(
    niuzibin@ubuntu:~/work/test1/dll/person$ g++ -fP]
    niuzibin@ubuntu:~/work/test1/dll$ g++ -o main mai
    niuzibin@ubuntu:~/work/test1/dll$ ./main
    Person[123-abc-Andy]
```

#### 名称遮掩

```
1、考虑下面的情况,
    class Animal
    {
        public:
            virtual void Say();
            virtual void Say(int a);
        };

    class Dog : public Animal
    {
        public:
            virtual void Say(int a);
```

```
};
   测试代码,如下:
   int main(int argc, char* argv[])
   {
      Dog* pDog = new Dog();
      pDog->Say();
      getchar();
      return 0;
   报错:error C2660: "Dog::Say": 函数不接受 0 个参数
2、pDog->Say(); 期望调用父类Animal的Say方法,而且Animal确。
   这是因为,子类同名方法会遮掩父类的方法,名称遮掩只考虑方法。
3、如何调用父类的Say()方法,显式调用,如下:
   pDog->Animal::Say();
   特别注意:显示调用不具备多态行为,在编译时就确定了方法,而
   pDog->Say(6); // 运行时,通过虚方法表决议,
   pDog->Animal::Say(6); // 编译时确定调用父类的方法
```

#### 对象池

- 1、考虑下面的场景,频繁地需要同一类型的大量对象,而且对象的生命 我们知道new操作符,操作系统要找到一块内存,然后调用构造方》
- 2、解决办法是:使用对象池,初始的时候,new出一批对象。用户请求
- 3、new出一批的时候,每个都初始化。用户还回来,再初始化。
- 4、怎么管理这批对象?
  - a、每个对象增加一个字段,标识是否已经使用。
  - b、只保存没有被使用的,还回来的再放入队列。
- 5、使用对象池,还可以监视同时有多少对象被使用,考虑,对象池初始 不可能同时很多被使用,那就是资源有泄露,用户用了没有还。

#### 崩溃产生core文件

- 1、Linux下产生core文件,只需要 ulimit -c unlimited
- 2、Windows下面崩溃不会自动产生core文件,注意:Windows下core要想产生dmp文件,需要设置未处理异常的回调,在回调方法中创建。
- 3、代码如下:

```
#include <stdio.h>
#include <Windows.h>
#include <DbgHelp.h>
#include <assert.h>
#pragma comment(lib, "Dbghelp.lib")
LONG WINAPI MyUnhandledExceptionFilter(PEXCEPTION
{
    SetErrorMode(SEM_NOGPFAULTERRORBOX);
    DWORD dwBufferSize = MAX PATH;
    SYSTEMTIME stLocalTime;
    GetLocalTime(&stLocalTime);
    HMODULE currentModule=GetModuleHandle(NULL);
    char exeFileName[512]={0};
    char workFullDir[512]={0};
    char szFileName[512]={0};
    if(!GetModuleFileNameA(currentModule,exeFileN
    {
        int res=GetLastError();
        if(0<res)
            res=0-res;
        res+=-10;
        return res;
    }
    strncpy(workFullDir,exeFileName,sizeof(workFi
    char* pos=strrchr(workFullDir,'\\');
    if(NULL==pos)
        return -2;
    ++pos;
    *pos=0;
    sprintf(szFileName, "%s-%04d%02d%02d-%02d%02c
        exeFileName,
```

```
stLocalTime.wYear, stLocalTime.wMonth, st
        stLocalTime.wHour, stLocalTime.wMinute, s
    HANDLE hDumpFile = CreateFileA(szFileName, GE
    MINIDUMP EXCEPTION INFORMATION ExpParam = {0}
    ExpParam.ThreadId = GetCurrentThreadId();
    ExpParam.ExceptionPointers = pExceptionPointe
    ExpParam.ClientPointers = FALSE;
    MINIDUMP TYPE MiniDumpWithDataSegs = (MINIDUN
    MiniDumpWriteDump(GetCurrentProcess(), GetCur
    CloseHandle(hDumpFile);
    return EXCEPTION_CONTINUE_SEARCH;
}
void DumpCore()
{
    SetUnhandledExceptionFilter(MyUnhandledExcept
    void *addr = (void *)GetProcAddress(GetModule
    if (addr != NULL)
    {
        unsigned char code [16] = \{0\};
        int size = 0;
        code[size++] = 0x33;
        code[size++] = 0xC0;
        code[size++] = 0xC2;
        code[size++] = 0x04;
        code[size++] = 0x00;
        DWORD dwOldFlag = 0, dwTmpFlag = 0;
        VirtualProtect(addr, size, PAGE_READWRITE
        WriteProcessMemory(GetCurrentProcess(), a
        VirtualProtect(addr, size, dwOldFlag, &dv
    }
```

```
void Check()
    {
        int ptrSize = sizeof(void*);
        printf("Pointer Size[%d]\n", ptrSize);
        #if defined WIN64 || __LP64___
        assert(ptrSize == 8);
        #else
        assert(ptrSize == 4);
        #endif
    }
int main(int argc, char* argv[])
{
    Check();
    DumpCore();
    int a = 0;
    return 1/0;
}
```

#### 异常保护

```
1、对抛出异常的方法,没有try catch,会导致程序崩溃,如下:
void aa()
{
    throw 1;
}

int main(int argc, char* argv[])
{
    aa();
    return 0;
}
2、修改为如下:
```

```
void aa()
    {
        throw 1;
    }
    int main(int argc, char* argv[])
    {
        try
        {
            aa();
        catch (...)
            //
        return 0;
3、现在考虑,在catch中又调用方法bb,而bb没有异常保护,也会导致
    void aa()
    {
        throw 1;
    }
    void bb()
    {
        throw 2;
    }
    int main(int argc, char* argv[])
    {
        try
        {
            aa();
        catch (...)
            bb();
```

```
return 0;
4、怎么解决这个问题?
   a、对这个代码段再加一层异常保护,如下:
       int main(int argc, char* argv[])
       {
           try
           {
               try
                   aa();
               catch (...)
                   bb();
           catch (int& ex)
           {
               int hh = 0;
           return 0;
   b、对bb方法进行单独保护,如下:
       int main(int argc, char* argv[])
       {
           try
               aa();
           catch (...)
           {
               try
                   bb();
```

```
catch (int& ex)
                 int gg = 0;
          return 0;
5、Effective C++中有一条,别让异常逃离析构方法。为什么?
   考虑下面的代码:
   Person::~Person()
   {
      throw 1;
   }
   int main(int argc, char* argv[])
   {
      Person p;
      return 0;
   这种情况,导致程序崩溃。解决这种问题,必须进行异常保护,如
   int main(int argc, char* argv[])
   {
      try
      {
          Person p;
      catch (int& ex)
      {
      return 0;
6、这种写法太丑陋了,而且析构方法抛出异常,还会导致其他问题。比
   在集合中有多个Person,这样的话就会一次性抛出多个异常,C++
   还有下列的情况,如下:
   void aa()
```

```
Person p;
throw 1;
}

int main(int argc, char* argv[])
{
   try
   {
      aa();
   }
   catch (...)
   {
   }
   return 0;
}
即使对aa进行了异常保护,也会崩溃。
因为throw 1 然后堆栈回滚,Person析构又抛出异常,也就是访所以说,千万不能让异常逃离析构方法。
```

### 成员方法的定义放在类定义内外的区别

- 1、区别一:放在类定义内,编译器会尝试编译为内联方法,提高方法执复杂不会成为内联方法。因此,很小的方法建议放在类定义内。当意
- 2、区别二:放在类定义内,成员方法是内部连接,被多个cpp文件包含 放在类定义外,成员方法是外部连接,被多个cpp文件包含会出现 无论是实例方法还是静态方法,都是一样的情况。

# 参见

# 时间2038年问题

- 1、unix操作系统使用有符号的int表示时间,这个时间是从unix的开始有符号的int最大值为21亿(21亿4千万左右),到了2038年,经
- 2、无符号的int最大值为42亿9千万

```
1、普通方法指针,绑定普通方法,如下:
    typedef int (*FunPtr)(int a, int userHandle);
    int Fool(int a,int userHanlde)
    {
        return a+1;
    }
    int main(int argc, char* argv[])
    {
        FunPtr fun = Fool;
        int hh = fun(3,0);
2、成员方法指针,绑定成员方法,如下:
    class Student
    public:
        Student(string name):_Name(name)
        {
        }
        void SayHello()
        {
            printf("Hello,I am %s",_Name.c_str());
    public:
        string _Name;
    };
    typedef void (Student::*SayHelloPtr)();
    int main(int argc, char* argv[])
    {
        Student a("Andy");
        SayHelloPtr sayHello = &Student::SayHello;
        (a.*sayHello)();
```

```
3、现在问题来了,想用普通方法指针,绑定成员方法,怎么办?
   不能直接绑定,增加一个间接层。如下:
   class Student
   {
   public:
       Student(string name):_Name(name)
       {
       }
       void SayHello()
           printf("Hello,I am %s",_Name.c_str());
   public:
       string _Name;
   };
   typedef int (*FunPtr)(int a, int userHandle);
   int StudentCallBack(int a,int userHandle)
   {
       Student* pStu = (Student*)userHandle;
       pStu->SayHello();
       return a+1;
   }
   int main(int argc, char* argv[])
   {
       Student a("Andy");
       FunPtr fun = StudentCallBack;
       int hh = fun(3,(int)&a);
4、可以采用更高级的C++特性bind,普通成员方法指针,期望一个普通
   对外暴露成一个普通的方法,对内调用对象的成员方法。
```

参见

#### 构造方法调用另一个构造方法

```
1、考虑下面的代码,如下:
    class AAA
    {
    public:
        AAA():_A1(0),_A2(0),_A3(0)
        {
             public:
                int _A1;
                int _A2;
                int _A3;
                };
                现在增加一个构造方法,AAA(int a3) 对_A3初始化,怎么办?
2、使用笨的办法,如下:
                 class AAA
                {
```

```
public:
        AAA(): A1(0), A2(0), A3(0)
        }
        AAA(int a3):\_A1(0),\_A2(0),\_A3(a3)
    public:
        int _A1;
        int _A2;
        int _A3;
    };
    这当然可行,有没有更好的办法,解决代码重复。
3、容易想到的办法如下:
    class AAA
    {
    public:
       AAA(): A1(0), A2(0), A3(0)
        }
        AAA(int a3)
        {
           AAA();
           _{A3} = a3;
    public:
        int _A1;
        int _A2;
        int _A3;
    };
```

```
int main()
   {
       AAA a1;
       AAA a2(6);
       return 0;
   这种方法是有大问题的,因为AAA a2(6); 表达的语义是构造一个
4、怎么办?
5、方法一是,提取出一个private的init方法,用于重复代码,如下
   class AAA
   {
   public:
       AAA()
       {
           init();
       }
       AAA(int a3)
           init();
           _{A3} = a3;
   public:
       int _A1;
       int _A2;
       int _A3;
   private:
       void init()
       {
           _{A1} = 0;
           _{A2} = 0;
           _{A3} = 0;
6、方法二,使用placement new,构造方法创建对象,有两个阶段,
```

```
第一个阶段分配内存,第二个阶段在分配的内存上根据构造方法初
   placement new 就是在第二个阶段执行。如下:
   class AAA
   {
   public:
      AAA(): A1(0), A2(0), A3(0)
      }
      AAA(int a3)
          new (this)AAA();
          A3 = a3;
   public:
      int _A1;
      int A2;
      int A3;
   注意:placement new 在linux下,需要包含头文件 #includ
7、高版本的VS(比如VS2013),已经支持构造方法转接,也就是构造。
   AAA(int a3):AAA()
   {
      A3 = a3;
   这种情况下,_A3不能使用初始化列表,必须赋值,因为AAA() E
```

# 查看内存的内容

```
1、在Debug模式下,栈上连续的两个对象,地址相差12个字节。如下:
    struct Person
    {
        int __Gentle;
        int __Age;
        int __Height;
```

```
int _Weight;
   };
   int main(int argc,char* argv[])
   {
      Person p1;
      p1._Gentle = 1;
      p1._Age = 18;
      p1._Height = 174;
      p1._Weight = 88;
      Person p2;
      p2._Gentle = 1;
      p2._Age = 18;
      p2._Height = 174;
      p2._Weight = 88;
      Person* pp1 = &p1;
      Person* pp2 = &p2;
      getchar();
      return 0;
   在Debug模式下,p1对象和p2对象的前后各有额外的4个字节,中
2、特别注意:对象p1内的字段之间,比如_Gentle和_Age并没有额外
   但是如果是栈上的两个int对象 Gentle和 Age,他们之间也会补
   也就是说,完整的两个对象之间有补充字节,对象内的字段之间没能
3、对象中的字段,顺序排列,字段中的字节逆序排列。
   注意:字节逆序排列是指从左到右,低位到高位。
4、堆和栈的区别
   栈的地址向下增长, 堆的地址向上增长
```

### 目标文件是什么样

- 1、.text是代码段,只读
- 2、.data是数据段,存储值不为0的静态对象
- 3、.bss,存储值为0的静态对象,初始化为0或者没有初始化的静态对复

- 4、为什么值为0的静态对象放在.data段,而不为0的放在.bss段? 二者对应的使用场景不一样。
  - 值不为0的静态对象,必须知道取值多少,必须分配空间,放在.da值为0的静态对象,既然取值确定了,没有必要分配空间,只要记住
- 5、.rodata read only data 只读数据段,不可修改,存储文本常
- 6、为什么代码段和数据段分开存放?
  - a、代码段只读,数据段可读写,分开存放防止程序修改代码段
  - b、空间局部性原理,可以提高缓存命中率
  - c、代码段是只读的,不存在读写冲突,可以多线程共享,进程的内

#### 禁止在栈上创建对象

- 1、有时候要求对象必须创建在堆上,比如对象包含一个很大的数组,如 因此,必须在堆上创建对象,让数组随同对象分配在堆上。
- 2、那么问题来了,如何禁止在栈上创建对象? 栈上的对象,超出作用域,会自动调用析构方法,只要把析构方法。 但是这样会导致,delete的时候报错,因为delete 会先调用析构 delete没法调用private的析构方法。
- 3、怎么解决?

增加一个public方法Destroy,在Destroy中调用delete,类的

4、示例代码如下:

```
public:
    int _Age;
};

int main()
{
    //Person aa; // 报错 无法访问 private 成员
    Person* pa = new Person();
    //delete pa; // 报错 无法访问 private 成员
    pa->Destroy();
    return 0;
}
```

5、注意:当前对象可以调用本身的private方法,也可以调用同类对象因为方法是代码段,相对于类而言的,private也是针对类的。调对象可以调用所属类的private方法,当然包含同类对象的priva

#### 程序退出的处理

```
1、解决的问题:
        向系统注册方法,在程序退出的时候,回调这个方法进行额外的处理,
2、示例代码:
        #include<stdio.h>
        #include<stdlib.h>

        void func1(void)
        {
            printf("call func1\n");
        }

        void func2(void)
        {
            printf("call func2\n");
        }

        void func3(void)
        {
```

```
printf("call func3\n");
}
int main()
{
    atexit(func1);
    atexit(func2);
    atexit(func3);

    printf("exit\n");
    return 0;
}
3、注意:回调的顺序和注册的顺序相反,这是由于参数压栈导致的。
最多注册32个方法。
```

#### 编译32位和64位差异有哪些

```
1、32位
   Windows:
      指针是4个字节,long是4个字节
   Linux:
      指针是4个字节,long是4个字节
2、64位
   Windows:
      指针是8个字节,long是4个字节
   Linux:
      指针是8个字节,long是8个字节
3、需要修改的地方:
   a、对于64位,指针强转为int,存在问题(应该是long long)
   b、对于64位,日志打印指针,存在问题(应该是11d)
   c、对于64位,同时Linux,日志打印long,存在问题(应该是1]
4、怎么解决?
   a、指针强转为int,修改为使用intptr_t,intptr_t跨32位和
   b、日志打印指针,使用%p,%p跨32位和64位,只不过打印出来是
   c、日志打印long,同理使用%p,
5、另外,操作系统预定义的宏如下:
   Operating system Predefined macro
```

DOS 16 bit	MSDOS	_MSDOS
Windows 16 bit	_WIN16	
Windows 32 bit	_WIN32	WINDOWS
Windows 64 bit	_WIN64	_WIN32
Linux 32 bit	unix	linux
Linux 64 bit	unix	linux
BSD	unix	BSD
Mac OS	APPLE	DARWIN
0S/2	0S2	
表示64位,使用 #if 表示64位并且是Linux 表示32位并且是Windo	,使用 #if defin	redLP64

#### 编译找不到头文件

1、报错如下:

fatal error C1083: Cannot open include file: 'xx>

2、双击错误,找到include这一行

右击 Open Document "xxx.h", 会列出头文件的查找目录。

3、检查目录是否正确,以及目录中是否包含头文件

# 调试断点不起作用

1、原因是:

pdb文件存在问题。

2、解决办法:

a、项目 -> 属性 -> 配置属性 -> 链接器 -> 调试 -> 「生成

b、项目 -> 属性 -> 配置属性 -> C/C++ -> [调试信息格式]

c、项目 -> 属性 -> 配置属性 -> C/C++ -> 优化 -> 「优化

# 隐式类型转化

1、考虑下面的代码: class Dog {

```
public:
       Dog(string name):_Name(name)
       {
          printf("I am a dog[%s]\n",_Name.c_str());
       }
   public:
       string _Name;
   };
   class Person
   public:
       Person(Dog dog):_Dog(dog)
       {
          printf("I have a dog[%s]\n",_Dog._Name.c_
       }
   public:
       Dog _Dog;
   };
   int main(int argc, char* argv[])
   {
       string dogName = "Hali";
       Person p1 = dogName;
2、对于 Person p1 = dogName; 左边的p1之前不存在,要调用构造
   构造方法的参数期望接收Dog对象,但是右边是一个string对象,
   因此,需要一个适配过程,中间产生一个临时对象。如下:
   dogName --> 临时的Dog对象 -->p1
3、对于Person p2 = "Hali"; 为什么会编译失败?
   因为这里需要多次适配, "Hali" --> string --> Dog --> Person
   假如可以适配多次,那么编译就要找到所有可能的适配路径,编译
   可能会存在循环的适配过程,而且如果存在两条适配路径,路径长,
4、考虑下面的情况:
   Dog hali = "Hali";
```

```
p1 = hali;
p1 = dogName;
5、根据只能适配一次的规则,得到:
p1 = hali; p1调用copy赋值,期望右边是一个Person对象,ī
p1 = dogName; 同样道理,p1调用copy赋值,期望右边是一个
编译报错:error C2679: 二进制"=": 没有找到接受"std::s
也就是说,没有找到从string到Person的copy赋值,也不存在同
```

参见

#### 隐式类型转换 临时对象

```
1、隐式类型转换和临时对象之间没有关系,是从两个不同的角度来看问
   类型不匹配,就要进行隐式类型转换,但是中间并不一定要产生临一
   临时对象是由于类型不匹配,需要中间适配一个对象。
   也就是说,隐式类型转换并不一定要产生临时对象,而产生临时对
2、考虑Person类有一个_Age字段,
   int main(int agrc,char* argv[])
   {
      Person p1(8);
      Person p2 = 8;
      Person p3 = Person(8);
      Person p4 = p1;
      p4 = p1;
      p4 = Person(9);
      p4 = 9;
      return 0;
3、有两个地方存在隐式类型转换, Person p2 = 8; 和 p4 = 9;
   使用explicit Person(int age): Age(age){} 会导致这两
4、p4 = 9; 会产生临时对象,中间需要适配一个临时的对象。
5、逐个分析如下:
   int main(int agrc,char* argv[])
   {
```

```
Person p1(8); // 直接调用构造方法
   Person p2 = 8;  // 直接调用构造方法,不产
Person p3 = Person(8); // 直接调用构造方法,而且
                         // 调用copy构造
   Person p4 = p1;
   p4 = p1;
                        // 调用copy赋值
                        // 先调用构造方法,产生图
   p4 = Person(9);
                        // 先调用构造方法,产生\
   p4 = 9;
   return 0;
也就是说,下面三行的执行过程是等价的,唯一的区别就是,对于国
如果构造方法使用了explicit,会导致Person p2 = 8; 这一行
Person p1(8);
Person p2 = 8;
Person p3 = Person(8);
```

#### 隐式类型转换 隐式类型转换操作符 操作符重载

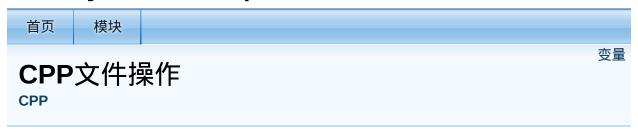
```
1、考虑下面的情况:有个Person类,内部有个 Age字段。正常情况下
   Person p = Person(8);
   p = Person(9);
   但是有个家伙很懒,他想这么写:
   Person p = 8;
   p = 9;
   这样写也是可以的,因为编译器知道你的意图,运行的时候会进行。
   但这样写不好,因为语义不通。要避免这种情况,使用explicit
2、现在在考虑下面的需求:判断p的年龄是否大于5,正常的写法是:
   if(p.GetAge()>5){....}
   但是,上面的家伙还是很懒,想直接这样写:
   if(p>5){....}
   这是情况,有两种解决办法,一种是让p转化为对应的_Age,一种:
   a、前一种办法,使用隐式类型转换操作符 operator int()
      这时候,需要提供一个隐式类型转换操作符,
      Person::operator int()
         return this->_Age;
```

```
    b、后一种办法,提供一个操作符重载,如下:
        bool operator>(const Person& lhs,const Person {
            return lhs.GetAge()>rhs.GetAge();
        }
        注意:如果两种方法同时提供,编译器会报错,因为编译器不知道.
3、操作符重载,本质上是方法,和GetName一样,operator+对应于而隐式类型转换操作符可以认为是一个特殊的成员方法,这个成员。
```

可以认为这个方法以operator开头,返回值是int,没有方法名,

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

ReadFromFile

ReadFromFile\_ByByte

WriteToFile

# 变量说明

#### ReadFromFile

```
void ReadFromFile(const string& filePath, string& out
{
   /*
       文件打开方式由r,w,a,t,b,+六个字符拼成,各字符的含义
       r(read):
                  读
       w(write):
                  写
       a(append): 追加
       t(text): 文本文件,可省略不写
       b(banary): 二进制文件
                  读和写
       +:
   */
   FILE* pFile = fopen(filePath.c_str(), "r"); // 读
   if (pFile == NULL)
   {
       return;
   }
   // 从文件结尾,向后偏移0个字节,也就是移到文件结尾
   if (fseek(pFile, 0, SEEK_END) == -1)
   {
       return;
   }
   // 获取当前位置,也就是文件的字节个数
   fpos_t fPos = 0;
   fgetpos(pFile, &fPos);
```

```
// 从文件开头,向后偏移0个字节,也就是移到文件开头
   fseek(pFile, 0, SEEK_SET);
   // 动态分配内存,
   char* buf = (char *)malloc(fPos+1); //要多分配一个:
   memset(buf, 0, fPos+1);
   // 把文件流读入到buf中,每次读取一个字节【sizeof(char)]
   fread(buf, sizeof(char), fPos, pFile);
   outStr = buf;
   fclose(pFile); // 不要忘记close文件句柄
   free(buf); // 不要忘记释放,使用free与malloc对应
}
分多次读取
void ReadFromFile_2(const string& filePath, string& or
   FILE* pFile = fopen(filePath.c_str(), "rb"); // i
   if (pFile == NULL)
   {
       return;
   }
   int sizeForOneRead = 1024*1024;
   char* buf = new char[sizeForOneRead];
   memset(buf, 0, sizeForOneRead);
   while(feof(pFile) == false)
   {
       int readSize = fread(buf, sizeof(char), sizeF
   }
   printf("%s\n", buf);
```

```
fclose(pFile); // 不要忘记close文件句柄
delete []buf; // 不要忘记释放,使用free与malloc对应
}
```

#### ReadFromFile\_ByByte

```
void ReadFromFile_ByByte(const string& filePath,strin
{
    FILE* pFile = fopen(filePath.c_str(), "r"); // 读
    if (pFile == NULL)
    {
        return;
    }
    char ch;

while(feof(pFile) == false)
    {
        ch = fgetc(pFile);
        outStr+=ch;
    }

fclose(pFile); // 不要忘记close文件句柄
}
```

#### WriteToFile

```
void WriteToFile(const string& filePath,const string&
{
    FILE* pFile = fopen(filePath.c_str(), "w");
    if (pFile == NULL)
    {
        return;
    }
    fwrite(inStr.c_str(), sizeof(char), inStr.size(),
    fclose(pFile);
}
```

```
使用fprintf
void WriteToFile_2(const string& filePath,const strinf
{
    static FILE* pFile = fopen(filePath.c_str(), "a+'
    if (pFile == NULL)
    {
        return;
    }
    fprintf(pFile,"%s",inStr.c_str());
    fflush(pFile);
    //fclose(pFile);
}
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 变量

【CPP沉思录】代理类

【CPP沉思录】句柄1

【CPP沉思录】句柄2

# 变量说明

#### 【CPP沉思录】代理类

```
1、考虑下面的场景:设计一个容器,包含一组类型不同但相互关联的对
2、容器一般只能包含一种类型的对象,使用vector<Animal> 会造成
3、经典的解决办法是:vector<Animal*>, 但是这会增加内存管理的
   Dog d;
   vec[i] = &d; // 局部对象d销毁, vec[i] 指向垃圾
   vec[i] = vec[j]; // 指向同一个对象, 在vec析构之前,需
4、怎么解决上面的问题,每次都创建一个新的对象。如下:
   Dog d;
   vec[i] = new Dog(d);
   vec[i] = new Animal(vec[j]); // 这里存在问题,由于ve
5、处理编译时类型未知的对象,使用虚方法。Animal 增加一个纯虚方
   不能实例化,同时要求子类必须重写clone方法。注:Animal 可
6、Animal* pa = new Dog; delete pa; 为了能够调用Dog的析材
   方法是虑方法:表面类型和真实类型不一致。
7、有没有更好的办法呢?
   使用代理类,对Animal*管理。也就是栈上对象管理动态资源,表
   在析构方法中释放资源。
8、代码如下:
AnimalProxy::AnimalProxy():_pa(NULL)
{
}
AnimalProxy::~AnimalProxy()
{
```

```
delete _pa;
}
// 注意:类中可以访问自己的private成员,也可以访问rhs的priva
AnimalProxy::AnimalProxy(const AnimalProxy& rhs)
{
   _pa = (rhs._pa != NULL ? rhs._pa->Clone():NULL);
}
AnimalProxy& AnimalProxy::operator=(const AnimalProxy
{
   if(this != &rhs) // 等同测试
    {
       delete pa; // delete NULL 也没有问题
       _pa = (rhs._pa != NULL ? rhs._pa->Clone():NUL
   return *this; // 返回引用
}
AnimalProxy::AnimalProxy(const Animal& animal):_pa(ar
{
}
```

参见

# 【CPP沉思录】句柄1

- 1、在【CPP沉思录】代理类中,使用了代理类,存在问题:
  - a、代理复制,每次创建一个副本,这个开销有可能很大
  - b、有些对象不能轻易创建副本,比如文件
- 2、怎么解决这个问题?
  - 使用引用计数句柄,对动态资源封装,句柄包含指针,多个句柄可
- 3、使用引用计数句柄,是为了避免不必要的对象复制,因此我们要知道 这样才能确定何时可以释放资源。
- 4、需要注意的是:引用计数不能是句柄的一部分,如果怎么做,当前句同时,引用计数不能成为对象的一部分,如果这样做,要求我们重要

```
5、以Point类为例说明,解决办法是:增加一个新的类UPoint,包含F
class Point
{
public:
   Point():_x(0),_y(0){}
   Point(int x, int y):_x(x),_y(y){}
   Point(const Point& rhs):_x(rhs._x),_y(rhs._y){}
   Point& SetX(int x)
   {
       _x = x;
       return *this;
   }
   int GetX()
   {
       return _x;
   }
   Point& SetY(int y)
   {
       _y = y;
       return *this;
   }
   int GetY()
   {
       return _y;
   }
private:
   int
         _x;
   int
           _y;
};
#include "point.h"
// UPoint的目的是对Point和引用计数封装,用户是不可见的
```

```
class UPoint
{
    friend class Handle_1;
private:
    Point p;
    int u;
    UPoint():u(1){}
    UPoint(int x,int y)
    {
        p.SetX(x);
        p.SetY(y);
        u = 1;
    }
    UPoint(const Point& rhs)
    {
        p = rhs;
        u = 1;
};
6、现在考虑Handle_1的实现细节,
#include "u_point.h"
class Handle_1
{
public:
    Handle_1():_up(new UPoint){}
    Handle_1(int x,int y):_up(new UPoint(x,y)){}
    Handle_1(const Point& rhs):_up(new UPoint(rhs)){}
    ~Handle_1()
```

```
subRef();
}
// copy构造,复制指针,增加引用
Handle_1(const Handle_1& rhs)
{
    addRef(rhs._up);
}
// copy赋值,左边减少引用计数,判断是否delete,右边增加。
Handle_1& operator=(const Handle_1& rhs)
{
    if(this != &rhs)
    {
        subRef();
        addRef(rhs._up);
    return * this;
}
int GetX()
{
    return _up->p.GetX();
}
int GetY()
{
    return _up->p.GetY();
}
Handle_1& SetX(int x)
{
   _up->p.SetX(x);
   return *this;
}
```

```
Handle_1& SetY(int y)
   {
       _up->p.SetY(y);
       return *this;
   }
private:
   void addRef(UPoint* up) // 复制指针,增加引用
   {
       _{up} = up;
       ++_up->u;
   }
                // 减少引用,判断是否delete
   void subRef()
   {
       if(--up->u == 0)
       {
          delete _up;
   }
private:
   UPoint* _up;
};
7、考虑下面的情况,
   Handle_1 h1(3,4);
   Handle_1 h2(h1);
   h2.SetX(5);
   int dd = h1.GetX();
   dd的值是5,也就是说,多个句柄指向同一个对象,避免了不必要£
   怎么解决这个问题?
8、使用写时拷贝,每次修改的时候重新创建一个对象。也就是说,修改
   如下:
   Handle_1& SetX(int x)
       //_up->p.SetX(x);
```

```
if(_up->u == 1) // 当前是唯一的引用
    {
        _up->p.SetX(x);
    }
    else
    {
        --_up->u;
        _up = new UPoint(x,_up->p.GetY());
    }
    return *this;
}
Handle_1& SetY(int y)
{
    //_up->p.SetY(y);
    if(_up->u == 1) // 当前是唯一的引用
    {
        _up->p.SetY(y);
    }
    else
    {
        --_up->u;
        _up = new UPoint(_up->p.GetX(),y);
    return *this;
}
```

参见

# 【CPP沉思录】句柄2

1、【CPP沉思录】句柄1 存在问题: 句柄为了绑定到Point的对象上,必须定义一个辅助类UPoint,女 2、有没有更简单的办法呢? 句柄使用Point\*直接绑定到Point对象上(包括子类),为了保持 3、代码如下:

```
#include "point.h"
class Handle_2
{
public:
    Handle_2():_p(new Point),_u(new int(1)){}
    Handle_2(int x,int y):_p(new Point(x,y)),_u(new i
    Handle_2(const Point& rhs):_p(new Point(rhs)),_u(
    ~Handle_2()
    {
        subRef();
    }
    Handle_2(const Handle_2& rhs)
    {
        addRef(rhs);
    }
    Handle_2& operator=(const Handle_2& rhs)
    {
        if(this != &rhs)
        {
            subRef();
            addRef(rhs);
        return * this;
    }
    int GetX()
    {
        return _p->GetX();
    }
    int GetY()
```

```
{
        return _p->GetY();
    }
    Handle_2& SetX(int x)
    {
        if(*_u == 1) // 当前是唯一的引用
        {
            _p->SetX(x);
        }
        else
        {
            --*_u;
            _p = \text{new Point}(x, _p -> \text{GetY}());
        return *this;
    }
    Handle_2& SetY(int y)
    {
        if(*_u == 1) // 当前是唯一的引用
        {
            _p->SetY(y);
        }
        else
        {
            --* u;
            _p = new Point(_p->GetX(),y);
        return *this;
    }
private:
    void addRef(const Handle_2& rhs) // 复制对象指针和引
        _p = rhs._p;
        _u = rhs._u;
```

```
++*_u;
   }
    void subRef()// 减少引用,判断是否delete对象和引用计数
    {
       if(--*_u == 0)
       {
           delete _p;
           delete _u;
       }
    }
private:
   Point* _p;
   int* _u;
};
4、这里要手动管理动态内存 int* _u; 同样道理,可以使用栈上对象
class UseCount
{
public:
    UseCount():_p(new int(1)){}
    UseCount(const UseCount& rhs)
    {
       addRef(rhs);
    }
   UseCount& operator=(const UseCount& rhs)
    {
       if(this != &rhs)
        {
           subRef();
           addRef(rhs);
       }
       return *this;
    }
    ~UseCount()
```

```
{
        subRef();
    }
    bool IsOnly()
    {
        return (*_p == 1);
    }
    void MakeOnly()
        if(IsOnly()) // 防止已经是only,用户还调用MakeOn.
            return;
        }
        --*_p;
        _p = new int(1);
    }
private:
    void addRef(const UseCount& rhs)
    {
        _p = rhs._p;
        ++*_p;
    }
    void subRef()
    {
        if(--*_p == 0)
            delete _p;
    }
private:
    int* _p;
};
```

```
#include "point.h"
#include "use_count.h"
class Handle 2
{
public:
    Handle_2():_p(new Point){}
    Handle_2(int x,int y):_p(new Point(x,y)){}
    Handle_2(const Point& rhs):_p(new Point(rhs)){}
    ~Handle_2()
    {
        subRef();
    }
    Handle_2(const Handle_2& rhs)
    {
        addRef(rhs);
    }
    Handle_2& operator=(const Handle_2& rhs)
    {
        if(this != &rhs)
        {
            subRef();
            addRef(rhs);
        return * this;
    }
    int GetX()
    {
        return _p->GetX();
```

```
int GetY()
    {
        return _p->GetY();
    }
    Handle_2& SetX(int x)
    {
        if(_u.IsOnly()) // 当前是唯一的引用
        {
            _p->SetX(x);
        else
        {
            _u.MakeOnly();
            _p = new Point(x,_p->GetY());
        return *this;
    }
    Handle_2& SetY(int y)
    {
        if(_u.IsOnly()) // 当前是唯一的引用
        {
            _p->SetY(y);
        else
        {
            _u.MakeOnly();
            _p = new Point(_p->GetX(),y);
        return *this;
    }
private:
    void addRef(const Handle_2& rhs)
        _p = rhs._p;
```

```
_u = rhs._u;
}

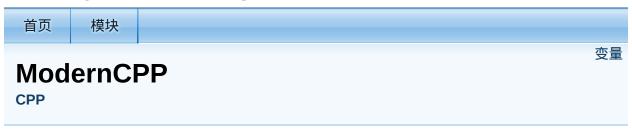
void subRef()
{
    if(_u.IsOnly())
     {
        delete _p;
     }
}

private:
    Point* _p;
    UseCount _u;
};
```

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

CPP遍历的方法汇总

CPP11的thread

move操作

### 变量说明

#### CPP11的thread

```
1、先看C++98的线程测试代码
    #include <pthread.h>
    #include <unistd.h>
    #include <stdio.h>
    #include <stdlib.h>
    using namespace std;
    void* TestThread1(void*);
    int main(int argc, char* argv[])
    {
        pthread_t hThread;
        pthread_attr_t pAttr;
        pthread_attr_init(&pAttr);
        pthread_attr_setdetachstate(&pAttr,PTHREAD_CF
        pthread_create(&hThread, &pAttr, TestThread1,
        printf("主线程:你好帅!!!\n");
        getchar();
        return 0;
    }
    void* TestThread1(void*)
    {
        sleep(5);
        exit(0);
```

```
2、编译,运行如下:
   root@ubuntu:/home/disk1/CPP_2/TestThread# g++ -o
    root@ubuntu:/home/disk1/CPP 2/TestThread# ./main
3、C++11的线程测试代码
   #include <thread>
   #include <unistd.h>
   using namespace std;
   void TestThread1();
   int main(int argc, char* argv[])
    {
       thread t1(TestThread1);
       t1.detach();
       printf("主线程:你好帅!!!\n");
       getchar();
       return 0;
   }
   void TestThread1()
    {
       sleep(5);
       exit(0);
4、编译,运行如下:
   root@ubuntu:/home/disk1/CPP 2/TestThread# q++ -o
   In file included from /usr/include/c++/4.8/threac
                   from main_11.cpp:1:
   /usr/include/c++/4.8/bits/c++0x_warning.h:32:2: €
   #error This file requires compiler and library su
    Λ
   main_11.cpp: In function 'int main(int, char**)':
   main 11.cpp:9:5: error: 'thread' was not declared
       thread t1(TestThread1);
   编译报错,使用C++11的thread,需要添加选项 -std=c++11
5、再次测试如下:
```

```
root@ubuntu:/home/disk1/CPP_2/TestThread# g++ -o root@ubuntu:/home/disk1/CPP_2/TestThread# ./main_terminate called after throwing an instance of 's what(): Enable multithreading to use std::the Aborted (core dumped)
6、运行报错,查了一下资料,链接pthread的时候,必须去掉 1,如1 root@ubuntu:/home/disk1/CPP_2/TestThread# g++ -o root@ubuntu:/home/disk1/CPP_2/TestThread# ./main_7、那么问题来了,-lpthread和-pthread到底什么区别?使用 g++ -v 输出详细信息,进行比较,发现存在一些区别。-pthread多了一些 -pthread, -D_REENTRANT
```

#### CPP遍历的方法汇总

```
1、示例代码如下:
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
// C语言的惯用方法
void traversal_1(vector<int>& intVec)
{
    for (int i = 0; i < intVec.size(); i++)
        printf("%d, ",intVec[i]);
    printf("\n");
}
// for循环使用迭代器
void traversal_2(vector<int>& intVec)
{
    for (vector<int>::iterator iter = intVec.begin();
        printf("%d, ", *iter);
```

```
printf("\n");
}
// 迭代器使用auto
void traversal_3(vector<int>& intVec)
   for (auto iter = intVec.begin(); iter != intVec.e
   {
       printf("%d, ", *iter);
   printf("\n");
CPP遍历的方法汇总
// foreach加lambda
// 特别注意: 这里的cbegin 返回const迭代器
void traversal_4(vector<int>& intVec)
{
   for_each(intVec.cbegin(), intVec.cend(), [](const
   printf("\n");
}
// for区间遍历
// 注意: 这种方法是 traversal_3的缩写, auto val = *iter
// 需要注意这里的 auto val, val是迭代器元素的副本, 如果for(
// 也就是说, auto可以理解为文本替换, 但是只能替换成类型, 如果要
void traversal_5(vector<int>& intVec)
{
   for(auto val : intVec)
   {
       printf("%d, ", val);
   }
   printf("\n");
}
```

```
int main()
{
    vector<int> intVec = { 0, 1, 2, 3, 4, 5 };

    traversal_1(intVec);
    traversal_2(intVec);
    traversal_3(intVec);
    traversal_4(intVec);
    traversal_5(intVec);

    return 0;
}
```

#### move操作

1、解决什么问题?

考虑下面的场景,甲乙住宾馆,甲对房间里面的家具不满意,对乙

2、怎么解决这个问题?

笨的办法是,先清空甲房间的家具,对乙房间的每个家具复制一个这就是copy赋值的做法。

- 3、上面的办法,显然成本比较大。更好的办法是: 把乙房间的钥匙给甲,不就0K了嘛,反正乙不再使用房间。
- 4、怎么实现?

相对于上面的copy赋值,需要增加一种copy赋值,暂且叫做move为了区分正常的copy赋值,需要区分构造参数,也就是乙不再使用而move方法就是把左值参数,转化为右值参数。

5、注意:

move只是把参数的属性修改一下,使它成为右值,触发调用接收在copy构造也是相同的道理。

6、也就是说,增加了移动构造和移动赋值。

# Andy Niu Help 1.0.0.0

## 变量

理解实例和静态

字段和方法\_\_实例和静态

抑制多态机制

多态行为的条件

### 变量说明

#### 多态行为的条件

- 1、第一是虚方法,第二表面类型和真实类型不一样,也就是使用引用或
- 2、考虑下面的代码, Say是虚方法

Dog d;

Animal a = d;

a.Say();

3、a.Say(); 不具备多态行为。为什么?

Animal a = d; 会发生对象切割,也就是d中的subobject会际但是特别注意,其中的vptr字段不会被复制,也就是vptr字段初级那么,a的表面类型和真实类型都是Animal,a.Say()调用被编译取出虚方法表的第2个槽位(注:第一个槽位[0]是type\_info,解引用,vptr指向Animal的虚方法表,然后传递this指针,作为如果编译器足够聪明,这里可以抑制多态行为,a.Say()转化为a.如果Animal::Say()是inline方法,效率会更高。

4、考虑多态行为,就是00(Object Oriented面向对象)。不考虑多

### 字段和方法 实例和静态

### 1、考虑字段:

- a、从语义来讲,实例字段对每个对象有意义,针对每个对象,大复静态字段对所有对象和类本身有意义,针对所有对象和类本身 举例来说,对于Person类,Age是实例字段,MaxAge是静态和Person类,最大的年龄是一样的(比如150岁)。
- b、从内存来讲,实例字段是每个对象有独属于自己的一块内存,至 2、考虑方法:
  - a、从语义来讲,实例方法可以访问自己的字段和静态的字段,静态

但是不能访问其他对象的实例字段。特别注意:这里的不能说如果方法的参数列表中包含其他对象,当然可以访问,甚至可(特别注意,private有两层含义:当前对象可以访问自己的必须是同类型,Dog对象可以访问其他Dog对象的private成静态方法,不依赖具体的对象,即使没有对象,也可以调用前

- b、从内存来讲,无论实例方法还是静态方法,在内存中只有一个排相对于普通方法,实例方法在形参表中在第一个位置增加一个相对于普通方法,静态方法可以认为是名称空间内的一个普通也就是只能访问同一个名称空间的字段。静态方法可以通过类
- 3、理解了静态方法,就知道为什么静态方法不能访问实例字段。静态方const方法解决什么问题,对于实例方法,this指针不能修改指向使用const方法,也就是让this指针指向const对象。但是静态方
- 4、字段和方法编译之后,也就是数据段和代码段,数据段存在多个内存代码段只有一个内存拷贝,而且是只读的,可以加工同一类型的数据

参见

### 抑制多态机制

```
1、我们知道,虚方法的调用,编译器会转化为虚方法表中对应的slot,

考虑,下面的情况

#include <stdio.h>

class Animal

{

public:

virtual void Say(int a);

};

#include "Animal.h"

void Animal::Say(int a)

{

printf("Animal::I am Say(int a)");

}

#include "Animal.h"

class Dog : public Animal
```

```
public:
       virtual void Say(int a);
   };
   #include "Dog.h"
   void Dog::Say(int a)
   {
       Say(a);
       printf("Dog::I am Say(int a)");
   }
   测试代码,
   int main(int argc, char* argv[])
   {
       Animal* pa = new Dog();
       pa->Say(1);
       return 0;
   在子类中调用Say方法,由于多态机制,又调用子类的Say方法,引
   如何解决?
   这个时候需要抑制多态机制,如下:
   void Dog::Say(int a)
   {
       Animal::Say(a);
       printf("Dog::I am Say(int a)");
3、考虑下面的情况,增加一个虚方法Play,在Dog的Play方法中,如<sup>T</sup>
   void Dog::Play()
   {
       Say(5);
       printf("Dog::I am Play()");
   }
   这里的Say会进行多态机制,调用Dog的Say方法。我们知道,这种
   void Dog::Play()
```

```
Dog::Say(5);
printf("Dog::I am Play()");
}
相对于上一种做法,这种做法减少了运行期对虚方法的决议过程(
```

参见

### 理解实例和静态

- 1、成员分为字段和方法
- 2、考虑字段,实例字段占用对象内存,每个对象有专属于自己的一块内静态字段不占用对象内存,内存中只有一份,类和所有对象共享。
- 3、考虑方法,和字段不同,方法是代码段。 无论是实例方法还是静态方法,并不占用对象的内存。在内存中只 方法是相对于类的,实例方法关联this指针,静态方法不关联thi
- 4、考虑方法内的static字段,是针对所有对象的,所有对象都会调用i 这个static字段不光是当前对象进来只执行一次,是所有对象进来

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 函数 | 变量 Example

# 函数

int **GetString** (string &outStr, const string &inStr) 函数的说明 更多...

# 重变

### 变量和命令

## 详细描述

## 函数说明

参见

参见其它的函数

## 变量说明

### 变量和命令

### 变量和命令的用法

参见

参见其它的变量和命令

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 模块 | **Linux** 

# 模块

Awk		
Cmake		
Flawfinder		
Gdb		
Git		
Grep		
Help		
IO多路复用		
Iptables		
Linux版本		
Nm		
Ntp		
Sed		
Shell变量		
Ssh		
Ubuntu		
Vim		

启动关机
常用的服务
常见问题
打包和压缩
文件与目录
文件系统和目录树
用户管理
管道和管道命令
编译 ————————————————————————————————————
网络有关
进程管理

# 重变

<u></u>	
dmesg	
Isof	
read	
strace	
top命令详解	
•	
valgrind	
•	
Linux数据流重定向	
,	
日期时间	

### 详细描述

### 变量说明

#### dmesg

- 1、dmesg【dmessage】显示内核缓冲区信息,包括设备启动,系统架。
- 2、通过dmesg常常诊断设备的故障,dmesg常常结合管道和其它命令一

#### 参见

#### Linux数据流重定向

- 1、标准输出:代码是1,使用>和>>,前者是覆盖,后者是追加。
  - xxx > aaa.txt
  - xxx >> aaa.txt
- 2、对于错误的数据,需要使用标准错误输出,标准错误输出:代码是2
- 3、正常信息和错误信息输出到不同的文件,如下:
  - xxx >info.txt 2>err.txt
- 4、正常信息和错误信息输出到同一个文件,
  - xxx >info.txt 2>info.txt 是错误的写法,info.txt的内容xxx >info.txt 2>&1
- 5、如果信息既不想显示也不想存储呢,比如操作的历史记录,为了安全使用黑洞设备,xxx >/dev/null 2>&1
- 6、标准输入,代码是0,使用<和<<,解决的问题将原本需要键盘输出的
- 7、cat >tmp.txt 从键盘输入数据到tmp.txt,ctrl+d结束输入。5
- 8、标准输入,<<表示遇到某个输入,结束输入。如下:
  - [root@localhost home]# cat >tmp.txt <<'0'</pre>
  - > aaa
  - > bbb
  - > 0

### Isof

1,	lsof list	open	file	列出系:	统打开的	勺文件	
2、	对于父进程」	监听的	端口转科	多到子进	程的例识	子,使用1s	of可以查看相主
	[root@lo	ocalho	ost ni	.u]# ne	tstat	-anp gre	ep 9000
	tcp	Θ		0.0.	0.0:90	900	(
	[root@lo	ocalho	ost ni	.u]# ls	of -ps	5334	
	COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE
	main	5334	root	cwd	DIR	8,2	4096 454
	main	5334	root	rtd	DIR	8,2	4096
	main	5334	root	txt		•	9025 454
	main	5334	root	mem	REG		125736 193
	main	5334	root	mem	REG	8,2	1611564 193
	main	5334	root	mem	REG	8,2	208352 193
	main	5334	root	mem	REG	8,2	936908 20
	main	5334	root	mem	REG	8,2	46636 193
	main	5334	root			•	
	main		root			136,4	
	main			2u		•	
	main			3u		1191891	
	[root@lo	ocalho	ost ni	.u]# ls	of -i	: 9000	
	COMMAND						SIZE NODE N
	main					1191891	
						1191891	
	ping			3u		1191891	
3、							可执行文件,;
		_		-	- '	-	,3[Socket监
4、							i使用close-o
							ep 9000只能看
5、							<b>了父进程打开文</b>
	-			_		-anp gre	· •
	tcp	0		0 0.0.			(
	[root@lo			_			0.775
	COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE NODE N

参见

#### read

```
1、考虑下面的需求,在shell脚本运行过程中,提示用户输入,并且打
   使用read如下:
   #! /bin/sh
   read -p "please input your name: " name
   echo "hello $name, welcome"
2、检查用户的输入,如果名字太短,提示重新输入,直到满足要求。如
   #! /bin/sh
   read -p "please input your name: " name
   while [ ${#name} -lt 4 ]
   do
          echo "sorry, your name is too short"
          read -p "try input again: " name
   done
   echo "hello $name, welcome"
3、如果用户没有输入,就一直卡在这里,怎么解决?
   给出超时时间,没有输入,直接跳过去。如下:
   #! /bin/sh
   read -t5 -p "please input your name: " name
   echo "hello $name, welcome"
4、考虑输入密码,密码的位数最长是6位,输入6位后,直接返回,不等
   #! /bin/sh
   read -n6 -p "please input your passwd: " passwd
   echo "hello, password is $passwd"
5、对于密码的输入,用户不期望显示输入的内容,使用-s 默读,如下
   #! /bin/sh
   read -s -p "please input your passwd: " passwd
   echo "hello, password is $passwd"
```

#### strace

```
1、strace跟踪进程运行时的系统调用和所接受的信息。
2、举例来说,示例代码:
   #include <stdio.h>
   int main(int argc, char* argv[])
    {
            printf("hello,world\n");
            getchar();
            return 0;
3、运行strace ./main,如下:
    [root@localhost strace]# strace ./main
   execve("./main", ["./main"], [/* 23 vars */]) = 0
   brk(0)
                                           = 0x99da@
   access("/etc/ld.so.preload", R_OK)
                                         = -1 ENOE
   open("/etc/ld.so.cache", O_RDONLY)
   fstat64(3, {st_mode=S_IFREG|0644, st_size=44224,
   mmap2(NULL, 44224, PROT_READ, MAP_PRIVATE, 3, 0)
   close(3)
   open("/usr/lib/libstdc++.so.6", O_RDONLY) = 3
   read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\
   fstat64(3, {st_mode=S_IFREG|0755, st_size=936908,
   mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIV/
   mmap2(0x21da000, 961720, PROT_READ|PROT_EXEC, MAF
   mmap2(0x22ba000, 20480, PROT READ|PROT WRITE, MAF
   mmap2(0x22bf000, 23736, PROT_READ|PROT_WRITE, MAF
   close(3)
                                            = 0
   open("/lib/libm.so.6", O_RDONLY)
   read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\
   fstat64(3, {st mode=S IFREG|0755, st size=208352,
   mmap2(0x817000, 155760, PROT_READ|PROT_EXEC, MAP_
   mmap2(0x83c000, 8192, PROT_READ|PROT_WRITE, MAP_F
   close(3)
                                            = 0
```

```
open("/lib/libgcc_s.so.1", O_RDONLY)
    read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\
    fstat64(3, {st_mode=S_IFREG|0755, st_size=46636,
    mmap2(0x7cee000, 48196, PROT_READ|PROT_EXEC, MAP_
    mmap2(0x7cf9000, 4096, PROT_READ|PROT_WRITE, MAP_
    close(3)
    open("/lib/libc.so.6", O_RDONLY)
                                             = 3
    read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\
    fstat64(3, {st_mode=S_IFREG|0755, st_size=1611564
    mmap2(0x6c9000, 1332676, PROT_READ|PROT_EXEC, MAF
    mprotect(0x808000, 4096, PROT_NONE)
    mmap2(0x809000, 12288, PROT_READ|PROT_WRITE, MAP_
    mmap2(0x80c000, 9668, PROT_READ|PROT_WRITE, MAP_F
    close(3)
                                             = 0
    mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIV/
    set_thread_area({entry_number:-1 -> 6, base_addr:
    mprotect(0x809000, 8192, PROT_READ)
                                             = 0
    mprotect(0x83c000, 4096, PROT_READ)
                                             = 0
    mprotect(0x22ba000, 16384, PROT_READ)
                                             = 0
    mprotect(0x6c5000, 4096, PROT_READ)
                                             = 0
    munmap(0xb7f68000, 44224)
                                             = 0
    fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev
    mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIV/
    write(1, "hello, world\n", 12hello, world
    )
                = 12
    fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev
    mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIV/
    read(0,
4、kill -9 6456,如下:
    read(0, <unfinished ...>
    +++ killed by SIGKILL +++
```

参见

【第一行】任务队列信息

16:44:36

up 113 days, 36 min

1 user

load average: 0.10, 0.04, 0.00

表示当前系统的时间 表示该机器已经启动了 当前连接到该机器的有 分别对应1、5、15分钟

关于该参数的详细说明:该参数是平常监控机器的运行状况的一个主要抗这里存放待执行的进程,只要不是在等待IO或者主动的wait就是可运行如果这样的进程越来越多,cpu就需要花很长的时间来处理这些进程。这样cpu的负载其实就是指正在执行的进程以及处于运行队列中的进程的

【第二行】进程的运行情况

各个参数的具体含义分别是:总进程数,正在执行的进程数,休眠的进程

Tasks: 111 total 进程总数;

2 running 正在运行的进程数;

108 sleeping睡眠的进程数;1 stopped停止的进程数;0 zombie僵尸进程数;

【第三行】cpu的利用率

 Cpu(s):
 0.0%us
 用户空间占用CPU百分比;

 0.0%ov
 中核空间上用CPU百分比;

0.0%sy 内核空间占用CPU百分比; 0.0%ni 用户进程空间内改变过优先级的进

99.9%id 空闲CPU百分比;

0.0%wa 等待输入输出的CPU时间百分比;

0.0%hi, 0.0%si, 0.0%st 不常用,不做介绍;

【第四行】物理内存的使用情况

Mem: 5242880k total 物理内存的总量;

4403900k used 838980k free

285712k buffers 用作内核缓存的内存量

【第五行】交换区的使用情况

Swap: 2097144k total

112k used

2097032k free 3127936k cached ╆╏┢┢╗╬╱╒┻╏╪┲╗╟

空闲内存总量;

交换区的总量;

已经使用的交换区的量;

已经使用的物理内存的总理;

空闲交换区的量; 缓冲的交换区总量。

内存中的内容被换出到交换区,而后又被换入到内存,但使用过的交换区该数据即为这些内容已经存在于内存中的交换区的大小。相应的内存再次

### 【第六行】 空行

### 【第七行】进程信息区

讲行信息区的信息		'各个进程目前的运行状况:
	名名含义	
a	PID	进程id
b	PPID	之 父进程id
С	RUSER	Real user name
d	UID	进程所有者的用户id
е	USER	进程所有者的用户名
f	GROUP	进程所有者的组名
g	TTY	启动进程的终端名。不是从终端启动的意
h	PR	优先级
i	NI	nice值。负值表示高优先级,正值表示
j	Р	最后使用的CPU,仅在多CPU环境下有意
k	%CPU	上次更新到现在的CPU时间占用百分比
1	TIME	进程使用的CPU时间总计,单位秒
m	TIME+	进程使用的CPU时间总计,单位1/100和
n	%MEM	进程使用的物理内存百分比
0	VIRT	进程使用的虚拟内存总量,单位kb。VI
p	SWAP	进程使用的虚拟内存中,被换出的大小
q	RES	进程使用的、未被换出的物理内存大小
r	CODE	
S	DATA	可执行代码以外的部分(数据段+栈)占属
t	SHR	共享内存大小,单位kb
u	nFLT	页面错误次数 图点,次图入到现 <i>在</i> ,被像次法统善无法
V	nDRT	最后一次写入到现在,被修改过的页面
W	S	进程状态。
		D=不可中断的睡眠状态
		R=运行
		S=睡眠 T 明瞭 //ē L
		T=跟踪/停止
V	COMMAND	Z=僵尸进程 今今夕/今今行
X	COMMAND	命令名/命令行 李汝洪积太晦明,则且于晦明中的系统。
У	WCHAN	若该进程在睡眠,则显示睡眠中的系统

任务标志 Flags 默认情况下仅显示的信息包括:PID USER PR NI VIRT RES 可以通过如下的快捷方式来更改显示的内容: 通过【f】键可选择显示的内容,按f后会显示列的列表,按a-z即可显示 通过【o】键可改变列的显示顺序。按小写的a-z可以将相应的列向右移过 按【F】或【O】,按a-z可将进程按相应的列进行排序,按【R】可将当 命令的使用方法 top [-] [d] [p] [c] [S] [s] 指定每两次屏幕信息刷新之间的时间间隔。当然用户可以使用 d 通过指定监控进程 ID来仅仅监控某个进程的状态; p 使top命令在安全模式下运行,这将免出人意料交互命令所带 S S 指定累积模式: i 使top不显示任何闲置或僵死的进程; 显示整个命令行而不仅仅是显示命令名。 注意:top默认只列出活跃的进程,如果要列出所有的进程,使用 top top的一些交互命令 显示帮助画面,给出一些简短的命令总结说明。 h或者? 终止一个进程,系统提示输入一个PID及需要发送给该过 k i 忽略闲置和僵死的进程,这是一个开关式命令; 退出程序; q 重新安排一个进程的优先级别。系统提示用户输入需要证 r S 切换到累积模式: 改变两次刷新之间的延迟时间,系统提示用户输入新的时 S 显示的进程数,系统会提示输入需要显示的进程数目; n 从当前显示项目中添加或删除项目: f 改变当前显示项目的顺序; 0 切换显示平均负载和启动时间的信息; 1 切换显示内存信息; m 切换显示进程和CPU状态信息; t 切换命令名称和完整命令行: С 根据驻留内存大小进行排序; М 根据CPU使用百分比大小进行排序; Р 根据时间/累积时间进行排序。 Т 选择其它的字段排序 F或者0

### 常用的选项:

- 1、输入1, 查看cpu的个数和使用情况
- 2、top -H -p7073 查看进程7073以及子进程的情况 注意:在多核设备上,进程的cpu使用超过100%有可能是正常的。 因为多个子进程在不同的cpu上面,每个都使用很低,在一起有可能

### valgrind

1、valgrind是什么? valgrind是Linux下仿真调试工具的集合,由内核和基于内核的ì 2、valgrind包括的工具: 这是valgrind应用最广泛的工具,一个重量级的 Memcheck: 比如,使用未初始化的内存,使用已经释放了的内 它主要用来检查程序中函数调用过程中出现的问题 Callgrind: Cachegrind:它主要用来检查程序中缓存使用出现的问题。 它主要用来检查多线程程序中出现的竞争问题。 Helgrind: Massif: 它主要用来检查程序中堆栈使用中出现的问题。 Extension: 可以利用core提供的功能,自己编写特定的内存。 3、Memcheck能够检测出内存问题,关键在于其建立了两个全局表。 Valid-Value表:记录内存的值是否初始化,是否有效。 Valid-Address表:记录该地址是否能够被读写。 4、重点介绍内存的检测情况,内存没有初始化。 代码: int main(int agc, char\* argv[]) { int\* pi; int a = \*pi;return 0; 测试: [root@localhost valgrind]# valgrind --tool=memche ==7422== Memcheck, a memory error detector ==7422== Copyright (C) 2002-2009, and GNU GPL'd, ==7422== Using Valgrind-3.5.0 and LibVEX; rerun v ==7422== Command: ./main ==7422== ==7422== Use of uninitialised value of size 4 ==7422== at 0x8048408: main (in /home/niu/valg

```
==7422==
   ==7422==
   ==7422== HEAP SUMMARY:
                 in use at exit: 0 bytes in 0 blocks
   ==7422==
               total heap usage: 0 allocs, 0 frees, (
   ==7422==
   ==7422==
   ==7422== All heap blocks were freed -- no leaks a
    ==7422==
   ==7422== For counts of detected and suppressed er
   ==7422== Use --track-origins=yes to see where uni
   ==7422== ERROR SUMMARY: 1 errors from 1 contexts
   指出上下文有一个错误,使用--track-origins=yes查看,栈上
    [root@localhost valgrind]# valgrind --tool=memche
   ==7483== Memcheck, a memory error detector
   ==7483== Copyright (C) 2002-2009, and GNU GPL'd,
   ==7483== Using Valgrind-3.5.0 and LibVEX; rerun v
   ==7483== Command: ./main
   ==7483==
   ==7483== Use of uninitialised value of size 4
                at 0x8048408: main (in /home/niu/valc
   ==7483==
   ==7483== Uninitialised value was created by a st
                at 0x8048402: main (in /home/niu/valç
   ==7483==
   ==7483==
   ==7483==
   ==7483== HEAP SUMMARY:
                 in use at exit: 0 bytes in 0 blocks
   ==7483==
               total heap usage: 0 allocs, 0 frees, 6
   ==7483==
   ==7483==
   ==7483== All heap blocks were freed -- no leaks a
   ==7483==
   ==7483== For counts of detected and suppressed er
    ==7483== ERROR SUMMARY: 1 errors from 1 contexts
5、内存泄露
   代码:
   int main(int agc, char* argv[])
           int* pi = new int[10];
```

```
return 0;
    测试:
    [root@localhost valgrind]# valgrind --tool=memche
    ==7636== Memcheck, a memory error detector
    ==7636== Copyright (C) 2002-2009, and GNU GPL'd,
    ==7636== Using Valgrind-3.5.0 and LibVEX; rerun v
    ==7636== Command: ./main
    ==7636==
    ==7636==
    ==7636== HEAP SUMMARY:
    ==7636==
                 in use at exit: 40 bytes in 1 blocks
               total heap usage: 1 allocs, 0 frees, 4
    ==7636==
    ==7636==
    ==7636== 40 bytes in 1 blocks are definitely lost
    ==7636==
                at 0x4005D2D: operator new[](unsigned
                by 0x8048490: main (in /home/niu/valc
    ==7636==
    ==7636==
    ==7636== LEAK SUMMARY:
    ==7636==
                definitely lost: 40 bytes in 1 blocks
                indirectly lost: 0 bytes in 0 blocks
    ==7636==
                  possibly lost: 0 bytes in 0 blocks
    ==7636==
    ==7636==
                still reachable: 0 bytes in 0 blocks
                     suppressed: 0 bytes in 0 blocks
    ==7636==
    ==7636==
    ==7636== For counts of detected and suppressed er
    ==7636== ERROR SUMMARY: 1 errors from 1 contexts
    内存泄露40个字节
6、内存申请释放不匹配
    代码:
    #include <stdlib.h>
    int main(int agc, char* argv[])
    {
            int* p1 = new int[10];
            delete p1;
            int* p2 = new int;
```

```
free(p2);
            return 0;
   测试:
    [root@localhost valgrind]# valgrind --tool=memche
   ==8162== Memcheck, a memory error detector
   ==8162== Copyright (C) 2002-2009, and GNU GPL'd,
   ==8162== Using Valgrind-3.5.0 and LibVEX; rerun v
   ==8162== Command: ./main
   ==8162==
   ==8162== Mismatched free() / delete / delete []
                at 0x4005234: operator delete(void*)
   ==8162==
                by 0x804852E: main (in /home/niu/valc
   ==8162==
   ==8162== Address 0x4017028 is 0 bytes inside a k
                at 0x4005D2D: operator new[](unsigned
   ==8162==
                by 0x8048520: main (in /home/niu/valc
   ==8162==
   ==8162==
   ==8162== Mismatched free() / delete / delete []
                at 0x400551D: free (vg_replace_malloc
   ==8162==
                by 0x8048548: main (in /home/niu/valc
   ==8162==
              Address 0x4017080 is 0 bytes inside a k
   ==8162==
                at 0x40060D5: operator new(unsigned i
   ==8162==
                by 0x804853A: main (in /home/niu/valc
   ==8162==
   ==8162==
   ==8162==
   ==8162== HEAP SUMMARY:
                 in use at exit: 0 bytes in 0 blocks
   ==8162==
               total heap usage: 2 allocs, 2 frees, 4
   ==8162==
   ==8162==
   ==8162== All heap blocks were freed -- no leaks a
   ==8162==
   ==8162== For counts of detected and suppressed er
   ==8162== ERROR SUMMARY: 2 errors from 2 contexts
7、踩内存
   代码:
   #include <stdlib.h>
```

```
#include <memory.h>
int main(int agc, char* argv[])
{
        char* p1 = new char[4];
        char* p2 = "abcde";
        memcpy(p1, p2, 5);
        return 0;
测试:
[root@localhost valgrind]# valgrind --tool=memche
==8387== Memcheck, a memory error detector
==8387== Copyright (C) 2002-2009, and GNU GPL'd,
==8387== Using Valgrind-3.5.0 and LibVEX; rerun v
==8387== Command: ./main
==8387==
==8387== Invalid write of size 1
            at 0x80484A9: main (in /home/niu/valc
==8387==
==8387== Address 0x401702c is 0 bytes after a b]
==8387==
            at 0x4005D2D: operator new[](unsigned
            by 0x8048490: main (in /home/niu/valc
==8387==
==8387==
==8387==
==8387== HEAP SUMMARY:
             in use at exit: 4 bytes in 1 blocks
==8387==
           total heap usage: 1 allocs, 0 frees, 4
==8387==
==8387==
==8387== 4 bytes in 1 blocks are definitely lost
            at 0x4005D2D: operator new[](unsigned
==8387==
            by 0x8048490: main (in /home/niu/valg
==8387==
==8387==
==8387== LEAK SUMMARY:
            definitely lost: 4 bytes in 1 blocks
==8387==
            indirectly lost: 0 bytes in 0 blocks
==8387==
              possibly lost: 0 bytes in 0 blocks
==8387==
            still reachable: 0 bytes in 0 blocks
==8387==
                 suppressed: 0 bytes in 0 blocks
==8387==
==8387==
```

==8387== For counts of detected and suppressed er ==8387== ERROR SUMMARY: 2 errors from 2 contexts

#### 日期时间

```
1、date 查看日期时间,如下:
   [root@localhost ~]# date
   Sat Sep 26 20:41:54 CST 2015
2、查看时区,使用-R,如下:
   [root@localhost ~]# date -R
   Sat, 26 Sep 2015 20:42:42 +0800
3、格式化输出时间
   [root@localhost ~]# date '+%Y-%m-%d %H:%M:%S'
   2015-09-26 20:43:36
   [root@localhost ~]# date '+%Y-%m-%d %I:%M:%S'
   2015-09-26 08:43:43
   小写m是月
   大写M是分钟
   大写H是24小时
   大写I是12小时
4、设置时间
   [root@localhost ~]# date -s '2015-09-26 20:47:00'
   Sat Sep 26 20:47:00 CST 2015
5、date打印出来的时间,看起来不直观,使用date '+%Y-%m-%d %F
   [root@localhost ~]# date
   Thu Dec 29 13:20:23 CST 2016
   [root@localhost ~]# alias date="date '+%Y-%m-%d 9
   [root@localhost ~]# date
   2016-12-29 13:20:32
   但是,假如我还想使用原始的date命令,怎么办?
   使用\date
考虑下面的需求,清除15天之前的日志文件
```

考虑下面的需求,清除15天之前的日志文件添加日志文件 #!/bin/sh set -x

```
i=0
nowSeconds=$(date +%s)
while [ $i -lt $1 ]
do
    nowStr=$(date +%Y-%m-%d\ %H:%M:%S -d "1970-01-01
    postfix=`date +%Y-%m-%d %H:%M:%S -d "$nowStr"`
    touch log_$postfix
    sleep 1
    ((i=i+1))
    ((nowSeconds=nowSeconds-86400*5))
done
删除日志文件
#! /bin/sh
set -x
LogFile_SaveDays=15
# clear old log file
logList=`ls -t|grep log_`
for log in $logList
do
        logDate=`echo $log|cut -d '_' -f 2`
        echo $logDate
        timeDiff=$(($(date +%s) - $(date +%s -d $log[
        if [ $[$timeDiff/86400] -gt $LogFile_SaveDays
                rm -f $log
        fi
        echo "$log"
done
考虑下面的需求,保留最新的三个dump core文件,其他的删除
添加core文件
#! /bin/sh
set -x
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

awk简单示例

## 详细描述

## 变量说明

### awk简单示例

```
1、考虑下面的文件,login_record记录了系统的登录名,从哪里来的
    [root@2 tmp]# more login_record
   username
                   from
                                  time
                   10.36.65.80
                                  2015-11-21 09:41:
   root
                   10.36.65.42 2015-11-20 15:16:
   andy
                  10.36.65.60 2015-10-13 10:36:
   bill
                  10.36.65.91
                                2015-11-18 19:25:
   caroline
2、获取所有的登录名,如下:
    [root@2 tmp]# cat login record | awk '{print $1}'
   username
   root
   andy
   bill
   caroline
   或者使用printf,如下:
    [root@2 tmp]# cat login record | awk '{printf ("9
   username
   root
   andy
   bill
   caroline
3、获取所有的登录名,登录时间,具体到小时
    [root@2 tmp]# cat login_record | grep -v user | awk
   root
           10.36.65.80 41
   andy 10.36.65.42 16
          10.36.65.60 36
   bill
```

carl 10.36.65.91 25 其中,-F "[\t]+|:" 指定分隔符,多个制表符或者冒号作为分

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

cmakey�����

**♦**���IIĿ¼CMAKE\_INSTALL\_PREFIX

cmake'��

## 变量说明

### cmake' ��

```
20000û0a 0000makefile010010
   00000cmake00cmake0000010CMakeLists.txt000000010ma
300' OPPOPE
   niuzibin@ubuntu:~/work/test1/cmake$ ll
   total 16
   drwxrwxr-x 2 niuzibin niuzibin 4096 Apr 9 00:17
   drwxrwxr-x 4 niuzibin niuzibin 4096 Apr 9 00:11
                                   45 Apr 9 00:12
   -rw-rw-r-- 1 niuzibin niuzibin
                                   89 Apr 9 00:12
   -rw-rw-r-- 1 niuzibin niuzibin
   niuzibin@ubuntu:~/work/test1/cmake$ more hello.c
   #include <stdio.h>
   int main(void)
   {
           printf("Hello world\n");
           return 0;
   niuzibin@ubuntu:~/work/test1/cmake$ more CMakeLis
   PROJECT(HELLO)
   ADD_EXECUTABLE(hello hello.c)
40000cmake00000£0
   niuzibin@ubuntu:~/work/test1/cmake$ cmake CMakeLi
   -- The C compiler identification is GNU 4.8.4
   -- The CXX compiler identification is GNU 4.8.4
   -- Check for working C compiler: /usr/bin/cc
   -- Check for working C compiler: /usr/bin/cc -- v
```

```
-- Detecting C compiler ABI info
    -- Detecting C compiler ABI info - done
    -- Check for working CXX compiler: /usr/bin/c++
    -- Check for working CXX compiler: /usr/bin/c++ -
    -- Detecting CXX compiler ABI info
    -- Detecting CXX compiler ABI info - done
    -- Configuring done
    -- Generating done
    -- Build files have been written to: /home/niuzik
   niuzibin@ubuntu:~/work/test1/cmake$ ll
    total 44
    drwxrwxr-x 3 niuzibin niuzibin
                                   4096 Apr 9 00:18
    drwxrwxr-x 4 niuzibin niuzibin
                                   4096 Apr 9 00:11
    -rw-rw-r-- 1 niuzibin niuzibin 11953 Apr 9 00:18
                                   4096 Apr 9 00:18
    drwxrwxr-x 5 niuzibin niuzibin
                                   1603 Apr 9 00:18
    -rw-rw-r-- 1 niuzibin niuzibin
                                      45 Apr 9 00:12
    -rw-rw-r-- 1 niuzibin niuzibin
                                      89 Apr 9 00:12
    -rw-rw-r-- 1 niuzibin niuzibin
    -rw-rw-r-- 1 niuzibin niuzibin 4592 Apr 9 00:18
5000000000h0v001���������00C0000 Makefile016
    niuzibin@ubuntu:~/work/test1/cmake$ make
   Scanning dependencies of target hello
    [100%] Building C object CMakeFiles/hello.dir/hel
   Linking C executable hello
    [100%] Built target hello
   niuzibin@ubuntu:~/work/test1/cmake$ ./hello
    Hello world
```

### cmakey������

### 1���-��QCQQQQQQQQQQQQQQQQQQQQQQQ

message(STATUS "the binary dir is  $PROJECT_BINAF$   $PROJECT_BI$ 

### **��� �** IIĿ¼CMAKE\_INSTALL\_PREFIX

```
100cmake00000 IIL400 '00cmake_INSTALL_PREFIX 🗘 🗘 0000 CMAKE_20000 h 🗘 ար 000 cmake -DCMAKE_INSTALL_PREFIX=/usr ../
3000 j 🌣 🗘 ար 000 cmake0 j 0000000 CMAKE_INSTALL_PREFIX < instal
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 变量

## flawfinder示例

## 变量说明

#### flawfinder示例

```
1、介绍
   Flawfinder是一个C/C 代码的开源安全审查工具,采用内建语法
   Flawfinder是一个非常简单的工具,采用的算法也不复杂,可以
   Flawfinder的开发者David Wheeler表示:很多软件开发者都很
2、下载
   https://www.dwheeler.com/flawfinder/
   找到 current released version flawfinder in .tar.
3、然后解压安装
   tar -zxvf flawfinder-2.0.6.tar.gz
   cd flawfinder-2.0.6/
   sudo make install
4、测试代码
   niuzibin@ubuntu:~/work/flawfinder$ vi main.cpp
   #include <stdio.h>
   #include <stdlib.h>
   int main(int argc, char* argv[])
   {
           printf("hello wolrd\n");
           int aa;
           int bb = aa + 10;
           char ch[50];
           gets(ch);
           scanf("%c",ch);
           return 1;
5、使用flawfinder,进行检测
```

niuzibin@ubuntu:~/work/flawfinder\$ flawfinder --c Flawfinder version 2.0.6, (C) 2001-2017 David A. Number of rules (primarily dangerous function nan Examining main.cpp

#### FINAL RESULTS:

main.cpp:9:2: [5] (buffer) gets:

Does not check for buffer overflows (CWE-120, CWE gets(ch);

main.cpp:8:2: [2] (buffer) char:

Statically-sized arrays can be improperly restrict overflows or other issues (CWE-119!/CWE-120). Perfunctions that limit length, or ensure that the smaximum possible length.

char ch[50];

#### ANALYSIS SUMMARY:

Hits = 2

Lines analyzed = 12 in approximately 0.01 seconds
Physical Source Lines of Code (SLOC) = 12

Hits@level = [0] 2 [1] 0 [2] 1 [3] 0 [4] Hits@level+ = [0+] 4 [1+] 2 [2+] 2 [3+] 1 Hits/KSLOC@level+ = [0+] 333.333 [1+] 166.667 [2+] Minimum risk level = 1

Not every hit is necessarily a security vulnerabithere may be other security vulnerabilities; revisee 'Secure Programming HOWTO'

(https://www.dwheeler.com/secure-programs) for mc 6、也可以生成html文档,保存一下

flawfinder --columns --context --html main.cpp> r 7、我们看一下,分析结果

有两处告警,并且指出违反的规则。

搜索 CWE-120, CWE-20, CWE-119

8、实际上,使用g++编译,我们也可以看到告警,甚至更多,比如这里 niuzibin@ubuntu:~/work/flawfinder\$ g++ -Wall -o n

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

gdb调试
gdb调试之list
gdb调试服务vru
gdb打印vector的内容
gdb打印stl的内容
gdb查看崩溃信息
gdb查看core堆栈,栈顶显示问号
gdb在动态库上打断点出现错误

## 详细描述

## 变量说明

### gdb在动态库上打断点出现错误

- 1, no source file named xxx
- 2、一种情况是动态库还没有加载进来
- 3、一种情况是因为找不到源文件,把需要的源文件拷贝到当前目录, 或者获取拷贝到一个目录,使用directory指定源文件的搜索路径

### gdb打印stl的内容

```
1、将下面的内容,保存到 ~/.gdbinit
#
#
    STL GDB evaluators/views/utilities - 1.03
#
    The new GDB commands:
#
        are entirely non instrumental
#
        do not depend on any "inline"(s) - e.g. size(
#
        are extremely tolerant to debugger settings
#
#
    This file should be "included" in .gdbinit as fol
#
    source stl-views.gdb or just paste it into your .
#
#
    The following STL containers are currently suppor
#
#
        std::vector<T> -- via pvector command
#
        std::list<T> -- via plist or plist_member con
#
        std::map<T,T> -- via pmap or pmap_member comn
#
        std::multimap<T,T> -- via pmap or pmap_member
#
```

```
#
        std::set<T> -- via pset command
        std::multiset<T> -- via pset command
#
        std::deque<T> -- via pdequeue command
#
        std::stack<T> -- via pstack command
#
        std::gueue<T> -- via pgueue command
#
        std::priority_queue<T> -- via ppqueue command
#
        std::bitset<n> -- via pbitset command
#
        std::string -- via pstring command
#
        std::widestring -- via pwstring command
#
#
    The end of this file contains (optional) C++ beau
#
    Make sure your debugger supports $argc
#
#
    Simple GDB Macros writen by Dan Marinescu (H-PhD)
#
    Inspired by intial work of Tom Malnar,
#
      Tony Novac (PhD) / Cornell / Stanford,
#
      Gilad Mishne (PhD) and Many Many Others.
#
    Contact: dan_c_marinescu@yahoo.com (Subject: STL)
#
#
    Modified to work with g++ 4.3 by Anders Elton
#
    Also added _member functions, that instead of pri
#
#
# std::vector<>
#
define pvector
    if \$argc == 0
        help pvector
    else
        set $size = $arg0._M_impl._M_finish - $arg0._
        set $capacity = $arg0._M_impl._M_end_of_storates
        set $size max = $size - 1
    end
    if \$argc == 1
```

```
set $i = 0
    while $i < $size
        printf "elem[%u]: ", $i
        p *($arg0. M impl. M start + $i)
        set $i++
    end
end
if \$argc == 2
    set $idx = $arg1
    if $idx < 0 \mid | $idx > $size_max
        printf "idx1, idx2 are not in acceptable
    else
        printf "elem[%u]: ", $idx
        p *($arg0._M_impl._M_start + $idx)
    end
end
if $argc == 3
  set $start_idx = $arg1
  set $stop_idx = $arg2
  if $start_idx > $stop_idx
    set $tmp_idx = $start_idx
    set $start_idx = $stop_idx
    set $stop_idx = $tmp_idx
  end
  if $start_idx < 0 || $stop_idx < 0 || $start_ic
    printf "idx1, idx2 are not in acceptable range
  else
    set $i = \$start idx
    while $i <= $stop_idx</pre>
        printf "elem[%u]: ", $i
        p *($arg0._M_impl._M_start + $i)
        set $i++
    end
  end
end
if $argc > 0
    printf "Vector size = %u\n", $size
```

```
printf "Vector capacity = %u\n", $capacity
        printf "Element "
        whatis $arg0._M_impl._M_start
    end
end
document pvector
    Prints std::vector<T> information.
    Syntax: pvector <vector> <idx1> <idx2>
    Note: idx, idx1 and idx2 must be in acceptable ra
    Examples:
    pvector v - Prints vector content, size, capacity
    pvector v 0 - Prints element[idx] from vector
    pvector v 1 2 - Prints elements in range [idx1..i
end
#
# std::list<>
#
define plist
    if \$argc == 0
        help plist
    else
        set $head = &$arg0._M_impl._M_node
        set $current = $arg0._M_impl._M_node._M_next
        set $size = 0
        while $current != $head
            if $argc == 2
                printf "elem[%u]: ", $size
                p *($arg1*)($current + 1)
            end
            if \$argc == 3
                if $size == $arg2
                    printf "elem[%u]: ", $size
                    p *($arg1*)($current + 1)
                end
```

```
end
            set $current = $current._M_next
            set $size++
        end
        printf "List size = %u \n", $size
        if $argc == 1
            printf "List "
            whatis $arg0
            printf "Use plist <variable name> <elemer</pre>
        end
    end
end
document plist
    Prints std::list<T> information.
    Syntax: plist <list> <T> <idx>: Prints list size,
    Examples:
    plist 1 - prints list size and definition
    plist l int - prints all elements and list size
    plist 1 int 2 - prints the third element in the 1
end
define plist_member
    if $argc == 0
        help plist_member
    else
        set $head = &$arg0._M_impl._M_node
        set $current = $arg0._M_impl._M_node._M_next
        set $size = 0
        while $current != $head
            if $argc == 3
                printf "elem[%u]: ", $size
                p (*(\$arg1*)(\$current + 1)).\$arg2
            end
            if \$argc == 4
                if $size == $arg3
                     printf "elem[%u]: ", $size
```

```
p (*($arg1*)($current + 1)).$arg2
                end
            end
            set $current = $current. M next
            set $size++
        end
        printf "List size = %u \n", $size
        if \$argc == 1
            printf "List "
            whatis $arg0
            printf "Use plist_member <variable_name>
        end
    end
end
document plist_member
    Prints std::list<T> information.
    Syntax: plist <list> <T> <idx>: Prints list size,
    Examples:
    plist_member 1 int member - prints all elements a
    plist_member 1 int member 2 - prints the third el
end
#
# std::map and std::multimap
#
define pmap
    if $argc == 0
        help pmap
    else
        set $tree = $arg0
        set $i = 0
        set $node = $tree._M_t._M_impl._M_header._M_]
        set $end = $tree._M_t._M_impl._M_header
        set $tree_size = $tree._M_t._M_impl._M_node_c
```

```
if \$argc == 1
    printf "Map "
    whatis $tree
    printf "Use pmap <variable name> <left el</pre>
end
if $argc == 3
    while $i < $tree_size</pre>
        set value = (void *)(snode + 1)
        printf "elem[%u].left: ", $i
        p *($arg1*)$value
        set $value = $value + sizeof($arg1)
        printf "elem[%u].right: ", $i
        p *($arg2*)$value
        if $node._M_right != 0
            set $node = $node._M_right
            while $node. M left != 0
                 set $node = $node._M_left
            end
        else
            set $tmp_node = $node._M_parent
            while $node == $tmp_node._M_right
                 set $node = $tmp_node
                 set $tmp_node = $tmp_node._M_
            end
            if $node._M_right != $tmp_node
                 set $node = $tmp node
            end
        end
        set $i++
    end
end
if \$argc == 4
    set $idx = $arg3
    set $ElementsFound = 0
    while $i < $tree size
        set value = (void *)(snode + 1)
        if *(\$arg1*)\$value == \$idx
```

```
printf "elem[%u].left: ", $i
            p *($arg1*)$value
            set $value = $value + sizeof($arç
            printf "elem[%u].right: ", $i
            p *($arg2*)$value
            set $ElementsFound++
        end
        if $node._M_right != 0
            set $node = $node. M right
            while $node._M_left != 0
                set $node = $node. M left
            end
        else
            set $tmp_node = $node._M_parent
            while $node == $tmp_node._M_right
                set $node = $tmp_node
                set $tmp_node = $tmp_node._M_
            end
            if $node._M_right != $tmp_node
                set $node = $tmp node
            end
        end
        set $i++
    end
    printf "Number of elements found = %u\n",
end
if \$argc == 5
    set $idx1 = $arg3
    set $idx2 = $arg4
    set \$ElementsFound = 0
    while $i < $tree size
        set value = (void *)(node + 1)
        set $valueLeft = *($arg1*)$value
        set valueRight = *(sarg2*)(value +
        if $valueLeft == $idx1 && $valueRight
            printf "elem[%u].left: ", $i
            p $valueLeft
```

```
printf "elem[%u].right: ", $i
                    p $valueRight
                    set $ElementsFound++
                end
                if $node. M right != 0
                    set $node = $node. M right
                    while $node. M left != 0
                        set $node = $node. M left
                    end
                else
                    set $tmp_node = $node._M_parent
                    while $node == $tmp node. M right
                        set $node = $tmp node
                        set $tmp node = $tmp node. M
                    end
                    if $node._M_right != $tmp_node
                        set $node = $tmp_node
                    end
                end
                set $i++
            end
            printf "Number of elements found = %u\n",
        printf "Map size = %u\n", $tree_size
    end
end
document pmap
    Prints std::map<TLeft and TRight> or std::multimate
    Syntax: pmap <map> <TtypeLeft> <TypeRight> <valLe
    Examples:
    pmap m - prints map size and definition
    pmap m int int - prints all elements and map size
    pmap m int int 20 - prints the element(s) with le
    pmap m int int 20 200 - prints the element(s) wit
end
```

```
define pmap_member
    if \$argc == 0
        help pmap member
    else
        set $tree = $arg0
        set $i = 0
        set $node = $tree._M_t._M_impl._M_header._M_]
        set $end = $tree._M_t._M_impl._M_header
        set $tree_size = $tree._M_t._M_impl._M_node_c
        if \$argc == 1
            printf "Map "
            whatis $tree
            printf "Use pmap <variable name> <left e]</pre>
        end
        if \$argc == 5
            while $i < $tree_size
                set value = (void *)(node + 1)
                printf "elem[%u].left: ", $i
                p (*($arg1*)$value).$arg2
                set $value = $value + sizeof($arg1)
                printf "elem[%u].right: ", $i
                p (*($arg3*)$value).$arg4
                if $node._M_right != 0
                     set $node = $node._M_right
                    while $node._M_left != 0
                         set $node = $node. M left
                     end
                else
                     set $tmp_node = $node._M_parent
                    while $node == $tmp_node._M_right
                         set $node = $tmp node
                         set $tmp_node = $tmp_node._M_
                     end
                     if $node. M right != $tmp node
                         set $node = $tmp_node
                     end
```

```
end
        set $i++
    end
end
if \$argc == 6
    set $idx = $arg5
    set $ElementsFound = 0
    while $i < $tree size
        set value = (void *)(node + 1)
        if *(\$arg1*)\$value == \$idx
            printf "elem[%u].left: ", $i
            p (*($arg1*)$value).$arg2
            set $value = $value + sizeof($arc
            printf "elem[%u].right: ", $i
            p (*($arg3*)$value).$arg4
            set $ElementsFound++
        end
        if $node._M_right != 0
            set $node = $node._M_right
            while $node. M left != 0
                set $node = $node._M_left
            end
        else
            set $tmp_node = $node._M_parent
            while $node == $tmp_node._M_right
                set $node = $tmp_node
                set $tmp_node = $tmp_node._M_
            end
            if $node._M_right != $tmp_node
                set $node = $tmp_node
            end
        end
        set $i++
    end
    printf "Number of elements found = %u\n",
end
printf "Map size = %u\n", $tree_size
```

```
end
end
document pmap member
    Prints std::map<TLeft and TRight> or std::multimate
    Syntax: pmap <map> <TtypeLeft> <TypeRight> <valLe
    Examples:
    pmap_member m class1 member1 class2 member2 - pri
    pmap_member m class1 member1 class2 member2 lvalu
end
#
# std::set and std::multiset
#
define pset
    if $argc == 0
        help pset
    else
        set $tree = $arg0
        set \$i = 0
        set $node = $tree._M_t._M_impl._M_header._M_]
        set $end = $tree._M_t._M_impl._M_header
        set $tree_size = $tree._M_t._M_impl._M_node_c
        if \$argc == 1
            printf "Set "
            whatis $tree
            printf "Use pset <variable_name> <element</pre>
        end
        if \$argc == 2
            while $i < $tree size
                set value = (void *)(node + 1)
                printf "elem[%u]: ", $i
                p *($arg1*)$value
                if $node._M_right != 0
                     set $node = $node._M_right
```

```
while $node._M_left != 0
                set $node = $node._M_left
            end
        else
            set $tmp_node = $node._M_parent
            while $node == $tmp_node._M_right
                set $node = $tmp node
                set $tmp_node = $tmp_node._M_
            end
            if $node._M_right != $tmp_node
                set $node = $tmp node
            end
        end
        set $i++
    end
end
if $argc == 3
    set $idx = $arg2
    set \$ElementsFound = 0
    while $i < $tree size
        set value = (void *)(node + 1)
        if *(\$arg1*)\$value == \$idx
            printf "elem[%u]: ", $i
            p *($arg1*)$value
            set $ElementsFound++
        end
        if $node._M_right != 0
            set $node = $node._M_right
            while $node. M left != 0
                set $node = $node. M left
            end
        else
            set $tmp_node = $node._M_parent
            while $node == $tmp node. M right
                set $node = $tmp node
                set $tmp_node = $tmp_node._M_
            end
```

```
if $node._M_right != $tmp_node
                         set $node = $tmp_node
                    end
                end
                set $i++
            end
            printf "Number of elements found = %u\n",
        end
        printf "Set size = %u\n", $tree_size
    end
end
document pset
    Prints std::set<T> or std::multiset<T> information
    Syntax: pset <set> <T> <val>: Prints set size, if
    Examples:
    pset s - prints set size and definition
    pset s int - prints all elements and the size of
    pset s int 20 - prints the element(s) with value
end
#
# std::dequeue
#
define pdequeue
    if \$argc == 0
        help pdequeue
    else
        set $size = 0
        set $start_cur = $arg0._M_impl._M_start._M_ci
        set $start last = $arg0. M impl. M start. M ]
        set $start_stop = $start_last
        while $start_cur != $start_stop
            p *$start cur
```

```
set $start cur++
            set $size++
        end
        set $finish_first = $arg0._M_impl._M_finish._
        set $finish cur = $arg0. M impl. M finish. M
        set $finish_last = $arg0._M_impl._M_finish._N
        if $finish cur < $finish last
            set $finish_stop = $finish_cur
        else
            set $finish_stop = $finish_last
        end
        while $finish_first != $finish_stop
            p *$finish first
            set $finish first++
            set $size++
        end
        printf "Dequeue size = %u\n", $size
    end
end
document pdequeue
    Prints std::dequeue<T> information.
    Syntax: pdequeue <dequeue>: Prints dequeue size,
    Deque elements are listed "left to right" (left-n
    Example:
    pdequeue d - prints all elements and size of d
end
#
# std::stack
#
define pstack
    if \$argc == 0
        help pstack
```

```
else
        set $start_cur = $arg0.c._M_impl._M_start._M_
        set $finish_cur = $arg0.c._M_impl._M_finish._
        set $size = $finish cur - $start cur
        set $i = $size - 1
        while $i >= 0
            p *($start_cur + $i)
            set $i--
        end
        printf "Stack size = %u\n", $size
    end
end
document pstack
    Prints std::stack<T> information.
    Syntax: pstack <stack>: Prints all elements and s
    Stack elements are listed "top to buttom" (top-mc
    Example:
    pstack s - prints all elements and the size of s
end
#
# std::queue
#
define pqueue
    if \$argc == 0
        help pqueue
    else
        set $start_cur = $arg0.c._M_impl._M_start._M_
        set $finish_cur = $arg0.c._M_impl._M_finish._
        set $size = $finish cur - $start cur
        set \$i = 0
        while $i < $size
            p *($start_cur + $i)
```

```
set $i++
        end
        printf "Queue size = %u\n", $size
    end
end
document pqueue
    Prints std::queue<T> information.
    Syntax: pqueue <queue>: Prints all elements and t
    Queue elements are listed "top to bottom" (top-mc
    Example:
    pqueue q - prints all elements and the size of q
end
# std::priority_queue
#
define ppqueue
    if \$argc == 0
        help ppqueue
    else
        set $size = $arg0.c._M_impl._M_finish - $arg@
        set $capacity = $arg0.c._M_impl._M_end_of_stc
        set $i = $size - 1
        while $i >= 0
            p *($arg0.c._M_impl._M_start + $i)
            set $i--
        end
        printf "Priority queue size = %u\n", $size
        printf "Priority queue capacity = %u\n", $car
    end
end
document ppqueue
```

```
Prints std::priority_queue<T> information.
    Syntax: ppqueue <priority_queue>: Prints all elem
    Priority_queue elements are listed "top to button
    Example:
    ppqueue pq - prints all elements, size and capaci
end
#
# std::bitset
define pbitset
    if $argc == 0
        help pbitset
    else
        p /t $arg0._M_w
    end
end
document pbitset
    Prints std::bitset<n> information.
    Syntax: pbitset <bitset>: Prints all bits in bits
    Example:
    pbitset b - prints all bits in b
end
#
# std::string
#
define pstring
    if \$argc == 0
        help pstring
```

```
else
                               printf "String \t\t= \"%s\"\n", $arg0._M_da
                               printf "String size/length \t= %u\n", $arg0._
                               printf "String capacity \t= %u\n", $arg0._M_r
                               printf "String ref-count \t= %d\n", $arg0._M_
               end
end
document pstring
               Prints std::string information.
               Syntax: pstring <string>
               Example:
               pstring s - Prints content, size/length, capacity
end
#
# std::wstring
#
define pwstring
               if \$argc == 0
                               help pwstring
               else
                               call printf("WString t = \"\sl "\n", \sl "\n
                               printf "WString size/length \t= %u\n", $arg0.
                               printf "WString capacity \t= %u\n", \$arg0._M_
                               printf "WString ref-count \t= %d\n", $arg0._\]
               end
end
document pwstring
               Prints std::wstring information.
               Syntax: pwstring <wstring>
               Example:
               pwstring s - Prints content, size/length, capacit
end
```

```
#
# C++ related beautifiers (optional)
#
set print pretty on
set print object on
set print static-members on
set print vtbl on
set print demangle on
set demangle-style gnu-v3
set print sevenbit-strings off

2、然后就可以使用这些命令 pvector, plist, pmap, pset, pde
```

参见

#### qdb打印vector的内容

```
1、高版本的gdb可以打印出vector的内容,但是低版本的vector还不
2、怎么解决?
   我们知道vector内部是使用数组实现的,我们可以打印出来arra
3、示例如下:
   [niu_zibin@localhost gdb]$ vi main.cpp
   1 #include <stdio.h>
   2 #include <vector.h>
   3 int main()
   4 {
   5
            vector<int> aVec;
            aVec.push_back(1);
   6
            aVec.push_back(2);
   7
   8
            aVec.push back(3);
   9
   10
             return 0;
   11 }
```

```
[niu_zibin@localhost gdb]$ g++ -g -o main main.cr
    [niu_zibin@localhost gdb]$ gdb main
    (qdb) b 10
    Breakpoint 1 at 0x804879b: file main.cpp, line 10
    (gdb) r
    Starting program: /home/niu zibin/test/gdb/main
    Breakpoint 1, main () at main.cpp:10
                    return 0;
    10
    (qdb) p aVec
    $1 = {<std:: Vector base<int, std::allocator<int>
       _M_impl = {<std::allocator<int>> = {<__qnu_c>
        _{M}start = 0x804b028, _{M}finish = 0x804b034,
    (gdb) p *(0x804b028)
    $2 = 1
    (gdb) p *(0x804b028)@1
   $3 = {1}
    (gdb) p *(0x804b028)@2
   $4 = {1, 2}
    (gdb) p *(0x804b028)@3
    $5 = \{1, 2, 3\}
    (gdb) c
    Continuing.
    Program exited normally.
    (gdb) q
4、现在考虑, vector元素是自定义的类型, 操作如下:
    [niu_zibin@localhost gdb]$ vi main.cpp
    #include <stdio.h>
    #include <vector.h>
    #include <string.h>
    using namespace std;
    struct Person
```

```
{
        int
                _Age;
        string _Name;
        Person(int age, string name)
        {
                _{Age} = age;
                _Name = name;
        }
};
int main()
{
        Person p1(1, "Andy");
        Person p2(2, "Bill");
        Person p3(3, "Caroline");
        vector<Person> aVec;
        aVec.push_back(p1);
        aVec.push_back(p2);
        aVec.push_back(p3);
        return 0;
}
[niu_zibin@localhost gdb]$ g++ -g -o main main.cr
[niu_zibin@localhost gdb]$ gdb main
(gdb) shell vi main.cpp
    1 #include <stdio.h>
    2 #include <vector.h>
    3 #include <string.h>
    4
    5 using namespace std;
    7 struct Person
```

```
9
                       _Age;
               int
    10
                string _Name;
                Person(int age, string name)
    11
    12
                {
                         _{Age} = age;
    13
    14
                         _{\text{Name}} = \text{name};
    15
                }
    16 };
    17
    18 int main()
    19 {
    20
                Person p1(1, "Andy");
    21
                Person p2(2, "Bill");
                Person p3(3, "Caroline");
    22
    23
    24
                vector<Person> aVec;
    25
                aVec.push_back(p1);
                aVec.push_back(p2);
    26
    27
                aVec.push_back(p3);
    28
    29
                return 0;
    30 }
(gdb) b 29
Breakpoint 1 at 0x8048b14: file main.cpp, line 29
(gdb) r
Starting program: /home/niu_zibin/test/gdb/main
warning: .dynamic section for "/lib/libc.so.6" is
warning: difference appears to be caused by preli
Breakpoint 1, main () at main.cpp:29
29
                 return 0;
(qdb) p aVec
$1 = {<std::_Vector_base<Person, std::allocator<F</pre>
    M impl = {<std::allocator<Person>> = {< qnu</pre>
    _{M_{start}} = 0x804c080, _{M_{start}} = 0x804c098,
(gdb) p *(0x804c080)
$2 = 1
```

```
(gdb) p *(Person*)(0x804c080)
3 = {\text{Age} = 1, \text{Name} = {\text{static npos} = 4294967295}}
    _M_dataplus = {<std::allocator<char>> = {<___c
    _{M_p} = 0x804c014 \text{ "Andy"}}
(gdb) p *(Person*)(0x804c080)@1
4 = {\text{Age} = 1, \text{Name} = \{\text{static npos} = 42949672\}
    _M_dataplus = {<std::allocator<char>> = {<___c
         _{M_p} = 0x804c014 "Andy"}}}
(gdb) p *(Person*)(0x804c080)@2
5 = {\{\_Age = 1, \_Name = \{static npos = 42949672\}\}}
    _M_dataplus = {<std::allocator<char>> = {<___(
         _{M_p} = 0x804c014 \text{ "Andy"}}, {_Age} = 2, _{N_p}
    _M_dataplus = {<std::allocator<char>> = {<__c
         M p = 0x804c02c "Bill"}}}
(gdb) p *(Person*)(0x804c080)@3
6 = {\{\_Age = 1, \_Name = \{static npos = 42949672\}\}
    _M_dataplus = {<std::allocator<char>> = {<__c
         _{M_p} = 0 \times 804 c014 \text{ "Andy"}}, {_Age} = 2, __N
    _M_dataplus = {<std::allocator<char>> = {<__c
         _{M_p} = 0 \times 804 c02 c "Bill" \} \}, {_Age = 3, _N}
    _M_dataplus = {<std::allocator<char>> = {<__c
        _{M_p} = 0x804c044 "Caroline"}}}
(gdb) c
Continuing.
Program exited normally.
(gdb) q
```

# gdb查看core堆栈,栈顶显示问号

```
1、使用gdb查看core文件的堆栈信息,使用bt(全称是backtrace回
```

2、这是怎么回事?

bt会显示方法的调用堆栈,栈顶为??()说明是调用的方法找不比如:方法指针被设置为NULL,或者其他无效的值

3、模拟代码如下:

```
int Sum(int a,int b)
{
```

```
return a+b;
          }
          int main()
          {
                     int(*pFun)(int,int)=Sum;
                     pFun=0; // 强制设置为0, 然后调用
                     // pFun=(int(*)(int,int))1234; 设置为无效的值,
                     int c = pFun(1,2);
                     return 0;
4、运行产生core文件
           [root@localhost niu5]# g++ -g -o main main.cpp
           [root@localhost niu5]# ulimit -c unlimited
           [root@localhost niu5]# ./main
          Segmentation fault (core dumped)
5、查看core文件,显示栈顶为??()
           [root@localhost niu5]# gdb main core.31041.main
          GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-23.
          Copyright (C) 2009 Free Software Foundation, Inc.
          License GPLv3+: GNU GPL version 3 or later <a href="http://www.gpl.num.gen.gov/">http://www.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gpl.num.gp
          This is free software: you are free to change and
          There is NO WARRANTY, to the extent permitted by
          and "show warranty" for details.
          This GDB was configured as "i386-redhat-linux-gnu
          For bug reporting instructions, please see:
          <a href="http://www.gnu.org/software/gdb/bugs/>...">http://www.gnu.org/software/gdb/bugs/>...</a>
          Reading symbols from /home/niu5/main...done.
          Reading symbols from /usr/lib/libstdc++.so.6...(r
          Loaded symbols for /usr/lib/libstdc++.so.6
          Reading symbols from /lib/libm.so.6...(no debuggi
          Loaded symbols for /lib/libm.so.6
          Reading symbols from /lib/libgcc_s.so.1...(no dek
          Loaded symbols for /lib/libgcc s.so.1
          Reading symbols from /lib/libc.so.6...(no debuggi
          Loaded symbols for /lib/libc.so.6
          Reading symbols from /lib/ld-linux.so.2...(no det
```

```
Loaded symbols for /lib/ld-linux.so.2
Core was generated by `./main'.
Program terminated with signal 11, Segmentation 1
#0 0x000000000 in ?? ()
(gdb) bt
#0 0x00000000 in ?? ()
#1 0x08048463 in main () at main.cpp:10
```

#### gdb查看崩溃信息

1、gdb常常用于查看崩溃信息,如下:

[root@localhost vru]# gdb vru\_main core.10504.vru

2、常用的命名

info thread

thread apply 1 bt

thread apply all bt

3、常见的coredump原因

Signal 4(SIGILL)

SIGILL信号是cpu在发现非法指令之后发出一个异常,然后是程序收到这个信号,一般就是报告 illegal instruction版本不一致, 比如依赖的共享库接口变了,但你仍在使用老而且你觉得不可能挂的地方,而通常是挂在你对依赖库的调用一个非常常见的原因是:崩溃的上下文,存在把string当成这种情况,采用排除法,注释掉会崩溃的代码,缩小范围,封

Signal 6(SIGABRT)

New失败:内存泄露造成内存不够

Delete失败:多次delete同一块内存

应用程序抛出的异常

Signal 11(SIGSEGV)

多为内存越界,访问已经被delete掉的内存

Signal 13(SIGPIPE)

写已经被删除的文件,写对方已经关闭的socket

4、更多的崩溃信号,如下:

编号为1-31的信号为传统UNIX支持的信号,是不可靠信号(非实际编号为32-63的信号是后来扩充的,称做可靠信号(实时信号)。

不可靠信号和可靠信号的区别在于前者不支持排队,可能会造成信下面我们对编号小于SIGRTMIN的信号进行讨论。

#### 1) SIGHUP

本信号在用户终端连接(正常或非正常)结束时发出,通常是不这时它们与控制终端不再关联。登录Linux时,系统会分配给在这个终端运行的所有程序,包括前台进程组和后台进程组,前台进程组和后台有对终端输出的进程将会收到SIGHUP信号。因此前台进程组和后台有终端输出的进程就会中止。不过可以这样就算退出了Linux登录,wget也能继续下载。

此外,对于与终端脱离关系的守护进程,这个信号用于通知它

2) SIGINT

程序终止(interrupt)信号, 在用户键入INTR字符(通常是

3) SIGQUIT

和SIGINT类似, 但由QUIT字符(通常是Ctrl-\)来控制. 过在这个意义上类似于一个程序错误信号。

4) SIGILL

执行了非法指令. 通常是因为可执行文件本身出现错误, 或礼

- 5) SIGTRAP 由断点指令或其它trap指令产生. 由debugger使用。
- 6) SIGABRT 调用abort函数生成的信号。
- 7) SIGBUS

非法地址,包括内存地址对齐(alignment)出错。比如访问但其地址不是4的倍数。它与SIGSEGV的区别在于后者是由于(如访问不属于自己存储空间或只读存储空间)。

8) SIGFPE

在发生致命的算术运算错误时发出. 不仅包括浮点运算错误,

9) SIGKILL

用来立即结束程序的运行. 本信号不能被阻塞、处理和忽略。

- 10) SIGUSR1 留给用户使用
- 11) SIGSEGV

试图访问未分配给自己的内存, 或试图往没有写权限的内存划

- 12) SIGUSR2 留给用户使用
- 13) SIGPIPE

管道破裂。这个信号通常在进程间通信产生,比如采用FIFO(写进程会收到SIGPIPE信号。此外用Socket通信的两个进程

14) SIGALRM

时钟定时信号, 计算的是实际的时间或时钟时间. alarm函

- 15) SIGTERM 程序结束(terminate)信号,与SIGKILL不同的是该信号可 shell命令kill缺省产生这个信号。如果进程终止不了,我们
- 17) SIGCHLD 子进程结束时,父进程会收到这个信号。如果父进程没有处理 但是还会在内核进程表中占有表项,这时的子进程称为僵尸进 (父进程或者忽略SIGCHILD信号,或者捕捉它,或者wait它 这时子进程的终止自动由init进程来接管)。
- 18) SIGCONT 让一个停止(stopped)的进程继续执行.本信号不能被阻塞。 可以用一个handler来让程序在由stopped状态变为继续执行
- 19) SIGSTOP 停止(stopped)进程的执行.注意它和terminate以及int 本信号不能被阻塞,处理或忽略.
- 20) SIGTSTP 停止进程的运行,但该信号可以被处理和忽略.用户键入SU
- 21) SIGTTIN 当后台作业要从用户终端读数据时,该作业中的所有进程会\
- 22) SIGTTOU 类似于SIGTTIN, 但在写终端(或修改终端模式)时收到.
- 23) SIGURG 有"紧急"数据或out-of-band数据到达socket时产生.
- 24) SIGXCPU 超过CPU时间资源限制. 这个限制可以由getrlimit/setr]
- 25) SIGXFSZ 当进程企图扩大文件以至于超过文件大小资源限制。
- 26) SIGVTALRM 虚拟时钟信号. 类似于SIGALRM, 但是计算的是该进程占用
- 27) SIGPROF 类似于SIGALRM/SIGVTALRM, 但包括该进程用的CPU时间以
- 28) SIGWINCH 窗口大小改变时发出.
- 29) SIGIO 文件描述符准备就绪,可以开始进行输入/输出操作.
- 30) SIGPWR
  Power failure

## 31) SIGSYS

非法的系统调用。

在以上列出的信号中,

程序不可捕获、阻塞或忽略的信号有:SIGKILL,SIGSTOP

不能恢复至默认动作的信号有:SIGILL,SIGTRAP

默认会导致进程流产的信号有:SIGABRT,SIGBUS,SIGFPE,SIG默认会导致进程退出的信号有:SIGALRM,SIGHUP,SIGINT,SIG默认会导致进程停止的信号有:SIGSTOP,SIGTSTP,SIGTTIN,S默认进程忽略的信号有:SIGCHLD,SIGPWR,SIGURG,SIGWINCH此外,SIGIO在SVR4是退出,在4.3BSD中是忽略;SIGCONT在进

#### qdb调试

#### 1、准备工作

编译调试版本的可执行程序(gcc加上-g参数即可,注意不要调试加

2、gdb调试方式

冷启动

gdb program e.g., gdb ./cs

gdb -p pid e.g., gdb -p `pidof cs`

gdb program core e.g., gdb ./cs core.xxx

热启动

(gdb) attach pid e.g., (gdb) attach 2313

命令行参数

qdb program --args arglist

(gdb) set args arglist

(gdb) run arglist

3、在gdb中调用shell命名,如下:

(gdb) shell vi main.cpp

进入vi

输入 :exit 退出vi

注意:有些shell命名执行完毕就返回了,这种情况不需要exit

4、设置断点

(gdb) break function: 在函数funtion入口处设置断点

(gdb) break linenum: 在当前源文件的第linenum行处设置图

(gdb) break filename:linenum: 在名为filename的源文(

(gdb) break filename:function: 在名为filename的源文

比如:

```
(gdb) break foo1
    (gdb) break 17
5、查看断点信息
    (qdb) info break
                           Disp Enb Address
    Num
            Type
                                               What
            breakpoint
    1
                           keep y
                                    0x080483fa in foc
            breakpoint
    2
                           keep y
                                    0x08048425 in foc
6、断点禁用启用
    (gdb) info break
                           Disp Enb Address
    Num
            Type
                                               What
            breakpoint
                           keep y
                                    0x080483fa in foc
    1
                           keep y
                                    0x08048425 in foc
            breakpoint
    (qdb) disable break 1
    (gdb) info break
    Num
                           Disp Enb Address
                                               What
            Type
            breakpoint
                           keep n
                                    0x080483fa in foc
    1
                           keep y
            breakpoint
    2
                                    0x08048425 in foc
    (gdb) enable break 1
    (gdb) info break
                           Disp Enb Address
    Num
            Type
                                               What
                           keep y
            breakpoint
                                    0x080483fa in foc
    1
    2
            breakpoint
                           keep y
                                    0x08048425 in foc
7、删除断点
    (gdb) info break
                           Disp Enb Address
    Num
            Type
                                               What
    1
            breakpoint
                           keep y 0x080483fa in foc
            breakpoint
                           keep y
                                    0x08048425 in foc
    (gdb) delete break 1-2
    (gdb) info break
8、调试
    n step over
    s step into
    c next break
9、打印,如下:
    (gdb) p a3
    $1 = 3
    (gdb) set print pretty
```

```
(gdb) p a3
    $2 = 3
10、转到堆栈
    (gdb) f 1
        0x08048449 in main () at main.cpp:24
    #1
                    foo3();
    24
11、gdb 寄存器,如下:
    (gdb) info register
                   0xbfffead4 -1073747244
    eax
                   0xbfffea50
                                  -1073747376
    ecx
    edx
                   0x1
                            1
    ebx
                   0x80aff4 8433652
                   0xbfffea24
                                    0xbfffea24
    esp
                                    0xbfffea38
    ebp
                   0xbfffea38
                   0x6c5ca0 7101600
    esi
    edi
                            0
                   0x0
    eip
                   0x8048405
                                    0x8048405 <main()
    eflags
                           [ PF SF IF ]
                   0x286
    CS
                   0x73
                            115
                   0x7b
                            123
    SS
    ds
                   0x7b
                            123
                            123
    es
                   0x7b
    fs
                   0x0
                            0
                   0x33
                            51
    gs
```

# gdb调试之list

```
1、列出某一行的上下文:
    (gdb) list 8
    3    int foo1(int a)
    4    {
    5         return a+1;
    6    }
    7    8
    9    int main()
    10    {
```

```
11
                  int a = 7;
   12
                  int b = foo1(a);
2、列出行的范围:
   (gdb) list 7,14
   7
   8
          int main()
   9
   10
                  int a = 7;
   11
                  int b = foo1(a);
   12
   13
                  return b;
   14
3、列出方法的上下文:
   (gdb) list foo1
          #include <stdio.h>
   1
   2
   3
          int foo1(int a)
   4
   5
                  return a+1;
          }
   6
   7
   8
          int main()
   9
   10
4、list是针对当前文件,一般是main.cpp,也可以指定其他cpp文件
   (gdb) list main.cpp:8
   (gdb) list main.cpp:7,14
   (gdb) list main.cpp:foo1
5、特别注意:list能列出源代码,必须满足条件,当前环境下有cpp文
   如果没有cpp文件,报错 xxx.cpp: No such file or direct
   也就是说,在设备A编译程序,然后把编译后的结果拷贝到设备B,
   解决办法是:从设备A把cpp文件拷贝到设备B的当前目录下。
```

## gdb调试服务vru

1、gdb ./vru\_start.sh 用法是错误的 [root@localhost vru]# gdb ./vru\_start.sh 用法是错误

```
(gdb) b 2
   No symbol table is loaded. Use the "file" commar
2、正确的用法:
    [root@localhost vru]# export LD_LIBRARY_PATH=../c
    [root@localhost vru]# gdb ./vru_main
    (qdb) set args -f vru linux.xml
    (qdb) r
3、另外一种办法,找到进程Id,先下断点,再continue,然后p
   如下:
    [root@localhost vru]# gdb -p 24847
    (qdb) b src/VruMaster.cpp:528
   Breakpoint 1 at 0x80d33b0: file src/VruMaster.cpg
    (qdb) c
   Continuing.
    [Switching to Thread 0xb097eb70 (LWP 24884)]
   Breakpoint 1, VruMaster::HandlePlan (this=0x93c65
           src/VruMaster.cpp: No such file or direct
   528
           in src/VruMaster.cpp
    (gdb) p ttpConfig
4、设置断点
    (gdb) info b
   No breakpoints or watchpoints.
    (gdb) b 6
   Breakpoint 8 at 0x80a6718: file src/VruApp.cpp, ]
    (gdb) b VruMaster.cpp:236
   Breakpoint 9 at 0x80d4649: file src/VruMaster.cpg
    (gdb) b src/VruMaster.cpp:238
   Breakpoint 10 at 0x80d465b: file src/VruMaster.cr
    (gdb) b ./src/VruMaster.cpp:246
   No source file named ./src/VruMaster.cpp.
   Make breakpoint pending on future shared library
    (qdb) b src/VruMaster.cpp:246
   Breakpoint 11 at 0x80d46f6: file src/VruMaster.cr
    (qdb) info b
                          Disp Enb Address
   Num
           Type
                                              What
           breakpoint
                          keep y <MULTIPLE>
```

```
8.1
                                   0x080a6718 in _
                             У
8.2
                                   0x080a6845 in_{-}
                            У
9
        breakpoint
                       keep y
                                 0x080d4649 in Vru
10
        breakpoint
                       keep y
                                 0x080d465b in Vru
11
        breakpoint
                       keep y
                                 0x080d46f6 in Vru
注意:b ./src/VruMaster.cpp:246 这种用法是错误的。
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 变量

git远程分支的删除和同步 git忽略文件权限的检查 本地创建分支,并和远程关联 svn与git对照 log命令 冲突的情况只有两种 我们的开发分支管理 push可能会有冲突 看看fetch做了什么事情 是不是只能merge提交的分支 本地的修改状态 push命令 git命令status git撤销修改 撤销本地修改

# 变量说明

# git命令status

1、执行的结果如下:

niuzibin@ubuntu:~/work/gittest\$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: readme.txt

Changes not staged for commit:

(use "git add <file>..." to update what will be ( use "git checkout -- <file>..." to discard chang

modified: readme.txt

2、解释如下:

On branch master 工作的分支

Changes to be committed: 暂存区没有提交的信息 Changes not staged for commit: 工作区没有添加到暂存

# git忽略文件权限的检查

- 1, git config core.fileMode false
- 2、可以查看配置文件

niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP\$ cat .git/
[core]

repositoryformatversion = 0

```
filemode = true
bare = false
logallrefupdates = true
```

#### 参见

#### git撤销修改

```
1、分为两种情况:
   第一种情况,工作区做了修改,还没有添加到暂存区。
   第二种情况,工作区做了修改,并且已经添加到暂存区,然后工作
2、对于第一种情况,直接使用checkout即可。如下:
   niuzibin@ubuntu:~/work/gittest$ git status
   On branch master
   Changes not staged for commit:
   (use "git add <file>..." to update what will be c
   (use "git checkout -- <file>..." to discard change
           modified: readme.txt
   no changes added to commit (use "git add" and/or
   niuzibin@ubuntu:~/work/gittest$ git diff
   diff --git a/readme.txt b/readme.txt
   index 4dd9cb5..56b6b46 100644
   --- a/readme.txt
   +++ b/readme.txt
   @@ -1 +1 @@
   -I am Andy
   +I am Bill
   niuzibin@ubuntu:~/work/gittest$ git checkout reac
   niuzibin@ubuntu:~/work/gittest$ git status
   On branch master
   nothing to commit, working directory clean
   niuzibin@ubuntu:~/work/gittest$ git diff
3、对于第二种情况,版本库是 I am Andy,暂存区是 I am Bill,
   // 比较暂存区和版本库的差别
```

```
niuzibin@ubuntu:~/work/gittest$ git diff --cachec
    diff --git a/readme.txt b/readme.txt
    index 4dd9cb5..56b6b46 100644
    --- a/readme.txt
    +++ b/readme.txt
    @@ -1 +1 @@
    -I am Andy
    +I am Bill
    // 比较工作区和暂存区的差别
    niuzibin@ubuntu:~/work/gittest$ git diff
    diff --git a/readme.txt b/readme.txt
    index 56b6b46..b688830 100644
    --- a/readme.txt
    +++ b/readme.txt
    @@ -1 +1 @@
    -I am Bill
    +I am Caroline
    // 比较工作区和版本库的差别
    niuzibin@ubuntu:~/work/gittest$ git diff HEAD
    diff --git a/readme.txt b/readme.txt
    index 4dd9cb5..b688830 100644
    --- a/readme.txt
    +++ b/readme.txt
    @@ -1 +1 @@
    -I am Andv
    +T am Caroline
4、撤销工作区的修改,如下:
    niuzibin@ubuntu:~/work/gittest$ git checkout reac
    niuzibin@ubuntu:~/work/gittest$ git diff --cachec
    diff --git a/readme.txt b/readme.txt
    index 4dd9cb5..56b6b46 100644
    --- a/readme.txt
    +++ b/readme.txt
    @@ -1 +1 @@
    -I am Andy
```

```
+I am Bill
   niuzibin@ubuntu:~/work/gittest$ git diff
   niuzibin@ubuntu:~/work/gittest$ git diff HEAD
   diff --git a/readme.txt b/readme.txt
   index 4dd9cb5..56b6b46 100644
   --- a/readme.txt
   +++ b/readme.txt
   @@ -1 +1 @@
   -I am Andy
   +I am Bill
5、撤销暂存区的修改,如下:
   niuzibin@ubuntu:~/work/gittest$ git reset HEAD re
   Unstaged changes after reset:
           readme.txt
   М
   niuzibin@ubuntu:~/work/gittest$ git diff --cachec
   niuzibin@ubuntu:~/work/gittest$ git diff
   diff --git a/readme.txt b/readme.txt
   index 4dd9cb5..56b6b46 100644
   --- a/readme.txt
   +++ b/readme.txt
   @@ -1 +1 @@
   -I am Andy
   +I am Bill
6、还需要再次checkout,如下:
   niuzibin@ubuntu:~/work/gittest$ git checkout reac
   niuzibin@ubuntu:~/work/gittest$ git diff --cachec
   niuzibin@ubuntu:~/work/gittest$ git diff
7、也就是说, checkout相当于拿暂存区的数据, 覆盖工作区。reset
8、考虑工作区、暂存区、版本库的内容分别为C、B、A
   现在要全部撤销,有两个办法:
   第一个办法:checkout结果为【B、B、A】-->reset HEAD结果
   第二个办法:reset HEAD结果为【C、A、A】-->checkout结果
```

# git远程分支的删除和同步

1、其他人已经删除某个分支,但是本地还能看到,这个时候fetch报错 niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP\$ git branc

```
develop
    * niuzibin
   remotes/origin/HEAD -> origin/develop
   remotes/origin/chenming
   remotes/origin/develop
   remotes/origin/heming
   remotes/origin/master
   remotes/origin/niuzibin
   remotes/origin/patch-1
   remotes/origin/patch-2
   remotes/origin/patch-3
   remotes/origin/patch-4
   remotes/origin/revert-77cdc6fe
   remotes/origin/revert-fa155359
   remotes/origin/tuchengyu
   remotes/origin/tuchengyu2
   remotes/origin/yangyang
   remotes/origin/yogurt
   remotes/origin/ywn
   niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP$ sudo git
   fatal: Couldn't find remote ref tuchengyu
   Unexpected end of command stream
2、这个错误的原因是,fetch已经删除的分支
3、如何同步呢?
   执行 git fetch -p,如下:
   From https://192.168.10.99/FLKDBEnc/FLKCDP
   x [deleted]
                       (none)
                                  -> origin/patch-2
   x [deleted]
                       (none) -> origin/patch-3
                       (none)
   x [deleted]
                       (none)
                                 -> origin/patch-4
                                  -> origin/tuchengy
   x [deleted]
   这个时候 git branch -a,就会少一些分支
4、查看分支的关联,如下:
   niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP$ git branc
   develop 0c0c9a2 [origin/develop: behind 119] Mer
    * niuzibin 4463d6a [origin/niuzibin] 删除无用的方法
```

#### log命令

```
1、显示log
   git log
2、显示最近的两次提交log
   qit loq -2
3、显示每次提交的内容差异
   git log -2 -p
4、显示某个文件的内容差异
   git log -2 -p .gitignore
5、也可以通过show,显示某一个提交的内容差异,如下:
   root@ubuntu:/home/disk1/GitTest# git log -1 -p .c
   commit 12fdff5b6cd259cff2952fce978f1555635d4bfc
   Author: nzbbody <nzbbody@163.com>
   Date: Wed Jun 6 10:27:04 2018 +0800
       增加忽略的文件
   diff --git a/.gitignore b/.gitignore
   index 259148f..645ee14 100644
   --- a/.gitignore
   +++ b/.gitignore
   @@ -30,3 +30,6 @@
    *.exe
    *.out
    *.app
   +# Other temp file
   +*.bak
   root@ubuntu:/home/disk1/GitTest# git show 12fdffs
   commit 12fdff5b6cd259cff2952fce978f1555635d4bfc
   Author: nzbbody <nzbbody@163.com>
   Date: Wed Jun 6 10:27:04 2018 +0800
```

```
增加忽略的文件

diff --git a/.gitignore b/.gitignore
index 259148f..645ee14 100644
--- a/.gitignore
+++ b/.gitignore
@@ -30,3 +30,6 @@
*.exe
*.out
*.app
+
+# Other temp file
+*.bak
```

#### push可能会有冲突

```
1、甲修改, commit push
2、乙修改, commit push, 报错
   To https://github.com/nzbbody/GitTest
                       master -> master (fetch first
    ! [rejected]
   error: failed to push some refs to 'https://githu
   hint: Updates were rejected because the remote co
   hint: not have locally. This is usually caused by
   hint: to the same ref. You may want to first inte
   hint: (e.g., 'git pull ...') before pushing agair
   hint: See the 'Note about fast-forwards' in 'git
3、提示,在push之前,先pull一下。我们使用fetch + merge
4、执行
   git fetch origin master
   git merge origin/master
5、merge有冲突,需要解决冲突,add commit push
   root@ubuntu:/home/disk1/GitTest# git add readme.t
   root@ubuntu:/home/disk1/GitTest# git commit -m"33
   root@ubuntu:/home/disk1/GitTest# git push origin
```

# push命令

- 1、git push命令用于将本地分支的更新,推送到远程主机。它的格式。git push <远程主机名> <本地分支名>:<远程分支名>
- 2、git push origin master 上面命令表示,将本地的master分支推送到origin主机的maste

### svn与git对照

- 1、svn是集中式,git是分布式
- 2、svn保存文件的差异,git保存文件的快照,如果没有变化,从上一次
- 3、svn创建分支,相当于整体拷贝,git创建分支,在任意一个提交点

#### 冲突的情况只有两种

- 1、两个分支都提交了,然后merge时 办法是:解决冲突+add+commit
- 2、push的时候,其实也是两个分支都提交了,需要merge 办法是:解决冲突+add+commit+push
- 3、你可能会想, master改了,提交。test1改了,先把master分支m。这种情况是不存在的。因为:test1修改了,没有提交的情况下,

# 我们的开发分支管理

- 1、远程有develop和niuzibin,在gitlab执行 merge niuzibin-
- 2、我的本地有 niuzibin,注意:本地的develop没有意义 本地的niuzibin分支,push到 origin/niuzibin
- 3、我在本地修改 niuzibin分支,
- 4、然后add commit,
- 5、从origin/develop进行fetch,然后merge,有冲突,解决冲突,
- 6、注意:4和5是不能颠倒顺序的,本地有修改,必须先提交,再merge

# 撤销本地修改

1、准备工作如下:

niuzibin@ubuntu:~/work/gittest\$ git init
Initialized empty Git repository in /home/niuzibi

niuzibin@ubuntu:~/work/gittest\$ git config --glok niuzibin@ubuntu:~/work/gittest\$ git config --glok niuzibin@ubuntu:~/work/gittest\$ git config user.r niuzibin niuzibin@ubuntu:~/work/gittest\$ git config user. nzbbody@163.com niuzibin@ubuntu:~/work/gittest\$ vi readme.txt niuzibin@ubuntu:~/work/gittest\$ git add readme.t> niuzibin@ubuntu:~/work/gittest\$ git commit -m"adc [master (root-commit) 7846ec0] add file readme.t> 1 file changed, 1 insertion(+) create mode 100644 readme.txt niuzibin@ubuntu:~/work/gittest\$ git log commit 7846ec02be0f61138d84b75aa5213c9f4421d4b4 Author: niuzibin <nzbbody@163.com> Date: Mon Apr 9 18:29:47 2018 -0700

add file readme.txt

2、注意:提交的时候,如果没有-m写日志,会提示输入日志。输入之后 3、修改,查看变化

niuzibin@ubuntu:~/work/gittest\$ git status
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be (
(use "git checkout -- <file>..." to discard chance

modified: readme.txt

Untracked files:
 (use "git add <file>..." to include in what will

1

no changes added to commit (use "git add" and/or niuzibin@ubuntu:~/work/gittest\$ git diff diff --git a/readme.txt b/readme.txt index 4dd9cb5..3f54eb5 100644

--- a/readme.txt +++ b/readme.txt @@ -1 +1,2 @@ I am Andy +hehe

4、撤销本地修改

niuzibin@ubuntu:~/work/gittest\$ git checkout -- r

5、上面的方法撤销一个文件的修改,如果撤销当前目录下的多个文件修 niuzibin@ubuntu:~/work/gittest\$ git checkout ./

# 是不是只能merge提交的分支

- 1、创建test1分支,修改文件,add、commit
- 2、转到master分支,也修改文件,这个是时候,diff test1 maste 注意:diff比较分支,是比较已经提交的分支。
- 3、先不提交, merge试一下, 报错

error: Your local changes to the following files readme.txt

Please, commit your changes or stash them before Aborting

4、提示:要求提交之后,再merge,如下:

root@ubuntu:/home/disk1/GitTest# git add readme.t root@ubuntu:/home/disk1/GitTest# git commit -m"ma [master 217992a] master

1 file changed, 1 insertion(+)

root@ubuntu:/home/disk1/GitTest# git merge test1
Auto-merging readme.txt

CONFLICT (content): Merge conflict in readme.txt Automatic merge failed; fix conflicts and then co

5、这个时候, merge有冲突, 需要解决冲突解决冲突, 然后再进行 add、commit

6、结论:

当前分支修改后不提交, merge报错, 要求先提交, 再merge 提交后再merge, 可能会冲突, 需要解决冲突, 然后add commit

# 本地创建分支,并和远程关联

```
1、示例如下:
niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP$ git branch -a
* develop
  niuzibin
  remotes/origin/HEAD -> origin/develop
  remotes/origin/chenming
  remotes/origin/develop
  remotes/origin/heming
  remotes/origin/master
  remotes/origin/niuzibin
  remotes/origin/patch-1
  remotes/origin/revert-77cdc6fe
  remotes/origin/revert-fa155359
  remotes/origin/tuchengyu2
  remotes/origin/yangyang
  remotes/origin/yogurt
  remotes/origin/ywn
  remotes/origin/ywn1
niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP$ git branch to
niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP$ git branch -\
            dc5f079 [origin/develop] Merge branch '\
* develop
  niuzibin
            4463d6a [origin/niuzibin] 删除无用的方法
  tuchengyu2 dc5f079 Merge branch 'yangyang' into 'de
niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP$ git branch --
Branch tuchengyu2 set up to track remote branch tuche
niuzibin@ubuntu:~/work/FLKDBEnc/FLKCDP$ git branch -\
* develop
             dc5f079 [origin/develop] Merge branch '\
            4463d6a [origin/niuzibin] 删除无用的方法
  niuzibin
  tuchengyu2 dc5f079 [origin/tuchengyu2: behind 14] N
```

# 参见

本地的修改状态

```
工作区----【add】----->暂存区----【commit】-----
工作区<---【checkout】---暂存区<---【reset HEAD】----
```

#### 看看fetch做了什么事情

- 1、git fetch origin master,通过fetch前后的文本对照和抓包 fetch会获取远程分支的commit id,也就是本地的文件FETCH\_
- 2、然后merge origin/master,会合并 origin/master 到 mer 也就是说,merge origin/master 并没有网络传输
- 3、git pull origin master = git fetch origin master + 但是,一般不要使用pull,在背后进行merge总是不好的。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

grep常用功能

grep查询制表符

grep或条件的查询

### 变量说明

#### grep常用功能

```
1、示例数据如下:
   Andy
   Bill
   Caroline
   David
   Eric
   Favad
2、查找包含i的行
   [niu zibin@localhost test]$ more aaa.txt | grep i
3、 查找A开头的行
   [niu_zibin@localhost test]$ more aaa.txt |grep
4、查找ne结尾的行
   [niu_zibin@localhost test]$ more aaa.txt |grep 'r
5、不区分大小写
   [niu_zibin@localhost test]$ more aaa.txt |grep -i
6、匹配一个字符,这个字符是任意字符
   [niu_zibin@localhost test]$ more aaa.txt |grep '\
7、前面一个字符,后面出现0次或多次
   [niu_zibin@localhost test]$ more aaa.txt |grep 'c
   注意:这种情况等价于不过滤,因为任何一行都满足d出现@次或者
   如果表示出现一次或多次,需要使用dd*
   还有一种办法,使用+表示出现1次或多次,但是grep不支持,需要
   [niu_zibin@localhost test]$ more aaa.txt |egrep |
   ?表示0次或者1次,也是egrep的语法
   也就是说,egrep是grep的扩展,支持+,?,
8、特别注意:通配符*与正则表达式*的区别,通配符*表示匹配0到多个
```

```
正则表达式*表示前一个字符出现0次或多次
```

9、转义字符

[niu\_zibin@localhost test]\$ more aaa.txt |grep \'

10、查找一个字符,属于集合中的一个

[niu\_zibin@localhost test]\$ more aaa.txt |grep v|

- 11、使用grep查找文件,在当前目录下的文件中,查找内容包含andy的 [niu\_zibin@localhost time]\$ grep -i andy ./\*
- 12、使用grep查找文件,在当前目录下,递归,查找内容包含andy的プ [niu\_zibin@localhost time]\$ grep -i andy -R ./

#### grep或条件的查询

```
1, [root@localhost \sim]# more aaa.txt |grep -n "111\|22
```

1:111

3:222

5:111

7:222

2\ [root@localhost ~]# more aaa.txt |grep -nE "111|22

1:111

3:222

5:111

7:222

3、[root@localhost ~]# more aaa.txt |grep -ne 111 -ne

1:111

3:222

5:111

7:222

#### grep查询制表符

1、考虑,如何使用grep查询包含制表符的行? 使用\t不行

2、第一种办法:输入制表符,但是在shell中,制表符(tab键有特殊) 使用ctrl+v+i,输入制表符,如下:

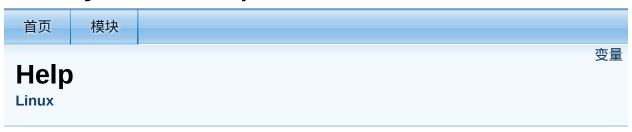
```
[root@localhost niu9]# more aaa.txt |grep "
```

**111** ddd

222 hhh

```
333
           kkkk
   444
           jjj
3、使用cat -A列出所有的字符,对于制表符,显示^I,grep ^I即可
    [root@localhost niu9]# cat -A aaa.txt
   111^Iddd$
   ddsdf$
   222^I^Ihhh$
   asasdf$
   333^Ikkkk$
   fdsf dsdsds
               adddsa$
   444<sup>1</sup>111115
    [root@localhost niu9]# cat -A aaa.txt|grep "\^I"
   111^Iddd$
   222^I^Ihhh$
   333^Ikkkk$
   444<sup>1</sup>111115
   注意:脱字符需要转义
4、对于制表符(水平制表符), ASCII取值为9, 可以通过这种方式过》
    [root@localhost niu9]# more aaa.txt | grep -P "\01
   111
           ddd
   222
                   hhh
   333
           kkkk
   444
           111
   注意:\0 表示8进制,如果是16进制使用 \x,如下:more aaa
5、常用的不可见字符,取值如下:
   水平制表符[\t]
                  9
   新行符[\n]
                 10 [牛思齐 new 10]
   回车符[\r]
                 13
   空格
                32
```

# Andy Niu Help 1.0.0.0



# 重变

man
info
help

## 详细描述

## 变量说明

#### help

```
一般的命令都提供了 xxx --help, 描述简单用法
```

参见

#### info

```
和man,一样,信息更全面
```

参见

man

#### man

```
命令如下:
man route
注意:man和info,都是使用less输出结果,q退出后,输出的结果不fexport PAGER='less -X' // 保留屏幕
export PAGER='less' // 不保留屏幕
对于less命令也是同样的道理,
```

less config.xml//退出不保留屏幕less -X config.xml//退出保留屏幕

注意:是否保留屏幕还和系统,终端有关,有可能默认就是保留。

## man是使用less输出结果,查找关键字参考less命令。

参见 info

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

iO多路复用 Linux

## 变量

IO多路复用\_\_总结

IO多路复用\_\_select

IO多路复用\_\_poll

IO多路复用\_\_epoll

## 变量说明

#### IO多路复用\_\_epoll

```
1, server.cpp
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include <errno.h>
   #include <netinet/in.h>
   #include <sys/socket.h>
   #include <arpa/inet.h>
   #include <sys/epoll.h>
   #include <unistd.h>
   #include <sys/types.h>
                       "127.0.0.1"
   #define IPADDRESS
   #define PORT
                       8787
   #define MAXSIZE
                       1024
   #define LISTENQ
                       5
   #define FDSIZE
                       1000
   #define EPOLLEVENTS 100
   //函数声明
   //创建套接字并进行绑定
   static int socket_bind(const char* ip,int port);
   //IO多路复用epoll
   static void do_epoll(int listenfd);
   //事件处理函数
   static void
```

```
handle_events(int epollfd, struct epoll_event *eve
//处理接收到的连接
static void handle_accpet(int epollfd,int listent
//读处理
static void do read(int epollfd,int fd,char *buf)
//写处理
static void do_write(int epollfd,int fd,char *buf
//添加事件
static void add_event(int epollfd,int fd,int stat
//修改事件
static void modify event(int epollfd,int fd,int s
//删除事件
static void delete_event(int epollfd,int fd,int s
int main(int argc,char *argv[])
{
    int
         listenfd;
    listenfd = socket_bind(IPADDRESS,PORT);
    listen(listenfd, LISTENQ);
    do_epoll(listenfd);
    return 0;
}
static int socket_bind(const char* ip,int port)
{
         listenfd;
    int
    struct sockaddr_in servaddr;
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    if (listenfd == -1)
    {
        perror("socket error:");
        exit(1);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, ip, &servaddr.sin_addr);
    servaddr.sin_port = htons(port);
```

```
if (bind(listenfd,(struct sockaddr*)&servaddr
    {
        perror("bind error: ");
        exit(1);
    return listenfd;
}
static void do_epoll(int listenfd)
{
    int epollfd;
    struct epoll_event events[EPOLLEVENTS];
    int ret;
    char buf[MAXSIZE];
    memset(buf, 0, MAXSIZE);
    //创建一个描述符
    epollfd = epoll_create(FDSIZE);
    //添加监听描述符事件
    add_event(epollfd, listenfd, EPOLLIN);
    for (;;)
    {
        //获取已经准备好的描述符事件
        ret = epoll_wait(epollfd, events, EPOLLEVEN
        handle_events(epollfd, events, ret, listenfo
    close(epollfd);
}
static void
handle_events(int epollfd, struct epoll_event *eve
{
    int i;
    int fd;
    //进行选好遍历
    for (i = 0; i < num; i++)
    {
        fd = events[i].data.fd;
```

```
//根据描述符的类型和事件类型进行处理
        if ((fd == listenfd) &&(events[i].events
            handle_accpet(epollfd, listenfd);
        else if (events[i].events & EPOLLIN)
            do_read(epollfd, fd, buf);
        else if (events[i].events & EPOLLOUT)
            do_write(epollfd, fd, buf);
    }
static void handle_accpet(int epollfd,int listenf
{
    int clifd;
    struct sockaddr_in cliaddr;
    socklen t cliaddrlen;
    clifd = accept(listenfd,(struct sockaddr*)&c]
    if (clifd == -1)
        perror("accpet error:");
    else
    {
        // 注意:cliaddr.sin_port的字节逆序,需要转换
        printf("accept a new client: %s:%d\n",
            inet_ntoa(cliaddr.sin_addr),
            ntohs(cliaddr.sin_port));
        //添加一个客户描述符和事件
        add_event(epollfd, clifd, EPOLLIN);
    }
}
static void do_read(int epollfd,int fd,char *buf)
{
    int nread;
    nread = read(fd, buf, MAXSIZE);
    if (nread == -1)
    {
        perror("read error:");
        close(fd);
        delete_event(epollfd,fd,EPOLLIN);
```

```
else if (nread == 0)
    {
        fprintf(stderr, "client close.\n");
        close(fd);
        delete_event(epollfd, fd, EPOLLIN);
    }
    else
    {
        printf("read message is : %s",buf);
        //修改描述符对应的事件,由读改为写
        modify_event(epollfd, fd, EPOLLOUT);
    }
}
static void do_write(int epollfd,int fd,char *bu1
{
    int nwrite;
    nwrite = write(fd, buf, strlen(buf));
    if (nwrite == -1)
    {
        perror("write error:");
        close(fd);
        delete_event(epollfd, fd, EPOLLOUT);
    }
    else
        modify_event(epollfd, fd, EPOLLIN);
    memset(buf, 0, MAXSIZE);
}
static void add_event(int epollfd,int fd,int stat
{
    struct epoll_event ev;
    ev.events = state;
    ev.data.fd = fd;
    epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev);
}
```

```
static void delete_event(int epollfd,int fd,int s
    {
        struct epoll event ev;
        ev.events = state;
        ev.data.fd = fd;
        epoll_ctl(epollfd, EPOLL_CTL_DEL, fd, &ev);
    }
    static void modify_event(int epollfd,int fd,int s
    {
        struct epoll event ev;
        ev.events = state;
        ev.data.fd = fd;
        epoll_ctl(epollfd, EPOLL_CTL_MOD, fd, &ev);
2、client.cpp
    #include <netinet/in.h>
    #include <svs/socket.h>
    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>
    #include <sys/epoll.h>
    #include <time.h>
    #include <unistd.h>
    #include <sys/types.h>
    #include <arpa/inet.h>
    #define MAXSIZE
                         1024
    #define IPADDRESS
                         "127.0.0.1"
    #define SERV_PORT
                         8787
    #define FDSIZE
                           1024
    #define EPOLLEVENTS 20
    static void handle_connection(int sockfd);
    static void
    handle_events(int epollfd, struct epoll_event *eve
```

```
static void do_read(int epollfd,int fd,int sockfc
static void do_read(int epollfd,int fd,int sockfc
static void do_write(int epollfd,int fd,int sockf
static void add event(int epollfd,int fd,int stat
static void delete event(int epollfd,int fd,int s
static void modify_event(int epollfd,int fd,int s
int main(int argc,char *argv[])
{
                        sockfd;
    int
                        servaddr;
    struct sockaddr in
    sockfd = socket(AF INET, SOCK STREAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin family = AF INET;
    servaddr.sin_port = htons(SERV_PORT);
    inet_pton(AF_INET, IPADDRESS, &servaddr.sin_adc
    connect(sockfd,(struct sockaddr*)&servaddr,si
    //处理连接
    handle_connection(sockfd);
    close(sockfd);
    return 0;
}
static void handle_connection(int sockfd)
{
    int epollfd;
    struct epoll_event events[EPOLLEVENTS];
    char buf[MAXSIZE];
    int ret;
    epollfd = epoll_create(FDSIZE);
    add event(epollfd,STDIN FILENO,EPOLLIN);
    for (;;)
    {
        ret = epoll_wait(epollfd, events, EPOLLEVEN
        handle_events(epollfd, events, ret, sockfd, t
    }
```

```
close(epollfd);
}
static void
handle_events(int epollfd, struct epoll_event *eve
{
    int fd;
    int i;
    for (i = 0; i < num; i++)
    {
        fd = events[i].data.fd;
        if (events[i].events & EPOLLIN)
            do_read(epollfd, fd, sockfd, buf);
        else if (events[i].events & EPOLLOUT)
            do_write(epollfd, fd, sockfd, buf);
    }
}
static void do_read(int epollfd,int fd,int sockfc
{
    int nread;
    nread = read(fd, buf, MAXSIZE);
    if (nread == -1)
    {
        perror("read error:");
        close(fd);
    else if (nread == 0)
    {
        fprintf(stderr, "server close.\n");
        close(fd);
    }
    else
    {
        if (fd == STDIN_FILENO)
            add_event(epollfd, sockfd, EPOLLOUT);
        else
```

```
delete_event(epollfd, sockfd, EPOLLIN);
            add_event(epollfd, STDOUT_FILENO, EPOLL
        }
    }
}
static void do_write(int epollfd,int fd,int sock1
{
    int nwrite;
    nwrite = write(fd, buf, strlen(buf));
    if (nwrite == -1)
    {
        perror("write error:");
        close(fd);
    }
    else
    {
        if (fd == STDOUT_FILENO)
            delete_event(epollfd, fd, EPOLLOUT);
        else
            modify_event(epollfd, fd, EPOLLIN);
    }
    memset(buf, 0, MAXSIZE);
}
static void add_event(int epollfd,int fd,int stat
{
    struct epoll_event ev;
    ev.events = state;
    ev.data.fd = fd;
    epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev);
}
static void delete_event(int epollfd,int fd,int s
    struct epoll_event ev;
```

```
ev.events = state;
        ev.data.fd = fd;
        epoll_ctl(epollfd, EPOLL_CTL_DEL, fd, &ev);
    }
    static void modify_event(int epollfd,int fd,int s
        struct epoll_event ev;
        ev.events = state;
        ev.data.fd = fd;
        epoll_ctl(epollfd, EPOLL_CTL_MOD, fd, &ev);
3、运行服务端
    [root@localhost Epoll]# g++ -o server server.cpp
    server.cpp:190:2: warning: no newline at end of 1
    [root@localhost Epoll]# ./server
    accept a new client: 127.0.0.1:51619
    read message is : 123
4、运行客户端
    [root@localhost Epoll]# g++ -o client client.cpp
    client.cpp:143:2: warning: no newline at end of 1
    [root@localhost Epoll]# ./client
    123
    123
```

#### IO多路复用 poll

```
1    server.cpp
    #include <stdio.h>
    #include <stdib.h>
    #include <string.h>
    #include <errno.h>
    #include <netinet/in.h>
    #include <sys/socket.h>
    #include <poll.h>
    #include <unistd.h>
    #include <sys/types.h>
```

```
#include <arpa/inet.h>
                    "127.0.0.1"
#define IPADDRESS
#define PORT
                    8787
#define MAXLINE
                    1024
#define LISTENO
                    5
#define OPEN MAX
                    1000
#define INFTIM
                    - 1
//函数声明
//创建套接字并进行绑定
static int socket_bind(const char* ip,int port);
//IO多路复用poll
static void do_poll(int listenfd);
//处理多个连接
static void handle_connection(struct pollfd *conr
int main(int argc,char *argv[])
{
    int listenfd, connfd, sockfd;
    struct sockaddr_in cliaddr;
    socklen_t cliaddrlen;
    listenfd = socket_bind(IPADDRESS, PORT);
    listen(listenfd, LISTENQ);
    do_poll(listenfd);
    return 0;
}
static int socket_bind(const char* ip,int port)
{
    int listenfd;
    struct sockaddr in servaddr;
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    if (listenfd == -1)
    {
        perror("socket error:");
        exit(1);
```

```
int reuse = 1;
    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSE
        return -1;
    }
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, ip, &servaddr.sin_addr);
    servaddr.sin port = htons(port);
    if (bind(listenfd,(struct sockaddr*)&servaddr
    {
        perror("bind error: ");
        exit(1);
    return listenfd;
}
static void do_poll(int listenfd)
{
    int connfd, sockfd;
    struct sockaddr_in cliaddr;
    socklen_t cliaddrlen;
    struct pollfd clientfds[OPEN_MAX];
    int maxi;
    int i;
    int nready;
    //添加监听描述符
    clientfds[0].fd = listenfd;
    clientfds[0].events = POLLIN;
    //初始化客户连接描述符
    for (i = 1; i < OPEN_MAX; i++)
        clientfds[i].fd = -1;
    \max i = 0;
    //循环处理
    for (;;)
```

```
//获取可用描述符的个数
nready = poll(clientfds, maxi+1, INFTIM);
if (nready == -1)
{
    perror("poll error:");
    exit(1);
//测试监听描述符是否准备好
if (clientfds[0].revents & POLLIN)
{
    cliaddrlen = sizeof(cliaddr);
    //接受新的连接
    if ((connfd = accept(listenfd,(struct
    {
        if (errno == EINTR)
            continue;
        else
        {
            perror("accept error:");
            exit(1);
        }
    fprintf(stdout, "accept a new client:
        inet_ntoa(cliaddr.sin_addr),
        ntohs(cliaddr.sin_port));
    //将新的连接描述符添加到数组中
    for (i = 1; i < OPEN_MAX; i++)
        if (clientfds[i].fd < 0)</pre>
        {
            clientfds[i].fd = connfd;
            break;
        }
    if (i == OPEN_MAX)
```

```
fprintf(stderr, "too many clients.
                exit(1);
            }
            //将新的描述符添加到读描述符集合中
            clientfds[i].events = POLLIN;
            //记录客户连接套接字的个数
            maxi = (i > maxi ? i : maxi);
            if (--nready \le 0)
                continue;
        //处理客户连接
        handle_connection(clientfds, maxi);
    }
}
static void handle_connection(struct pollfd *conr
{
    int i,n;
    char buf[MAXLINE];
    memset(buf, 0, MAXLINE);
    for (i = 1; i \le num; i++)
    {
        if (connfds[i].fd < 0)</pre>
            continue;
        //测试客户描述符是否准备好
        if (connfds[i].revents & POLLIN)
        {
            //接收客户端发送的信息
            n = read(connfds[i].fd,buf,MAXLINE);
            if (n == 0)
            {
                close(connfds[i].fd);
                connfds[i].fd = -1;
                continue;
            // printf("read msg is: ");
            write(STDOUT_FILENO, buf, n);
```

```
//向客户端发送buf
                write(connfds[i].fd,buf,n);
            }
        }
2、client.cpp
    #include <netinet/in.h>
    #include <sys/socket.h>
    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>
    #include <poll.h>
    #include <time.h>
    #include <unistd.h>
    #include <sys/types.h>
    #include <arpa/inet.h>
    #define MAXLINE
                        1024
    #define IPADDRESS
                        "127.0.0.1"
    #define SERV_PORT
                        8787
    #define max(a,b) (a > b) ? a : b
    static void handle_connection(int sockfd);
    int main(int argc,char *argv[])
    {
        int
                             sockfd;
                            servaddr;
        struct sockaddr in
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin family = AF INET;
        servaddr.sin_port = htons(SERV_PORT);
        inet pton(AF INET, IPADDRESS, & servaddr. sin add
        connect(sockfd,(struct sockaddr*)&servaddr,si
        //处理连接描述符
        handle_connection(sockfd);
```

```
return 0;
}
static void handle_connection(int sockfd)
{
            sendline[MAXLINE], recvline[MAXLINE];
    char
            maxfdp, stdineof;
    int
    struct pollfd pfds[2];
    int n;
    //添加连接描述符
    pfds[0].fd = sockfd;
    pfds[0].events = POLLIN;
    //添加标准输入描述符
    pfds[1].fd = STDIN_FILENO;
    pfds[1].events = POLLIN;
    for (; ;)
    {
        poll(pfds, 2, -1);
        if (pfds[0].revents & POLLIN)
        {
            n = read(sockfd, recvline, MAXLINE);
            if (n == 0)
            {
                fprintf(stderr, "client: server is
                close(sockfd);
            write(STDOUT_FILENO, recvline, n);
        }
        //测试标准输入是否准备好
        if (pfds[1].revents & POLLIN)
        {
            n = read(STDIN FILENO, sendline, MAXLIN
            if (n == 0)
            {
                shutdown(sockfd, SHUT_WR);
                continue;
            }
```

```
write(sockfd,sendline,n);
}
}
3、运行服务端
[root@localhost poll]# ./server
accept a new client: 127.0.0.1:54849
abc
4、运行客户端
[root@localhost poll]# ./client
abc
abc
abc
```

#### IO多路复用\_\_select

```
1, server.cpp
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <errno.h>
    #include <netinet/in.h>
    #include <arpa/inet.h>
    #include <sys/socket.h>
    #include <sys/select.h>
    #include <sys/types.h>
    #include <unistd.h>
    #include <assert.h>
    #define IPADDR
                        "127.0.0.1"
    #define PORT
                        8787
    #define MAXLINE
                        1024
    #define LISTENQ
                        5
    #define SIZE
                        10
    typedef struct server_context_st
        int cli_cnt;
                           /*客户端个数*/
```

```
int clifds[SIZE]; /*客户端的个数*/
   fd_set allfds; /*句柄集合*/
                       /*句柄最大值*/
    int maxfd;
} server context st;
static server_context_st *s_srv_ctx = NULL;
static int create_server_proc(const char* ip,int
{
    int fd;
    struct sockaddr in servaddr;
   fd = socket(AF_INET, SOCK_STREAM, 0);
    if (fd == -1) {
       fprintf(stderr, "create socket fail, error
           errno, strerror(errno));
        return -1;
   }
    /*一个端口释放后会等待两分钟之后才能再被使用,SO REU
    int reuse = 1;
   if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR,
        return -1;
   }
   bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
   // ip和port进行转化
   inet_pton(AF_INET, ip, &servaddr.sin_addr);
    servaddr.sin_port = htons(port);
   if (bind(fd,(struct sockaddr*)&servaddr,sizec
        perror("bind error: ");
        return -1;
    }
   listen(fd, LISTENQ);
```

```
return fd;
}
static int accept client proc(int srvfd)
{
    struct sockaddr_in cliaddr;
    socklen_t cliaddrlen;
    cliaddrlen = sizeof(cliaddr);
    int clifd = -1;
    printf("accpet clint proc is called.\n");
ACCEPT:
    clifd = accept(srvfd,(struct sockaddr*)&cliac
    if (clifd == -1) {
        if (errno == EINTR) {
            goto ACCEPT;
        } else {
            fprintf(stderr, "accept fail,error:%s
            return -1;
        }
    }
    // ip和port进行转化
    fprintf(stdout, "accept a new client: %s:%d\r
        inet_ntoa(cliaddr.sin_addr), ntohs(cliaddr
    //将新的连接描述符添加到数组中
    int i = 0;
    for (i = 0; i < SIZE; i++) {
        if (s_srv_ctx->clifds[i] < 0) {</pre>
            s srv ctx->clifds[i] = clifd;
            s_srv_ctx->cli_cnt++;
            break;
        }
```

```
if (i == SIZE) {
        fprintf(stderr, "too many clients.\n");
        return -1;
    }
}
static int handle_client_msg(int fd, char *buf)
{
    assert(buf);
    printf("recv buf is :%s\n", buf);
   write(fd, buf, strlen(buf) +1);
    return 0;
}
static void recv_client_msg(fd_set *readfds)
{
    int i = 0, n = 0;
    int clifd;
    char buf[MAXLINE] = \{0\};
    for (i = 0;i <= s_srv_ctx->cli_cnt;i++) {
        clifd = s_srv_ctx->clifds[i];
        if (clifd < 0) {
            continue;
        /*判断客户端套接字是否有数据*/
        if (FD_ISSET(clifd, readfds)) {
            //接收客户端发送的信息
            n = read(clifd, buf, MAXLINE);
            if (n <= 0) {
                /*n==0表示读取完成,客户都关闭套接字*/
                FD_CLR(clifd, &s_srv_ctx->allfds)
                close(clifd);
                s_srv_ctx->clifds[i] = -1;
                continue;
```

```
handle_client_msg(clifd, buf);
       }
   }
static void handle_client_proc(int srvfd)
   int clifd = -1;
    int retval = 0;
   fd_set *readfds = &s_srv_ctx->allfds;
    struct timeval tv;
   int i = 0;
   while (1) {
       /*每次调用select前都要重新设置文件描述符和时间。
       FD ZERO(readfds);
       /*添加监听套接字*/
       FD_SET(srvfd, readfds);
       s srv ctx->maxfd = srvfd;
       tv.tv\_sec = 30;
       tv.tv\_usec = 0;
       /*添加客户端套接字*/
       for (i = 0; i < s\_srv\_ctx->cli\_cnt; i++)
           clifd = s_srv_ctx->clifds[i];
           /*去除无效的客户端句柄*/
           if (clifd != -1) {
               FD_SET(clifd, readfds);
           s_srv_ctx->maxfd = (clifd > s_srv_ct>
       }
       /*开始轮询接收处理服务端和客户端套接字*/
       retval = select(s srv ctx->maxfd + 1, rea
       if (retval == -1) {
           fprintf(stderr, "select error:%s.\n",
           return;
```

```
if (retval == 0) {
            fprintf(stdout, "select is timeout.\r
            continue;
        if (FD_ISSET(srvfd, readfds)) {
            /*监听客户端请求*/
            accept_client_proc(srvfd);
        } else {
            /*接受处理客户端消息*/
            recv_client_msg(readfds);
        }
   }
}
static void server_uninit()
{
    if (s_srv_ctx) {
        free(s_srv_ctx);
        s_srv_ctx = NULL;
    }
}
static int server_init()
{
    s_srv_ctx = (server_context_st *)malloc(size(
    if (s_srv_ctx == NULL) {
        return -1;
    }
   memset(s_srv_ctx, 0, sizeof(server_context_st
    int i = 0;
    for (;i < SIZE; i++) {
        s_srv_ctx->clifds[i] = -1;
    }
```

```
return 0;
    }
    int main(int argc,char *argv[])
    {
        int srvfd;
        /*初始化服务端context*/
        if (server_init() < 0) {</pre>
            return -1;
        /*创建服务,开始监听客户端请求*/
        srvfd = create_server_proc(IPADDR, PORT);
        if (srvfd < 0) {
            fprintf(stderr, "socket create or bind fa
            goto err;
        /*开始接收并处理客户端请求*/
        handle_client_proc(srvfd);
        server_uninit();
        return 0;
    err:
        server_uninit();
        return -1;
2、client.cpp
    #include <netinet/in.h>
    #include <arpa/inet.h>
    #include <sys/socket.h>
    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>
    #include <sys/select.h>
    #include <time.h>
    #include <unistd.h>
    #include <sys/types.h>
    #include <errno.h>
```

```
#define MAXLINE 1024
#define IPADDRESS "127.0.0.1"
#define SERV_PORT 8787
#define max(a,b) (a > b) ? a : b
static void handle_recv_msg(int sockfd, char *bu1
{
    printf("client recv msg is:%s\n", buf);
    sleep(5);
   write(sockfd, buf, strlen(buf) +1);
}
static void handle_connection(int sockfd)
{
    char sendline[MAXLINE], recvline[MAXLINE];
    int maxfdp, stdineof;
    fd_set readfds;
    int n;
    struct timeval tv;
    int retval = 0;
    while (1) {
        FD_ZERO(&readfds);
        FD_SET(sockfd,&readfds);
        maxfdp = sockfd;
        tv.tv\_sec = 5;
        tv.tv\_usec = 0;
        retval = select(maxfdp+1,&readfds,NULL,NL
        if (retval == -1) {
            return ;
        }
```

```
if (retval == 0) {
            printf("client timeout.\n");
            continue;
        }
        if (FD_ISSET(sockfd, &readfds)) {
            n = read(sockfd, recvline, MAXLINE);
            if (n <= 0) {
                fprintf(stderr, "client: server is
                close(sockfd);
                FD_CLR(sockfd,&readfds);
                return;
            }
            handle_recv_msg(sockfd, recvline);
        }
    }
}
int main(int argc,char *argv[])
{
    int sockfd;
    struct sockaddr_in servaddr;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERV_PORT);
    inet_pton(AF_INET, IPADDRESS, &servaddr.sin_add
    int retval = 0;
    retval = connect(sockfd,(struct sockaddr*)&se
    if (retval < 0) {
        fprintf(stderr, "connect fail,error:%s\n'
        return -1;
    }
```

```
sockaddr_in clientAddr;
        sockaddr_in serverAddr;
        socklen t addrLen = sizeof(sockaddr in);
        // 获取本地ip和port
        getsockname(sockfd, (struct sockaddr*)&client
        // 获取对端ip和port
        getpeername(sockfd, (struct sockaddr*)&server
        printf("client[%s:%d] connect server[%s:%d]\r
            inet ntoa(clientAddr.sin addr),
            ntohs(clientAddr.sin_port),
            inet_ntoa(serverAddr.sin_addr),
            ntohs(serverAddr.sin_port));
        printf("client send to server.\n");
        char data[64] = \{0\};
        sprintf(data, "hello server[%s:%d], this is cli
            inet_ntoa(serverAddr.sin_addr),
            ntohs(serverAddr.sin_port),
            inet_ntoa(clientAddr.sin_addr),
            ntohs(clientAddr.sin_port));
        write(sockfd, data, 64);
        handle_connection(sockfd);
        return 0;
3、运行服务端
    [root@localhost Select]# g++ -o server server.cpr
    [root@localhost Select]# ./server
    accpet clint proc is called.
    accept a new client: 127.0.0.1:42108
    recv buf is :hello server[127.0.0.1:8787], this is
    accpet clint proc is called.
```

accept a new client: 127.0.0.1:42109

recv buf is :hello server[127.0.0.1:8787], this is

recv buf is :hello server[127.0.0.1:8787], this is recv buf is :hello server[127.0.0.1:8787], this is

recv buf is :hello server[127.0.0.1:8787], this is

4、运行客户端

[root@localhost Select]# g++ -o client client.cpr client[127.0.0.1:42108] connect server[127.0.0.1: client send to server.

client recv msg is:hello server[127.0.0.1:8787],t
client recv msg is:hello server[127.0.0.1:8787],t

5、注意:if (setsockopt(fd, SOL\_SOCKET, SO\_REUSEADDR, 没有这段代码,先关闭服务端,这时候连接处于 TIME\_WAIT状态 [root@localhost poll]# ./server

bind error: : Address already in use 加上上面的这段代码,就不会有这个问题。

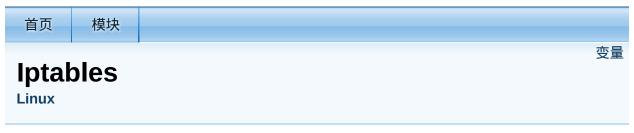
#### IO多路复用 总结

- 1、I0多路复用解决什么问题?
  - 一个连接创建一个线程,用于这个连接的交互,但是线程的开销比!
- 2、怎么解决这个问题?

可以使用一个线程管理多个连接,一个连接对应一个socket句柄,

- 3、select特点
  - a、监视的FD(文件描述符)个数有限,1024个
  - b、对于FD集合,需要从用户态拷贝到内核态,同时进行线性扫描
- 4、poll特点
  - a、poll可以认为是加强版的select,可以监视的FD个数不限,[
  - b、整体拷贝和线性扫描的问题,还是同样存在
- 5、epoll特点
  - a、epol1解决了上述问题,可以监视的FD个数不限
  - b、从用户态到内核态只拷贝一次
  - c、select、poll不断主动轮询,把就绪的FD设置一下,在应用和epoll也就是主动轮询,但是只把就绪的FD放入就绪链表中,

# Andy Niu Help 1.0.0.0



# 重变

理解iptables
iptables常用的操作
设置了DNAT但是telnet不通
多网卡网线直连网络异常
以linux主机为跳板登录windows主机
一个iptables示例
iptables的常见问题
iptables连接和抓包的问题
iptables的NAT设置
NAT与路由器的区别

## 详细描述

## 变量说明

#### iptables常用的操作

```
1、添加规则链
    [root@localhost ~]# iptables -t nat -A PREROUTIN(
    [root@localhost ~]# iptables -t nat -A PREROUTIN(
    [root@localhost ~]# service iptables status
    Table: filter
    Chain INPUT (policy ACCEPT)
                prot opt source
    num target
                                                  des
    Chain FORWARD (policy ACCEPT)
                   prot opt source
    num target
                                                  des
    Chain OUTPUT (policy ACCEPT)
        target prot opt source
                                                  des
    num
    Table: nat
    Chain PREROUTING (policy ACCEPT)
   num target
                   prot opt source
                                                  des
                    tcp -- 0.0.0.0/0
                                                  \Theta . \Theta
    1
        DNAT
                                                  \Theta . \Theta
    2
        DNAT
                    tcp -- 0.0.0.0/0
    Chain POSTROUTING (policy ACCEPT)
        target
                    prot opt source
    num
                                                  des
    Chain OUTPUT (policy ACCEPT)
    num target
                    prot opt source
                                                  des
2、查看某个条件的规则链
```

[root@localhost ~]# iptables -t nat -L PREROUTIN( Chain PREROUTING (policy ACCEPT) prot opt source target des num tcp -- anywhere 1 DNAT any DNAT tcp -- anywhere any 3、删除某个规则链 [root@localhost ~]# iptables -t nat -D PREROUTIN( [root@localhost ~]# iptables -t nat -L PREROUTIN( Chain PREROUTING (policy ACCEPT) target prot opt source des num **DNAT** 1 tcp -- anywhere any 4、清除规则链 [root@localhost ~]# iptables -F -t nat [root@localhost ~]# iptables -t nat -L PREROUTIN( Chain PREROUTING (policy ACCEPT) prot opt source des num target

#### iptables的NAT设置

1、NAT解决什么问题?

NAT分为DNAT和SNAT

2、DNAT是从公网访问NAT设备后面的内网设备,在NAT上把目的IP地址 修改目的IP地址可以根据目的IP或者目的端口。

根据目的IP来修改的场景是:

路由器后面有一个主控板,把对路由器的访问都转到主控板, 根据目的端口来修改的场景是:

主控板后面有一组业务板卡(主控板与它们是背板直通),在主控板上根据目的端口,映射到不同的业务板卡上,映射的

3、SNAT从内网访问公网,在NAT对源IP地址修改,修改为NAT的公网II

### iptables的常见问题

#### 1、第一个问题:

对于12U设备,主控板对外暴露外网ip地址eth0:1为10.22.4.20dmu业务板的内网ip地址eth0为192.168.1.160,在主控板设置主控板eth0和业务板eth0内网连接,不是网线直连或者说同一个3而是做在硬件上,背板直通,通过主控板上的交换芯片连在一起。

(注意:/proc/sys/net/ipv4/ip\_forward必须设置为1) 我现在从10.36.65.80去连接10.22.4.201:9820,是可以成功 现在我从主控板10.22.4.201上去连接10.22.4.201:9820,能

- 2、不能成功。为什么?
  - 从外部来的请求(源IP不是10.22.4.201),路由怎么走之前,先也就是说,这里有一个路由判断的过程,而从主控板10.22.4.20至在IP层短路,直接放入IP的接收队列中,不存在路由判断过程,没
- 3、现在考虑,从内网,比如vru的板卡192.168.1.163,去连接10.2 源IP是192.168.1.163,目的IP是10.22.4.201,因此需要路 从192.168.1.163到10.22.4.201:9820有两个过程,SNAT和I 然后,SANT在路由后决定,源IP修改为10.22.4.201,现在报文
- 4、也就是说,这里外部来的请求是指,可以认为除了当前主控板(10.2 包括外网来的请求和自己内部业务板卡来的请求。因为内网来的请定 怎么理解?因为源IP不影响选路,在选路后,再修改源IP。

因此,业务板卡可以去连接主控板的外网地址[10.22.4.201],ì

5、第二个问题:

当前主机A为10.36.65.80,主机B为10.36.65.81,主机C为10在主机B设置DNAT(根据端口进行映射,目标网络地址转换),主想一想,DNAT能否实现上述的功能。

6、答案是不能,为什么?

考虑,主机A连接主机B的1734端口,主机B映射到主机C的1733,现在请求发给了C,请求流程是可以的。但是回复流程走不通,因为主机C直接回复给主机A,也就是说请求流程是A-->B-->C,但是区这显然不对。

那么,你会说,我把主机C的网关设置为主机B,强制回复流程为C-这也是不行的,因为只有跨网络的时候,才使用网关,主机C到主机

#### iptables连接和抓包的问题

- 1、考虑主机A为NAT(外网172.16.3.150,内网192.168.3.150),
- 2、在主机A设置iptables,从主机B连接外网主机C(10.36.65.80) 在主机B可以看到连接,是到10.36.65.80,如下:

[root@localhost ~]# netstat -anp|grep 1237
tcp 0 192.168.3.151:37825

从主机C也可以看到连接,连接是来自主机A的外网172.16.3.15(C:\Users\25697>netstat -ano|findstr 1237

TCP 10.36.65.80:1237 0.0.0.0:0

TCP 10.36.65.80:1237 172.16.3.150:37825

特别注意:在主机A上看不到连接信息。

3、现在抓包来分析,如下:

从主机B可以看到192.168.3.151到10.36.65.80的数据包

从主机C可以看到172.16.3.150到10.36.65.80的数据包

特别注意:在主机A上可以看到,两份数据包,是重复的。 一份是是192.168.3.151到10.36.65.80,另一份是172.16.3 为什么? 因为NAT在网卡之间进行转发,这些数据都被抓到。

#### NAT与路由器的区别

1、NAT和路由器都是技术手段,为了解决某个问题。它们分别解决什么

2、NAT本来是为了解决IP地址不够的问题,把一组设备封装一下,对外 封装的口子就是NAT设备,这就产生问题,内部设备怎么和外部设备

3、这需要在NAT设备上使用iptables设置DNAT和SNAT

对于外部到NAT设备的请求,根据目的IP地址(也就是NAT的外网] 我们在设备A上面部署所有的服务。有时候,服务需要部署到不同的根据目的端口映射到不同的内网设备上。

如果从内网访问公网,需要在NAT设备上,把源IP地址修改为NATE然后NAT转给对应的内网设备。

注意:NAT只需要设置请求的映射,回复的映射NAT会自动处理。

- 4、路由器是为了连接不同的网络,本质上是架构逻辑,不分什么公网私也就是从一个节点到一个节点,怎么走。这个路径往往是动态计算计算出一条新的路径,这条新路径往往是最优的。
- 5、简单总结就是:转换和转发。

NAT会根据需要转换源IP或者目的IP。

而路由器只是当前节点记录到另一个节点,应该经过哪个节点,也是转发的过程中,源和目的IP地址不会变化,确定下一跳的mac地址也就是,网络层源和目的IP不变,链路层mac地址发生变化,就是是

6、现实的场景往往是:

局域网内有多个小网络,每个小网络关联一个自己的路由器,路由当去连接公网的时候,有一个专门的NAT负责与外部公网通信。

在公网上,又会有多个路由器连接,内部来的请求,经过NAT发到公

- 7、NAT和路由器的功能有重合,都能实现从一个网络访问另一个网络, PC1<-->R1<-->R2<-->PC2
  - 如果只是路由,PC1要和PC2通信,PC1的网关设置为R1,PC2的网R1记录到PC2的报文,要经过R2,同理,R2记录到PC1的报文,要PC1发报文给PC2,源IP是PC1,目的IP是PC2,整个转发过程中指首先要发给R1,这个时候目的mac地址就是R1,当R1转发给R2的即但是,如果在R1上设置了NAT,当PC1发送报文经过R1出来的时候当R2回复的时候,看到的是R1的IP地址,直接发给R1。因为不是结果
- 8、之所以很容易把NAT和路由器搞混淆,是因为目前的路由器具备很多。
- 9、NAT是在路由器上增加了转换功能,因此NAT肯定是路由器,但路由

#### 一个iptables示例

- 1、这里-i,-o后面跟的是网络接口,也就是eth0,或者eth1
- 2、注意:出去和进来的转换时机是不同的。

出去是修改源IP,源IP不影响选路,先确定路由怎么走,再修改源 进来是修改目的IP,目的IP确定了路由选择,因此先把目的IP改筑

3、有时候把iptables的设置脚本,从Windows主机上传到Linux主机原因是,上传过程中有某些不可见的无效字符,解决办法是:在Li

```
#!/bin/sh -x

case $1 in
start)
echo "start iptables server"

#clear nat rules
iptables -t nat -F

#out
iptables -t nat -A POSTROUTING -s 192.168.250.0/24 -c

#in
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
```

```
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p tcp
iptables -t nat -A PREROUTING -d 10.1.208.200 -p udp
# 下面一行特别关键,默认ip_forward取0,是禁止数据包转发的,要
echo "1" >/proc/sys/net/ipv4/ip_forward
;;
stop)
echo "stop iptables server"
iptables -t nat -F
;;
```

\*)
echo "Usage: \$0" {start|stop}

#### 以linux主机为跳板登录windows主机

- 1、考虑下面的场景:现场有linux主机和windows主机,二者内网通过也就是说,在外网只能访问linux主机。那么从外网如何访问winc
- 2、现在考虑其他的场景:

esac

两台设备都是linux: 在其中一台linux主机上通过ssh登录另一两台设备都是windows:在其中一台windows主机上通过远程桌面一台windows,一台linux,windows暴露外网地址:在windows

- 3、现在的问题是:一台windows,一台linux,linux暴露外网地址,
- 4、使用iptables进行目标地址转换(DNAT),对linux主机的3389i
- 5、windows远程桌面服务的监听端口是 3389,内网直连的ip地址分别

linux: 192.167.1.108

windows: 192.167.1.109

在linux主机设置DNAT,如下:

iptables -t nat -A PREROUTING -p tcp --dport 同时在windows主机设置默认网关为 192.167.1.108

- 6、这个时候远程桌面linux主机,即进入到windows主机。
- 7、特别注意,这种情况下,在远端,linux主机,windows主机查看te在远端: TCP 10.36.65.80:40542 172.

在linux主机: 没有连接,因为进行了转换

在windows主机: TCP 192.167.1.109:3389 10.36

8、另外注意:根据端口转换,端口可以不一样。如下:

iptables -t nat -A PREROUTING -d 10.22.4.201 -p t 这个时候查看tcp连接,分别是:

在远端: TCP 10.36.65.80:29315 10.2

跳板linux主机: 没有连接,因为进行了转换

目标linux主机: tcp 0 296 192.168.1.152: 从10.22.4.201:1734 到 192.168.1.152:1733 就是转换的

#### 多网卡网线直连网络异常

1、Linux两个网卡,子网掩码不一样,但是前缀一样,导致网络异常。 eth0 192.168.1.108 255.255.0.0 eth1 192.168.1.213 255.255.255.0

当前主机的网络设置为192.168.1.109 255.255.0.0, 然后与

2、ping程序使用网络层协议ICMP(Internet控制消息协议),网线i 首先从109到达108是可以的,问题出在从108回来的路上。查看L

[root@localhost IBP]# route -n

Kernel IP routing table

 Destination
 Gateway
 Genmask

 192.168.1.0
 0.0.0.0
 255.255.255.0

 192.168.0.0
 0.0.0.0
 255.255.0.0

问题是:路由匹配是最长匹配,也就是先找小范围,再找大范围, 108主机和eth0直连,当然收不到eth1的回复。

- 3、怎么验证这个问题?
- 4、在Linux将eth1关闭,使路由从eth0发出数据,网络正常。如下: ifconfig eth1 down
- 5、增加一条更加匹配的路由,使路由从eth0发出数据,网络正常。如7 route add -net 192.168.1.109 netmask 255.255.255.
- 6、特别说明,正常情况下,多网卡不应该这么配置,应该是子网掩码不

#### 参见

#### 理解iptables

1、解决什么问题?

对网络数据包进行管理,包括状态跟踪,标记,地址转换,过滤等。

2、怎么解决?

iptables使用规则管理数据包,规则就是模式/行为的概念,模式针对iptables就是,检查一个数据包是否满足某个条件,如果满足

3、可以从不同的角度对规则进行分组。从功能的角度来看,分为:

raw表: 确定是否对该数据包进行状态跟踪

mangle表: 为数据包设置标记

nat表: 修改数据包中的源、目标IP地址或端口

filter表: 确定是否放行该数据包(过滤)

4、根据处理的时机,也就是从处理流程的角度来看,分为5个节点(也家

PREROUTING 路由前

INPUT 数据包流入口

FORWARD 转发管卡

OUTPUT 数据包出口 POSTROUTING: 数由后

注意:每个节点所拥有的功能是不一样的,如下:

PREROUTING mangle, nat
INPUT mangle, filter
FORWARD mangle, filter

OUTPUT mangle, nat, filter

POSTROUTING: mangle, nat

从另外一个角度看,列出每个功能可以在那些节点使用。如下: [root@localhost dll3]# iptables -t filter --list

Chain INPUT (policy ACCEPT)

target prot opt source destinat DROP icmp -- anywhere anywhere

Chain FORWARD (policy ACCEPT)

target prot opt source destinat

Chain OUTPUT (policy ACCEPT)

target prot opt source destinat

5、Iptables命令的语法格式:

6、禁止10.36.65.80连接本机的tcp端口9000,如下:

[root@localhost dll3]# iptables -t filter -A INPl
[root@localhost dll3]# service iptables status

Table: filter

Chain INPUT (policy ACCEPT)

num target prot opt source des 1 DROP tcp -- 0.0.0.0/0 0.6

Chain FORWARD (policy ACCEPT)

num target prot opt source des

Chain OUTPUT (policy ACCEPT)

num target prot opt source des

注意:这里的policy有两种情况:

- ACCEPT:是黑名单策略,假定都能做,没说不能做,就都能DROP: 是白名单策略,假定都不能做,说了你能做,你不
- 7、我能够ping通10.36.65.80,但是不让10.36.65.80 ping通我。 ping的实现是使用icmp协议,请求是8,回复是0,不让10.36.6 在入口禁掉请求(类型是8),或者在出口禁掉回复(类型是0), iptables -t filter -A INPUT -p icmp --icmp-type { iptables -t filter -A OUTPUT -p icmp --icmp-type
- 8、反过来一下,让10.36.65.80能够ping通我,但是我不能ping通1 办法是:在出口禁掉请求(类型是8),或者在入口禁掉回复(类型 iptables -t filter -A OUTPUT -p icmp --icmp-type iptables -t filter -A INPUT -p icmp --icmp-type (
- 9、注意:我们系统的iptables设置脚本在 /usr/local/appfs/bi

#### 设置了DNAT但是telnet不通

- 1、场景如下:两台设备A和B,其中设备A开启外网172.16.2.29,设备设备A的内网192.167.1.108,设备B的内网192.167.1.109
- 2、设备B在端口9812监听,现在从外网172.16.2.101 去连接端口98 也就是DNAT(Destination Network Address Translatio iptables -t nat -A PREROUTING -p tcp --dport 9812
- 3、现在从172.16.2.101 进行 telnet 172.16.2.29 9812 发现 抓包发现,发送了sync同步包,但是没有收到ack回复,tcp发送
- 4、进一步定位发现,同步包sync从172.16.2.101发到了172.16.2. 转发给了 192.167.1.109,但是192.167.1.109 没有发送ac
- 5、问题也就是,192.167.1.109 没有把包发送出来。查看路由表,如 [root@localhost ~]# route

Kernel IP routing table

Destination Gateway Genmask F 192.167.1.0 \* 255.255.255.0

6、原因找到了,也就是从192.167.1.109 向172.16.2.101 回复a 到目标网路192.167.1.0的路由表,因此需要增加默认路由,如了 [root@localhost ~]# route add default gw 192.167. [root@localhost ~]# route -n

Kernel IP routing table

Destination	Gateway	Genmask	F
192.167.1.0	0.0.0.0	255.255.255.0	Į
0	102 167 1 108	0000	- 1

然后telnet成功。

7、注意:删除路由表,如下:

route del default gw 192.167.1.108

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

Linux内核版本

Linux发行版本

## 详细描述

## 变量说明

#### Linux内核版本

Linux版本分为内核版本和发行版本(distribution),我们通常说能查看内核版本的方法有:

- 1, uname -a
- 2、cat /proc/version

#### 参见

#### Linux发行版本

Linux发行版本很多,常用的有:RedHat, CentOS, SuSE, Debian RedHat企业版是收费的, CentOS可以认为是RedHat免费的源代码编译查看发行版本的方法有:

- 1、lsb\_release -a // lsb是 linux standard base
- 2、cat /etc/issue
- 3、对于redhat发行版,可以使用 more /etc/redhat-release rpm -qa|grep -i redhat-release
- 4、注:对于任何一个程序,使用file 也可以查看相关信息 file /bin/bash file /bin/tar

#### 参见

# Andy Niu Help 1.0.0.0



# 重变

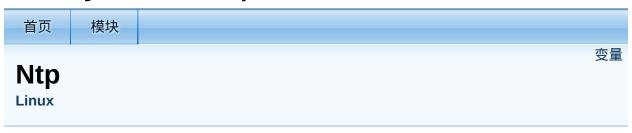
## nm命令

## 变量说明

#### nm命令

```
1、示例代码
   #include <stdio.h>
   void myprint()
    {
       printf("hello world");
   }
   int main(int argc, char* argv[])
    {
       myprint();
       return 0;
2\ niuzibin@ubuntu:~/work/test1/nm$ nm main
3、nm -u main
   显示main 中的未定义符号,需要和其他对象文件进行链接,如下
   niuzibin@ubuntu:~/work/test1/nm$ nm -u main
                w __gmon_start___
                w _ITM_deregisterTMCloneTable
                w _ITM_registerTMCloneTable
                w _Jv_RegisterClasses
                U __libc_start_main@@GLIBC_2.2.5
                U printf@@GLIBC_2.2.5
```

# Andy Niu Help 1.0.0.0



# 重变

ntp介绍

ntpdate报错

## 变量说明

#### ntpdate报错

```
1、在172.16.2.16正常开启ntp服务,
   [root@localhost ~]# ntpdate 172.16.2.16
   29 Dec 14:52:38 ntpdate[9002]: no server suitable
2、当用ntpdate -d 来查询时会发现导致 no server suitable f
3、错误一:Server dropped: Strata too high
   在ntp客户端运行ntpdate serverIP,出现no server suita
   在ntp客户端用ntpdate -d serverIP查看,发现有"Server
   这是因为NTP server还没有和它的server,或者其自身同步上。
   通过ntpstat可以看到如下:
   [root@localhost ~]# ntpstat
   unsynchronised
   time server re-starting
   poll interval unknown
   解决办法:是让NTP Server和其自身保持同步,如果在/ntp.co
   server 127.127.1.0
   fudge 127.127.1.0 stratum 8
4、错误二:Server dropped: no data
   从客户端执行netdate -d时有错误信息如下:
   transmit(192.168.30.22)
   transmit(192.168.30.22)
   transmit(192.168.30.22)
   transmit(192.168.30.22)
   transmit(192.168.30.22)
   192.168.30.22: Server dropped: no data
   server 192.168.30.22, port 123
```

28 Jul 17:42:24 ntpdate[14148]: no server suitab]

出现这个问题的原因可能有2:

a、检查ntp的版本,如果你使用的是ntp4.2(包括4.2)之后的原 使用以下命令检查ntp的版本:

# ntpq -c version

下面是来自ntp官方网站的说明:

The behavior of notrust changed between versi In 4.1 (and earlier) notrust meant "Don't tru In 4.2 (and later) notrust means "Ignore all 解决:

把notrust去掉。

b、检查ntp server的防火墙。可能是server的防火墙屏蔽了up 可以用命令

service iptables stop

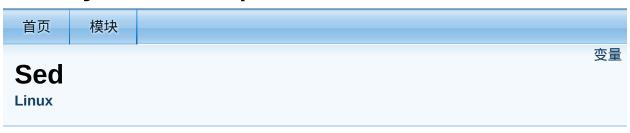
来关掉iptables服务后再尝试从ntp客户端的同步,如果成

#### ntp介绍

- 1、Network Time Protocol (NTP),解决什么问题? 考虑下面的需求,一个局域网内要求每台机器的时间完全一致。你以后时间也肯定会保持一致。这种说法是错误的,因为计算机运行时分的时候大家时间一致,运行一段时间,每个机器的时间可能都是此需要一个电脑时间,作为基准的时间。
- 2、真正能够精确地测算时间的还是原子钟,但由于原子钟十分的昂贵, 解决办法就是,局域网内布置一台设备A,去和原子钟机器进行时能
- 3、这就需要使用NTP服务,原子钟机器开启NTP服务(ntpd),设备Ai 然后开启NTP客户端(ntpd),与原子钟时间同步。
- 4、原子钟的ntpd作为服务端,设备A的ntpd作为客户端,对时间进行证
- 5、ntpdate进行时间校准,设备A上面的ntpdate与ntpd使用的场景7 ntpdate进行时间校准,ntpd进行时间微调。 时间同步的时候,先进行ntpdate,再进行ntpd
- 6、如果没有使用ntpdate,直接使用ntpd,可能时间差距很大,导致行
- 7、使用ntpdate还会导致一个问题,事件发生跳变。比如当前时间为1章 导致问题:日志的记录时间前面是12点,后面是8点。
- 8、时间同步使用udp协议,端口是123, ntpdate只有手动执行一次, 而ntpd要一直与原子钟的ntpd交互,进行微调,因此一直占用12:

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 量变

原理

替换

提取

匹配优先的问题

## 详细描述

## 变量说明

#### 匹配优先的问题

- 1、考虑下面的情况,对于 "Tom" is short for "Thomas" 我想提取第一个名称,即Tom
- 2、容易想到的办法是:
  - echo \"Tom\" is short for \"Thomas\" | sed -r "s/ 结果却是:Tom" is short for "Thomas
- 3、为什么?原因是 .\* 是匹配优先,即贪婪模式,也就是说会尽量多吃
- 4、怎么解决?使用忽略优先,也就是告诉正则表达式引擎,尽量少吃。
- 5、但是, sed并不支持忽略优先,怎么办? 可以尽量多吃,但是吃的内容不能包含",这种策略吃到第一个就像 echo \"Tom\" is short for \"Thomas\" | sed -r "s/

#### 参见

#### 原理

- 1、sed读取一行到模式空间,判断是否满足条件【地址行数或者匹配包于否则不处理。如果没有条件,都处理。
- 2、读取下一行之前,把模式空间的内容输出到屏幕。如果确定哪些行不使得接下来的输出为空。如果确定哪些行要显示,其它的行不显示-n为安静模式,也就是说,读取下一行之前,不在输出当前行的内
- 3、sed有三层循环:

第一层遍历所有的行

第二层遍历一组模式/行为

第三层遍历行为中的一组命令

#### 参见

#### 提取

直接使用sed,查找满足条件的一行,替换,然后打印 sed -nr '/(server-id=)/{s/(server-id=)(.\*)/\2/;p}' my

另一种办法:使用管道,过滤出一行,替换 cat mysql.cnf|grep 'server-id='|sed -r 's/(server-id=

#### 参见

#### 替换

将 server-id=78 替换为 server-id=66,如下: sed -r 's/(server-id=)([0-9]+)/\166/' -i mysql.cnf

#### 注:

-r表示支持正则表达式

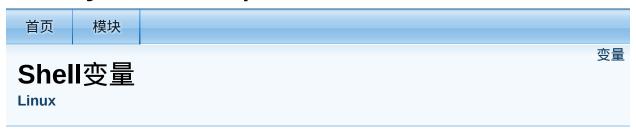
\166 表示 找到(server-id=)([0-9]+),两个分组,反向引用\1, 提取出server-id=,再加上66,也就是相当于用66替换原来的数字 -i 直接修改文件

上面的替换只是,替换行中的第一个,如果要替换行中的每一个,使用 sed -r 's/(server-id=)([0-9]+)/\166/g' -i mysql.cnf

#### 参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

变量的设置

变量内容的操作

变量的有效范围

## 变量说明

#### 变量内容的操作

```
[root@localhost log]# var=/dir1/dir2/file.txt
1、对变量取值
   [root@localhost log]# echo ${var}
   /dir1/dir2/file.txt
2、求字符串的长度
   [root@localhost log]# echo ${#var}
   19
3、提取位置3到最后的字符
   [root@localhost log]# echo ${var:3}
   r1/dir2/file.txt
4、从位置3,开始提取5个字符
   [root@localhost log]# echo ${var:3:5}
   r1/di
5、从左边开始,删除最短匹配r,这里的*表示删除
   [root@localhost log]# echo ${var#*r}
   1/dir2/file.txt
6、从左边开始,删除最短匹配dir
   [root@localhost log]# echo ${var#*dir}
   1/dir2/file.txt
7、从左边开始,删除最短匹配数字,支持正则表达式
   [root@localhost log]# echo ${var#*[0-9]}
   /dir2/file.txt
8、从左边开始,删除最长匹配di,这里的*表示删除
   [root@localhost log]# echo ${var##*di}
   r2/file.txt
9、从右边开始,删除最短匹配di,注意:这里的*在后面。 #在$的左ì
```

[root@localhost log]# echo \${var%di\*}
/dir1/

- 10、从右边开始,删除最长匹配di [root@localhost log]# echo \${var%%di\*} /
- 11、用aa替换第一个匹配的di [root@localhost log]# echo \${var/di/aa} /aar1/dir2/file.txt
- 12、用bbb替换所有匹配的dir [root@localhost log]# echo \${var//dir/bbb} /bbb1/bbb2/file.txt
- 13、用bbb替换前缀/dir,注意:/本身需要使用转义字符 [root@localhost log]# echo \${var/#\/dir/bbb} bbb1/dir2/file.txt
- 14、用bbb替换前缀le,注意:/本身需要使用转义字符 [root@localhost log]# echo \${var/%\le\*.txt/bbb} /dir1/dir2/fibbb
- 15、用bbb替换后缀2\*.txt,支持正则表达式 [root@localhost log]# echo \${var/%2\*.txt/bbb} /dir1/dirbbb

#### 变量的有效范围

- 1、变量的范围分为环境变量和自定义变量,也可以认为是全局变量和局
- 2、当前进程name为Andy Niu,进入子进程,子进程只会继承父进程的 也就是说,在子进程看不到name,离开子进程,又可以看到name。 使用export,把当前进程的自定义变量name导出,作为环境变量 export name
- 3、export的原理是:
  - a、启动一个shell为A,操作系统分配一块内存给shell使用,可
  - b、A执行export,相当于把自定义变量写到上述的内存块中,作为
  - c、启动一个子shell为B, B会将A内存块中的环境变量导入自己的
- 4、env查看环境变量, set查看环境变量和自定义变量。

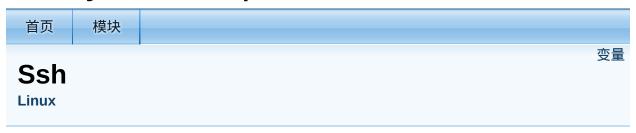
#### 变量的设置

```
[root@localhost test]# echo $name
Andy Niu
[root@localhost test]# name=$name:'Bill Lee' //累加
[root@localhost test]# echo $name
Andy Niu:Bill Lee
[root@localhost test]# unset name //取消
[root@localhost test]# echo $name

注意:一般情况下,系统默认变量为大写,用户自定义变量为小写。
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

ssh介绍

ssh登录

ssh登录错误

## 变量说明

#### ssh介绍

- 1、shell是对Linux内核的封装,对外暴露接口。用户调用shell脚本
- 2、考虑下面的场景,我想要远程调用shell脚本,怎么办?
- 3、这就是ssh, Secure Shell, ssh是客户端/服务端模式,使用tcp服务端是sshd进程,客户端软件很多,客户端向sshd服务器发送ishhd是守护进程(daemon),默认在22端口监听,请求到来的时ssh和mysql服务的情况类似。
- 4、为什么叫Secure Shell? 因为ssh的交互过程是加密的。
- 5、在本地机器上,使用ssh进入到A机器,在A机器上,通过ssh可以再使用exit,退出当前shell
- 6、使用SecureCRT可以建立多个ssh连接,如果显示器连接Linux主机 使用Alt+F1 Alt+F2 Alt+F3 Alt+F4 Alt+F5

## ssh登录

- 1、ssh解决什么问题?
  - 考虑下面的情况,主备两台机器A,B,通过内网直连,对外只暴露-设备A外网激活,设备B外网处于失效状态,我在外网,怎么访问设
- 2、解决办法是从外网进入设备A,再以设备A为跳板,使用ssh,通过内ssh root@192.168.1.101 -p1733

## ssh登录错误

1、登录的时候,用户名和密码都是正确的,但是登录报错,具体的错误 The server has disconnected with an error. Serve A protocol error occurred. Change of username or 2、错误原因:

是因为新建会话的时候,用户名输入错误,然后在登录的时候,即<sub>3</sub>3、解决办法:

修改会话选项中的用户名,使用正确的用户名。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

如何解决apt get中Unmet\_dependencies问题

安装ssh

## 详细描述

## 变量说明

## 如何解决apt get中Unmet\_dependencies问题

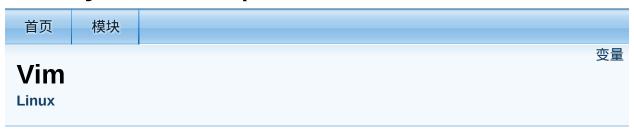
```
sudo apt --fix-broken install
sudo apt-get update
sudo apt-get upgrade
```

## 安装ssh

apt-get install openssh-server

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

<u> </u>		
查找		
±±+4		
替换		
区域选择		
行号		
删除、复制、粘贴		
转移		
撤销与重做		
magic		
正则表达式		

## 详细描述

## 变量说明

## magic

- 1、在vim中,对于正则表达式,+表示+字符本身,要表示匹配1个或多
- 2、这与正则表达式的使用习惯不一致,在正则表达式中,+表示匹配1个
- 3、怎么解决这个问题,使用:set magic,就和正则表达式的使用习情
- 4、使用 :set nomagic 切换回去到vim的默认模式。

### 参见

## 删除、复制、粘贴

- 1、小写x相当于del,大写X相当于backspace
- 2、dd删除当前行,也是剪切当前行

:1,5d 删除第1行到第5行 :1,\$d 删除第1行到最后一行,也就是所有的行

5dd 当前位置向下删除5行

- 3、yy复制当前行,5yy从当前行向下复制5行
- 4、小写p在下一行粘贴,大写P在上一行粘贴
- 5、复制粘贴是 yy p 剪切粘贴是 dd p

## 参见

## 区域选择

- 1、小写v:字符选择
- 2、大写V【shift+v】: 行选择

- 3、ctrl+v:矩形选择
- 4、在区域选择的情况,d删除选择的区域,y复制选择的区域,小写p在

#### 参见

## 撤销与重做

- 1、u 撤销
- 2、Ctrl+r 重做

### 参见

#### 替换

- 1、当前行第一个111替换为222,:s/111/222/
- 2、当前行所有的111替换为222,:s/111/222/g
- 3、表示范围从1行到5行,:1,5xxx,\$表示最后一行,xxx表示 s/11
- 4、整个文档,:%xxx
- 5、反向引用,将ab替换为ba,:%s/\(a\)\(b\)/\2\1/g (注:小
- 6、利用反向引用的特点,可以统计单词出现的个数,如下:
  - :%s/ab/&/g 其中&反向引用前面的单词ab,也就是说替换并没有按下Enter会显示替换了多少个,3 substitutions on 3 lir如果不显示替换了多少个,按下u会显示撤销了几个变化,3 chan
- 7、删除行首的空白字符(包括空格符和制表符),如下:
  - :%s/^\s\+//g
- 8、删除空行
  - :%g/^\s\*\$/d

## 参见

## 查找

- 1、查找111,/111 然后enter,n下一个,N上一个
- 2、vim支持正则表达式,如/[0-9]\+
- 3、表示出现次数:\*【0次或多次】 \+【1次或多次】 \?或者\=【0次

4、vim默认是区分大小写的,怎么不区分大小写呢?
:set ic 【:set ignorecase 】 不区分大小写:set noic【:set noignorecase】 区分大小写上面的设置会导致全局起效,如果只想当前搜索起效,使用如下:/hau\c 小写c,当前搜索不区分大小写,等价于 /\chau/hau\C 大写C,当前搜索区分大小写,等价于 /\chau/chau
5、搜索结果高亮显示:set hls 【:set hlsearch 】 高亮显示:set nohls【:set nohlsearch】 取消高亮显示注:只有vim才有这个效果,vi没有这个效果

### 参见

### 正则表达式

```
1、量词
                        意义
   vim
             Perl
                        ⊙个或多个(匹配优先)
                        1个或多个(匹配优先)
   \+
             +
                        0个或1个(匹配优先)
   \?或\=
             ?
                        n个到m个(匹配优先)
   \{n,m\}
             {n,m}
                        最少n个(匹配优先)
   \n,
             {n,}
                        最多m个(匹配优先)
   \setminus \{, m\}
             {,m}
   \n
             {n}
                        恰好n个
                        n个到m个(忽略优先)
   \{-n,m}
             {n,m}?
   \{-}
             *?
                        ⊙个或多个(忽略优先)
                        1个或多个(忽略优先)
   \{-1,}
             +?
             ??
   \{-,1}
                        0个或1个(忽略优先)
2、环视、固化分组、分组不捕获
             Perl
   vim
                        意义
             (?=
   \@=
                        顺序环视
              (?!
   \@!
                        顺序否定环视
   \@<=
              (?<=
                        逆序环视
             (?<!
   \@<!
                        逆序否定环视
                        固化分组
   \@>
              (?>
                        非捕获型括号
              (?:
   \%(atom\)
```

## 参见

## 行号

1、显示行号 :set nu 全写 :set number 2、取消行号 :set nonu 全写 :set nonumber

## 参见

## 转移

1、0转到行头

2、\$转到行尾

3、:15 转到第15行

## 参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

自页 模块 启动关机 Linux

## 重变

启动

关机重启

运行级别

## 变量说明

### 关机重启

1、reboot是重启,平时用的最多。

2、注意:关机之前通过下面的命令,查看系统的使用状态

谁在线: who

联网状态: netstat -a 后台执行的程序: ps -aux

3、shutdown命令

-t:后面跟着秒数,比如-t 10,表示发送关机警告信息后,等待注意:真正的关机时间取决于必选项时间,这个时间单位是分

shutdown now # 立即关机

shutdown +2 # 2min 后关机

shutdown 10:01 # 10:01关机

shutdown +2 "The machine will shutdown" # 2mi

-h:halt,将系统关机

-r:重启

-k:只发关机的警告信息,但并不关机

4、halt相当于 shutdown -h now

5、poweroff是halt的软连接,如下:

[root@localhost ~]# 11 /sbin/poweroff

lrwxrwxrwx 1 root root 4 Dec 28 2011 /sbin/power

6、使用init, init是所有进程的祖先, 他是Linux系统操作中不可缺定的进程号始终为1, 通过init可以切换系统的运行等级, 如下:

run level 0: 关机

run level 1: 单用户模式,系统维护

run level 3: 纯文本模式

run level 5: 含图形接口模式

run level 6: 重新启动

#### 参见

### 启动

1、启动执行过程:/etc/rc.d/rc.sysinit,然后 /etc/rc.d/rc

2、在我们的系统中是,

rc.sysinit 调用 sh /usr/local/configfs/config.sh config.sh 执行 \cp -fr /usr/local/configfs/etc/\*

## 参见

#### 运行级别

1、Linux系统有7个运行级别(runlevel)

0:系统停机状态,系统默认运行级别不能设为0,否则不能正常启

1:单用户工作状态,root权限,用于系统维护,禁止远程登陆

2:多用户状态(没有NFS)

3:完全的多用户状态(有NFS), 登陆后进入控制台命令行模式

4:系统未使用,保留

5:X11控制台,登陆后进入图形GUI模式

6:系统正常关闭并重启,默认运行级别不能设为6,否则不能正常

2、runlevel可以查看当前运行级别,如下:

[root@localhost ~]# runlevel
N 5

3、在/etc/rc.d目录下面有下面一些目录,如下:

[root@localhost rc.d]# 11
total 108

drwxr-xr-x 2 root root 4096 Apr 16 11:07 init.d

-rwxr-xr-x 1 root root 2255 Jul 4 2009 rc

drwxr-xr-x 2 root root 4096 Apr 16 10:16 rc0.d

drwxr-xr-x 2 root root 4096 Apr 16 10:16 rc1.d

drwxr-xr-x 2 root root 4096 Apr 16 10:16 rc2.d drwxr-xr-x 2 root root 4096 Apr 16 10:16 rc3.d

drwxr-xr-x 2 root root 4096 Apr 16 10:16 rc4.d

drwxr-xr-x 2 root root 4096 Apr 16 10:16 rc5.d drwxr-xr-x 2 root root 4096 Apr 16 10:16 rc6.d -rwxr-xr-x 1 root root 347 Sep 24 11:30 rc.locaterwxr-xr-x 1 root root 27239 Jul 4 2009 rc.sysi

- 4、init.d目录下面是service脚本文件,而rcN.d中的文件都是连接 这些连接文件的命名规则为K+nn+服务名或S+nn+服务名,系统根 对于K开头的,终止对应的服务 对于S开头的,启动对应的服务
- 5、在文件/etc/inittab中,保存了系统默认的运行级别,如下: id:5:initdefault:

注意:这里的运行级别不能是6,如果是6,导致关闭重启,关闭重

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

常用的服务 Linux

## 重变

查看所有的服务

网络存储有关

## 详细描述

## 变量说明

### 查看所有的服务

- 1, netstat -tunlp
- 2、service --status-all
- 3、chkconfig --list

### 网络存储有关

- 1、我们知道进程是直接读写内存的,其他的数据是放在硬盘上的。存在
- 2、在设备上的硬盘插槽里添加硬盘。存在问题,插槽个数有限,硬盘还
- 3、解决办法是:通过网络访问,把其它设备的文件系统当成本地文件进 这就是NAS(Network Attached Storage 网络附加存储), 注意:这和ftp有本质的区别,ftp是交互式,读写权限,副本。 比如:对于一个很大的文件,只修改一行。ftp需要通过网络从远和 而NAS,可以直接修改,给人感觉就是修改本地的文件,实际上是是 因此,ftp只适合做下载用。
- 4、对于NAS,在linux上的实现是NFS(Network File System), CIFS可以简单理解为windows上的共享文件夹。 在linux上挂载windows的共享文件夹就是 mount -t cifs ... 在windows上挂载其它windows的共享文件夹,操作如下: Windows资源管理器-->工具-->映射网络驱动器,选择本地驱动管 然后本地就会多出一个目录,查看属性,可以看到文件系统是NTFS 特别注意:这个NTFS是New Technology File System,是一部 那么问题来了,如果在windows上访问linux的目录,就像访问其 这就要使用Samba
- 5、把局域网内一个设备上的目录当成本地目录来使用,还是存在问题, 这就需要SAN (Storage Area Networks 储存局域网络),也就

和NAS对照, NAS是挂载某个特定设备的目录。

SAN会搜索局域网内所有设备暴露出来的硬盘,挂载SAN会多出一个6、对于SAN,实现是iSCSI(internet SCSI),SCSI是Small Co可以理解为使用数据线直连存储器的通用标准接口,而iSCSI是基iSCSI是客户/服务器模式,服务端作为iSCSI target,也就是i注意:在iSCSI中作为客户端的iSCSI initiator,往往是作为。也就是,对多个设备的硬盘,进行融合,从新暴露出来,供其它客

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 **常见问题** Linux

## 变量

打包deb包 运行缺少库 shell如何输入制表符 Linux换行符 linux下printf函数输出不及时 获取脚本所在的绝对路径 Linux环境常用库的介绍 linux进程通信 发送信号的函数 虚拟内存和常驻内存 Linux文件命名 **system**与**popen** tty\_\_pty\_\_pts\_\_ptmx linux提示符设置 Linux统计文件、文件夹的个数 linux查看历史操作记录 linux下有关的目录

## 变量说明

## linux下printf函数输出不及时

```
1、考虑下面的代码:
   #include <stdio.h>
   int main(int argc, char* argv[])
   {
       printf("hello ");
       printf("world");
       getchar();
       return 0;
2、在windows下面没有问题,正常输出。但是在linux下面, printfi
3、原因是:在linux系统下, printf函数是行缓冲式的输出, 当prin
   才会将缓冲区里的内容刷新到标准输出(stdout).
4、怎么解决这个问题?
   a、输出的结尾加上\n
   b、或者在printf后面,调用 fflush(stdout)
   如下:
   #include <stdio.h>
   int main(int argc, char* argv[])
   {
          printf("hello \n");
          printf("world");
          fflush(stdout);
          getchar();
          return 0;
5、奇怪的是,调用过一次 fflush(stdout); 之后,以后都会实时刷
```

### linux下有关的目录

/ : this is root directory root 用户根目录

/bin : commands in this dir are all system insta /sbin : commands in this dir are all system insta /usr/bin: user commands for applications 后期安装的 /usr/sbin: super user commands for applications

主目录:/root、/home/username

用户可执行文件:/bin、/usr/bin、/usr/local/bin 系统可执行文件:/sbin、/usr/sbin、/usr/local/sbin

其他挂载点:/media、/mnt

配置:/etc 临时文件:/tmp

内核和Bootloader:/boot 服务器数据:/var、/srv 系统信息:/proc、/sys

共享库:/lib、/usr/lib、/usr/local/lib

每个用户都拥有一个主目录。所有用户的个人文件(配置、数据甚至应用根的主目录为/root。大多数非根主目录包含在 /home 树中,通常以厚重要的二进制位于 /bin(用户二进制)以及 /sbin(系统二进制)中不重要的二进制(如图形环境或Office 工具)安装在 /usr/bin 和进行这种分隔是为了尽可能地缩小根分区。使用源代码编译的软件通常包含。

注意:usr很多人误以为是user的简写,实际上是unix system resc

## Linux换行符

- 1、空白字符包括:空格符,制表符,换行符
- 2、Windows换行符为\r\n, 使用notepad++显示所有字符,显示CRL
- 3、Linux换行符为\n, 使用notepad++显示所有字符,显示LF,使用
- 4、在Linux下面,如何去除^M?

我们知道^M这个字符就是\r,因此可以对\r进行替换 sed "s/\r//g" -i 111 或者直接对^M进行替换,需要注意的是,不能直接输入^和M,必须 sed "s/^M//g" -i 111

### 参见

### linux提示符设置

```
1、新建用户登陆后,发现命令提示符如下:
```

-bash-3.2\$

2、设置命令提示符,选项如下:

\d : 代表日期, 格式为weekday month date, 例如: "Mon A

\H:完整的主机名称。例如:我的机器名称为:fc4.linux,则

\h : 仅取主机的第一个名字,如上例,则为fc4,.linux则被省

\t :显示时间为24小时格式,如:HH:MM:SS

\T:显示时间为12小时格式

\A :显示时间为24小时格式:HH:MM

\u : 当前用户的账号名称

\v : BASH的版本信息

\w : 完整的工作目录名称。家目录会以 ~代替

\W : 利用basename取得工作目录名称,所以只会列出最后一个E

\#:下达的第几个命令

\\$:提示字符,如果是root时,提示符为:#,普通用户则为:

3、示例如下:

-bash-3.2\$ export PS1='[\u @\W]'

[niu @~]cd /home

[niu @home]

## Linux文件命名

- 1、xxx.d是目录,这里的d是directory,比如rc.d
- 2、xxxd是守护进程文件,也就是服务进程文件,这里的d是daemo,比
- 3、rc文件,这个有争议,一说是run command,一说是run control

## linux查看历史操作记录

```
1、命令是history,列出当前用户最新的10条操作记录,如下:
    [root@localhost ~]# history 10
   691* 2016-09-27 16-27-29 root:
   692* 2016-09-27 16-28-23 root:
   693
        2016-09-27 16-31-03 root: history
   694 2016-09-27 16-31-24 root: history 10
   695
        2016-09-27 16-37-58 root: history
   696
        2016-09-27 16-38-03 root: history 10
   697 2016-09-27 16-38-48 root: 11
   698
        2016-09-27 16-38-50 root: ll -a
   699
        2016-09-27 16-39-06 root: more .bash history
   700
        2016-09-27 16-39-52 root: history 10
   会列出操作者和操作时间。
2、history只能列出当前用户的操作记录,也就是说,用户A不能查看。
3、注意:连续两次相同的操作,不会重复记录,如下:
    [root@localhost dmu]# history 10
        2016-09-27 16-42-15 root: history
   702
        2016-09-27 16-45-11 root: cd /usr/local/data
   703
   704 2016-09-27 16-45-14 root: cd dmu
   705 2016-09-27 16-45-17 root: ./dmu start.sh
        2016-09-27 16-45-29 root: history 10
   706
        2016-09-27 16-45-52 root: history|grep dmu_s
   707
   708 2016-09-27 16-46-44 root: 11
   709
        2016-09-27 16-46-56 root: history
   710
        2016-09-27 16-47-03 root: ll
   711
        2016-09-27 16-47-06 root: history 10
    [root@localhost dmu]# history 10
        2016-09-27 16-42-15 root: history
   702
        2016-09-27 16-45-11 root: cd /usr/local/data
   703
   704
        2016-09-27 16-45-14 root: cd dmu
   705
        2016-09-27 16-45-17 root: ./dmu start.sh
        2016-09-27 16-45-29 root: history 10
   706
        2016-09-27 16-45-52 root: history|grep dmu_s
   707
   708
        2016-09-27 16-46-44 root: ll
   709
        2016-09-27 16-46-56 root: history
   710
        2016-09-27 16-47-03 root: 11
```

```
2016-09-27 16-47-06 root: history 10
4、根据操作序号,引用操作,如下:
    [root@localhost dmu]# history 10
    702
        2016-09-27 16-42-15 root: history
        2016-09-27 16-45-11 root: cd /usr/local/data
   703
   704 2016-09-27 16-45-14 root: cd dmu
   705
        2016-09-27 16-45-17 root: ./dmu start.sh
   706
        2016-09-27 16-45-29 root: history 10
   707 2016-09-27 16-45-52 root: history|grep dmu_s
   708 2016-09-27 16-46-44 root: 11
        2016-09-27 16-46-56 root: history
   709
   710
        2016-09-27 16-47-03 root: ll
   711
        2016-09-27 16-47-06 root: history 10
    [root@localhost dmu]# !711
   history 10
   702
        2016-09-27 16-42-15 root: history
        2016-09-27 16-45-11 root: cd /usr/local/data
   703
   704
        2016-09-27 16-45-14 root: cd dmu
   705
        2016-09-27 16-45-17 root: ./dmu start.sh
   706 2016-09-27 16-45-29 root: history 10
   707 2016-09-27 16-45-52 root: history|grep dmu_s
   708
        2016-09-27 16-46-44 root: ll
   709
        2016-09-27 16-46-56 root: history
   710
        2016-09-27 16-47-03 root: 11
        2016-09-27 16-47-06 root: history 10
   711
```

## Linux环境常用库的介绍

头文件	描述
<gl gl.h=""></gl>	实现到
<gl glu.h=""></gl>	实现到
<imlib.h></imlib.h>	实现也
无	实现材
<com_err.h></com_err.h>	实现材
<crypt.h></crypt.h>	实现材
<curses.h></curses.h>	实现材
<curses.h></curses.h>	光标字
<db.h></db.h>	<b>创建</b> 和
	<gl gl.h=""> <gl glu.h=""> <imlib.h> 无 <com_err.h> <crypt.h> <curses.h></curses.h></crypt.h></com_err.h></imlib.h></gl></gl>

libdl.so	<dlfcn.h></dlfcn.h>	让程序
libform.so	<form.h></form.h>	实现字
libglib.so	<glib.h></glib.h>	Glib
libgthread.so	<glib.h></glib.h>	实现对
libgtk.so	<gtk gtk.h=""></gtk>	GIMP <sup>-</sup>
libhistory.so	<readline readline.h=""></readline>	实现G
libjpeg.so	<jpeglib.h></jpeglib.h>	定义到
libm.so	<math.h></math.h>	实现标
libmenu.so	<menu.h></menu.h>	提供在
libncurses.so	<ncurses.h></ncurses.h>	使用n
libnss.so	<nss.h></nss.h>	用于名
libpanel.so	<panel.h></panel.h>	提供在
libpbm.so	<pbm.h></pbm.h>	可移植
libpgm.so	<pgm.h></pgm.h>	可移植
libpng.so	<png.h></png.h>	用于编
libpnm.so	<pnm.h></pnm.h>	可移植
libppm.so	<ppm.h></ppm.h>	可移植
libpthread.so	<pthread.h></pthread.h>	实现了
libreadline.so	<readline readline.h=""></readline>	GNU r
libresolv.so	<resolv.h></resolv.h>	提供佷
libslang.so	<slang.h></slang.h>	提供方
libtiff.so	<tiffio.h></tiffio.h>	读写T
libz.so	<zlib.h></zlib.h>	通用圧
libvga.so	<vga.h></vga.h>	Linux

## Linux统计文件、文件夹的个数

```
1、当前目录下文件的个数
        [niu_zibin@localhost time]$ ll|grep '^-'|wc -l
2、当前目录下文件夹的个数
        [niu_zibin@localhost time]$ ll|grep '^d'|wc -l
3、在当前目录下递归,统计文件的个数
        [niu_zibin@localhost time]$ ll -R|grep '^-'|wc -l
4、在当前目录下递归,统计文件夹的个数
        [niu_zibin@localhost time]$ ll -R|grep '^d'|wc -l
```

- 1、信号是unix中最古老的进程通信的一种方式,他是软件层次上对中型信号可以实现用户空间进程和内核空间进程的交互,内核进程可以是我们可以任何时候给进程发送信号而无需知道进程的状态,如果进程是阻塞状态,那么信号传递会被延迟,直到阻塞结束时才是
- 2、查看所有的信号,如下:

[root@localhost mysql]# kill -1

63) SIGRTMAX-1 64) SIGRTMAX

- 1) SIGHUP 2) SIGINT 3) SIGQUIT 7 6) SIGABRT 7) SIGBUS 8) SIGFPE 12) SIGUSR2 11) SIGSEGV 13) SIGPIPE 1 16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 21) SIGTTIN 22) SIGTTOU 26) SIGVTALRM 27) SIGPROF 23) SIGURG 28) SIGWINCH 34) SIGRTMIN 35) SIGRTMIN+1 31) SIGSYS 38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 4 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 5 53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4
- 3、前32种信号会有各种不同的名称,后32种会以"SIGRTMIN"或者"SI前者是从unix继承下来的信号,称为不可靠信号(也称为非实时信后者为了解决"不可靠信号"的问题进行了更改和扩充的信号形成了
- 4、如果想要了解可靠与不可靠信号,需要了解信号的生命周期:
  - 一个完整的信号周期可以分为三个重要阶段,三个重要阶段有四个! 信号产生,信号在进程中注册,信号在进程中注销,执行信号处理!
- 5、可靠信号与不可靠信号的区别:

不可靠信号如果发现信号已经在进程中注册,就会忽略该信号,因可靠信号发送给一个进程时,不管该信号是否已经在进程中注册, 所有可靠信号都支持排队,所有不可靠信号都不支持排队。

注:这里信号的产生,注册,注销等是指信号的内部的实现机制, 所以信号注册与否,与本节后面讲到的发送信号函数(kill等)以及

6、用户进程对信号的响应有三种方式:

忽略信号,对信号不做任何处理,但是又两个信号是不能忽略的, 捕捉信号,定义信号处理函数,当信号发送时,执行相应的自定义: 执行缺省操作,linux对每种信号都规定了默认操作 7、kill()方法

kill()函数和熟知的kill系统命令一样,可以发送信号给信号和需要注意的是他不仅可以终止进程(发送SIGKILL信号),也可以向

8、raise()方法

与kill函数不同的是raise()函数允许进程向自身发送信号。

9、alarm()方法

alarm()也称为闹钟函数,可以在进程中设置一个定时器,等到时注意的是一个进程只能有一个闹钟时间,如果调用alarm()之前已

10、pause()方法

此函数用于将进程挂起直到捕捉到信号为止,这个函数很常用,通

### shell如何输入制表符

```
1、ctrl,按下v,再按下i
```

## system与popen

```
1、脚本如下:
    [root@localhost localconfig]# more niu.sh
    #! /bin/sh
    set -x
    sleep 5
    echo 'over'
2、使用system调用,如下:
    #include <stdio.h>
    #include <stdlib.h>
    int main(int argc,char* argv[])
    {
            char script[256] = \{0\};
            sprintf(script,"./niu.sh");
            system(script);
            return 0;
3、使用popen调用,如下:
    #include <stdio.h>
```

```
#include <stdlib.h>
   int main(int argc,char* argv[])
   {
          char script[256] = \{0\};
          sprintf(script,"./niu.sh");
          printf("Exe Script[%s]", script);
          FILE* fp = popen(script, "r");
          pclose(fp);
          return 0;
   }
4、这里存在问题,由于脚本执行需要5秒钟,导致程序卡住5秒,怎么办
   把脚本放入后台运行,如下:
   sprintf(script,"./niu.sh &");
5、system与popen的区别:
   system和popen都是执行了类似的运行流程,大致是fork->exe
   但是我们看到system在执行期间调用进程会一直等待shell命令扩
   但是popen无须等待shell命令执行完成就返回了。
   我们可以理解system为串行执行,在执行期间调用进程放弃了"控
   popen中的子进程没人给它收尸了啊?是的,如果你没有在调用po
6、另外一个重要的区别是:popen可以与标准IO流进行交互。如果存在
   #include <stdio.h>
   #include <stdlib.h>
   int main(int argc,char* argv[])
   {
          char script[256] = \{0\};
          sprintf(script,"./niu.sh &");
          system(script);
   //
          printf("Exe Script[%s]", script);
          FILE* fp = popen(script, "r");
          char outBuf[256] = \{0\};
```

```
fread(outBuf,1024,1,fp);
printf("Exe Result:%s",outBuf);
pclose(fp);

return 0;
}
为什么?
很好理解,因为要进行输入,或者获取输出。
```

## tty\_\_pty\_\_pts\_\_ptmx

1、tty(终端设备的统称)

tty一词源于Teletypes,或者teletypewriters,原来指的是后来这东西被键盘与显示器取代,所以现在叫终端比较合适。

终端是一种字符型设备,它有多种类型,通常使用tty来简称各种影

2、pty (虚拟终端):

但是如果我们远程telnet到主机或使用xterm时不也需要一个终端

3、pts/ptmx(pts/ptmx结合使用, 进而实现pty)

pts(pseudo-terminal slave)是pty的实现方法,与ptmx(p

## 打包deb包

```
1、示例脚本如下:
#! /bin/sh
set -x
set -e

# read -p "please input version like 1.0.0: " version
name="masterkeymgr"
version="1.0.0"
underline="_"
platform="amd64"
projName=${name}${underline}${version}${underline}${projName=${projName}".deb"

# 脚本所在目录
```

```
shPath=$(cd $(dirname $0);pwd)
cd $shPath/../
if [ ! -d "./dist" ]; then
        mkdir "./dist"
fi
cd ./dist
rm -fr *
mkdir DEBIAN
cat>"./DEBIAN/control"<<EOF
package: $name
version: $version
architecture: $platform
maintainer: niuzibin
description: this is masterkeymgr deb
EOF
mkdir -p ./usr/lib
cp $shPath/../lib/libmasterkeymgr.so ./usr/lib
mkdir -p ./usr/include
cp $shPath/../src/masterkeymgr/masterkeymgr.h ./usr/i
cd $shPath/../
sudo dpkg -b dist $debName
```

## 获取脚本所在的绝对路径

- 1、pwd是工作目录,不是脚本所在的目录,所谓工作目录就是:在哪个考虑脚本aa.sh中有 echo \$(pwd),从aa.sh所在目录调用aa.d但是,从目录A去调用aa.sh,回显目录A,从目录B去调用aa.sh 特别注意:从不同的目录调用aa.sh,后面的参数不同,因为需要也就是说,从哪个目录去调用【也就是工作目录】,pwd就是哪个
- 2、下面使用python来举例说明:
  - a、os.getcwd():获取当前工作目录,也就是在哪个目录下运行

```
举例来说:假设F:\aaa\bbb\ccc目录下有个文件 TestDi
      在F:\aaa\bbb\ccc运行 python TestDir.py,输出F:\
      在F:\aaa\bbb运行 python ccc/TestDir.py,输出F:\
      在F:\aaa运行 python bbb/ccc/TestDir.py,输出F:\
   b、如果我想获取TestDir.py文件所在的目录,怎么办?
      使用sys.path[0] 如下:
      import os
      print(os.getcwd())
      import sys
      print(sys.path[0], end='')
3、如何获取脚本所在的决定路径呢?
   shPath=$(cd $(dirname $0);pwd)
   echo $shPath
   cd $(dirname $0) 进入脚本所在目录,工作目录变化,pwd获
4、注意: $(cd $(dirname $0); pwd)返回后,工作目录还是原来的。
   要想改变工作目录,使用cd $(dirname $0),不能获取命令的证
   为什么?
   可以认为,获取命令的返回值,相当于进入新的工作目录,返回后
5、$(dirname $0) 返回第一个参数的目录,第一个参数就是脚本。>>
   SHELL HOME=`dirname $0`
```

## 虚拟内存和常驻内存

```
1、考虑下面的代码,如下:
    #include <string>
    #include <vector>
    using namespace std;
    int main(int argc, char* argv[])
    {
        int i=0;
        vector<char*> pVec;
        while(true)
        {
            printf("num=%d\n",i++);
            //fflush(stdout);
            usleep(2000*1000);
            //sleep(2);
```

```
char* pc = new char[1024*1024];
                pc = new char[1024*1024];
                memset(pc, 0, 1024*1024);
                pVec.push back(pc);
          return 0;
2、程序每次new出1M,然后再new出1M,前面的1M不使用,使用后面的
   32596 root
                15 0 13084 6016 760 S 0.0
                                           1.
                15 0 15140 7048
                                           1.
   32596 root
                                760 S
                                      0.0
                15 0 17196 8080
                                760 S 0.0
                                           1.
   32596 root
   32596 root
                15 0 19252 9112
                                 760 S
                                      0.0
                                           1.
                15 0 21308 9.9m
                                760 S 0.0
                                           2.
   32596 root
                15 0 23364 10m
                                           2.
   32596 root
                                760 S 0.0
                15 0 25420 11m
                                           2.
   32596 root
                                 760 S
                                      0.0
                15 0 27476 12m
   32596 root
                                760 S 0.0
                                           2.
                15 0 29532 13m
                                           2.
   32596 root
                                760 S 0.0
                15 0 31588 14m
                                           3.
   32596 root
                                 760 S 1.0
   32596 root
                15
                     0 31588 14m
                                 760 S 0.0
                                           3.
                     0 33644 15m
   32596 root
                15
                                760 S 0.0
                                           3.
3、可以看到VIRT每次增加2M,而RES每次增加1M,SHR是共享内存,不
   刚开始的时候,单位是KB,RES大于10M,使用M作为单位。
4、为什么VIRT每次增加2M,而RES每次增加1M?
   VIRT是虚拟内存,包括常驻内存和交换到硬盘的内存。
   RES是常驻内存,每次new出2M,只有1M使用,在常驻内存,另外。
   可以认为Linux操作系统很聪明,没有使用的1M直接交换到硬盘中
5、怎么看出交换出去的内存大小?
   top命令后,按下f,按下p选中p: SWAP
                                  = Swapped s.
   32661 root
                15
                     0 29532 13m 760 S 1.0
                                           2.
   VIRT
          29532 KB
   RES
          13M
   SWAP
          14M
6、看进程是否内存泄露,要看VIRT虚拟内存,因为程序有些内存分配了
7、同样的程序,在Windows下面,表现不一样,在Windows虚拟内存和
   在windows如何查看虚拟内存?
   任务管理器 - ->查看 - ->选择列 - ->内存 - 提交大小就是虚拟内存
   而内存-专用工作集就是常驻内存
```

## 运行缺少库

- 1、运行缺少库,需要添加依赖库的路径,有两种办法。
- 2、第一种办法:

export  $LD_LIBRARY_PATH = xxx$ 

3、第二种办法:

修改 /etc/ld.so.conf 添加路径, 然后执行 ldconfig

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

打包和压缩 Linux

## 重变

压缩

tar

## 变量说明

#### tar

```
1、区分打包和压缩的概念,打包是把一堆文件合成一个文件,压缩是把
   经典的压缩算法是哈夫曼算法。
2、压缩程序一般只能对一个文件进行压缩,因此,对于一堆文件,要先
3、tar既可以打包也可以压缩,压缩可认为调用其他的命令进行压缩。
4、z对应gzip,j对应bzip2,压缩与解压的算法要一致,否者出错。
5、常用选项:
        创建压缩包
   - C
   -x 解压
   -t 查看压缩的内容
-r 在打包的文件中追加
   -u 在打包的文件中更新
         选项取值是压缩的文件,必须是最后一个选项
   注意:-r -u只能针对打包的文件,压缩后的文件,不能执行。另
   但是解压会覆盖,只有一个。
6、例子:
   tar -zcvf aaa.tar.gz 111 222 // 压缩111 222
   tar -zxvf aaa.tar.gz // 解压到当前文件
tar -zxvf aaa.tar.gz -C ./hhh // 解压到./hhh
tar命令示例,考虑下面的情况,当前目录有文件111.txt和aaa.txt
1、打包成IBP.tar,如下:
```

-rw-r--r-- 1 root root 2 2015-10-15 10:28 111.txt -rw-r--r-- 1 root root 5 2015-10-15 10:28 aaa.txt

[root@localhost tmp]# tar -cvf IBP.tar ./\*

[root@localhost tmp]# 11

total 8

```
./111.txt
    ./aaa.txt
    [root@localhost tmp]# 11
    total 20
    -rw-r--r-- 1 root root 2 2015-10-15 10:28 111
-rw-r--r-- 1 root root 5 2015-10-15 10:28 aaa
    -rw-r--r-- 1 root root 10240 2015-10-15 10:29 IBF
2、解包
    [root@localhost tmp]# rm -f *txt
    [root@localhost tmp]# 11
    total 12
    -rw-r--r-- 1 root root 10240 2015-10-15 10:29 IBF
    [root@localhost tmp]# tar -xvf IBP.tar
    ./111.txt
    ./aaa.txt
    [root@localhost tmp]# 11
    total 20
    -rw-r--r-- 1 root root 2 2015-10-15 10:28 111
    -rw-r--r-- 1 root root 5 2015-10-15 10:28 aaa
    -rw-r--r-- 1 root root 10240 2015-10-15 10:29 IBF
3、查看包的内容
    [root@localhost tmp]# rm -f *txt
    [root@localhost tmp]# tar -tvf IBP.tar
                          2 2015-10-15 10:28
    -rw-r--r-- root/root
    -rw-r--r-- root/root
                                 5 2015-10-15 10:28
    [root@localhost tmp]# 11
    total 12
    -rw-r--r-- 1 root root 10240 2015-10-15 10:29 IBF
4、把111.txt和aaa.txt,使用gzip压缩成IBP.tar.gz
    [root@localhost tmp]# 11
    total 8
    -rw-r--r-- 1 root root 2 2015-10-15 10:28 111.txt
    -rw-r--r-- 1 root root 5 2015-10-15 10:28 aaa.txt
    [root@localhost tmp]# tar -zcvf IBP.tar.qz ./*
    ./111.txt
    ./aaa.txt
    [root@localhost tmp]# 11
```

```
total 12
    -rw-r--r-- 1 root root 2 2015-10-15 10:28 111.t
-rw-r--r-- 1 root root 5 2015-10-15 10:28 aaa.t
    -rw-r--r-- 1 root root 145 2015-10-15 10:34 IBP.t
5、解压到IBP,先创建目录IBP
    [root@localhost tmp]# mkdir IBP
    [root@localhost tmp]# tar -zxvf IBP.tar.gz -C ./]
    ./111.txt
    ./aaa.txt
    [root@localhost tmp]# 11
    total 4
    drwxr-xr-x 2 root root 0 2015-10-15 10:37 IBP
    -rw-r--r-- 1 root root 145 2015-10-15 10:34 IBP.t
6、把目录IBP压缩成IBP.tar.gz,并且压缩包里面没有目录IBP,进入
    [root@localhost tmp]# cd IBP
    [root@localhost IBP]# tar -zcvf ../IBP.tar.gz ./'
    ./111.txt
    ./aaa.txt
    [root@localhost IBP]# cd ..
    [root@localhost tmp]# 11
    total 4
    drwxr-xr-x 2 root root 0 2015-10-15 10:37 IBP
    -rw-r--r-- 1 root root 145 2015-10-15 10:39 IBP.t
7、把目录IBP压缩成IBP.tar.gz,压缩包里面有目录IBP
    [root@localhost tmp]# 11
    total 0
    drwxr-xr-x 2 root root 0 2015-10-15 10:37 IBP
    [root@localhost tmp]# tar -zcvf IBP.tar.gz IBP
    IBP/
    IBP/aaa.txt
    IBP/111.txt
    [root@localhost tmp]# tar -ztvf IBP.tar.gz
    drwxr-xr-x root/root
                                0 2015-10-15 10:37 ]
    -rw-r--r-- root/root 5 2015-10-15 10:28 ]
                          2 2015-10-15 10:28 ]
    -rw-r--r-- root/root
8、解压到当前目录
    [root@localhost tmp]# tar -zxvf IBP.tar.gz
```

```
IBP/
IBP/aaa.txt
IBP/111.txt
[root@localhost tmp]# ll
total 4
drwxr-xr-x 2 root root 0 2015-10-15 10:37 IBP
-rw-r--r-- 1 root root 181 2015-10-15 10:46 IBP.t
```

#### 压缩

```
1、文件格式:
   * .Z
                compress程序压缩的文件
   *.gz
                gzip程序压缩的文件
                bzip2程序压缩的文件
   *.bz2
   *.tar
                tar程序打包的文件,没有压缩
                tar程序打包的文件,并使用gzip压缩
   *.tar.gz
                tar程序打包的文件,并使用bzip2压缩
   *.tar.bz2
2、压缩效果:
   compress < gzip < bzip2</pre>
3、以gzip为例说明,
                // 压缩
   qzip 111
   gzip -d 111.gz // 解压,等价于 gunzip 111.gz
   注:使用gzip压缩解压之后,原文件不见了,要想保留原文件,使
   gzip -c 111 >aaa.gz // 111后面有个空格,保留了原:
   gzip -dc aaa.gz >111 // aaa.gz后面有个空格,保留了
4、zcat可以查看gzip压缩文件,显示的是解压后的文本内容。bzip2
5、bzip2的用法和gzip类似。
```

文件与目录 Linux

## 模块

文件与目录管理

文件内容查看

## 详细描述

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

**文件与目录管理** Linux » 文件与目录

## 重变

ср

scp

查看文件和目录的大小

Linux搜索文件

## 变量说明

#### Ср

- 1、cp命令,目标已经存在,每次都提示是否覆盖,怎么办?
- 2、cp --help 可以看到选项-i的时候,才会提示,但是这里并没有--
- 3、原因是:这里执行的cp是一个别名,通过alias 命令可以看到 ali
- 4、怎么解决这个问题?
  - a、使用原始的cp命令,\cp 1 2
  - b、通过alias命令,临时取消别名,alias cp='cp'
  - c、要想重新登录或者重启有效,需要修改文件~/.bashrc,注意
- 5、注意:cp -f表示存在目标,不能打开,删除目标,再试一次。
- 6、考虑下面的需求,把当前目录的111,拷贝到./bbb/ccc目录,而...如果不存在,新建目录,然后拷贝。
  - 解决办法:
  - a if [! -d './bbb/ccc']; then mkdir -p './bbb/
  - b、mkdir -p ./bbb/ccc/; \cp 111 \$\_;
  - 注:\$\_引用前一个命令的最后一个参数
    - mkdir,目录不存在创建,如果存在,就不创建,返回值都是
- 7、\cp 111 ./bbb/ccc/ 把111拷贝到./bbb/ccc/, 如果重命名为

### 参见

### Linux搜索文件

- 1、whereis 只能查找可执行文件,联机帮助文档,源代码文件。在数于但是数据库mlocate.db不是实时更新的,而是一天更新一次(cr要想准确,需要执行一把updatedb,但是这又失去了高效性。
- 2、locate是whereis的加强版,增加一些选项,比如正则表达式。

- 3、which \$PATH配置了可执行文件的目录,有多个目录,which在这些
- 4、type,检查一个命令是否为shell内置命令,如果不是shell内置行
- 5、find 遍历磁盘查找,因此很准确,但是效率很差。find的功能很强 a、find / -name aaa 在当前目录查找名称为aaa的文件,忽 b、find查找是精确查找,文件名必须相同。如果只知道文件名的
  - ? 和 \* ?表示匹配一个字符, \*表示匹配任意个字符。如:注意:通配符与正则表达式不是一个概念,在正则表达式中, mysql查询支持通配符和正则表达式,在mysql中下划线"\_"
  - c、使用正则表达式, find ./ -regex '.\*a', 特别注意,正则 文件路径就是要搜索的路径。
  - d、使用通配符或者正则表达式,查找的目标,必须使用单引号或者如果没有括起来,相当于特殊字符作为文件名的一部分,当然
- 6、特别注意:find与grep的区别。

find是根据文件属性,查找文件 grep是根据文件内容,查找文件

#### scp

- 1、cp只是在本机上进行,我想把本机的一个文件,拷贝到远程的一个机
- 2、使用scp,如下:
  - scp ./aaa root@10.65.200.52:/tmp
- 3、接下来输入机器10.65.200.52的登录密码,登录名是root,需要 scp默认的端口是 22,如果10.65.200.52的端口是1733,需要 scp -P1733 ./aaa root@10.65.200.52:/tmp

### 查看文件和目录的大小

- 1、df -h 查看整个文件的使用情况
- 2、du 查看当前目录的情况,包括每个文件,目录的大小注:一个总的目录往往下面有非常多的子目录和文件,使用du --max-depth=2 只查看两层

 首页
 模块

 文件内容查看

 Linux » 文件与目录

## 重变

cat		
head		
less		
more		
nl		
od		
tac		
tail		

## 变量说明

#### cat

```
1、cat一次显示整个文档的内容,而more、less是可以分页显示的。
2、cat有创建文件的功能,创建文件的同时,可以输入数据。如下:
   [root@localhost niu]# 11
   总计 0
   [root@localhost niu]# cat >111
   hello world
   [root@localhost niu]# 11
   总计 4
   -rw-r--r-- 1 root root 12 09-01 18:42 111
   [root@localhost niu]# cat 111
   hello world
3、注意:cat >111 最后一行的输入,不写入文件。怎么解决?
   可以多输出一个空行
   也可以使用EOF,告诉shell后续的输入作为子命令的输入,遇到E
   [root@localhost niu]# cat >111 <<EOF</pre>
   > ddd
   > dfdf
   > E0F
   [root@localhost niu]# cat 111
   ddd
   dfdf
   注意:在shell下输入的时候,删除前一个字符,不能使用后退ba
   而进入子命令输入时,可以使用后退符。
4、cat可以修改文件内容的功能,可以覆盖或者追加。
   cat >111 // 覆盖
```

cat >>111 //追加

5、上面都是cat从标准输入接收输入数据,也可以从其他文件接收输入

cat 111 222 >333 cat 111 222 >>333

注意:222后面有一个空格

- 6、cat常用的选项
  - -n 列出行数
  - -A 列出所有的字符,包含不可见的空白字符

#### 参见

tac

#### head

- 1、head显示文档的开头n行,如下:
  head -n5 111 // 显示文档的开头5行
- 2、head默认查看的行数是10行

#### 参见

tail

#### less

1、上一行下一行

上一行:【向上方向键】或者【y】

下一行:【向下方向键】或者【enter】

2、上一页下一页

上一页:【b】

下一页:【space】

3、上半页下半页

上半页:【u】 下半页:【d】

怎么记忆,上下的关系通过按键的上下位置,u在d上面,b在spac 大写D只有大写B的一半,因此表示半页

4、/Andy 向下查找Andy, n是下一个, N是上一个?Andy 相反,向上查找

- 5、:v 使用vi编辑当前文件
- 6、less比more的功能更强大。
- 7、注意:在有些终端上使用less,退出less后,之前显示的内容不见使用 echo \$TERM 查看取值,如果是 "xterm",执行export在选项-->会话选项-->终端-->仿真-->终端,可以看到TERM的耳

#### 参见

#### more

#### more

1、和cat相比, more支持分页显示

上一页:【b】

下一页:【space】

2、enter转到下一行,不能转到上一行

#### 参见

less

#### nl

显示文件内容,并显示行号,等价于 cat -n

### 参见

cat

#### od

- 1、使用字符的值显示,比如:字符a是16机制的61,也就是97
- 2、示例如下:

[root@localhost niu]# cat 444

123123

456

abc

[root@localhost niu]# od -tx 444 0000000 31333231 340a3332 610a3635 0a0a6362 0000020

1对应31,2对应32,a对应61,b对应62,换行符\n 对应0a,高(3、特殊字符的值:

\n: [0a] [10] \r: [0d] [13]

记忆方法, a与n构造单词 an

参见

cat

#### tac

tac与cat相反,从最后一行到第一行,反向显示。

参见

cat

#### tail

- 1、与head相反,显示文档的结尾n行,如下: tail -n5 111 // 显示文档的结尾5行
- 2、tail有一个很常用的选项 -f【--follow】 显示文档增加的内容,tail -f 日志文件,就可以实时查看日志É

参见

head

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

文件系统和目录树 Linux

## 变量

文件系统
目录树
mount
mount
thin

## 变量说明

#### ln

- 1、连接文件分为两种方式:硬连接和软连接
- 2、我们知道,文件名关联文件系统的inode,硬连接相当于多一个文件通过11 -i,可以看到文件名关联的inode相同,同时inode的连
- 3、软连接相当于新建一个文件,软连接文件名关联inode,inode指向
- 4、从C++指针和引用计数的角度理解:

硬连接相当于指针复制,inode的引用计数加1,删除文件名的时候软连接相当于指针的指针,比如原文件名为A,软连接为B,B指向/

- 5、使用硬连接比较安全,但是硬连接存在问题:
  - a、不能跨越文件系统,原因是:因为只有一个文件系统中,两个了b、不能硬连接到目录,原因是:如果可以,那就要保证目录下面的 这显然太复杂。
- 6、软连接是指针的指针,可认为是Windows下面的快捷方式,没有硬连
- 7、ln的语法,ln [-sf] 源文件 目标文件
  - -s 是软连接,不加参数就是硬连接
  - -f 如果目标文件存在,删除目标文件再重新创建

#### 参见

#### mount

- 1、将文件系统(或者其他目录)与目录树结合的操作,成为挂载。 也就是目录树上面的一个目录,作为挂载点,挂载一个文件系统, 注:挂载点必须是目录树上面的目录。
- 2、示例,如下:

mount //10.36.65.80/80\_Linux\_Dir ~/80\_Linux\_Dir -

#### mount报错不能分配内存

1、mount报错如下:

[root@localhost datafs]# mount //10.65.200.168/iv
mount: Cannot allocate memory

2、错误原因是:

//10.65.200.168/ivss 目录太大,不能进行挂载

3、解决办法:

修改windows的注册表

修改注册表 HKEY\_LOCAL\_MACHINE\System\CurrentControl如果不存在IRPStackSize项,新建一个DWORD值,取名为IRPSt然后重启电脑。

#### umount

#### 取消挂载:

umount //10.36.65.80/80\_Linux\_Dir

#### 文件系统

1、Linux的文件系统包含三个部分:

inode:记录文件的属性,包括访问模式(rwx),所有者和所有约 最重要的是包含指针(多个指针以及间接指针),指向文件的data block:实际记录文件的内容,若文件较大,会占用多个dasuper block 记录文件系统的整体信息,包括inode/data block

2、Linux的这种文件系统,称为索引式文件系统,由inode可以定位到内容。而Windows的文件系统可认为是链表式文件系统,读取到danackinux基本上不需要。如果data block分散的很厉害而Linux基本上不需要。

### 参见

### 目录树

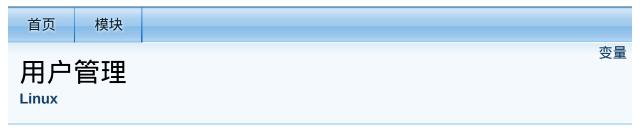
- 1、Linux内存中维护一棵目录树,使用VFS,目录包含两个部分:目录:
- 2、目录关联一个inode, inode记录目录的权限和属性, inode关联da

## 的文件名和该文件名占用的inode

- 3、inode中并不记录文件名,而是文件名关联inode,inode记录文件
- 4、定位文件的过程是:当前目录A-->目录A的inode-->目录A的data -->找到文件名对应的inode-->找到文件的data block-->读取

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.



## 变量

用户名和密码

查询用户登录信息

用户的增删改

用户身份切换

## 详细描述

## 变量说明

#### 查询用户登录信息

```
1、查询当前登陆的用户
    [root@localhost rc.d]# w
    17:49:10 up 1:31, 5 users,
                                  load average: 0.37,
    USER
             TTY
                      FROM
                                         LOGIN@
                                                  IDLE
             :0
                                        16:19
                                                ?xdm?
    root
             pts/1
                                        16:20
                                                 1:23n
    root
                      :0.0
             pts/2
                      192.168.1.100
                                        16:22
                                                52:21
    root
             pts/3
                      192.168.1.100
                                        16:57
                                                51:45
    root
    root
             pts/4
                      192.168.1.100
                                        16:57
                                                 0.005
    [root@localhost rc.d]# who
             :0
                          2015-12-28 16:19
    root
                          2015-12-28 16:20 (:0.0)
    root
             pts/1
             pts/2
                          2015-12-28 16:22 (192.168.1
    root
             pts/3
                          2015-12-28 16:57 (192.168.1
    root
    root
             pts/4
                          2015-12-28 16:57 (192.168.1
    [root@localhost rc.d]# whoami
    root
    [root@localhost rc.d]# who am i
                          2015-12-28 16:57 (192.168.1
    root
             pts/4
2、区别如下:
                show who is logged on and what they a
    W
                show who is logged on
    who
                print effective userid
    whoami
                print actual userid
    who am i
    Consider, when a user logs in as a root across the
    However, when a user abc logs in remotely and rur
```

```
3、lastlog 查询所有用户或者某一个用户,最后一次的登陆地址和登
    [root@localhost ~]# lastlog
                              From
                     Port
    Username
                                               Latest
    root
                     pts/5
                              192.168.1.100
                                               Tue De
    bin
                                                **Neve
    daemon
                                                **Neve
4、last对于登录过的用户,显示登陆有关的信息。
    [root@localhost ~]# last
    root
             pts/5
                          192.168.1.100
                                           Tue Dec 29
             pts/4
                          192.168.1.100
                                           Mon Dec 28
    root
    root
             pts/3
                          192.168.1.100
                                           Mon Dec 28
                          192.168.1.100
             pts/2
                                           Mon Dec 28
    root
             pts/2
                          :0.0
                                           Mon Dec 28
    root
    root
             pts/1
                          :0.0
                                           Mon Dec 28
             :0
                                           Mon Dec 28
    root
             :0
                                           Mon Dec 28
    root
                          2.6.18-194.el5
    reboot
             system boot
                                           Mon Dec 28
    zhanxin
             pts/1
                          :0.0
                                           Wed Jul
    zhanxin
             :0
                                           Wed Jul
5、另外id和finger也能显示当前登陆的用户信息,如下:
    [root@localhost ~]# id
    uid=0(root) gid=0(root) groups=0(root),1(bin),2(c
    [root@localhost ~]# finger
   Login
                                  Idle
                                        Login Time
              Name
                         Tty
                        *:0
                                        Dec 28 16:19
    root
              root
                                 17:47
                                        Dec 28 16:20
    root
                         pts/1
              root
                                        Dec 28 16:22
    root
              root
                         pts/2
                                 17:15
                                 17:15
                                        Dec 28 16:57
                         pts/3
    root
              root
                         pts/4
                                        Dec 28 16:57
    root
              root
                                     9
                                        Dec 29 10:03
    root
              root
                         pts/5
```

### 用户名和密码

1、登录Linux主机的时候,我们使用的是用户名,而Linux使用UID和

[root@2 ~]# more /etc/passwd

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin passwd每行表示一个用户,有7个字段

2、passwd每行中的第二个字段是登录密码,这里都是x,真正的密码係root:\$6\$dC4C06mrthJsxgWE\$q7SlxiQvgB7oQRLSQBInr7iN

bin:\*:15628:0:99999:7:::

daemon: \*:15628:0:99999:7:::

adm: \*:15628:0:99999:7:::

shadow每行表示一个用户对应的密码,有9个字段,第二个字段是

3、Linux如果忘记密码,怎么办?

忘记普通用户的密码,这个很简单,root登录进去,使用passwd 忘记root用户的密码,有两个办法:

- a、重新启动进入用户维护模式,Linux系统给予root权限的bask
- b、以Live CD开机挂载根目录,修改shadow文件, root的密码等
- 4、与用户对应,用户组的信息分别在文件 /etc/group 和 /etc/gs

#### 用户的增删改

1、添加用户

useradd niu

注意:useradd是二进制程序,而adduser是useradd的软连接。

2、设置密码

passwd niu

输入密码,然后确认

使用ssh远程登录,确认 ssh root@192.168.1.101

3、删除用户

userdel niu

4、修改用户名称

usermod -lniu2 niu 把用户名niu修改为niu2

### 用户身份切换

- 1、考虑下面的需求,当前以普通用户登陆,先查看shadow的内容,如-bash-3.2\$ cat /etc/shadow
  - cat: /etc/shadow: Permission denied
- 2、这个文件只有root用户才有权限,怎么办?
- 3、第一种方法,使用sudo,以其他用户的身份执行命令,通常是使用r

```
-bash-3.2$ sudo -uroot cat /etc/shadow
   [sudo] password for niu:
   niu is not in the sudoers file. This incident wi
4、这里报错,原因是用户niu 没有借用其他用户身份的权限,需要配置
   使用visudo在root下面的一行添加niu的信息,如下:
   ## Allow root to run any commands anywhere
          ALL=(ALL)
                         ALL
   root
   niu
          ALL=(ALL)
                          ALL
   再次执行, sudo -uroot cat /etc/shadow
5、第二种方法,使用su,切换到其他身份,如下:
   -bash-3.2$ env|grep -E -i 'mail|home'
   MAIL=/var/spool/mail/niu
   PWD=/home/niu
   HOME=/home/niu
   -bash-3.2$ su root
   Password:
   bash: alais: command not found
   [root@localhost niu]# env|grep -E -i 'mail|home'
   MAIL=/var/spool/mail/niu
   PWD=/home/niu
   HOMF=/root
   这个时候发现,虽然用户niu的身份已经切换到了root,但是环境
6、使用su - root(中划线后面有空格),如下:
   -bash-3.2$ env|grep -E -i 'mail|home'
   MAIL=/var/spool/mail/niu
   PWD=/home/niu
   HOME=/home/niu
   -bash-3.2$ su - root
   Password:
   -bash: alais: command not found
   [root@localhost ~]# env|grep -E -i 'mail|home'
   MAIL=/var/spool/mail/root
   HOME=/root
7、特别注意一点,使用su 改变的只是有效用户,并没有改变真实用户。
   -bash-3.2$ who am i
                        2015-12-29 11:42 (192.168.1
            pts/3
   niu
   -bash-3.2$ whoami
```

```
niu
-bash-3.2$ su root
Password:
bash: alais: command not found
[root@localhost niu]# who am i
niu pts/3 2015-12-29 11:42 (192.168.1
[root@localhost niu]# whoami
root
特别注意:用户niu的提示符是-bash-3.2$,而用户root的提示
8、通过上面的测试知道,使用su,切换到root,可能会导致比较大的f
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

管道和管道命令 Linux

## 重变

cut

xargs

## 变量说明

#### cut

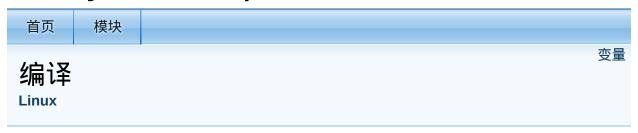
```
1、考虑下面的需求,获取当前时间的小时取值,如下
   [root@localhost Pacific]# date '+%Y-%m-%d %H:%M:9
   2016-01-19 18:17:56
   [root@localhost Pacific]# date '+%Y-%m-%d %H:%M:9
   18
2、获取时间,如下:
   [root@localhost Pacific]# date '+%Y-%m-%d %H:%M:9
   2016-01-19 18:18:35
   [root@localhost Pacific]# date '+%Y-%m-%d %H:%M:9
   18:18:44
   [root@localhost Pacific]# date '+%Y-%m-%d %H:%M:9
   18:18:49
3、上面都是使用-c取出字符的范围,数据是的格式是确定的,也就是占
   考虑export,取出所有的取值,如下:
   [root@localhost Pacific]# export |head -n 5
   declare -x CVS RSH="ssh"
   declare -x G BROKEN FILENAMES="1"
   declare -x HISTSIZE="1000"
   declare -x HOME="/root"
   declare -x HOSTNAME="localhost.localdomain"
   取出"ssh","1"等,怎么办?
4、使用-d分割,然后-f取出第几段,如下:
   [root@localhost Pacific]# export |head -n 5|cut -
   "ssh"
   "1"
   "1000"
```

```
"/root"
"localhost.localdomain"
```

#### xargs

```
1、xarqs的作用是把前一个命令的输出,作为后一个命令的输入,这不
2、举例来说,如下:
   [root@localhost tmp]# 11
   total 4
   -rw-r--r-- 1 root root 17 2015-10-28 17:13 aaa
   [root@localhost tmp]# ll|grep -v total|cut -d ' '
   aaa
   [root@localhost tmp]# ll|grep -v total|cut -d ' '
   This is file aaa
3、如上所示 .
   只使用管道, cat把前面的输出, 只是作为字符串, 输出aaa, 这个
   使用xargs, cat把前面的输出,作为一个文件,输出文件的内容。
   这里就看出使用xargs的区别了。
   也就是说:
   管道是把前面的标准输出作为后面的标准输入。
   而xargs是把前面的标准输出作为后一个命令的参数。
4、这样就能解释,rm一部分文件,必须使用xargs,如下:
   [root@localhost tmp]# 11
   total 0
   -rw-r--r-- 1 root root 0 2015-10-28 17:37 aaa
   [root@localhost tmp]# ll|grep -v total|cut -d ' '
   rm: missing operand
   Try `rm --help' for more information.
   [root@localhost tmp]# ll|grep -v total|cut -d ' '
   之所以ll|grep -v total|cut -d ' ' -f 8|rm失败,是因
5、注意:在windows下面的cmd,也有xargs命令。
```

# Andy Niu Help 1.0.0.0



# 变量

链接静态库只链接调用的方法
编译的头文件和库文件路径
编译告警
编译报错的解决思路
PHONY的作用
编译示例
Linux静态链接库
Linux动态链接库
Linux动态加载库
makefile
编译间接依赖的库
静态库与动态库的链接
编译链接的有关问题
方法地址找错

# 变量说明

#### Linux动态加载库

1、动态加载库,解决什么问题? 静态链接库和动态链接库是在编程时,直接调用对应的方法。链接 而动态加载库在编程时,需要使用dlopen等函数来获取库里面方法 对于没有提供头文件的动态库,只能dlopen等函数来调用。

```
2、测试代码:
    test.cpp
    #include <string>
    std::string GetName()
    {
            return std::string("Andy");
    }
    main.cpp
    #include <string>
    #include <dlfcn.h>
    using namespace std;
    typedef string (*Func) ();
    int main(int argc, char* argv[])
            void* pHandle = dlopen("./libtest.so", RT
            void* error = dlerror();
            printf("Error[%s]\n", error);
            Func pFunc = (Func)dlsym(pHandle, "GetName
            error = dlerror();
```

```
printf("Error[%s]\n", error);
                                   string name = pFunc();
                                   printf("name[%s]\n", name.c_str());
                                   return 0;
3、生成动态库so文件
            [root@localhost dll2]# g++ -fPIC -shared -o libte
            [root@localhost dll2]# 11
           total 16
            -rwxr-xr-x 1 root root 5158 Jan 31 22:38 libtest.
           -rw-r--r-- 1 root root 401 Jan 31 22:37 main.cpr
            -rw-r--r-- 1 root root 73 Jan 31 22:33 test.cpr
4、生成可执行文件
            [root@localhost dll2]# g++ -o main main.cpp -ldl
            [root@localhost dll2]# 11
           total 24
            -rwxr-xr-x 1 root root 5158 Jan 31 22:38 libtest.
            -rwxr-xr-x 1 root root 6222 Jan 31 22:38 main
            -rw-r--r-- 1 root root 401 Jan 31 22:37 main.cpr
            -rw-r--r-- 1 root root 73 Jan 31 22:33 test.cpg
            -1d1就是 -1 d1,表示动态加载d11
5、运行
            [root@localhost dll2]# ./main
           Error[(null)]
           Error[./libtest.so: undefined symbol: GetName]
           Segmentation fault (core dumped)
           报错没有找到定义的符号,明明有这个方法,为什么没有找到?
           这是因为G++编译的时候,进行了名称重整,生成libtest.so的F
           可能变成了__string_GetName, 所以dlsym(pHandle, "GetName, must be supplied to the control of the contro
6、怎么解决?
           需要抑制名称重整,修改如下:
           #include <string>
           extern "C"
           std::string GetName()
```

```
return std::string("Andy");
   }
   };
   重新执行一遍,运行如下:
   [root@localhost dll2]# g++ -fPIC -shared -o libte
   [root@localhost dll2]# g++ -o main main.cpp -ldl
   [root@localhost dll2]# ./main
   Error[(null)]
   Error[(null)]
   name[Andy]
7、上面的做法存在问题,因为C编译器并不认识extern "C",因此需要
   #include <string>
   #ifdef __cplusplus
   extern "C"
   {
   #endif
   std::string GetName()
   {
          return std::string("Andy");
   }
   #ifdef __cplusplus
   };
   #endif
8、静态链接、动态链接和动态加载的比较:
   a、对于静态库,实现发生改变,依赖它的exe(或者dll)文件必
   b、对于动态库的链接,实现发生改变,依赖它的exe(或者dll)
      如果接口发生变化,必须重新构建,否则报错,如下:
       ./main: symbol lookup error: ./main: undefine
   c、对于动态库的加载,和动态库的链接是同样道理。二者的区别是
      但是有些场景下,需要在程序运行过程中,按照需要加载进来
      相比较动态链接,动态加载的优点是:延迟和按照需要进行。
```

# Linux动态链接库

```
1、测试代码:
test.h
#include <string>
```

```
std::string GetName();
    test.cpp
   #include "test.h"
   std::string GetName()
    {
            return std::string("Andy");
   }
   main.cpp
   #include "test.h"
   using namespace std;
   int main(int argc, char* argv[])
    {
            string name = GetName();
            printf("name[%s]\n", name.c_str());
            return 0;
2、生成动态库so文件
    [root@localhost dll]# g++ -fPIC -shared -o libtes
    [root@localhost dll]# 11
    total 20
    -rwxr-xr-x 1 root root 5158 Jan 31 20:58 libtest.
    -rw-r--r-- 1 root root 149 Jan 31 20:46 main.cpg
    -rw-r--r-- 1 root root 74 Jan 31 20:43 test.cpr
    -rw-r--r-- 1 root root 41 Jan 31 20:55 test.h
3、生成可执行文件
    [root@localhost dll]# g++ -o main -ltest -L./ mai
    [root@localhost dll]# 11
    total 28
    -rwxr-xr-x 1 root root 5158 Jan 31 20:58 libtest.
    -rwxr-xr-x 1 root root 5992 Jan 31 20:59 main
    -rw-r--r-- 1 root root 149 Jan 31 20:46 main.cpg
    -rw-r--r-- 1 root root 74 Jan 31 20:43 test.cpr
    -rw-r--r-- 1 root root 41 Jan 31 20:55 test.h
4、运行
    [root@localhost dll]# export LD_LIBRARY_PATH=$(pv
```

### Linux静态链接库

```
1、测试代码:
    test.h
    #include <string>
    std::string GetName();
    test.cpp
    #include "test.h"
    std::string GetName()
    {
            return std::string("Andy");
    }
    main.cpp
    #include "test.h"
    #include <stdio.h>
    using namespace std;
    int main(int argc, char* argv[])
    {
            string name = GetName();
            printf("name[%s]\n", name.c_str());
            return 0;
2、生成静态库
    [root@localhost lib]# 11
```

```
total 12
    -rw-r--r-- 1 root root 149 Jan 31 21:27 main.cpp
    -rw-r--r-- 1 root root 73 Jan 31 21:27 test.cpp
    -rw-r--r-- 1 root root 41 Jan 31 21:27 test.h
    [root@localhost lib]# q++ -c test.cpp
    [root@localhost lib]# 11
    total 16
    -rw-r--r-- 1 root root 149 Jan 31 21:27 main.cpr
    -rw-r--r-- 1 root root 73 Jan 31 21:27 test.cpr
    -rw-r--r-- 1 root root 41 Jan 31 21:27 test.h
    -rw-r--r-- 1 root root 1384 Jan 31 21:30 test.o
    [root@localhost lib]# ar -rc libtest.a test.o
    [root@localhost lib]# 11
    total 20
    -rw-r--r-- 1 root root 1532 Jan 31 21:30 libtest.
    -rw-r--r-- 1 root root 149 Jan 31 21:27 main.cpr
    -rw-r--r-- 1 root root 73 Jan 31 21:27 test.cpr
    -rw-r--r-- 1 root root 41 Jan 31 21:27 test.h
    -rw-r--r-- 1 root root 1384 Jan 31 21:30 test.o
3、生成可执行文件
    [root@localhost lib]# g++ -o main main.cpp libtes
    [root@localhost lib]# 11
    total 28
    -rw-r--r-- 1 root root 1532 Jan 31 21:30 libtest.
    -rwxr-xr-x 1 root root 6317 Jan 31 21:31 main
    -rw-r--r-- 1 root root 149 Jan 31 21:27 main.cpg
    -rw-r--r-- 1 root root 73 Jan 31 21:27 test.cpr
-rw-r--r-- 1 root root 41 Jan 31 21:27 test.h
    -rw-r--r-- 1 root root 1384 Jan 31 21:30 test.o
4、特别注意:链接静态库,上面使用的方式是,后面直接添加静态库文
   也可以使用 -1的方式链接静态库,如下:
   q++ -o main main.cpp -L./ -ltest
   也就是说,【g++ -o main main.cpp -L./ -ltest】与【g+
   但是,有些特殊情况,需要明确标识是静态库还是动态库。
   使用-Wl,-Bstatic或者-Wl,-Bdynamic
    -Wl.option 此选项传递option给连接程序;如果option中间
5、运行
```

#### makefile

```
1、有以下三个文件: main.cpp, student.h, student.cpp
2、makefile文件如下:
app:main.o student.o
   g++ -o app main.o student.o
main.o:main.cpp student.h
   g++ -c main.cpp
student.o:student.cpp student.h
   q++ -c student.cpp
clean:
   rm -fr app *.o
3、app是目标,依赖main.o student.o,要生成目标,执行的命令是
   main.o是同样道理。
4、make xxx 执行对应的命令,
   make app 执行 g++ -o app main.o student.o
   make main.o 执行 g++ -c main.cpp
   make 默认执行 make app
5、目标main.o 和 student.o可以去掉, makefile在生成app的时,
6、make会自动在当前目录下面,查找makefile文件,如果存在多个make
   make -f makefile2
7、打印makefile中的变量
   makefile与shell脚本不一样,具体方法如下:
all: desc $(DLIBTARGET)
```

```
$(CP) $(DLIBTARGET) $(TARGET_BASEPATH)/lib_linux/
desc:
   echo aaaaaaaaaaa
   @echo $(TARGET BASEPATH)
   echo bbbbbbbbbbb
8、在makefile中调用shell脚本,如下:
all: desc
   echo '1111111'
   echo 22222222
desc:
   hh=kkkkkk;\
   echo $$hh
   ddddd; \
   ret=$$?;if [ $$ret -ne 0 ]; then echo "OK"; fi
   echo aaaaaaaaaaa
   @echo $(TARGET_BASEPATH)
   echo bbbbbbbbbb
   在makefile中使用shell脚本,有两点特别注意:
   a、一行必须是一个完整的shell语句,可以使用反斜杠折行
   b、为了区分makefile中的变量,引用变量使用 $$var
9、如果你想要看到make构建的详细过程,可以使用make VERBOSE=1
```

# PHONY的作用

```
1、避免语义冲突,告诉make不是创建目标文件,而是执行一些命令,也考虑,下面的情况,makefile内容如下:
clean:
    rm -fr *.o
执行:
    niuzibin@ubuntu:~/work/test1/phony$ ll
total 12
drwxrwxr-x 2 niuzibin niuzibin 4096 Apr 8 23:09
drwxrwxr-x 3 niuzibin niuzibin 4096 Apr 8 23:07
-rw-rw-r-- 1 niuzibin niuzibin 0 Apr 8 23:08
-rw-rw-r-- 1 niuzibin niuzibin 0 Apr 8 23:08
```

```
-rw-rw-r-- 1 niuzibin niuzibin
                                19 Apr
                                        8 23:07
niuzibin@ubuntu:~/work/test1/phony$ more makefile
clean:
        rm -fr *.o
niuzibin@ubuntu:~/work/test1/phony$ make clean
rm -fr *.o
niuzibin@ubuntu:~/work/test1/phony$ 11
total 12
drwxrwxr-x 2 niuzibin niuzibin 4096 Apr
                                       8 23:10
drwxrwxr-x 3 niuzibin niuzibin 4096 Apr
                                        8 23:07
-rw-rw-r-- 1 niuzibin niuzibin
                                        8 23:07
                                19 Apr
假设当前目录有个clean文件,如下:
niuzibin@ubuntu:~/work/test1/phony$ touch clean
niuzibin@ubuntu:~/work/test1/phony$ 11
total 12
drwxrwxr-x 2 niuzibin niuzibin 4096 Apr
                                        8 23:12
drwxrwxr-x 3 niuzibin niuzibin 4096 Apr
                                        8 23:07
-rw-rw-r-- 1 niuzibin niuzibin
                                 0 Apr
                                       8 23:12
-rw-rw-r-- 1 niuzibin niuzibin
                                 0 Apr
                                        8 23:12
-rw-rw-r-- 1 niuzibin niuzibin
                                 0 Apr
                                        8 23:12
-rw-rw-r-- 1 niuzibin niuzibin
                                19 Apr
                                        8 23:07
niuzibin@ubuntu:~/work/test1/phony$ make clean
make: `clean' is up to date.
这个时候的make clean , make读取到makefile中的clean , 检
这不是我们所期望的,怎么解决这个问题?
告诉make,这个clean是个伪目标,强制执行下面的命令。如下:
niuzibin@ubuntu:~/work/test1/phony$ ll
total 12
drwxrwxr-x 2 niuzibin niuzibin 4096 Apr 8 23:23
drwxrwxr-x 3 niuzibin niuzibin 4096 Apr
                                        8 23:07
                                 0 Apr
-rw-rw-r-- 1 niuzibin niuzibin
                                        8 23:23
-rw-rw-r-- 1 niuzibin niuzibin
                                 0 Apr
                                        8 23:23
-rw-rw-r-- 1 niuzibin niuzibin
                                 0 Apr
                                        8 23:12
-rw-rw-r-- 1 niuzibin niuzibin
                                32 Apr
                                        8 23:23
niuzibin@ubuntu:~/work/test1/phony$ more makefile
.PHONY:clean
```

```
clean:
           rm -fr *.o
   niuzibin@ubuntu:~/work/test1/phony$ make clean
    rm -fr *.o
2、解决间接依赖的问题。
   考虑下面的情况,当前包含了四个目录test、add、sub、includes
   test、add、sub三个目录分别包含了三个源程序test.c、add.
   niuzibin@ubuntu:~/work/test1/phony$ ll
    total 28
   drwxrwxr-x 6 niuzibin niuzibin 4096 Apr
                                           8 23:42
   drwxrwxr-x 3 niuzibin niuzibin 4096 Apr 8 23:07
   drwxrwxr-x 2 niuzibin niuzibin 4096 Apr 8 23:42
   drwxrwxr-x 2 niuzibin niuzibin 4096 Apr 8 23:34
    -rw-rw-r-- 1 niuzibin niuzibin 249 Apr 8 23:37
   drwxrwxr-x 2 niuzibin niuzibin 4096 Apr 8 23:42
   drwxrwxr-x 2 niuzibin niuzibin 4096 Apr 8 23:42
   niuzibin@ubuntu:~/work/test1/phony$ more ./incluc
   #ifndef HEAD H
   #define _HEAD_H_
   extern int add(int,int);
   extern int sub(int,int);
   #endif
   niuzibin@ubuntu:~/work/test1/phony$ more ./add/ac
   #include "../include/heads.h"
   int add(int a,int b)
    {
           return (a+b);
   niuzibin@ubuntu:~/work/test1/phony$ more ./add/ma
    add.o :add.c ../include/heads.h
           qcc -c -o $@ $<
    .PHONY: clean
```

```
clean:
        rm -f *.o
niuzibin@ubuntu:~/work/test1/phony$ more ./sub/su
#include "../include/heads.h"
int sub(int a,int b)
{
        return a-b;
}
niuzibin@ubuntu:~/work/test1/phony$ more ./sub/ma
sub.o:sub.c ../include/heads.h
        qcc -c -o $@ $<
.PHONY: clean
clean:
        rm -f *.o
niuzibin@ubuntu:~/work/test1/phony$ more ./makefi
OBJS = ./add/add.o ./sub/sub.o ./test/test.o
program: $(OBJS)
        gcc ./test/test.o ./add/add.o ./sub/sub.c
$(OBJS):
        make -C $(dir $@)
.PHONY: clean
clean:
        make -C ./add clean
        make -C ./sub clean
        make -C ./test clean
        rm -f program
运行,如下:
niuzibin@ubuntu:~/work/test1/phony$ make
make -C add/
make[1]: Entering directory `/home/niuzibin/work/
qcc -c -o add.o add.c
make[1]: Leaving directory `/home/niuzibin/work/t
```

```
make -C sub/
make[1]: Entering directory `/home/niuzibin/work/
gcc -c -o sub.o sub.c
make[1]: Leaving directory `/home/niuzibin/work/t
make -C test/
make[1]: Entering directory `/home/niuzibin/work/
qcc -c -o test.o test.c
make[1]: Leaving directory `/home/niuzibin/work/t
gcc ./test/test.o ./add/add.o ./sub/sub.o -o proc
注意:makefile的规则,是目标,依赖什么,怎么做,怎么做前ī
现在修改 ./add/add.c实现,如下:
niuzibin@ubuntu:~/work/test1/phony$ vi ./add/add.
#include "../include/heads.h"
int add(int a,int b)
{
       return (a+b+100);
再次执行make,如下:
niuzibin@ubuntu:~/work/test1/phony$ make
make: `program' is up to date.
当代码实现修改了,我们期望重新构建,但是并没有重新构建。分:
构建program的时候,检查被依赖的项 add sub test是否更新
解决办法就是,告诉make,./add/add.o./sub/sub.o./te
OBJS = ./add/add.o ./sub/sub.o ./test/test.o
program: $(OBJS)
       gcc ./test/test.o ./add/add.o ./sub/sub.c
.PHONY: $(OBJS)
$(OBJS):
       make -C $(dir $@)
.PHONY: clean
clean:
       make -C ./add clean
       make -C ./sub clean
```

```
make -C ./test clean
rm -f program
```

### 方法地址找错

1、两个动态库,含有相同的C接口方法名,在linux下,找方法地址可能

#### 编译告警

```
1、测试代码
    niuzibin@ubuntu:~/work/flawfinder$ vi main.cpp
    #include <stdio.h>
    #include <stdlib.h>
    int main(int argc, char* argv[])
    {
            printf("hello wolrd\n");
            int aa;
            int bb = aa + 10;
            char ch[50];
            qets(ch);
            scanf("%c",ch);
            return 1;
2、输出所有的告警 -Wall,这里是-大写W 加上单词all
    niuzibin@ubuntu:~/work/flawfinder$ g++ -Wall -o n
    main.cpp: In function 'int main(int, char**)':
    main.cpp:9:2: warning: 'char* gets(char*)' is der
    gets(ch);
    main.cpp:9:9: warning: 'char* gets(char*)' is der
    gets(ch);
    main.cpp:7:6: warning: unused variable 'bb' [-Wur
    int bb = aa + 10;
    main.cpp:7:16: warning: 'aa' may be used uninitia
    int bb = aa + 10;
```

```
/tmp/ccUdCIgt.o: In function `main':
   main.cpp:(.text+0x39): warning: the `gets' functi
3、禁止所有的告警 -w,在linux中,大小写往往表示相反的意思,-W
   niuzibin@ubuntu:~/work/flawfinder$ q++ -w -o mair
   /tmp/ccVjBsPa.o: In function `main':
   main.cpp:(.text+0x39): warning: the `gets' functi
   注意:有些告警比较严重,即使带上-w选项,也不能禁止,如上。
4、每一种告警,默认值(是否输出是不一样的),如下:
   niuzibin@ubuntu:~/work/flawfinder$ g++ -Wall -o n
   main.cpp: In function 'int main(int, char**)':
   main.cpp:9:2: warning: 'char* gets(char*)' is der
   gets(ch);
    Λ
   main.cpp:9:9: warning: 'char* gets(char*)' is der
   gets(ch);
   main.cpp:7:6: warning: unused variable 'bb' [-Wur
   int bb = aa + 10;
   main.cpp:7:16: warning: 'aa' may be used uninitia
   int bb = aa + 10;
   /tmp/ccic7wY5.o: In function `main':
   main.cpp:(.text+0x39): warning: the `gets' functi
   niuzibin@ubuntu:~/work/flawfinder$ g++ -o main ma
   main.cpp: In function 'int main(int, char**)':
   main.cpp:9:2: warning: 'char* gets(char*)' is der
   gets(ch);
   main.cpp:9:9: warning: 'char' gets(char')' is der
   gets(ch);
   /tmp/ccLU9M12.o: In function `main':
   main.cpp:(.text+0x39): warning: the `gets' functi
    可以看到,在默认情况下,-Wdeprecated-declarations是输向
5、如何输出默认不输出的?
```

指定-Wxxx强制输出,如下:

g++ -o main main.cpp -Wunused-variable

特别注意:一般情况,要输出全部的告警 -Wall,因为告警往往意

#### 编译报错的解决思路

1、cmake编译报错,如下:

/home/niuzibin/work/heming/cryptdb/src/util/onior
/home/niuzibin/work/heming/cryptdb/src/util/onior

2、我直接执行一把

g++ -o KeyManager.o -c KeyManager.cpp -I../incluc 同样的错误,如下:

/home/niuzibin/work/heming/cryptdb/src/util/onior
/home/niuzibin/work/heming/cryptdb/src/util/onior

3、加上-std=c++11,再来测试一把,如下:

g++ -std=c++11 -o KeyManager.o -c KeyManager.cpp 报错如下:

/home/niuzibin/work/heming/cryptdb/external\_libs/
my\_ptrdiff\_t l\_offset= (my\_ptrdiff\_t) (table->s/home/niuzibin/work/heming/cryptdb/external\_libs/
struct TABLE;

原因是:在前置声明的情况下,调用了方法。

4、那就要思考一个问题了,为什么直接在 heming/cryptdb中可以构 make VERBOSE=1

cd /home/niuzibin/work/heming/cryptdb/build/src/k

5、修改一下,拿到本地执行一把,

/usr/bin/c++ -DDBUG\_OFF -DEMBEDDED\_LIBRARY -DH/ 可以执行成功

6、那么对照一下,和上面的构建命令有啥区别?

一点一点向正确的靠近。

错误的

g++ -std=c++11 -o KeyManager.o -c KeyManager.cpp 正确的

/usr/bin/c++ -DDBUG\_OFF -DEMBEDDED\_LIBRARY -DH/

- 7、最终找到错误的地方,需要添加头文件的搜索路径,添加宏-DMYS(g++ -std=c++11 -o KeyManager.o -c KeyManager.cpp
- 8、事后总结:

#### 这里编译报错

error: invalid use of incomplete type 'struct error: forward declaration of 'struct TABLE'

直接原因是 在前置声明的情况下,调用了方法。

间接原因是 缺少预定义宏导致的问题,需要添加预定义宏 - DMYS

### 编译的头文件和库文件路径

1、编译,引用的头文件路径使用 - I 有一部分头文件路径,g++会默认引用,如何查看?

`g++ -print-prog-name=cc1plus` -v

2、链接时使用-L指定库的路径,使用-1指定库,运行时使用LD\_LIBR/目前没有找到方法,显示g++默认引用的库文件路径。步骤是: g++会去找-L

再找gcc的环境变量LIBRARY\_PATH

再找内定的目录 /lib /usr/lib /usr/local/lib 这是当初。

3、由于g++搜索默认的头文件和库文件路径,所以有时候不指定,构建

# 编译示例

```
1、C++代码:
#include <iostream>
int main(int argc,char* argv[])
{
       printf("Hello,world\n");
2、编译 g++ -c hello.cpp
   生成hello.o
3、链接 q++ -o hello hello.o
   生成hello,可执行程序
4、运行hello, 打印Hello world
5、编译链接,一起执行q++ -o hello hello.cpp
6、g++ hello.cpp 默认生成 a.out
7、编译链接的时候,需要知道头文件的路径,库的名称和路径,分别使
   g++ -o main main.cpp -I/usr/include -lpthread -L/
8、宏定义使用-D选项,如下:
   g++ -o main main.cpp -DDEBUG
```

9、特别注意:Windows下面包含头文件,可以使用斜杠或者反斜杠,不但是Linux下面,不能使用反斜杠,而且区分大小写。如下,#inc 因此,会出现Windows下面编译成功,而在Linux下面编译失败,

# 编译链接的有关问题

- 1、普通方法,只声明,不定义,没有问题,只要不调用方法,就不会链
- 2、虚方法,只声明,不定义,即使不调用方法,链接也报错。为什么? 为了实现多态性,必须在类的虚方法表里设置这个虚方法的地址( 注意:子类重写会在他的虚方法表里重新设置这个地址。
- 3、纯虚方法,只声明,不定义,没有问题。为什么? 对于纯虚方法,可以认为在类的虚方法表里,对于的纯虚方法slot 这个slot值为0,导致两个问题:
  - a、当前类不能实例化,考虑如果可以实例化,调用对应的虚方法, 也就是说,纯虚方法会导致当前类为抽象类(即使提供方法实 编译报错:不能实例化抽象类,并且会指出纯虚方法。
  - b、子类必须重写虚方法,子类的虚方法表对父类进行整体拷贝。表注意:纯虚方法也可以提供实现,相当于对于的纯虚方法slot值由

# 编译间接依赖的库

```
1、main依赖test.so, test.so依赖link.so, 示例代码如下:
    [root@localhost link]# more link.h
    #include <string>
    std::string GetAAA();

    [root@localhost link]# more link.cpp
    #include "link.h"
    std::string GetAAA()
    {
        return std::string("Andy");
    }

    [root@localhost link]# more test.h
    #include <string>
    std::string GetName();
```

```
[root@localhost link]# more test.cpp
   #include "test.h"
   #include "link.h"
   std::string GetName()
    {
       return GetAAA();
   }
    [root@localhost link]# more main.cpp
   #include "test.h"
   using namespace std;
   int main(int argc, char* argv[])
    {
       string name = GetName();
       printf("name[%s]\n", name.c_str());
       return 0;
2、对于这种情况,构建有两种方式。
3、第一种方式:先生成link.so,然后生成test.so的时候,链接lir
   然后生成main的时候,只需要链接test.so,如下:
    [root@localhost link]# g++ -fPIC -shared -o libli
    [root@localhost link]# g++ -fPIC -shared -o libte
    [root@localhost link]# g++ -o main main.cpp -L./
    [root@localhost link]# 11
   total 44
    -rwxr-xr-x 1 root root 5157 Aug 18 18:07 liblink.
   -rwxr-xr-x 1 root root 4557 Aug 18 18:08 libtest.
    -rw-r--r-- 1 root root 74 Aug 18 17:42 link.cpg
    -rw-r--r-- 1 root root 40 Aug 18 17:42 link.h
    -rwxr-xr-x 1 root root 5992 Aug 18 18:08 main
   -rw-r--r-- 1 root root 153 Aug 18 17:33 main.cpg
    -rw-r--r-- 1 root root 81 Aug 18 17:42 test.cp;
    -rw-r--r-- 1 root root 41 Aug 18 17:33 test.h
    [root@localhost link]# ./main
   name[Andy]
4、第二种方式:先生成link.so,然后生成test.so的时候,不链接l
   然后生成main的时候,同时链接test.so和link.so,如下:
```

```
[root@localhost link]# g++ -fPIC -shared -o libte
    [root@localhost link]# g++ -o main main.cpp -L./
    .//libtest.so: undefined reference to `GetAAA()'
    collect2: ld returned 1 exit status
    [root@localhost link]# g++ -o main main.cpp -L./
    [root@localhost link]# 11
    total 44
    -rwxr-xr-x 1 root root 5157 Aug 18 18:10 liblink.
    -rwxr-xr-x 1 root root 4533 Aug 18 18:10 libtest
    -rw-r--r-- 1 root root 74 Aug 18 17:42 link.cpg
    -rw-r--r-- 1 root root
                             40 Aug 18 17:42 link.h
    -rwxr-xr-x 1 root root 6016 Aug 18 18:11 main
    -rw-r--r-- 1 root root 153 Aug 18 17:33 main.cpg
    -rw-r--r-- 1 root root 81 Aug 18 17:42 test.cpr
    -rw-r--r-- 1 root root 41 Aug 18 17:33 test.h
    [root@localhost link]# ./main
    name[Andy]
5、也就是, main-->test.so-->link.so, 可以在每一步都链接, t
    每一步都链接,会导致test.so会稍微大一点,因为里面保存了链
    [root@localhost link]# g++ -fPIC -shared -o libte
    [root@localhost link]# ldd libtest.so
            linux-gate.so.1 => (0x00ab3000)
            liblink.so \Rightarrow ./liblink.so (0x00973000)
            libstdc++.so.6 => /usr/lib/libstdc++.so.6
            libm.so.6 => /lib/libm.so.6 (0x005a2000)
            libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x00)
            libc.so.6 => /lib/libc.so.6 (0x00ca3000)
            /lib/ld-linux.so.2 (0x006ab000)
    [root@localhost link]# g++ -fPIC -shared -o libte
    [root@localhost link]# ldd libtest.so
            linux-gate.so.1 \Rightarrow (0x00d7a000)
            libstdc++.so.6 => /usr/lib/libstdc++.so.6
            libm.so.6 => /lib/libm.so.6 (0x00c08000)
            libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x00)
            libc.so.6 => /lib/libc.so.6 (0x00110000)
            /lib/ld-linux.so.2 (0x006ab000)
```

[root@localhost link]# g++ -fPIC -shared -o libli

每一步都链接,test中多了一项链接 liblink.so => ./libl.

- 6、每一步都链接,只是相当于最后编译的时候少一个链接-llink,但是
- 7、不管是每步都链接,还是最后一起链接,能否定位到方法的实现,都也就是说,对于每步都链接,链接link,生成test.so的时候,艮最后一步生成main的时候,才报错找不到方法实现。
- 8、特别注意一点:对于动态库,其中的代码实现不会合并到任何其他动动态库中的代码段只会在内存中加载一次,所有依赖该动态库的主比如,定位到方法的入口。解决这个问题,有两种办法:
  - a、动态链接,编译的时候使用占位符,然后链接的时候修改占位符 b、动态加载,运行的时候,使用系统库根据方法名称直接找到方法
- 被依赖的动态库,暴露出来的接口不能进行名称重整,否者抄 9、再次强调,对于动态库,其中的代码实现不会合并到任何其他动态库
- 链接静态库只链接调用的方法
- 1、动态库或者可执行文件链接静态库,只会链接调用的方法,对于没有 比如静态库暴露了10个方法,动态库或者可执行文件只调用了一个

```
2、示例代码如下:
    root@ubuntu:/home/disk1/CPP 2/Link# more test.h
    int add(int a, int b);
    int sub(int a, int b);
    root@ubuntu:/home/disk1/CPP 2/Link# more test1.cr
    #include "test.h"
    int add(int a, int b)
    {
            return a+b;
    }
    root@ubuntu:/home/disk1/CPP 2/Link# more test2.cr
    #include "test.h"
    int sub(int a, int b)
    {
            return a-b;
    }
    root@ubuntu:/home/disk1/CPP_2/Link# more main.cpg
```

```
#include "test.h"
   #include <stdio.h>
   int main()
   {
           int c = add(1,2);
           printf("c[%d]\n", c);
           return 0;
3、测试如下:
   root@ubuntu:/home/disk1/CPP_2/Link# g++ -c test1.
   root@ubuntu:/home/disk1/CPP 2/Link# q++ -c test2.
   root@ubuntu:/home/disk1/CPP_2/Link# ar -rc libtes
   root@ubuntu:/home/disk1/CPP 2/Link# q++ -o main n
   root@ubuntu:/home/disk1/CPP 2/Link# ./main
   c[3]
   root@ubuntu:/home/disk1/CPP_2/Link# objdump -t ma
                         F text 000000000000014
   0000000000400562 q
   root@ubuntu:/home/disk1/CPP 2/Link# objdump -t mage
   root@ubuntu:/home/disk1/CPP 2/Link#
   可以看到, main中并没有sub方法的代码实现。
4、如果我想把静态库中的所有方法都链接进来(这种需求比较奇葩,也
   使用 --whole-archive编译选项,如下:
   root@ubuntu:/home/disk1/CPP_2/Link# g++ -o main n
   root@ubuntu:/home/disk1/CPP_2/Link# objdump -t max
   root@ubuntu:/home/disk1/CPP_2/Link# g++ -o main n
   root@ubuntu:/home/disk1/CPP 2/Link# objdump -t mage
   0000000000400576 g
                         F .text 000000000000016
   可以看到,加上--whole-archive链接静态库时,会把没有调用
5、特别说明,上面是可执行文件链接静态库,动态库链接静态库,道理
   动态库和可执行文件没有本质区别,可执行文件多了一个程序入口
   测试如下:
   root@ubuntu:/home/disk1/CPP 2/Link# more aaa.cpp
   #include "test.h"
   #include <stdio.h>
   int aaa()
```

```
int c = add(1,2);
           printf("c[%d]\n", c);
           return 0;
   }
   root@ubuntu:/home/disk1/CPP_2/Link# g++ -fPIC -sh
   root@ubuntu:/home/disk1/CPP_2/Link# objdump -t li
   root@ubuntu:/home/disk1/CPP 2/Link# ll |grep liba
   -rwxr-xr-x 1 root root 8082 Jun 14 15:47 libaaa
   root@ubuntu:/home/disk1/CPP_2/Link# g++ -fPIC -sh
   root@ubuntu:/home/disk1/CPP 2/Link# objdump -t li
                          F .text 000000000000016
   0000000000000780 q
   root@ubuntu:/home/disk1/CPP 2/Link# ll |grep liba
   -rwxr-xr-x 1 root root 8149 Jun 14 15:47 libaaa.
   可以看到,加上--whole-archive,生成的动态库,包含所有的。
6、特别需要注意的是:
   链接静态库的最小单元是 o文件,上面的两个方法,分别在两个cr
   如果生成一个o文件呢?
   示例代码
   root@ubuntu:/home/disk1/CPP_2/Link# more test1.cr
   #include "test.h"
   int add(int a, int b)
   {
           return a+b;
   root@ubuntu:/home/disk1/CPP_2/Link# more test2.cr
   #include "test.h"
   int sub(int a, int b)
   {
           return a-b;
   root@ubuntu:/home/disk1/CPP 2/Link# more test.cpr
   #include "test.h"
   int add(int a, int b)
           return a+b;
```

```
int sub(int a, int b)
   {
           return a-b;
   测试如下:
   root@ubuntu:/home/disk1/CPP_2/Link# rm -fr *.o *.
   root@ubuntu:/home/disk1/CPP 2/Link# q++ -c test1.
   root@ubuntu:/home/disk1/CPP 2/Link# q++ -c test2.
   root@ubuntu:/home/disk1/CPP_2/Link# ar -rc libtes
   root@ubuntu:/home/disk1/CPP 2/Link# q++ -o main n
   root@ubuntu:/home/disk1/CPP 2/Link# objdump -t max
   root@ubuntu:/home/disk1/CPP 2/Link#
   root@ubuntu:/home/disk1/CPP 2/Link# rm -fr *.o *.
   root@ubuntu:/home/disk1/CPP_2/Link# g++ -c test.
   root@ubuntu:/home/disk1/CPP 2/Link# ar -rc libtes
   root@ubuntu:/home/disk1/CPP_2/Link# g++ -o main n
   root@ubuntu:/home/disk1/CPP_2/Link# objdump -t ma
                       F .text 0000000000000016
   0000000000400576 q
   也就是说,链接的最小单元是 o文件(目标文件)
7、得出的结论是:
   链接静态库的方法,默认情况下(不加--whole-archive),是
   但是,链接的最小单元是目标文件,需要目标文件中的一个方法,
```

# 参见

# 静态库与动态库的链接

```
测试场景,Test, lib1, lib2, dll1, dll2
分为下面四种情况:
1、Test->lib1->lib2
lib1编译自己的代码,对lib2的部分,只需要lib2的头文件,对
生成Test链接的时候,把lib1的代码实现包含进来,再递归,把l
运行Test的时候,不再需要lib1和lib2。
也就是说,lib2不合并到lib1中,等到exe的时候,一起合并到e
2、Test->lib1->dll2
```

lib1编译自己的代码,对dll2的部分,只需要dll2的头文件,对 生成Test链接的时候,把lib1的代码实现包含进来,但是,dll2 运行Test的时候,不需要lib1,但是需要dll2【不需要dll的li 也就是说,dll2不合并到lib1中,等到exe的时候,把lib1合并

3、Test->dll1->lib2

dll1编译自己的代码,对lib2的部分,需要lib2的头文件和实现 生成Test链接的时候,dll1中的代码不包含进来,也不再需要lib 【可以这样测试,生成lib2,生成dll1,生成test,删除lib2, 可以成功,说明生成test.exe,链接的时候,根本不需要li 也就是说,静态库会被链接到动态库或者exe中,但是不会链 运行Test的时候,需要dll1,但是不需要lib2

也就是说,lib2合并到dll1中,等到exe的时候,不需要合并dl]

4. Test->dll1->dll2

dll1编译自己的代码,对dll2的部分,需要dll2的头文件和lib 生成Test链接的时候,Test需要dll1的lib,dll1需要dll2的] 运行Test的时候,需要dll1和dll2【不需要他们的lib】

也就是说,dll2不合并到dll1中,等到exe的时候,二者都不需要

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 **网络有关** Linux

# 重变

ifconfig
route
tcpdump
traceroute
arp
arping
nc
在我们的系统安装netcat

# 详细描述

# 变量说明

#### arp

查看ip地址对应mac地址的缓存

# arping

1、目的是在局域网内,查找ip地址对应的mac地址,因为在数据链路层 2、-U是请求模式,请求ip地址对应的mac地址,如下: 在10.65.200.89上,发送请求,请求10.65.200.34对应的mac [root@localhost tmp]# arping 10.65.200.34 -s10.65 ARPING 10.65.200.34 from 10.65.200.89 eth0 Unicast reply from 10.65.200.34 [90:02:AA:A9:36:1 Unicast reply from 10.65.200.34 [90:02:AA:A9:36:1 在10.65.200.34抓包,收到请求,并回复 [root@localhost ~]# tcpdump -i any -s 0 host 10.6 tcpdump: verbose output suppressed, use -v or -vv listening on any, link-type LINUX\_SLL (Linux cook 11:26:11.648254 ARP, Request who-has 10.65.200.34 .........0d... A.Y..... A."......... 11:26:11.648307 ARP, Reply 10.65.200.34 is-at 90: . . . . . . . . . . . . 6 . A.".0d... A.Y 11:26:12.645916 ARP, Request who-has 10.65.200.34

	0d
	A.Y6.
	A."
	11:26:12.645962 ARP, Reply 10.65.200.34 is-at 90:
	6 .
	A.".0d
	A.Y
3、	-A是回复模式,主动告诉对方,自己ip地址对应的mac地址,如下:
	在10.65.200.89,发送回复(也就是通知),告诉10.65.200.
	<pre>[root@localhost tmp]# arping 10.65.200.34 -s10.65</pre>
	ARPING 10.65.200.34 from 10.65.200.89 eth0
	^CSent 7 probes (7 broadcast(s))
	Received 0 response(s)
	ナ. 。 。 。 。
	在10.65.200.34,不停地收到,10.65.200.89汇报的mac地址
	[root@localhost ~]# tcpdump -i any -s 0 host 10.6
	tcpdump: verbose output suppressed, use -v or -v
	listening on any, link-type LINUX_SLL (Linux cook
	11:30:51.593318 ARP, Reply 10.65.200.89 is-at 00:
	A.Y.0d
	A."
	11:30:52.591462 ARP, Reply 10.65.200.89 is-at 00:
	A.Y.0d
	A."
4、	-A的使用场景:
	已经知道主机H的ip/mac地址,告诉主机H,自己的ip/mac地址(
	-U的使用场景:
	对于一个ip地址,我不知道它对应的mac地址,因此,我要发送一
	开始的时候是广播,因为不知道谁拥有这个ip地址,一旦收到有人
_	如何保持广播呢?使用-b
5、	-A是告诉某个主机,自己的ip和mac映射。
	-U是请求某个主机的ip和mac映射。
	那么如何通知所有主机,自己的ip和mac映射呢?
	对于-U,当前主机使用广播,请求一个ip地址对应的mac地址,收
	在广播请求的时候,当前主机会把自己的ip/mac地址带着,在当前

利用这个特点,可以在局域网广播自己的ip和mac的映射。 也就是在局域网内广播请求自己的mac地址,局域网内的主机(包扎 6、如何知道是广播呢? 我们知道ip地址分成两部分,前缀是网络部分,后面是主机部分。 7、利用arping的特点,请求自身ip地址对应的mac地址,如果有人回约 8、windows下面没有arping,可以制作一个arping,如下: @echo off :loop ping -n 1 -w 500 %1 >nul for /f "tokens=1,2 delims= " %%i in ('arp -a') do arp -d qoto loop 把上面的内容,保存为arping.bat,在cmd窗口执行arping 10 也可以在批处理中,使用变量定义dstIp,直接运行arping.bat @echo off set dstIp=10.36.65.91 :loop ping -n 1 -w 500 %dstIp% >nul for /f "tokens=1,2 delims= " %%i in ('arp -a') do arp -d

# ifconfig

1、查看网络配置:

ifconfig

goto loop

2、设置网络配置:

ifconfig eth0 10.65.200.52 netmask 255.255.0.0

3、设置多个IP地址:

ifconfig eth0:1 10.65.200.53 netmask 255.255.0.0

特别注意的是,在arping.bat中并没有向10.36.65.91发送arg

#### nc

- 1、nc也叫netcat,用于传输层的端口检测。
- 2、nc既可以作为客户端,也可以作为服务端。 作为客户端需要指定远程的ip和port,也可以指定本地的端口,-

作为服务端使用-1表示监听,需要指定本地的监听端口。

nc支持tcp协议和udp协议,默认使用tcp协议,如果使用udp协议

3、作为tcp客户端,如下:

nc 10.36.65.80 12345

如果要指定本地的ip和port,如下:

nc -s10.36.65.60 -p12346 10.36.65.80 12345

特别注意:如果要指定本地的ip和port,在这种情况下,tcp连接这段时间内端口12346是不能使用的,报错如下:

[root@localhost ~]# nc -s10.36.65.60 -p12346 10.3

nc: bind failed: Address already in use

要过一段时间才能使用端口12346

4、作为tcp的服务端,使用-1 表示监听,如下:

nc -1 10.36.65.60 12345

5、作为udp的客户端,使用-u表示udp,如下:

nc -u 10.36.65.80 12345

特别注意:作为udp的客户端,如果没有指定udp的端口,必须先发服务端收到数据,知道了客户端的udp端口,才能发送发送数据。如果让服务端先发送数据,作为客户端的udp也要指定端口,如下nc-u-p12346 10.36.65.80 12345

6、作为udp的服务端,如下:

nc -1 -u 10.36.65.60 12345

7、扫描端口,如下:

[root@localhost ~]# nc -v -w5 0.0.0.0 -z 20-25

nc: connect to 0.0.0.0 port 20 (tcp) failed: Conr nc: connect to 0.0.0.0 port 21 (tcp) failed: Conr Connection to 0.0.0.0 22 port [tcp/ssh] succeeded nc: connect to 0.0.0.0 port 23 (tcp) failed: Conr nc: connect to 0.0.0.0 port 24 (tcp) failed: Conr Connection to 0.0.0.0 25 port [tcp/smtp] succeeded

#### route

# 查看路由:

route -n

# 设置路由:

route add default gw 10.65.0.1

或者 route add -net 0.0.0.0 netmask 0.0.0.0 gw 10

## 添加路由报错,如下:

[root@localhost ~]# ifconfig eth0 10.65.200.52 ne [root@localhost ~]# route add default gw 10.65.0

SIOCADDRT: Network is unreachable

错误原因是:10.65.0.1是不可达的,为什么不能达?

添加的路由(或者说网关),当前网卡必须能够访问,能够访问必这里的网络掩码是255.255.255.0,对于10.65.200.52同一个

### tcpdump

```
1、tcpdump -i any -s 0 tcp port 9845 -w test.pcap //
```

- 2 tcpdump -i any -s 0 tcp port 9845 -A //
- 3、-s0 获取全部数据包

默认的话tcpdump只显示部分数据包,参数-s snaplen 就是控制要显示全部数据包,使用-s0

- 4、-a是输出交互过程,没有分组的具体内容
- 5、-A以ASCII格式打印所有的分组
- 6、-x表示16进制显示,如下:

[root@localhost ~]# tcpdump -i any -s0 tcp port tcpdump: verbose output suppressed, use -v or -v\ listening on any, link-type LINUX\_SLL (Linux cool 10:29:40.237592 IP 10.36.65.80.italk > 172.16.2.1

0x0000: 4500 007a 1082 4000 3d06 3368 0a

0x0010: ac10 0210 3039 8b93 2d26 270d 9c

0x0020: 8018 0104 7738 0000 0101 080a 0<sup>2</sup>

0x0030: ff52 6789 6162 6364 6566 6761 62

0x0040: 6667 6162 6364 6566 6761 6263 6<sup>2</sup>

0x0050: 6162 6364 6566 6761 6263 6465 66 0x0060: 6364 6566 6761 6263 6465 6667 61

0x0070: 6566 6761 6263 6465 6667

10:29:40.237624 IP 172.16.2.16.35731 > 10.36.65.8

0x0000: 4510 0034 6183 4000 4006 df9c ac

0x0010: 0a24 4150 8b93 3039 9c2e 08f3 2c

0x0020: 8010 00e5 e01f 0000 0101 080a f1

0x0030: 04ee 29f9

7、注意:-v会显示抓到报的个数,这个选项很有用,判断当前已经抓到

#### traceroute

- 1、traceroute程序的设计是利用ICMP及IP header的TTL(Time 1 首先,traceroute送出一个TTL是1的IP datagram (其实,每次送出的为3个40字节的包,包括源地址,目的地址和1 当路径上的第一个路由器(router)收到这个datagram时,检查此时,TTL变为0了,所以该路由器会将此datagram丢掉,并送回(包括发IP包的源地址,IP包的所有内容及路由器的IP地址),t便知道这个路由器存在于这个路径上,接着traceroute 再送出5
- 2、traceroute 每次将送出的datagram的TTL加1来发现另一个路由 这个重复的动作一直持续到某个datagram 抵达目的地。当datag 因为它已是目的地了,那么traceroute如何得知目的地到达了呢
- 3、traceroute在送出UDP datagrams到目的地时,它所选择送达的局导码(30000 以上),所以当此UDP datagram 到达目的地后,而当traceroute 收到这个消息时,便知道目的地已经到达了。凡
- 4、traceroute提取发 ICMP TTL到期消息设备的IP地址并作域名解构 包括所经过的路由,设备的域名及 IP地址,三个包每次来回所花时
- 5、traceroute 有一个固定的时间等待响应(ICMP TTL到期消息)。 它将打印出一系列的\*号表明:在这个路径上,这个设备不能在给员 然后,traceroute给TTL记数器加1,继续进行。

注:没有在给定的时间内发出ICMP TTL到期消息的响应,往往是[

6、知道traceroute的原理,就能明白:

中间任何一个router上如果封了ICMP Echo Request, trace 如果封了type 11 (TTL-expired), 中间的router全看不到如果封了ICMP Echo Reply,中间的全能看到,最后的destina

7、注意:windows下对应的命令是tracert

# 在我们的系统安装netcat

1、netcat也叫nc,安装如下:

installing package netcat-1.10-1.i386 nee

2、报错,在根目录需要276KB,查看目录挂载情况,都没有根目录/ [root@localhost tmp]# df -h Filesystem Size Used Avail Use% Mount

```
tmpfs
                        3.9G
                              244K
                                    3.9G
                                          1% /dev/
   /dev/sda5
                        259M
                                         38% /usr/
                              93M
                                    153M
   /dev/sda3
                                    861G
                        916G
                             9.0G
                                          2% /usr/
   /dev/sda6
                         54M
                              4.9M
                                     46M
                                         10% /usr/
3、怎么解决?
   直接解压出来,如下:
    [root@localhost tmp]# rpm2cpio netcat-1.10-1.i386
   解压后,进入/usr/sbin 当前有
    [root@localhost sbin]# 11
   total 140
    -rwxr-xr-x 1 root root 142308 2016-10-11 11:07 nc
   drwxr-xr-x 2 root root
                              0 2016-10-11 11:07 ne
4、nc只能在当前目录下执行,如何能在其他地方执行
    [root@localhost bin]# echo $PATH
   /usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sk
   因此执行环境没有当前目录,在/usr/local/bin增加一个软连接
    [root@localhost bin]# pwd
   /usr/bin
    [root@localhost bin]# ln -s /root/tmp/usr/sbin/nc
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 进程管理 Linux

## 变量

如何杀掉僵尸进程
孤儿进程与僵尸进程
父进程监听的端口转移到子进程
父进程监听的端口转移到子进程测试
linux程序崩溃,没有产生dmp文件
杀死进程 
proc目录
ps查看进程
pstree查看进程树
pstack查看线程死锁
Linux生成core文件
进程在后台运行
Linux下进程间通信
如何判断一个进程是谁启动的
理解nohup

### 变量说明

#### Linux下进程间通信

- 1、进程间的通信方法有:管道、消息队列、共享内存、信号、信号量、
- 2、管道

管道是进程间通信中最古老的方式,它包括无名管道和有名管道两 前者用于父进程和子进程间的通信,后者用于运行于同一台机器上

3、消息队列

消息队列用于运行于同一台机器上的进程间通信,它和管道很相似,是一个在系统内核中用来保存消息的队列,它在系统内核中是以消事实上,它是一种正逐渐被淘汰的通信方式,我们可以用流管道或;

4、共享内存

共享内存是运行在同一台机器上的进程间通信最快的方式,因为数据 通常由一个进程创建一块共享内存区,其余进程对这块内存区进行前一种方式不给系统带来额外的开销,但在现实中并不常用,因为 在Linux系统下,这只有通过限制Linux系统存取的内存才可以做 常用的方式是通过shmXXX函数族来实现利用共享内存进行存储的。

5、信号

发送信号给某个进程,这个进程接收到信号后,进行相应的处理。

6、信号量

通过PV操作来协调不同进程间的数据对象的,而最主要的应用是共本质上,信号量是一个计数器,它用来记录对某个资源(如共享内)。

7、套接字

套接字(socket)是进程间通信的主要方式,分为网络套接字(TUNIX域套接字解决什么样的问题?

对于CS架构,一般情况下客户端和服务端在不同的机器上,因此,但是有些情况下,客户端和服务端在同一台机器上,二者没有必要UNIX域套接字比网络套接字的效率更高,UNIX域数据报服务是可能实际上,UNIX域套接字是套接字和管道之间的混合物,同时与网络

#### Linux生成core文件

- 1、linux程序崩溃,发生segmentation fault段错误,会把程序运 函数调用堆栈等信息保存为一个coredump文件,通过gdb可以查看
- 2、首先一点,要生成coredump文件,必须设置 ulimit -c unlimi ulimit -c unlimited 表示生成coredump文件,不限制生成能 否则会因为要生成的coredump文件过大,导致系统不生成。
- 3、生成的coredump文件,生成的位置在执行文件的当前目录,如果要是程序可能会多次崩溃,如果每次生成的文件名相同,会导致这一次。
- 4、这就需要设置生成的coredump文件目录和文件名,通过/proc/sysecho "./core.%p.%e" > /proc/sys/kernel/core\_patte选项表示的意义如下:
  - %% 单个%字符
  - %p 所dump进程的进程ID
  - %u 所dump进程的实际用户ID
  - %g 所dump进程的实际组ID
  - %s 导致本次core dump的信号
  - %t core dump的时间 (由1970年1月1日计起的秒数)
  - %h 主机名
  - %e 程序文件名

#### 一个启动脚本如下:

ulimit -c unlimited

ulimit -n 65535

PROXY\_HOME=`dirname \$0`

cd \${PROXY\_HOME}

if [ -d "/mnt/data\_bank" ]

then

echo "/mnt/data\_bank/core.%p.%e" > /proc/sys/
else

echo "./core.%p.%e" > /proc/sys/kernel/core\_
fi

export LD\_LIBRARY\_PATH=./:\$LD\_LIBRARY\_PATH
./vmu\_main -f vmu\_linux.xml

5、注意:在/proc/sys/kernel目录,除了文件core\_pattern,还 [root@localhost kernel]# pwd

```
/proc/sys/kernel
[root@localhost kernel]# 11 |grep core
-rw-r--r-- 1 root root 0 2015-10-21 15:18 core_pa
-rw-r--r-- 1 root root 0 2015-10-21 15:13 core_pi
-rw-r--r-- 1 root root 0 2015-10-21 15:13 core_us
[root@localhost kernel]# more core_pattern
./core.%p.%e
[root@localhost kernel]# more core_pipe_limit
0
[root@localhost kernel]# more core_uses_pid
1
```

- 6、对于文件core\_uses\_pid,如果这个文件的内容被配置成1,那么思最后生成的coredump文件名仍会加上进程ID。
- 7、core\_pipe\_limit和core\_pattern结合使用,当core\_patter 这种情况,coredump不是生成文件,而是作为一个程序的标准输。

当配置成管道core文件模式时,内核通过管道搜集core信息,偶定为了能够安全的获得数据,不能过早的清除崩溃程序的/proc/pic反过来,如果用户空间的一个行为不正确的数据搜集程序从/proc就可能一直阻止内核对/proc/pid的崩溃进程进行回收。

core\_pipe\_limit这个文件定义了, coredump处理程序可以并发的就是有多少个并发的崩溃程序可以通过管道模式传递给指定的co如果超过了指定数,则后续的程序将不会处理,只在内核日志中做0是个特殊的值,当设置为0时,不限制并行捕捉崩溃的进程,但不(就是说,崩溃程序的相关信息可能随时被回收,搜集的信息可能

#### linux程序崩溃,没有产生dmp文件

- 1、一般是没有设置 ulimit -c unlimited
- 2、如果设置了 ulimit -c unlimited,程序崩溃,可执行程序的当
- 3、如果目录有写文件的权限,则很大概率是生成在其它目录。

检查文件 /proc/sys/kernel/core\_pattern

[root@HikvisionOS VTDUTest]# more /proc/sys/kerne
/opt/components\_core/core-%e-%p-%t

4、通过下面的信息,确认崩溃如下:

[root@HikvisionOS niuzb]# cat /var/log/messages |

```
Jan 29 14:52:38 localhost kernel: traps: main[101 [root@HikvisionOS niuzb]# ll /opt/components_core-rw----- 1 root root 413696 1月 29 14:52 c
```

#### proc目录

```
1、proc文件系统是伪文件系统,不存在硬盘上,而是在内存中。它以为
    允许调用者访问系统内核数据。
2、比如查看cpu信息,如下:
    [root@localhost proc]# more /proc/cpuinfo
    processor
                    : 0
   vendor_id
                    : GenuineIntel
   cpu family
                    : 6
   model
                    : 60
                  : Intel(R) Core(TM) i7-4712MQ CPL
   model name
                    : 3
   stepping
   cpu MHz
                    : 2294.740
   cache size
                   : 6144 KB
   fdiv_bug
                    : no
   hlt bug
                   : no
   f00f bug
                  : no
   coma buq
                 : no
                    : yes
   fpu
   fpu exception
                    : yes
   cpuid level
                    : 13
                    : yes
   wp
                    : fpu vme de pse tsc msr pae mce
   flags
   sc up ida nonstop tsc arat pni ssse3 sse4 1 sse4
   bogomips
                    : 4589.48
3、查看进程信息,需要使用进程Id,如下:
    [root@localhost proc]# pgrep cmu
   8155
    [root@localhost proc]# 11 ./8155
    total 0
   dr-xr-xr-x 2 root root 0 Aug 28 11:02 attr
    -r----- 1 root root 0 Aug 28 11:02 auxv
-r--r-- 1 root root 0 Aug 28 11:01 cmdline
```

```
-rw-r--r-- 1 root root 0 Aug 28 11:02 coredump_1
-r--r-- 1 root root 0 Aug 28 11:02 cpuset
           1 root root 0 Aug 28 11:02 cwd -> /us
lrwxrwxrwx
           1 root root 0 Aug 28 11:02 environ
-r-----
lrwxrwxrwx 1 root root 0 Aug 28 11:02 exe -> /us
           2 root root 0 Aug 28 11:01 fd
dr-x----
           1 root root 0 Aug 28 11:02 io
-r--r--r--
           1 root root 0 Aug 28 11:02 limits
-r-----
           1 root root 0 Aug 28 11:02 loginuid
-rw-r--r--
-r--r-- 1 root root 0 Aug 28 11:02 maps
           1 root root 0 Aug 28 11:02 mem
-rw-----
           1 root root 0 Aug 28 11:01 mounts
-r--r--r--
           1 root root 0 Aug 28 11:02 mountstats
-r-----
           1 root root 0 Aug 28 11:02 oom_adj
-rw-r--r--
           1 root root 0 Aug 28 11:02 oom_score
-r--r--r--
lrwxrwxrwx 1 root root 0 Aug 28 11:02 root -> /
-r--r-- 1 root root 0 Aug 28 11:02 schedstat
-r--r-- 1 root root 0 Aug 28 11:02 smaps
-r--r-- 1 root root 0 Aug 28 11:01 stat
-r--r-- 1 root root 0 Aug 28 11:02 statm
-r--r-- 1 root root 0 Aug 28 11:01 status
dr-xr-xr-x 10 root root 0 Aug 28 11:02 task
-r--r-- 1 root root 0 Aug 28 11:01 wchan
```

#### pstack查看线程死锁

```
1、测试代码如下:
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>

pthread_mutex_t mutexA;
pthread_mutex_t mutexB;

void* thread_func1(void*)
{
```

```
pthread_mutex_lock(&mutexA);
        printf("thread1 lock A\n");
        sleep(5);
        {
                pthread_mutex_lock(&mutexB);
                printf("thread1 lock B\n");
                sleep(5);
                pthread_mutex_unlock(&mutexB);
        }
        pthread_mutex_unlock(&mutexA);
void* thread_func2(void*)
{
        pthread_mutex_lock(&mutexB);
        printf("thread2 lock B\n");
        sleep(5);
        {
                pthread_mutex_lock(&mutexA);
                printf("thread2 lock A\n");
                sleep(5);
                pthread_mutex_unlock(&mutexA);
        }
        pthread_mutex_unlock(&mutexB);
}
int main()
{
        pthread_t thread1;
        pthread_t thread2;
        pthread_mutex_init(&mutexA, NULL);
        pthread_mutex_init(&mutexB, NULL);
```

```
if(pthread_create(&thread1, NULL, thread_f)
            {
                    printf("create thread1 error !\n'
                    exit(1);
            }
            if(pthread_create(&thread2, NULL, thread_fi
                    printf("create thread2 error!\n")
                    exit(1);
            }
            pthread_join(thread1, NULL);
            pthread_join(thread2, NULL);
            pthread_mutex_destroy(&mutexA);
            pthread_mutex_destroy(&mutexB);
            return 0;
2、编译
    g++ -o main main.cpp -lpthread
3、运行,使用pstack查看,每个线程的调用堆栈。
    [root@localhost ~]# pstack 26438
    Thread 3 (Thread 0xb7f4db90 (LWP 26439)):
        0x00f53402 in __kernel_vsyscall ()
    #0
        0x0084c6e9 in __lll_lock_wait () from /lib/li
    #1
        0x00847d9f in _L_lock_885 () from /lib/libpth
    #2
    #3
        0x00847c66 in pthread_mutex_lock () from /lik
        0x08048822 in thread_func1(void*) ()
    #4
        0x00845832 in start_thread () from /lib/libpt
    #5
    #6
        0x0079ae0e in clone () from /lib/libc.so.6
    Thread 2 (Thread 0xb754cb90 (LWP 26440)):
    #0
        0x00f53402 in __kernel_vsyscall ()
        0x0084c6e9 in __lll_lock_wait () from /lib/li
    #1
        0x00847d9f in L lock 885 () from /lib/libpth
    #2
    #3
        0x00847c66 in pthread mutex lock () from /lik
```

```
#4 0x080487ba in thread_func2(void*) ()
#5 0x00845832 in start_thread () from /lib/libpt
#6 0x0079ae0e in clone () from /lib/libc.so.6
Thread 1 (Thread 0xb7f4e6d0 (LWP 26438)):
#0 0x00f53402 in __kernel_vsyscall ()
#1 0x00846a77 in pthread_join () from /lib/libpt
#2 0x0804874a in main ()
4、一共3个线程,每个线程从下到上,显示调用堆栈,可以看到Threac
```

#### pstree查看进程树

```
1、pstree查看进程树,如下:
    [root@localhost ~]# pstree
    init——atd
         -auditd----{auditd}
          -crond
         -dbus-daemon
         —ibpctrl——hau_main——5*[{hau_main}]
         -ibpctrl---dmu_main---27*[{dmu_main}]
         -ibpctrl---vtdu_main---19*[{vtdu_main}]
         -ibpctrl-vru_main-24*[{vru_main}]
         -ibpctrl---uas_main---14*[{uas_main}]
         -ibpctrl---lku_main---7*[{lku_main}]
         -ibpctrl--cmu_main--5*[{cmu_main}]
         -ibpctrl---uac_main---26*[{uac_main}]
         -ibplogbk.sh---sleep
         -java---93*[{java}]
         -login---bash
         -5*[mingetty]
         -mysqld_safe---mysqld---13*[{mysqld}]
         -rsyslogd---3*[{rsyslogd}]
         -sshd<del>---</del>5*[sshd----bash]
                -sshd---bash---7*[tcpdump]
-sshd---bash---pstree
         -udevd---2*[udevd]
        ∟watch dog
2、查看进程已经子进程,如下:
```

```
[root@localhost init.d]# pstree -p 3677
sshd(3677)—sshd(10819)—bash(9700)—ping(1171)
                       -bash(10822)
                        -bash(13914)----pstree(11
注意:p后面必须有一个空格
```

```
ps查看进程
1、ps列出进程,常用的两个选项为 aux 和 -ef
2、使用aux
    [root@localhost ~]# ps aux|grep main
                            3808
    root
             2972 0.0 0.0
                                  716 pts/3
                                             SH
    root
                  0.6 0.1
                                             S]
            15577
                           85148
                                 3964 ?
                  3.0 0.4 317540 16952 ?
            19929
                                             S]
    root
                                             S]
                  1.7 0.4 226520 15824 ?
    root
            19989
            20032 1.1 0.4 281116 14948 ?
                                             S]
    root
                                             S]
                  0.3 0.5 171728 19020 ?
    root
            20133
                  0.7 0.3 102256 13064 ?
                                             S]
            20174
    root
            20379
                  0.4 0.4 81384 16220 ?
                                             S]
    root
                  2.8
                                             S]
    root
            27523
                      0.7 307272 26228 ?
    表示意思分别如下:
    USER:该 process 属于那个使用者账号的
    PID :该 process 的号码
    %CPU:该 process 使用掉的 CPU 资源百分比
    %MEM:该 process 所占用的物理内存百分比
    VSZ :该 process 使用掉的虚拟内存量 (Kbytes)
    RSS :该 process 占用的固定的内存量 (Kbytes)
    TTY :该 process 是在那个终端机上面运作,若与终端机无关
           若为 pts/0 等等的,则表示为由网络连接进主机的程,
    STAT:该程序目前的状态,主要的状态有
```

R:该程序目前正在运作,或者是可被运作

S:该程序目前正在睡眠当中 (可说是 idle 状态), (

T:该程序目前正在侦测或者是停止了

Z:该程序应该已经终止,但是其父程序却无法正常的终

START:该 process 被触发启动的时间

TIME :该 process 实际使用 CPU 运作的时间

COMMAND:该程序的实际指令

```
3、使用-ef,可以查看父子进程之间的关系
   [root@localhost ~]# ps -ef|grep main
            6397 29980
   root
                       0 20:08 pts/3
                                      00:00:00
   root
           15577 15574 0 Oct19 ?
                                      00:10:32
           19929 19926 3 17:26 ?
   root
                                      00:04:58
           19989 19986 1 17:26 ?
                                      00:02:49
   root
   root
           20032 20029 1 17:26 ?
                                      00:01:52
           20133 20130 0 17:26 ?
                                      00:00:29
   root
   root 20174 20171 0 17:26 ?
                                      00:01:16
   root
           20379 20376 0 17:26 ?
                                      00:00:47
           27523 27520 2 17:38 ?
                                      00:04:14
   root
4、查看进程启动时间和运行时间,如下:
   先找到进程pid:
   [root@localhost ~]# ps aux|grep ping
                                 668 pts/2
            3131 0.0 0.0
                           3808
                                             SH
   root
           18421 0.0 0.0
                           3812 740 pts/3
                                             SH
   root
   然后列出启动时间和运行时间:
   [root@localhost ~]# ps -eo pid,lstart,etime|grep
   3131 Sun Sep 18 16:20:25 2016 16:07
   [root@localhost ~]# date
   Sun Sep 18 16:36:33 CST 2016
   运行时间为16:07,也就是当前时间【16:36:33】减去启动时间
```

#### 如何判断一个进程是谁启动的

- 1、进程A启动进程B,进程A就是进程B的父进程。
- 2、使用ps -ef|grep dmu列出父子进程之间的关系,就可以找到一个

#### 如何杀掉僵尸进程

- 1、僵尸进程是指:杀掉了当前进程,但是父进程没有等待wait,导致⅓
- 2、僵尸进程已经不是一个正常的进程了,只关联一些进程描述符,因此
- 3、注意:僵尸的父进程还是存在的,只不过它没有等待僵尸进程。 因此要杀掉僵尸进程,解决办法是杀掉它的父进程。

#### 孤儿讲程与僵尸讲程

- 1、我们知道在unix/linux中,正常情况下,子进程是通过父进程创建子进程的结束和父进程的运行是一个异步过程,即父进程永远无法当一个进程完成它的工作终止之后,它的父进程需要调用wait() 項
- 2、孤儿进程:一个父进程退出,而它的一个或多个子进程还在运行,那 孤儿进程将被init进程(进程号为1)所收养,并由init进程对它们
- 3、僵尸进程:一个进程使用fork创建子进程,如果子进程退出,而父过那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸
- 4、unix/linux提供了一种机制可以保证只要父进程想知道子进程结束 在每个进程退出的时候,内核释放该进程所有的资源,包括打开的 ID,退出状态the termination status of the process, 直到父进程通过wait/waitpid来取时才释放。
- 5、但这样就导致了问题,如果进程不调用wait/waitpid的话,那么一但是系统所能使用的进程号是有限的,如果大量的产生僵尸进程,是此即为僵尸进程的危害,应当避免。
- 6、孤儿进程是没有父进程的进程,孤儿进程这个重任就落到了init进程 每当出现一个孤儿进程的时候,内核就把孤儿进程的父进程设置为 这样,当一个孤儿进程凄凉地结束了其生命周期的时候,init进程
- 7、也就是说,孤儿进程没有危害,因为init会管理。僵尸进程有危害, 等待父进程处理,进程描述符一直没有释放。系统所能使用的进程:
- 8、任何一个子进程(init除外)在exit()之后,并非马上就消失掉,而这是每个子进程在结束时都要经过的阶段。如果子进程在exit()之如果父进程能及时处理,可能用ps命令就来不及看到子进程的僵尸
- 9、如果父进程在子进程结束之前退出,则子进程将由init接管。init
- 10、僵尸进程危害场景:

例如有个进程,它定期的产生一个子进程,这个子进程需要做的引因此这个子进程的生命周期很短。但是,父进程只管生成新的子进这样,系统运行上一段时间之后,系统中就会存在很多的僵尸进程,严格地来说,僵尸进程并不是问题的根源,罪魁祸首是产生出大量因此,当我们寻求如何消灭系统中大量的僵尸进程时,答案就是把(也就是通过kill发送SIGTERM或者SIGKILL信号啦)。枪毙了这些孤儿进程会被init进程接管,init进程会wait()这些孤儿进这样,这些已经僵尸的孤儿进程就能瞑目而去了。

11、测试孤儿进程,如下:

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <unistd.h>

```
int main()
   {
       pid_t pid;
       //创建一个进程
       pid = fork();
       //创建失败
       if (pid < 0)
       {
           perror("fork error:");
           exit(1);
       }
       //子进程
       if (pid == 0)
       {
           printf("I am the child process.\n");
           //输出进程ID和父进程ID
           printf("pid: %d\tppid:%d\n",getpid(),getr
           printf("I will sleep five seconds.\n");
           //睡眠5s,保证父进程先退出
           sleep(5);
           printf("pid: %d\tppid:%d\n", getpid(), get;
           printf("child process is exited.\n");
       }
       //父进程
       else
       {
           printf("I am father process.\n");
           //父进程睡眠1s,保证子进程输出进程id
           sleep(1);
           printf("father process is exited.\n");
       return 0;
12、测试僵尸进程,如下:
   #include <stdio.h>
   #include <unistd.h>
```

```
#include <errno.h>
    #include <stdlib.h>
    int main()
    {
       pid_t pid;
        pid = fork();
       if (pid < 0)
        {
           perror("fork error:");
           exit(1);
        }
       else if (pid == 0)
        {
           printf("I am child process.I am exiting.")
           exit(0);
        printf("I am father process.I will sleep two
        //等待子进程先退出
       sleep(2);
       //输出进程信息
        system("ps -o pid,ppid,state,tty,command");
        printf("father process is exiting.\n");
        return 0;
13、父进程循环创建子进程,子进程退出,造成多个僵尸进程。
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include <errno.h>
    int main()
    {
       pid_t pid;
       //循环创建子进程
       while(1)
```

```
pid = fork();
           if (pid < 0)
           {
               perror("fork error:");
               exit(1);
           else if (pid == 0)
               printf("I am a child process.\nI am \epsilon
               //子进程退出,成为僵尸进程
               exit(0);
           }
           else
           {
               //父进程休眠20s继续创建子进程
               sleep(20);
               continue;
           }
       return 0;
14、怎么解决僵尸进程?
15、通过信号机制,子进程退出时向父进程发送SIGCHILD信号,父进程
   #include <stdio.h>
   #include <unistd.h>
   #include <errno.h>
   #include <stdlib.h>
   #include <signal.h>
   #include <sys/types.h>
   #include <sys/wait.h>
   static void sig_child(int signo);
   int main()
    {
       pid_t pid;
       //创建捕捉子进程退出信号
```

```
signal(SIGCHLD, sig_child);
       pid = fork();
       if (pid < 0)
       {
           perror("fork error:");
           exit(1);
       else if (pid == 0)
       {
           printf("I am child process,pid id %d.I an
           exit(0);
       printf("I am father process.I will sleep two
       //等待子进程先退出
       sleep(2);
       //输出进程信息
       system("ps -o pid,ppid,state,tty,command");
       printf("father process is exiting.\n");
       return 0;
   }
   static void sig_child(int signo)
    {
       pid_t
                    pid;
       int
                  stat;
       //处理僵尸进程
       while ((pid = waitpid(-1, &stat, WNOHANG)) > (
       {
           printf("child %d terminated.\n", pid);
       }
16、fork两次,原理是A进程fork出来B进程,同时在B进程中fork出表
   这样的话,C,D,E等进程进程成为孤儿进程,由init来管理。
   #include <stdio.h>
   #include <stdlib.h>
   #include <unistd.h>
   #include <errno.h>
```

```
#include <sys/types.h>
#include <sys/wait.h>
int main()
{
   pid_t pid;
   //创建第一个子进程
   pid = fork();
   if (pid < 0)
   {
       perror("fork error:");
       exit(1);
   }
   //第一个子进程
   else if (pid == 0)
   {
       //子进程再创建子进程
       printf("I am the first child process.pid:
       pid = fork();
       if (pid < 0)
       {
           perror("fork error:");
           exit(1);
       //第一个子进程退出
       else if (pid >0)
       {
           printf("first procee is exited.\n");
           exit(0);
       //第二个子进程
       //睡眠3s保证第一个子进程退出,这样第二个子进程的
       sleep(3);
       printf("I am the second child process.pic
       exit(0);
   //父进程处理第一个子进程退出
```

```
if (waitpid(pid, NULL, 0) != pid)
{
        perror("waitepid error:");
        exit(1);
    }
    exit(0);
    return 0;
}
```

#### 杀死进程

```
1、ps 结合管道,使用grep找到pid,kill -9 pid 强制杀死进程,
   [root@localhost lib_linux]# ps aux|grep cmu
                  0.0 0.1
   root
            4971
                            4492
                                  792 pts/2
                                              S
                  0.4
                      2.3
                                              S]
                           97156 12156 pts/2
   root
            4973
            4988
                  0.0
                      0.1
                            3920
                                              R⊦
   root
                                  708 pts/2
   [root@localhost lib_linux]# kill -9 4973
   [root@localhost lib_linux]# ps aux|grep cmu
   root
            5004 0.0
                      0.1
                            3920
                                  688 pts/2
                                              R⊦
   注:kill的工作原理是,向Linux系统的内核发送一个系统操作信
   然后系统内核就可以对指定的进程进行操作。通常终止前台进程我们
2、使用pidof 找到进程id,然后杀死,需要注意的是,pidof 必须约
   [root@localhost lib linux]# pidof cmu main
   5549
   [root@localhost lib_linux]# kill -9 5549
3、使用pgrep直接找到pid,杀死进程,pgrep就是 process grep
   [root@localhost lib linux]# pgrep cmu
   5240
   [root@localhost lib linux]# kill -9 5240
   [root@localhost lib_linux]# pgrep cmu
4、使用xargs对找到pid直接杀死,如下:
   [root@localhost lib_linux]# pgrep mu
   6175
   6212
   [root@localhost lib_linux]# pgrep mu|xargs kill -
   [root@localhost lib_linux]# pgrep mu
   注意:xargs相当于遍历前面的输出,作为下一个命令的输入,逐
```

5、使用pkill, pkill=pgrep+kill, 根据进程名称, 杀死进程,如<sup>-</sup> [root@localhost lib\_linux]# pgrep dmu
7407

[root@localhost lib\_linux]# pkill -9 dmu
[root@localhost lib\_linux]# pgrep dmu

6、使用killall,根据进程名称杀死进程,如下:

[root@localhost lib\_linux]# pgrep dmu
7453

[root@localhost lib\_linux]# killall -9 dmu

dmu: no process killed

[root@localhost lib\_linux]# killall -9 dmu\_main

[root@localhost lib\_linux]# pgrep dmu

需要注意的是:和pidof类似,使用killall必须给出进程的全名

#### 父进程监听的端口转移到子进程

#### 1、存在的问题:

virgo调用rpu\_sdk接口启动其它服务(比如cmu),rpu\_sdk的 当关闭virgo之后,virgo的监听端口9875转移到脚本进程上(即 9875端口被占用,导致virgo不能启动。

我在linux下模拟了场景,测试结果一样。

#### 2、原因是:

rpu\_sdk使用system或者popen来执行脚本,system或者poper问题出在fork,fork后子进程会对父进程的数据段,堆和栈信息(注意:由于子进程fork之后经常跟着exec执行另一个程序,从了对于文件描述符,fork的一个特性是父进程所有打开的文件描述符对于父子进程中相同的文件描述符,复制了FD标志和文件表指针,

对于从父进程复制过来的文件描述符,在子进程中可以使用,接着这个时候kill掉父进程,子进程成为孤儿进程,由init管理,但是表现为父进程的文件描述符转移到子进程上面。

3、怎么解决这个问题?

在fork子进程之后, exec之前, 在子进程上, close关闭掉无用的

4、上面的方法存在一个问题,在复杂的系统中,fork子进程的时候,不我们期望的是能在fork子进程前,也就是打开某个文件的时候,指这就是所谓的 close-on-exec,意为如果对文件描述符设置了F□此描述符被关闭,不能再使用它。但是在使用fork调用的子进程中

- 5、close-on-exec的功能,只需要调用系统的fcntl就能实现,如下int fd=open("foo.txt",O\_RDONLY);
  int flags = fcntl(fd, F\_GETFD);
  flags |= FD\_CLOEXEC;
  fcntl(fd, F\_SETFD, flags);
  这样,当fork子进程后,仍然可以使用fd。但执行exec后系统就
- 6、但是我们的场景是, virgo调用rpu\_sdk, rpu\_sdk并不知道监听9 文件描述符一般从3开始, 因为按照惯例0, 1, 2分别是标准输入, 因此遍历到1000, 每个数值FD都执行一把fcntl

#### 父进程监听的端口转移到子进程 测试

1、测试代码: linux\_socket.h

/\*

\* 文件功能: linux 下套接字简化操作函数

\* 文件名称: linux\_socket.h

\* 建立时间: 2007 年 07 月 19 号

\* 创建作者: wlzqi

\* 使用语言: C 或 C++ 语言

\* 使用环境: Linux + Windows

\* 函数要求:

- \* + 函数相对比较底层
- \* + 只使用系统API和C库,不能使用任何第三方库
- \* + 不可以用全局变量
- \* + 使用比较频繁
- \* + 具有模块化(函数不要嵌套)
- \* + 要有足够强壮性和高效
- \* + 要经过一定强度的本地测试
- \* + 尽量不使用动态分配内存(特殊情况可慎重添加)
- \* + 所有变量必须字节对齐
- \* 代码要求:
- \* + 尽量减少临时变量
- \* # 算法要精炼
- ' + 临时变量名要使用英文或英文缩写或英文词组单词首字母 , 并」
- \* + 函数名要求将函数功能描述清楚,单词之间要用 \_ 连接。例如

- execute\_sql 执行SQL语句、get\_database\_field\_na
- + 所有函数均使用小写字母拼写
- + 符号 ,和; 后要有一空格、符号 & \* 要紧挨右边的变量、符 两边都要留有空格
  - + if 和 for 、where 等要和紧挨的 '(' 符号间留有一空格
- + 合理运用空行使代码清晰易读
- 注释风格:
  - + 注释要得当风格要统一
- + 注释只能在代码之上
- + 如果注释和代码同行,则要求必须空出两个 Tab 键
- + 注释内容和注释符之间要有一空格
- + 如果注释要分行写则要求具有以下样子
  - + 函数功能注释必须有:函数说明、参数说明、返回说明
  - \* 注意事项:
  - + 如遇本文件中的函数与系统函数功能相同时,因优先考
  - + '必须'保证编译时不出现警告信息
  - + 缺省情况下,给一个已经断开连接(非法断开)的地址;
    - 缺省设置。建议应用根据需要处理 SIGPIPE 信号,至少
  - ,系统缺省的处理方式是退出进程,这样你的应用就很难
  - 如果调用 signal(SIGPIPE, SIG\_IGN); 那么程序在
  - 如果调用 signal(SIGPIPE, function\_name); 那
  - 如果在发送数据中对方断开,那么发送端会先返回已发送
    - 建议使用 signal 处理信号,避免进程莫名退出和描述
  - 已加模块:
  - + socket\_format\_err 格式化 socket 错误信息
  - + socket\_format\_herr 自定义格式化 socket 错误

  - + socket\_connect 连接 Socket
  - + socket\_bind 邦定本地端口
  - + socket\_listen 监听本地端口描述符
  - + socket\_accept 接受 Socket 连线 + socket\_send 发送数据包

  - + socket\_recv 接收数据包

  - + socket\_send\_pack 打包发送数据包 + socket\_recv\_pack 接收打包的数据包
  - + close\_socket 关闭套接字

```
#ifndef LINUX_SOCKET_H_
#define LINUX SOCKET H
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <signal.h>
#include <stdio.h>
     /*-----
     /* 定义 SOCKET 变量 */
     typedef int SOCKET;
#ifndef boolean
#define boolean
     /* 定义布尔型变量 */
#endif
#ifndef FALSE
#define FALSE (0)
#endif
#ifndef TRUE
#define TRUE (!FALSE)
#endif
     /* 定义布尔型变量 */
#ifndef BOOLLEAN
#define BOOLLEAN
     typedef int BOOL;
```

```
#endif
     /* 定义NULL字符 */
#ifndef NULL
#define NULL 0
#endif
     /* 函数功能: 格式化 socket 错误信息
     * 参数说明: 无 * 返回说明: 返回错误描述信息文本
     * 注意事项:
     * 使用举例:
     * */
     char * socket_format_err()
     {
        return (char *)hstrerror(h_errno);
     }
     /* 函数功能: 自定义格式化 socket 错误信息
     * 参数说明: pszErr 要在系统错误信息描述前添加的自己的
     * 返回说明: 无
     * 注意事项:
     * 使用举例: socket_format_herr("Warning");
          输出 Warning: xxxx.....
     * */
     void socket_format_herr(const char * pszErr)
     {
        herror(pszErr);
     /* 函数功能: 创建 Socket
     * 参数说明: nTimeR 接收超时(秒),0表示不受限
            nTimeS 发送超时(秒),0表示不受限
```

```
* 返回说明: 返回-1表示失败。否则返回被创建的socket
     * 注意事项: 函数内使用了端口重新绑定
     * 使用举例: 1. socket_create(AF_INET, SOCK_STRE
             2. socket create(AF INET, SOCK DGRAM,
      * */
     SOCKET socket create(const int nAf, const int r
         static SOCKET m sock;
#ifndef WIN32
         struct timeval tv;
#endif
         m sock = -1;
         m_sock = socket(nAf, nType, nProtocol);
         if (m sock == -1) return -1;
#ifdef WIN32
         nTimeS *= 1000;
         nTimeR *= 1000;
         /* 发送时限 */
         if (setsockopt(m_sock, SOL_SOCKET, SO_SNDT)
             return -1;
         /* 接收时限 */
         if (setsockopt(m_sock, SOL_SOCKET, SO_RCVT)
             return -1;
#else
         tv.tv\_usec = 0;
         tv.tv_sec = nTimeS;
         /* 发送时限 */
         if (setsockopt(m_sock, SOL_SOCKET, SO_SNDT)
             return -1;
         tv.tv_sec = nTimeR;
         /* 接收时限 */
         if (setsockopt(m_sock, SOL_SOCKET, SO_RCVT]
```

```
return -1;
#endif
         return m_sock;
     }
       函数功能: 连接 Socket
      参数说明: socket 套接字
              pcszIp
                      ΙP
              nPort 端口
     * 返回说明: 返回 false 表示失败。
     * 注意事项:
     * 使用举例: 1. SOCKET socke;
            if ((socke = socket_create(AF_INET, SOCk))
             return false;
            if (socket_connect(socke, "192.168.12.11
     bool socket_connect(const SOCKET socket, const
     {
         struct sockaddr_in svraddr;
         svraddr.sin_family = AF_INET;
         svraddr.sin_addr.s_addr = inet_addr(pcszIp)
         svraddr.sin_port = htons(nPort);
         if (connect(socket, (struct sockaddr *)&svr
         return true;
     /* 函数功能:邦定本地端口到描述符
     * 参数说明:nPort 需要邦定的本地端口
     * 返回说明:返回 false 表示失败。
     * 注意事项:
     * 使用举例:
           if ((socke = socket_create(AF_INET, SOCK_
```

```
return false;
     if (socket_bind(socke, 9000) == false) re
bool socket_bind(const SOCKET socket, const uns
{
    int nOpt = 1;
    struct sockaddr_in svraddr;
    svraddr.sin_family = AF_INET;
    svraddr.sin_addr.s_addr = INADDR_ANY;
    svraddr.sin port = htons(nPort);
   if (setsockopt(socket, SOL_SOCKET, SO_REUSE
    if (bind(socket, (struct sockaddr*)&svraddr
    return true;
/* 函数功能:监听本地端口描述符
* 参数说明:nBacklog 设置请求排队的最大长度,常用量为
* 返回说明:返回 false 表示失败。
* 注意事项:
* 使用举例:
     if ((socke = socket_create(AF_INET, SOCK_
       return false;
      }
     if (socket_bind(socke, 9000) == false) re
     if (socket_listen(socke, 5) == false) ret
* */
bool socket_listen(const SOCKET socket, const i
{
   if (listen(socket, nBacklog) == -1) return
    return true;
```

```
/* 函数功能:接受 Socket 连线
* 参数说明:pszCliIp [OUT] 对方IP地址
* 返回说明:成功返回客户 socket 描述符,否则返回 -1。
* 注意事项:
* 使用举例:
     SOCKET m_sock;
     char szCliIP[16];
     if ((socke = socket_create(AF_INET, SOCK_
       return false;
     if (socket bind(socke, 9000) == false) re
     if (socket_listen(socke, 5) == false) ret
     memset(szCliIP, 0, 16);
     m_sock = socket_accept(socke, szCliIP);
* */
SOCKET socket_accept(const SOCKET socket, char
{
   static SOCKET m_sock;
   struct sockaddr_in cliaddr;
   socklen_t addrlen = sizeof(cliaddr);
   m_sock = accept(socket, (struct sockaddr*)&
   if (m_sock == -1) return -1;
   if (pszCliIp != NULL) sprintf(pszCliIp, "%s
   return m_sock;
}
/* 函数功能: 发送 Socket 包
* 参数说明: socket 套接字
        pszBuff 待发送缓冲区
        nLen 待发送包长度
* 返回说明: 返回-1表示失败。否则返回实际已发送的字节数
* 注意事项: nFlags一般取 0
* 使用举例: 1. SOCKET socke;
      char szBuff[512];
```

```
memset(szBuff, 0, 512);
      if ((socke = socket_create(AF_INET, SOCH))
       return false;
      if (socket_connect(socke, "192.168.12.11
      // 拷贝待发送字节到缓冲区
      socket_send(socke, szBuff, strlen(szBuff
int socket_send(const SOCKET socket, const char
{
   int nBytes = 0;
   int nCount = 0;
   while (nCount < nLen) {</pre>
       nBytes = send(socket, pszBuff + nCount,
       if (nBytes \le 0) {
           return -1;
       nCount += nBytes;
   }
   return nCount;
}
  函数功能: 接收 Socket 包
 参数说明: socket
                  套接字
     pszBuff 待接收缓冲区
            待接收包长度
     nLen
* 返回说明: 返回-1表示失败。否则返回实际接收的字节数
* 注意事项: nFlags一般取 0
* 使用举例: 1. SOCKET socke;
      char szBuff[512];
      memset(szBuff, 0, 512);
```

```
if ((socke = socket_create(AF_INET, SOCk))
       return false;
      if (socket_connect(socke, "192.168.12.11
      // 拷贝待发送字节到缓冲区
      socket_send(socke, szBuff, strlen(szBuff
      socket_recv(socke, szBuff, strlen(szBuf1
int socket_recv(const SOCKET socket, char * psz
{
   return recv(socket, pszBuff, nLen, nFlags);
/* 函数功能: 打包发送 Socket 包
* 参数说明: socket
                  套接字
     pszBuff 待发送缓冲区
     nLength_all 待发送包总长度
     unPack Size 一次发送的字节数
* 返回说明: 返回-1表示失败。否则返回实际发送的字节数
* 注意事项:
* 使用举例: 1. SOCKET socke;
      char szBuff[512];
      memset(szBuff, 0, 512);
      if ((socke = socket_create(AF_INET, SOC))
       return false;
      if (socket_connect(socke, "192.168.12.11
      // 拷贝待发送字节到缓冲区
      socket_send_pack(socke, szBuff, strlen(s
unsigned long socket_send_pack(const SOCKET soc
   int nLength = 0;
```

```
long nAddLengths = 0;
   while(nAddLengths < nLength_all) {</pre>
       if ((nLength = socket_send(socket, pszE
           break;
       nAddLengths += nLength;
   return nAddLengths;
}
  函数功能: 接收打包 Socket 包
* 参数说明: socket 套接字
     pszBuff 待接收缓冲区
     nLength_all 待接收包总长度
     unPack_Size 一次接收的字节数
* 返回说明: 返回-1表示失败。否则返回实际接收的字节数
* 注意事项:
* 使用举例: 1. SOCKET socke;
      char szBuff[512];
      memset(szBuff, 0, 512);
      if ((socke = socket_create(AF_INET, SOC))
       return false;
      if (socket_connect(socke, "192.168.12.11
      // 拷贝待发送字节到缓冲区
      socket_send_pack(socke, szBuff, strlen(s
      socket_recv_pack(socke, szBuff, 2048, 51
int socket_recv_pack(const SOCKET socket, char
{
   int nLength = 0;
```

```
unsigned long nAddLengths;
   nAddLengths = 0;
   while (nAddLengths < nLength_all) {</pre>
       if ((nLength = socket_recv(socket, pszE
           break;
       nAddLengths += nLength;
   return nAddLengths;
                 /* 函数功能: 关闭 Socket
* 参数说明: socket 套接字
* 返回说明: 返回false表示失败。否则返回实际接收的字节:
* 注意事项:
* 使用举例: 1. SOCKET socke;
      char szBuff[512];
      memset(szBuff, 0, 512);
      if ((socke = socket_create(AF_INET, SOCk))
       return false;
      if (socket_connect(socke, "192.168.12.11
      // 拷贝待发送字节到缓冲区
      socket_send_pack(socke, szBuff, strlen(s
      socket_recv_pack(socke, szBuff, 2048, 51
      socket_close(socke);
bool socket_close(const SOCKET socket)
{
   return close(socket) == 0 ? true : false;
```

```
/*-----#endif /*LINUX_SOCKET_H_*/
```

main.cpp

```
int main(int argc, char* argv[])
{
    SOCKET servSocket = socket_create(AF_INET, SOCK_S
    if(servSocket == -1)
    {
        return -1;
    }
    if(socket_bind(servSocket, 9000) == false)
    {
        return -1;
    }
    if(socket_listen(servSocket, 5) == false)
    {
        return -1;
    }
    system("./test.sh");
    getchar();
    return 0;
}
```

test.sh

```
#! /bin/sh
ping 10.36.65.80
```

2、测试

编译运行

```
[root@localhost socket]# g++ -o main main.cpp
[root@localhost socket]# ./main
查看监听端口转移
[root@localhost test2]# netstat -anplgrep 9000
tcp
                  0 0.0.0.0:9000
                                                (
[root@localhost test2]# kill -9 17826
[root@localhost test2]# netstat -anp|grep 9000
                  0 0.0.0.0:9000
tcp
           0
                                                (
[root@localhost test2]# kill -9 17827
[root@localhost test2]# netstat -anp|grep 9000
                  0 0.0.0.0:9000
                                                (
tcp
[root@localhost test2]# kill -9 17828
[root@localhost test2]# netstat -anp|grep 9000
```

#### 3、解决办法,如下:

```
#include "linux_socket.h"
#include <stdlib.h>
#include <fcntl.h>
int main(int argc, char* argv[])
{
    SOCKET servSocket = socket_create(AF_INET, SOCK_S
    if(servSocket == -1)
    {
        return -1;
    }
    if(socket_bind(servSocket, 9000) == false)
    {
        return -1;
    }
    if(socket_listen(servSocket, 5) == false)
        return -1;
```

```
printf("servSocket[%d]\n", servSocket);
int flags = fcntl(servSocket, F_GETFL, 0);
fcntl(servSocket, F_SETFD, flags|FD_CLOEXEC);
printf("exe[%d]\n", FD_CLOEXEC);

system("./test.sh");
getchar();
return 0;
}
```

#### 理解nohup

- 1、在终端(包括仿真终端)上启动进程,当用户退出或者关闭终端,这而对于这个信号,默认的处理方式是进程退出。表现为终端上的进
- 2、如何解决这个问题?

使用nohup命令启动进程,让这个进程忽略SIGHUP信号,从而不补

#### 进程在后台运行

Linux 技巧:让进程在后台可靠运行的几种方法

- 1、考虑下面的需求,通过ssh连接10.65.40.186,然后ping 10.65 Ctrl+c 结束ping程序。
- 2、Ctrl+c 会终止ping程序,现在不想终止ping程序,怎么办? 使用 ping 10.65.200.168 & 在后台运行。
- 3、ping 10.65.200.168 & 在后台运行,但是结果还是打印在标准转把程序的输出,重定向到设备黑洞,如下: ping 10.65.200.168 >/dev/null & 注意:设备黑洞是/dev/null,而不是nul,通过 ll | grep nu
- 4、上面的方式存在一个问题:通过ssh连接,然后执行 ping 10.65. 也就是说,ping程序是ssh连接的子进程,因为ping是通过ssh连 当ssh连接断开,ping程序也会自动终止。
- 5、具体如下: 建立一个ssh连接A,执行ping,如下: [root@localhost~]# ping 10.65.200.168 >/dev/nul]

```
[1] 10995
   再建立一个ssh连接B,查看进程关系,10977是ssh连接A,1099
    [root@localhost dev]# ps -ef|grep ping
            10995 10977 0 17:11 pts/0
   root
                                          00:00:00 r
                         0 17:15 pts/2
   root
            12035 4157
                                         00:00:00
    [root@localhost dev]# pstree -p
   init(1)—atd(1040)
            -auditd(956)—\{auditd\}(957)
            -crond(1029)
            -dbus-daemon(989)
            -\text{login}(1071) --\text{bash}(1083) --\text{tailf}(2299)
                                      └_tailf(3032)
            -mingetty(1073)
            -rsyslogd(972) - \{rsyslogd\}(973)
                            -{rsyslogd}(974)
                           \vdash{rsyslogd}(975)
             -sshd(1021)---sshd(4155)---bash(4157)---
                                      —bash(10977)—
                         -sshd(7687)---bash(7691)
                         -sshd(10495)----bash(10501)-
                         -sshd(11553)---sftp-server(
            -udevd(143)---udevd(741)
-udevd(1075)
            -watch_dog(1057)
   进程关系如下:init(1)-sshd(1021)-sshd(4155)-bash(10
   特别注意:sshd下面是bash,然后才是ping
6、也就是说,关闭10977,10995也会终止。怎么办?
7、使用小括号,让ping程序成为init的子进程,而不是当前终端的子。
    (ping 10.65.200.168 >/dev/null &)
   查看进程的关系,如下:
    [root@localhost dev]# ps -ef|grep ping
                         0 17:26 pts/0
            15484
                      1
                                          00:00:00 r
   root
                         0 17:26 pts/2
            15508
                   4157
                                          00:00:00
   root
   通过pstree,也可以看到15484直接挂在init下面
```

```
通过这种方式,关闭终端连接A,也不会影响到ping程序。
8、上面情况的原理是: 当用户注销或者网络断开,终端收到HUP (hand
   使用小括号,相当于让ping程序不是终端的子进程,而是init的一
   有没有其他的办法呢?
9、使用nohup, 让提交的ping命令忽略 hangup 信号,如下:
   在终端A,如下:
   [root@localhost ~]# nohup ping 10.65.200.168 >/de
   [1] 18524
   [root@localhost ~]# ps -ef |grep ping
           18524 18506 0 17:36 pts/0
   root
                                     00:00:00 r
           18543 18506
                      0 17:36 pts/0
                                     00:00:00
   root
   在终端B,如下:
   [root@localhost dev]# ps -ef|grep ping
           18524 18506 0 17:36 pts/0
   root
                                      00:00:00 r
                      0 17:37 pts/2
   root
           19013
                 4157
                                      00:00:00
   关闭终端A,如下:
   [root@localhost dev]# ps -ef|grep ping
   root
           18524
                    1
                      0 17:36 ?
                                     00:00:00 r
   root
           19018
                 4157
                      0 17:37 pts/2
                                     00:00:00 c
10、使用nohup,特别注意:
   a、关闭终端A, ping的父进程变为init, 也就是说, init会收留
   b、如果不输出到设备黑洞, nohup ping 10.65.200.168 & 5
11、nohup是忽略hangup信号的影响,另一种思路,让ping程序不要原
   使用setsid,如下:
   setsid ping 10.65.200.168 >/dev/null &
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

CPP调用lua

lua调用CPP

lua协程

### 变量说明

#### CPP调用lua

```
1、hello.lua,如下:
    niuzibin@ubuntu:~/work/test1$ vi hello.lua
    str = "I am so cool"
    tbl = {name = "shun", id = 20114442}
    function add(a,b)
        return a + b
    end
2、test.cpp,如下:
    niuzibin@ubuntu:~/work/test1$ vi test.cpp
    #include <iostream>
    #include <string.h>
    using namespace std;
    extern "C"
    {
        #include "lua.h"
        #include "lauxlib.h"
        #include "lualib.h"
    int main()
    {
        //1. 创建Lua状态
        lua_State *L = luaL_newstate();
        if (L == NULL)
        {
            return 0 ;
```

```
//2.加载Lua文件
int bRet = luaL loadfile(L, "hello.lua");
if(bRet)
{
    cout<<"load file error"<<endl;</pre>
    return 0 ;
}
//3.运行Lua文件
bRet = lua\_pcall(L, 0, 0, 0);
if(bRet)
{
    cout<<"pcall error"<<endl;</pre>
    return 0 ;
}
//4.读取变量
lua_getglobal(L,"str");
string str = lua_tostring(L,-1);
cout<<"str = "<<str.c_str()<<endl;</pre>
                                          1/5
//5.读取table
lua_getglobal(L,"tbl");
lua_getfield(L, -1, "name");
str = lua_tostring(L, -1);
cout<<"tbl:name = "<<str.c_str()<<endl; //tbl</pre>
//6.读取函数
lua_getglobal(L, "add"); // 获取函数,压
lua_pushnumber(L, 10); // 压入第一个参
                               // 压入第二个参
lua_pushnumber(L, 20);
int iRet= lua_pcall(L, 2, 1, 0);// 调用函数, 证
if (iRet)
                                // 调用出错
{
    const char *pErrorMsg = lua_tostring(L, -
```

```
cout << pErrorMsg << endl;</pre>
          lua_close(L);
          return 0 ;
      }
      if (lua_isnumber(L, -1)) //取值输出
      {
          double fValue = lua_tonumber(L, -1);
          cout << "Result is " << fValue << endl;</pre>
      }
      //至此, 栈中的情况是:
      // 索引 类型
                      值
      // 4 int:
                       30
      // 3 string: shun
      // 2 table:
// 1 string:
                      tbl
                       I am so cool~
      //7.关闭state
      lua_close(L);
      return 0 ;
   }
3、构建,运行:
   niuzibin@ubuntu:~/work/test1$ g++ -o test test.cr
   niuzibin@ubuntu:~/work/test1$ ./test
   str = I am so cool
   tbl:name = shun
   Result is 30
```

### lua协程

1、解决什么问题? 考虑下面的应用场景,在车间加工一个产品,分为多个步骤,每个 2、示例代码 niuzibin@ubuntu:~/work/lua\$ vi coroutine.lua function foo (a)

```
print("foo", a)
            return coroutine.yield(2*a)
    end
    co = coroutine.create(function (a,b)
    print("co-body1", a, b)
    local r = foo(a+1)
    print("co-body2", r)
    local r, s = coroutine.yield(a+b, a-b)
    print("co-body3", r, s)
    return b, "end"
    end)
    print("1----")
    print("main", coroutine.resume(co, 1, 10))
    print("2----")
    print("main", coroutine.resume(co, "r"))
    print("3----")
    print("main", coroutine.resume(co, "x", "y"))
    print("4----")
    print("main", coroutine.resume(co, "x", "y"))
3、运行
    niuzibin@ubuntu:~/work/lua$ lua coroutine.lua
    1 - - - -
    co-body1
                    1
                            10
    foo
            2
    main
            true
                    4
    2----
    co-body2
                    r
                    11
                            - 9
    main
          true
    3----
    co-body3
                    Χ
                            У
    main
            true
                    10
                            end
    4 - - - -
    main
           false
                    cannot resume dead coroutine
4、代码分析
    resume 多次进来,每次进来,拿着数据,在现场继续处理。
    yield 返回当前步骤的处理结果。
```

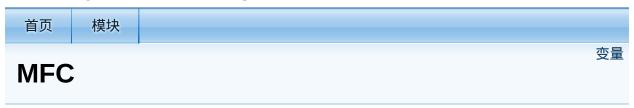
#### lua调用CPP

```
1、h文件
    hello world!niuzibin@ubuntu:~/work/test2$ vi aaa.
    #pragma once
    extern "C" {
    #include "lua.h"
    #include "lualib.h"
    #include "lauxlib.h"
    }
    extern "C" LUA_API int luaopen_aaa(lua_State *L);
2、cpp文件
    niuzibin@ubuntu:~/work/test2$ vi aaa.cpp
    #include <stdio.h>
    #include "aaa.h"
    static int averageFunc(lua_State *L)
    {
        int n = lua_gettop(L);
        double sum = 0;
        int i;
        /* 循环求参数之和 */
        for (i = 1; i \le n; i++)
            sum += lua_tonumber(L, i);
        lua_pushnumber(L, sum / n); //压入平均值
                                      //压入和
        lua_pushnumber(L, sum);
                                       //返回两个结果
        return 2;
    }
    static int sayHelloFunc(lua_State* L)
    {
        printf("hello world!");
        return 0;
    }
```

```
static const struct lual_Reg myLib[] =
    {
       {"average", averageFunc},
       {"sayHello", sayHelloFunc},
       {NULL, NULL} //数组中最后一对必须是{NULL,
   };
   int luaopen_aaa(lua_State *L)
   {
       luaL_register(L, "ss", myLib);
       return 1; // 把myLib表压入了栈中,所以就需要
   }
3、lua文件
   niuzibin@ubuntu:~/work/test2$ vi test.lua
   require "aaa"
   local ave, sum = ss.average(1,2,3,4,5) -- 参数对应均
   print(ave, sum) -- 3 15
   ss.sayHello() -- hello world!
4、构建,运行
   niuzibin@ubuntu:~/work/test2$ q++ -fPIC -shared -
   niuzibin@ubuntu:~/work/test2$ lua test.lua
   3
           15
   特别注意:这里的动态库名称,不带lib,因为要和luaopen aaa
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

模态对话框与非模态对话框

获取和设置EditControl的值

确认框

### 变量说明

#### 模态对话框与非模态对话框

```
1、模态对话框,关闭对话框之前,不能进行其它工作。
非模态对话框,对话框打开的情况下,可以进行其它工作。
```

```
2、创建模态对话框
```

```
CDlgAlarm dlg;
dlg.DoModal();
```

3、创建非模态对话框

```
CDlgAlarm* dlg = new CDlgAlarm;
dlg->Create(IDD_DIALOG_ALARM); // IDD_DIALOG_ALA
dlg->ShowWindow(SW_SHOWNORMAL);
```

### 确认框

- 1、MFC中有MessageBox, WinAPI也有MessageBox
- 2、AfxMessageBox是MFC中的,是对MFC中MessageBox的封装,示例 AfxMessageBox("和vru没有连接");

MFC中MessageBox是模态的,也就是必须确认之后,才能点击其1

3、怎么创建非模态的?

需要使用WinAPI中的MessageBox,第一个参数设置为NULL,如::MessageBox(NULL,"和vru没有连接","Caption",MB\_OK)

### 获取和设置EditControl的值

1、添加变量的时候选择类别是Control,多出一个变量 CEdit aaa; CString str; aaa.GetWindowText(str);

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 模块 模块 | 変量

# 模块

10000
API调用
Mysql事务
优化
Mysql分区表
Mysql常见错误
Mysql常见问题
Mysql数据类型
Mysql方法
Mysql有关文件
查询操作
Mysql语法
Mysql复制
ODBC
OTL
事件调度器
执行计划
索引

## 重变

mysql变量

mysql编码

mysql编码测试

### 变量说明

### mysql变量

```
用户变量
1、用户变量,用户变量和数据库连接有关,连接后声明变量,连接断开
2、用户变量以@开头, select一个没有赋值的用户变量, 返回NULL, t
   mysql> select @var;
   +---+
   l @var l
   +---+
   | NULL |
3、用户变量赋值有三种方式:
   a、set @var=(select user from user limit 0,1);
   b, select user into @var from user limit 0,1;
   c、select @var:=user from user limit 0,1;
   注意:a和bc的区别,比如当前@var取值为123,查询出来的记录
4、需要注意的是, mysql的变量类似于动态语言, 在赋值的时候, 确定
   mysql> set @a=123;
   Query OK, 0 rows affected (0.00 sec)
   mysql> select @a;
   +---+
   | @a
   +---+
   | 123 |
   +----+
   1 row in set (0.00 sec)
   mysql> set @a='abc';
   Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select @a;
   +---+
   I @a
   +---+
   l abc
   +---+
   1 row in set (0.00 sec)
系统变量
系统变量两个@@,系统变量的作用域分为全局作用域和会话作用域,以&
1、新建一个连接,只有全局autocommit,会话autocommit还不存在
2、修改全局autocommit,只会对新建立的连接起作用,对已经存在的
3、修改会话autocommit,只对当前连接起作用。
4、不管是全局autocommit还是会话autocommit,通过客户端命令设
   mysql服务重启,还是原来的配置。要想对服务起作用,必须写入
5、注意:系统变量和连接有关,才有会话作用域。有些系统变量和连接
   只有全局作用域,没有会话作用域。
6、全局autocommit的设置和获取
   mysql> set @@global.autocommit=1;
   Query OK, 0 rows affected (0.00 sec)
   mysql> select @@global.autocommit;
   | @@global.autocommit |
   1 row in set (0.00 sec)
   mysql> set @@global.autocommit=0;
   Query OK, 0 rows affected (0.00 sec)
   mysql> select @@global.autocommit;
   | @@global.autocommit |
```

```
1 row in set (0.00 sec)
7、会话autocommit的设置和获取
   mysql> set @@session.autocommit =1;
   Query OK, 0 rows affected (0.00 sec)
   mysql> select @@session.autocommit;
    | @@session.autocommit |
   1 row in set (0.00 sec)
   mysql> set @@session.autocommit =0;
   Query OK, 0 rows affected (0.00 sec)
   mysql> select @@session.autocommit;
    | @@session.autocommit |
   +----+
   1 row in set (0.00 sec)
8、注意:不带作用域修饰,默认是指会话作用域。
    (特别注意:有些系统变量,不带作用域,设置不成功,比如tx_j
   mysql> select @@global.autocommit;
         - - - - - - - - - - - +
   | @@global.autocommit |
   1 row in set (0.00 sec)
   mysql> select @@session.autocommit;
   +----+
   | @@session.autocommit |
```

```
+-----+
   1 row in set (0.00 sec)
   mysql> set @@autocommit=0;
   Query OK, 0 rows affected (0.00 sec)
   mysql> select @@autocommit;
   | @@autocommit |
   +----+
   +----+
   1 row in set (0.00 sec)
   mysql> select @@global.autocommit;
   | @@global.autocommit |
   +----+
   +----+
   1 row in set (0.00 sec)
   mysql> select @@session.autocommit;
   +----+
   | @@session.autocommit |
   +-----+
   1 row in set (0.00 sec)
9、注意:set @@global.autocommit=1; 还有一种写法: set g
   对于session是同样道理。
10、对于系统变量,除了select,还有一种办法:show variables
   mysql> show global variables like '%autocommit%';
   +----+
   | Variable_name | Value |
```

```
| autocommit | ON
  +-----+
  1 row in set (0.00 sec)
  mysql> show session variables like '%autocommit%'
  | Variable_name | Value |
  | autocommit | OFF
  +----+
  1 row in set (0.00 sec)
  mysql> show variables like '%autocommit%';
  | Variable name | Value |
  +----+
  | autocommit | OFF
  1 row in set (0.00 sec)
11、和select类似,可以show global和session,没有指定作用域
系统参数
  系统变量对应于启动时的选项,也就是系统参数。
状态变量
  sql运行过程中,还有状态变量。
```

参见

### mysql编码

- 1、分为几个层次来看:
  - a、Mysql服务端数据保存的编码,小范围的设置会覆盖大范围的设
  - b、当前环境的编码,比如VS2005或者ssh客户端的编码
  - c、交互的编码,交互的编码又分为两部分,请求和回复。

- 2、现在考虑交互的编码,如下:
  - a、Mysql客户端发送的SQL语句,由character\_set\_client来
  - b、Mysql服务端收到SQL语句,会把编码集从character\_set\_c 然后转化为字段设置的编码,保存起来。
  - c、Mysql服务端执行完SQL语句,把数据库中查询的结果,转化为
- 3、特别注意: 乱码往往不是转化导致的, 而是因为用一种编码去解释另另一种编码中没有这个字符。
- 4、Mysql的整个交互过程都是转化,转化过程中有可能出现乱码。只有a、当前环境有自己的编码,Mysql客户端用character\_set\_cl
  - b、结果返回来的编码是character\_set\_results,当前环境用
- 5、测试场景1:

```
字段保存编码utf8
set names utf8
当前环境编码为utf8
```

插入查询,都正确。

mysql> insert into t1(v1) values('cn中国'); Query OK, 1 row affected (0.00 sec)

a、现在把character\_set\_results修改为gb2312,查询,如<sup>-</sup> mysql> set character\_set\_results=gb2312; Query OK, 0 rows affected (0.00 sec)

```
1 row in set (0.00 sec)
       原因是: 当前环境用utf8去解释gb2312, 当然乱码
   b、当前环境修改为gb2312,查询如下:
       mysql> select * from t1;
       | V1 |
       I cn中国 I
       1 row in set (0.00 sec)
       原因是:服务端把utf8转为qb2312,传给客户端,客户端用
6、测试场景2:
   字段保存编码utf8
   set names utf8
   当前环境编码为utf8
   设置character_set_client=gb2312; 插入查询,如下:
       mysql> set names utf8;
       Ouery OK, 0 rows affected (0.00 sec)
       mysql> set character_set_client=gb2312;
       Query OK, 0 rows affected (0.00 sec)
       mysql> insert into t1(v1) values('cn中国');
       Query OK, 1 row affected (0.00 sec)
       mysql> select * from t1;
       | V1 |
       +----+
       | cn涓?? |
       1 row in set (0.00 sec)
       原因是:Mysql客户端用gb2312来解释utf8
7、测试场景3:
   字段保存编码utf8
   set names utf8
   当前环境编码为utf8
```

```
设置character_set_connection=gb2312; 插入查询,如下
       mysql> set names utf8;
       Query OK, 0 rows affected (0.00 sec)
       mysql> set character_set_connection=gb2312;
       Query OK, 0 rows affected (0.00 sec)
       mysql> insert into t1(v1) values('cn中国');
       Query OK, 1 row affected (0.00 sec)
       mysql> select * from t1;
       +----+
       | v1
       +----+
       | cn中国 |
       +---+
       1 row in set (0.00 sec)
       原因是:Mysql客户端用utf8来解释utf8,然后转化为gb2。
8、总结,也就是说,character_set_client要与当前环境的编码保
   当前环境的编码要与character set results保持一致,通过s
```

参见

### mysql编码测试

```
insert into t1(v1) values('cn中国'); select * from t1;

1、输入gbk,交互latin1,数据库latin1 insert,客户端把gbk的输入当成latin1来解释,传给服务,服务不需select,服务不转码,返回给客户端,客户端latin1当成gbk解释,怎2、输入gbk,交互latin1,数据库gbk insert 报错:ERROR 1366 (HY000): Incorrect string valu原因是:客户端把gbk的输入当成latin1来解释,传给服务,服务转码3、输入gbk,交互latin1,数据库utf8和1一样,这是因为utf8英文字符一个字节,多字节字符三个字节。
```

```
4、输入gbk,交互gbk,数据库latin1
insert 报错:ERROR 1366 (HY000): Incorrect string value
原因是:客户端把gbk的输入当成gbk来解释,传给服务,服务把gbk转
5、输入abk,交互abk,数据库abk
insert,客户端把qbk的输入当成qbk来解释,传给服务,服务不需要等
select,服务不转码,返回给客户端,客户端gbk当成gbk解释,显示i
6、输入qbk,交互qbk,数据库utf8
insert,客户端把gbk的输入当成gbk来解释,传给服务,服务把gbk等
select,服务把utf8转成gbk,返回给客户端,客户端gbk当成gbk解。
7、输入gbk,交互utf8,数据库latin1
insert 报错:ERROR 1366 (HY000): Incorrect string value
原因是:客户端把qbk的输入当成utf8来解释,解释错误。
8、输入abk,交互utf8,数据库abk
同7
9、输入qbk,交互utf8,数据库utf8
同7
考虑下面的情况,输入gbk,交互gbk,数据库gbk,插入一条记录,查
mysql> select * from t1;
+----+
| V1 |
+---+
| cn中国 |
+----+
1 row in set (0.00 sec)
mysql> set names utf8;
Query OK, 0 rows affected (0.00 sec)
mysql> select * from t1;
+----+
| v1 |
+---+
+----+
1 row in set (0.00 sec)
```

```
这是因为把utf8的中国,当成gbk解释,就是显示涓 浗,也就是说, ſ
mysql> insert into t1(v1) values('cn涓 浗');
Query OK, 1 row affected (0.08 sec)
mysql> select * from t1;
+----+
| v1 |
+---+
+----+
2 rows in set (0.00 \text{ sec})
mysql> set names gbk;
Query OK, 0 rows affected (0.00 sec)
mysql> select * from t1;
| v1 |
+---+
I cn中国 I
| cn中国 |
+---+
2 rows in set (0.00 sec)
```

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

### free结果集

### 变量说明

### free结果集

```
1、对于select语句和存储过程,释放结果集的做法不一样,如下:
    int DBMysql::Test()
    {
        if(mysql_query(_conn, "select devid from devid
        {
            return FAILURE;
        }
        MYSQL_RES* results = mysql_store_result(_conr
        if(results == NULL)
        {
            return FAILURE;
        }
        MYSQL_ROW row;
        while((row = mysql_fetch_row(results)) != NUL
        {
            LogInfo(DBLog, "%s", *row);
        }
        freeResults(results, false);
        return SUCCESS;
    }
    int DBMysql::TestProcedure()
```

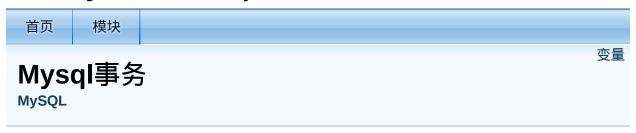
```
char* sql = "call niu1";
    if(mysql_query(_conn, sql) != SUCCESS)
    {
        return FAILURE;
    }
    MYSQL_RES* results = mysql_store_result(_conr
    if(results == NULL)
    {
        return FAILURE;
    }
    MYSQL_ROW row;
    while((row = mysql_fetch_row(results)) != NUL
    {
        LogInfo(DBLog, "%s", *row);
    }
    freeResults(results, true);
    return SUCCESS;
}
int DBMysql::freeResults(MYSQL_RES* results,bool
{
    if(isProcedure == true)
    {
        do
        {
            results = mysql_store_result(_conn);
            mysql_free_result(results);
        while((0 == mysql_next_result(_conn)));
    }
    else
    {
        mysql_free_result(results);
    }
```

# return SUCCESS;

}

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

事务隔离级别

分布式事务

锁的粒度

### 变量说明

#### 事务隔离级别

### read-uncommitted(未提交读)

#### 测试流程:

- 1、A设置read-uncommitted, start transaction
- 2、B执行start transaction,修改一条记录,
- 3、A查询记录,得到了以为正确的记录
- 4、B回滚。

问题:A读到了B没有提交的记录,也就是脏读。

### read-committed(已提交读)

### 测试流程:

- 1、A设置read-committed, start transaction
- 2、B执行start transaction,修改一条记录,查询记录,记录
- 3、A查询记录,结果还是老的记录
- 4、B提交事务
- 5、A再次查询记录,结果是新的记录。

问题:解决了脏读的问题,但是出现一个新问题,A在一个事务中,

### repeatable-read(可重复读)

### 测试流程:

- 1、A设置repeatable-read, start transaction,查询记录
- 2、B执行start transaction,修改一条记录,查询记录,记录
- 3、A查询记录,结果还是老的记录
- 4、B提交事务
- 5、A再次查询记录,结果还是老的记录。

问题:可以重复读,A在事务过程中,即使B修改了数据,并且com注意:这里可能会存在一个新的问题,A在事务过程中,B增加一条这里只是可能,具体取决于数据库的实现。mysql的repeatable

serializable (可串行化)

#### 测试流程:

- 1、A设置serializable, start transaction,查询记录,纟
- 2、B执行start transaction,修改一条记录,B卡在这里,要
- 3、A查询记录,结果还是老的记录,A提交。
- 4、B的修改操作才进行下去。

注意:B在等待过程中,会出现lock超时。这种情况,存在性能问题

注意:事务隔离级别,是在不同事务之间隔离的概念,而不是针对当前事务的隔离级别是提交读,插入一条,在当前事务中可以查询

注意:事务隔离的实现,基本上是两种方法:

- 1、访问数据之前加锁,阻止其他事务对数据的访问。
- 2、不加任何锁,对于一个请求,通过一定机制生成当前时间点的数从用户的角度来看,好像是数据库提供了一份数据的多个版本MVCC(MultiVersion Concurrency Control)

事务隔离是在一定程度上进行串行化处理,这与并行矛盾,要根据

### 分布式事务

```
//test为分布式事务的标识, 'add student'和'add teacher'为结
//创建一个分布式事务的一个分支
mysql> xa start 'test', 'add student';
Query OK, 0 rows affected (0.00 sec)
//student表插入一条记录
mysql> insert student(school, name, age) value('Num_1',
Query OK, 1 row affected (0.00 sec)
//第一阶段提交,进入prepeare状态
mysql> xa end 'test', 'add student';
Query OK, 0 rows affected (0.00 sec)
mysql> xa prepare 'test', 'add student';
Query OK, 0 rows affected (0.07 sec)
//查看分支状态
mysql> xa recover\G
formatID: 1
```

### mysql的分布式事务还存在缺陷,比如:

- 1、分支进入prepare状态,这个时候数据库异常,重新启动mysql,重但是提交的事务没有写入binlog(二进制日志),如果存在数据原
- 2、多个分支进入prepare状态,其中一个分支连接异常,其他的分支成而对于连接异常的分支,mysql的处理方式是回滚,这就导致了其导致分布式事务的不完整。

### 锁的粒度

1、锁的类型分为读锁和写锁,这个很好区分。可以这样认为:如果有增

```
Query OK, 0 rows affected (0.00 sec)
mysql> update test set num=num+1 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
事务B:
mysql> select @@session.tx_isolation;
+----+
| @@session.tx_isolation |
| READ-COMMITTED |
1 row in set (0.00 sec)
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql> update test set num=num+1 where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
杳询事务:
mysql> select * from information_schema.innodb_trx\G
trx_id: DAB09
trx state: RUNNING
trx started: 2015-07-02 08:41:28
trx_requested_lock_id: NULL
trx wait started: NULL
trx weight: 3
trx_mysql_thread_id: 3
trx query: NULL
trx operation state: NULL
trx tables in use: 0
trx tables locked: 0
trx lock structs: 2
trx_lock_memory_bytes: 320
```

```
trx_rows_locked: 1
trx_rows_modified: 1
trx_concurrency_tickets: 0
trx isolation level: READ COMMITTED
trx unique checks: 1
trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
trx_adaptive_hash_latched: 0
trx adaptive hash timeout: 10000
trx id: DAB08
trx state: RUNNING
trx started: 2015-07-02 08:41:00
trx requested lock id: NULL
trx wait started: NULL
trx_weight: 3
trx_mysql_thread_id: 2
trx_query: NULL
trx_operation_state: NULL
trx_tables_in_use: 0
trx_tables_locked: 0
trx_lock_structs: 2
trx_lock_memory_bytes: 320
trx_rows_locked: 1
trx_rows_modified: 1
trx_concurrency_tickets: 0
trx isolation level: READ COMMITTED
trx_unique_checks: 1
trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
trx_adaptive_hash_latched: 0
trx adaptive hash timeout: 10000
2 rows in set (0.00 sec)
两个事务之间没有lock wait,说明read-committed是行锁
测试 repeatable-read , 结果是表锁
事务A:
```

```
mysql> select @@session.tx_isolation;
| @@session.tx_isolation |
| REPEATABLE-READ |
1 row in set (0.01 sec)
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql> update test set num=num+1 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
事务B:
mysql> select @@session.tx_isolation;
| @@session.tx_isolation |
+-----+
| REPEATABLE-READ |
+----+
1 row in set (0.00 sec)
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql> update test set num=num+1 where id=2;
查询事务:
mysql> select * from information_schema.innodb_trx\G
trx id: DAB1A
trx state: LOCK WAIT
trx started: 2015-07-02 09:31:29
```

```
trx_requested_lock_id: DAB1A:0:100841:2
trx_wait_started: 2015-07-02 09:31:29
trx_weight: 2
trx mysql thread id: 3
trx query: update test set num=num+1 where id=2
trx_operation_state: starting index read
trx tables in use: 1
trx tables locked: 1
trx_lock_structs: 2
trx_lock_memory_bytes: 320
trx rows locked: 1
trx_rows_modified: 0
trx concurrency tickets: 0
trx isolation level: REPEATABLE READ
trx_unique_checks: 1
trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
trx_adaptive_hash_latched: 0
trx_adaptive_hash_timeout: 10000
trx id: DAB19
trx state: RUNNING
trx started: 2015-07-02 09:31:21
trx_requested_lock_id: NULL
trx_wait_started: NULL
trx_weight: 3
trx_mysql_thread_id: 2
trx_query: NULL
trx operation state: NULL
trx tables in use: 0
trx tables locked: 0
trx lock structs: 2
trx lock memory bytes: 320
trx rows locked: 4
trx rows modified: 1
trx concurrency tickets: 0
trx isolation level: REPEATABLE READ
```

```
trx_unique_checks: 1
trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
trx adaptive hash latched: 0
trx adaptive hash timeout: 10000
2 rows in set (0.00 \text{ sec})
事务B处于lock wait状态,说明repeatable-read是表锁
测试 repeatable-read, 查询条件使用主键, 也就是id为primary
mysql> desc test;
+----+
| Field | Type | Null | Key | Default | Extra |
+----+
| ID | int(11) | NO | | 0 | |
+----+
2 rows in set (0.01 sec)
mysql> alter table test add primary key(id);
Query OK, 0 rows affected (14.70 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> desc test;
+----+
| Field | Type | Null | Key | Default | Extra |
+----+
| ID | int(11) | NO | PRI | 0 | |
+----+---+---+
2 \text{ rows in set } (0.01 \text{ sec})
事务A:
mysql> select @@session.tx_isolation;
| @@session.tx_isolation |
```

```
| REPEATABLE-READ |
1 row in set (0.00 \text{ sec})
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql> update test set num=num+1 where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
事务B:
mysql> select @@session.tx_isolation;
+----+
| @@session.tx_isolation |
| REPEATABLE-READ |
+-----
1 row in set (0.00 sec)
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql> update test set num=num+1 where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
查询事务:
mysql> select * from information_schema.innodb_trx\G
trx id: DAB23
trx state: RUNNING
trx started: 2015-07-02 09:38:48
trx requested lock id: NULL
trx wait started: NULL
trx_weight: 3
```

```
trx_mysql_thread_id: 3
trx_query: NULL
trx_operation_state: NULL
trx tables in use: 0
trx tables locked: 0
trx lock structs: 2
trx_lock_memory_bytes: 320
trx rows locked: 1
trx rows modified: 1
trx_concurrency_tickets: 0
trx isolation level: REPEATABLE READ
trx_unique_checks: 1
trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
trx_adaptive_hash_latched: 0
trx_adaptive_hash_timeout: 10000
trx_id: DAB22
trx state: RUNNING
trx started: 2015-07-02 09:38:45
trx_requested_lock_id: NULL
trx_wait_started: NULL
trx_weight: 3
trx_mysql_thread_id: 2
trx_query: NULL
trx_operation_state: NULL
trx_tables_in_use: 0
trx_tables_locked: 0
trx lock structs: 2
trx_lock_memory_bytes: 320
trx rows locked: 1
trx rows modified: 1
trx concurrency tickets: 0
trx isolation level: REPEATABLE READ
trx_unique_checks: 1
trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
```

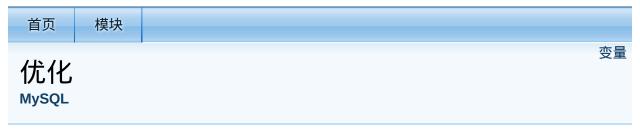
trx\_adaptive\_hash\_latched: 0
trx\_adaptive\_hash\_timeout: 10000

2 rows in set (0.00 sec)

表test的id为主键,上面的两个事务,没有lock wait,说明通过主锁这是因为innodb主键索引(聚集索引),可以直接定位相应的行,不需也就是说innodb 使用索引,只会去锁相应的行(有可能还包括当前行政治意:查询加锁信息,除了表innodb\_trx,还有表innodb\_locks,

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

in与exists的区别

count有关

查询缓存

mysql千万级大数据SQL查询优化30条经验

### 变量说明

### count有关

- 1、count有两个作用:统计某个字段有值的记录数;统计结果集的记录
- 2、count括号内的表达式不为null,就是统计结果集的记录数。也就是都是等价的,对于说法,count(1)是第一个字段,count(\*)展注:括号内的表达式为null,即count(null)返回 0
- 3、count(\*) 统计结果集的记录数,也就是总的行数,而主键是不允认count(\*) 也就可以认为对主键扫描,与count(key)一样,而i□
- 4、因此, count(\*)与count(key)的效率基本一样。而对于一般的与 但是这要根据实际的需求,如果是查询总的行数,使用count(\*)
- 5、有些情况下,不要求精确值,只需要记录的近似值。这个时候可以使 desc select count(\*) from student; 效率非常高,返回扫描的行数是个近似值。
- 6、查询执行计划返回的信息记录在information\_schema.tables, select table\_name,table\_rows from information\_sch
- 7、特别注意:对于MyISAM引擎, select count(\*) from studen 因此直接返回。根据这个特点,考虑id<5的记录很少,对于 select count(\*) from student;减去 select count(\*) 这样大大地减少了扫描的行数。

### 参见

### in与exists的区别

1、使用5.6版本,准备表和存储过程 mysql> create table a(ID int,NAME varchar(64)); Query OK, 0 rows affected

```
mysql> create table b(ID int, NAME varchar(64));
  Query OK, 0 rows affected
  BEGIN
  declare vI int default 0;
  start transaction;
  while(vI < 100000) do
     insert into a(id, name) values(vI, concat('And)
     set vI= vI+1;
  end while;
  commit;
  END
2、添加10万条记录,下面两个查询是等价的,大概耗时不到0.1秒
  select sql_no_cache * from a where name ='Andy10@
  select sql_no_cache * from a where name in ('Andy
3、对name加上索引, create index index_a_name on a(name)
  查看执行计划,可以看到使用了a表上面的索引index_a_name,
  mysql> desc select sql_no_cache * from a where na
  | id | select_type | table | type | possible_key
  | 1 | SIMPLE | a | range | index_a_name
  1 row in set
4、理解in的原理,in的执行流程如下:
  对于in后面的集合,遍历,对于每一个元素,使用索引index_a_
5、但是特别注意的是:对于not in没有办法执行上面的流程,因为没有
  只能先遍历表a的记录,检查不在in的集合中,因此没有办法使用实
  执行计划如下:
  mysql> desc select sql_no_cache * from a where na
  +---+----
   | id | select_type | table | type | possible_keys
```

```
1 | SIMPLE | a | ALL | index_a_name
  1 row in set
  也就是说,对于in,使用表a的索引。
6、现在考虑, a, b两个表都不使用索引,分别加上10万条记录, 1千条
  select sql_no_cache * from a where a.name in (se)
  select sql_no_cache * from a where a.name not in
  这是为什么?
7、查看执行计划,使用desc extended + show warnings,如下:
  mysql> desc extended select sql_no_cache * from a
  +---+----+-----
  | <subquery2> | ALL | NULL
  | 1 | SIMPLE
  1 | SIMPLE | a | ALL | NULL
   2 | MATERIALIZED | b
               | ALL | NULL
  +---+----+----
  3 rows in set
  mysql> show warnings;
  | Level | Code | Message
  1 row in set
  mysql> desc extended select sql_no_cache * from a
  | id | select_type | table | type | possible_keys
  1 | PRIMARY | a | ALL
   2 | SUBQUERY | b | ALL | NULL
  2 rows in set
  mysql> show warnings;
```

```
| Level | Code | Message
  | Note | 1003 | /* select#1 */ select sql_no_cac
  1 row in set
  原因找到了,也就是对于not in, mysql使用了<in_optimize
8、对表a的name字段加上索引,查询时间如下:
  select sql_no_cache * from a where a.name in (sel
  select sql no cache * from a where a.name not in
  也就是说,对表a的name字段加上索引,大幅度提高效率,性能提定
  查看执行计划,如下:
  mysgl> desc extended select sgl no cache * from a
  | id | select_type | table | type | possible_keys
  | 1 | SIMPLE | b | ALL | NULL
    1 | SIMPLE | a | ref | index_a_name
  2 rows in set
  mysql> show warnings;
  | Level | Code | Message
  | Note | 1003 | /* select#1 */ select sql_no_cac
  +-----
  1 row in set
9、现在考虑使用exists,如下:
  select sql_no_cache * from a where exists (select
  执行计划,如下:
  mysql> desc extended select sql_no_cache * from ε
  +---+----+----
  +---+----+----
            a | ALL | NULL
  | 1 | PRIMARY
  | 2 | DEPENDENT SUBQUERY | b | ALL | NULL
```

++	·	++	
遍历表a的记录,把字段name拿到ex要解决问题,需要在表b上,建立索	kists后面 別。		,检查是?
mysql> desc extended select	sql_no		
id   select_type	table	type	possik
1   PRIMARY   2   DEPENDENT SUBQUERY	a b	ALL	NULL index_
2 rows in set 总结in与exists,如下: a、in的执行流程:对于in后面的集 b、exists的执行流程:遍历前面表 既然是检查是否满足条件,不完 c、也就是说,in使用前面表的索引 not in不能使用前面表的索引	合,遍历每 的记录,持 心后面的 ,exists ,因为后面	事一个元素 把字段拿到 select内 使用后面表 面的集合是	,去和前  exists/ 容,可以 長的索引。
select sql_no_cache * 1 表a的记录个数是10万,表b的 in相当于使用了1千次10万记录 exists相当于使用了10万次1	from a wi 记录个数是 R的索引 千记录的索	here exi ≟1干。 ≷引	sts (se
(可以想象二叉树是log2的关 因此,肯定是exists的速度慢 e、根据上面的推论,使用exists, select sql_no_cache * 1	系,多叉树 很多。 反过来查i <sup>∓</sup> rom b w	t是是1ogr 旬速度就很	的关系) !快,
	2 rows in set 理解exists的原理,exists的执行遍历表a的记录,把字段name拿到ex要解决问题,需要在表b上,建立索引建立索引之后,查询耗时1秒,执行证mysql> desc extended select ++ —— id   select_type	2 rows in set 理解exists的原理,exists的执行流程如下: 遍历表a的记录,把字段name拿到exists后面要解决问题,需要在表b上,建立索引。 建立索引之后,查询耗时1秒,执行计划如下: mysql> desc extended select sql_no++	理解exists的原理,exists的执行流程如下: 遍历表a的记录,把字段name拿到exists后面的查询中要解决问题,需要在表b上,建立索引。 建立索引之后,查询耗时1秒,执行计划如下: mysql> desc extended select sql_no_cache * ++

## mysql千万级大数据SQL查询优化30条经验

1、对查询进行优化,应尽量避免全表扫描,首先应考虑在 where 及

- 2、应尽量避免在 where 子句中对字段进行 null 值判断,否则将导 如:select id from t where num is null可以在num上设 确保表中num列没有null值,然后这样查询:select id from
- 3、应尽量避免在 where 子句中使用!=或<>操作符,否则引擎将放弃(
- 4、应尽量避免在 where 子句中使用or 来连接条件,否则将导致引擎如:select id from t where num=10 or num=20
- 可以这样查询:select id from t where num=10 union a 5、in 和 not in 也要慎用,否则会导致全表扫描,如:select id 能用 between 就不要用in了:select id from t where n
- 6、下面的查询也将导致全表扫描: select id from t where name
- 7、如果在 where 子句中使用参数,也会导致全表扫描。因为SQL只有但优化程序不能将访问计划的选择推迟到运行时;它必须在编译时然 而,如果在编译时建立访问计划,变量的值还是未知的,因而对如下面语句将进行全表扫描:select id from t where num=可以改为强制查询使用索引:select id from t with(index
- 8、应尽量避免在 where 子句中对字段进行表达式操作,这将导致引擎如:select id from t where num/2=100应改为:select
- 9、应尽量避免在where子句中对字段进行函数操作,这将导致引擎放弃如:select id from t where substring(name,1,3)='a 应改为: select id from t where name like 'abc%'
- 10、不要在 where 子句中的"="左边进行函数、算术运算或其他表达i
- 11、在使用索引字段作为条件时,如果该索引是复合索引,那么必须使身
- 12、不要写一些没有意义的查询,如需要生成一个空表结构:select 这类代码不会返回任何结果集,但是会消耗系统资源的,应改成这种
- 13、很多时候用 exists 代替 in 是一个好的选择: select num f 用下面的语句替换: select num from a where exists(se]
- 14、并不是所有索引对查询都有效,SQL是根据表中数据来进行查询优保 SQL查询可能不会去利用索引,如一表中有字段sex,male、fema 那么即使在sex上建了索引也对查询效率起不了作用。
- 15、索引并不是越多越好,索引固然可以提高相应的 select 的效率 因为 insert 或 update 时有可能会重建索引,所以怎样建索引一个表的索引数最好不要超过6个,若太多则应考虑一些不常使用到
- 16、应尽可能的避免更新 clustered 索引数据列,因为 clustered 一旦该列值改变将导致整个表记录的顺序的调整,会耗费相当大的 若应用系统需要频繁更新 clustered 索引数据列,那么需要考虑
- 17、尽量使用数字型字段,若只含数值信息的字段尽量不要设计为字符? 这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符

- 18、尽可能的使用 varchar/nvarchar 代替 char/nchar ,因为证 其次对于查询来说,在一个相对较小的字段内搜索效率显然要高些,
- 19、任何地方都不要使用 select \* from t , 用具体的字段列表代:
- 20、尽量使用表变量来代替临时表。如果表变量包含大量数据,请注意第
- 21、避免频繁创建和删除临时表,以减少系统表资源的消耗。
- 22、临时表并不是不可使用,适当地使用它们可以使某些例程更有效,信息,对于一次性事件,最好使用导出表。
- 23、在新建临时表时,如果一次性插入数据量很大,那么可以使用 sel 避免造成大量 log ,以提高速度;如果数据量不大,为了缓和系:
- 24、如果使用到了临时表,在存储过程的最后务必将所有的临时表显式制 然后 drop table ,这样可以避免系统表的较长时间锁定。
- 25、尽量避免使用游标,因为游标的效率较差,如果游标操作的数据超让
- 26、使用基于游标的方法或临时表方法之前,应先寻找基于集的解决方题
- 27、与临时表一样,游标并不是不可使用。对小型数据集使用 FAST\_I 尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括 如果开发时间允许,基于游标的方法和基于集的方法都可以尝试-
- 28、在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ,在 无需在执行存储过程和触发器的每个语句后向客户端发送DONE\_IN
- 29、尽量避免大事务操作,提高系统并发能力。
- 30、尽量避免向客户端返回大数据量,若数据量过大,应该考虑相应需求

### 查询缓存

```
+----+
   1 row in set
   mysql> set @@global.query_cache_size=1000000;
   Query OK, 0 rows affected
   mysql> select @@global.query_cache_size;
   | @@global.query_cache_size |
   +----+
   +----+
   1 row in set
5、query_cache_limit 控制查询结果的最大值,也就是说,查询结果
   query_cache_limit只有全局作用域。
                              // 这是状态变量,也
6\ show status like 'qcache%';
   记录查询缓存的相关信息,其中Qcache_queries_in_cache 是
7、考虑下面的需求,在查询性能测试的时候,不希望有查询缓存,否则
   a、设置会话query_cache_type=off; 注意是会话query_cac
   b、每次查询的时候,使用reset query cache,清除缓存信息
   c、使用sql_no_cache,告诉mysql不要缓存查询结果,Qcache
      select sql_no_cache count(*) from student;
8、注意:Mysql只对查询和非常简单的存储过程(只包含简单的selec
   // 进行缓存
   BEGIN
      select count(*) from student;
   END
   // 不进行缓存
   BEGIN
   declare i int default 0;
   while (i<1) do
      select count(*) from student;
      set i = i+1;
   end while;
```

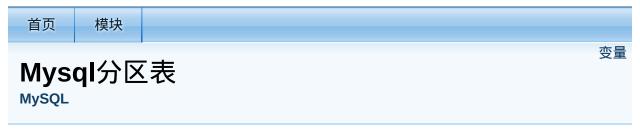
**END** 

- 9、某个操作会导致缓存中的记录无效,考虑下面的情况, select count(\*) from student; delete from student where id = 100; select count(\*) from student; 这种情况,Mysql执行select count(\*) from student;缓存 Mysql认为缓存记录select count(\*) from student;无效 select count(\*) from student; 重新执行一次。
- 10、注意:上面缓存的测试使用mysql客户端进行的,使用Navicat的 先执行一下set @@session.query\_cache\_type=off;

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

创建分区表

动态管理分区

### 变量说明

### 创建分区表

```
1、考虑下面的需求, alarmlog记录告警日志信息, 用户查询的时候,
   alarmlog的数据量很大,如果都写到一个表里面,会导致查询效?
2、应用程序控制,每天建立一个表。但是这样导致一个问题,客户端查
   如果客户端查询的时间段跨多天,应用程序就需要找出对应的表,
3、有没有更好的办法?
   使用分区表,逻辑上是一个表,物理上是多个表组成。
4、创建表的时候,创建分区,如下:
   create table alarmlog(ID int, MSG varchar(32), T1
   engine=innodb
   partition by range(to_days(TIME))
       partition p1 values less than (to_days('2015-
       partition p2 values less than (to_days('2015-
       partition p3 values less than (to_days('2015-
       partition p4 values less than (to_days('2015-
       partition p5 values less than (to days('2015-
       partition p6 values less than (to_days('2015-
       partition p7 values less than (to_days('2015-
   );
5、添加测试数据
   BEGIN
       declare vI int default 0;
       declare vRand int default 0;
       start transaction;
       while (vI <= 100) do
```

```
set vI = vI +1;
       set vRand=floor(rand()*6);
       insert into alarmlog(id, msg, time) value(\)
     end while;
     commit;
  END
6、查看执行计划
  mysql> desc partitions select * from alarmlog whe
  +---+----+----
  | id | select_type | table | partitions | type
  +---+----+----
  1 row in set
  mysql> desc partitions select * from alarmlog whe
  +---+----+----
  | id | select_type | table | partitions | type
  +---+----+----
  1 row in set
  可以看到mysql直接定位到分区表,进行扫描,缩小了扫描的范围
7、删除分区
  alter table alarmlog drop partition p1;
8、添加分区
  alter table alarmlog add partition (partition p8
  注意:对表增加一个分区,表之前必须创建过分区。
9、随着时间的过去,运行到 2015-12-27,考虑插入一条告警,如下
  mysql> insert into alarmlog(id, msg, time) value(10
  1526 - Table has no partition for value 736324
  也就是说,没有分区存放这条记录。
  因此,需要动态增加分区。同时,很早以前的分区,数据不再使用
10、动态管理分区,删除老的分区,增加新的分区。使用mysql的事件ì
```

```
1、python生成分区表,如下:
    #coding=gb18030
    from datetime import *
    class GeneMysqlPartition:
        def Gene(self):
            pFile = open('MysqlPartition.txt','w')
            i=0
            curDate=date.today()-timedelta(days=10)
            while(i<10):
                ss= "partition p%04d%02d%02d values ]
                pFile.write(ss)
                i=i+1
                curDate=curDate+timedelta(days=1)
            pFile.close()
    if __name__ == '__main__':
        geneA = GeneMysqlPartition();
        geneA.Gene();
2、创建表,如下:
    use niu_test;
    create table alarm_log(ID int, MSG varchar(32), 1
    engine=innodb
    partition by range(to_days(TIME))
        partition p20151220 values less than (to_days
    );
3、动态管理分区表
    DROP PROCEDURE IF EXISTS `exe_dynamic_sql`;
    DELIMITER ;;
    CREATE DEFINER=`root`@`%` PROCEDURE `exe_dynamic_
    BEGIN
        set @exeSql = iDynamicSql;
        prepare stmt from @exeSql;
        execute stmt;
        deallocate prepare stmt; -- 释放掉预处理段
    END
```

```
DELIMITER;
DROP PROCEDURE IF EXISTS `manage_partitions`;
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `manage_parti
BEGIN
    declare vPartitionName varchar(32) default ''
    declare vIsOver int default 0;
    declare vCurDate date default curdate();
    declare v0ldDate date default curdate();
    declare vLastDate date default curdate();
    declare vExpectDate date default curdate();
    declare vLastPartitionName varchar(32) defaul
    declare vDynamicSql text default '';
    declare vCursorPartitionName cursor for selection
    declare continue handler for sglstate '02000'
    -- delete old partition
    open vCursorPartitionName;
    fetch vCursorPartitionName into vPartitionNam
    while(vIsOver = 0) do
        set v0ldDate = substr(vPartitionName, 2);
        -- select datediff(vCurDate, vOldDate);
        if(datediff(vCurDate, vOldDate)> iOldParti
            set vDynamicSql = concat('alter table
            -- select vDynamicSql;
            call exe_dynamic_sql(vDynamicSql);
        end if;
        fetch vCursorPartitionName into vPartitic
    end while;
    close vCursorPartitionName;
```

```
-- delete old partition
        -- add new partition
        set vLastPartitionName = (select max(partitic)
        set vLastDate = substr(vLastPartitionName, 2)
        select date_add(curdate(),interval iPartitior
        while(vLastDate < vExpectDate) do</pre>
            set vLastDate = vLastDate + interval iPar
            set vDynamicSql = concat('alter table ',i
            -- select vDynamicSql;
            call exe dynamic sql(vDynamicSql);
        end while;
        -- add new partition
    END
    ;;
    DELIMITER;
4、计划任务
    DROP EVENT IF EXISTS `event_day`;
    DELIMITER ;;
    CREATE EVENT `event_day` ON SCHEDULE EVERY 1 DAY
    call manage_partitions('niu_test', 'alarm_log', 365
    call manage_partitions('niu_test','realdata_log',
    end
    DELIMITER;
```

# Andy Niu Help 1.0.0.0

首页 模块 模块 模块 | 変量 Mysql常见错误 MysQL

# 模块

## 复制有关的错误

# 重变

Error_1045
navicat执行utf8格式的sql脚本第一行报错
执行SQL文件DELIMITER报错
Error_1130
Error_1172
Error_1265
Error_1418
Error_2002

### 变量说明

#### **Error 1045**

ERROR 1045 access denied for user 'root'@'localhost'原因是:root的密码错误了。

解决思路:关闭mysql服务,重新启动mysql服务,启动mysql的时候,然后登陆mysql,修改密码,退出。再重新启动mysql服务。

- 1、关闭mysql
- 2、启动mysql,指定不需要校验密码

C:\Program Files\MySQL\MySQL Server 5.1\bin>mysq]
Files\MySQL\MySQL Server 5.1\my.ini" --console --

- 3、再开一个窗口, root登陆, 修改密码
  - C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql (解释: -uroot mysql 后面这个mysql指的是连接到mysql库 mysql>
  - mysql> UPDATE user SET Password=PASSWORD('newpass
    mysql> FLUSH PRIVILEGES;
  - mysql> quit

特别注意:修改密码之后,不要忘记FLUSH PRIVILEGES; 否者i

- 4、再重新启动mysql
- 5、linux下面类似,执行 service mysql start --skip-grant
- 6、特别注意:1045是密码错误, using password yes 表示使用了 using password no 表示没有输入密码。

要与1130错误区分,1130是远程连接的用户没有权限,不允许连扩mysql服务先检查,远程连接的用户是否有权限(也就是是否允许)

### 参见

### **Error 1130**

ERROR 1130 (00000): Host '172.16.1.9' is not allowed

错误原因是:不允许从其他主机连接mysql服务

解决办法是:本机登录,授权,允许从其他主机登录。如下:

grant all privileges on \*.\* to 'root'@'%' identified flush privileges;

上面赋予的权限过大,可以使用下面的方法,赋予适当的权限。 允许myuser使用mypassword从任何主机连接到mysql服务器 GRANT ALL PRIVILEGES ON \*.\* TO 'myuser'@'%' IDENTIFIE FLUSH PRIVILEGES;

允许用户myuser从ip为10.22.4.45的主机连接到mysql服务器,并使GRANT ALL PRIVILEGES ON \*.\* TO 'myuser'@'10.22.4.45' FLUASH PRIVILEGES;

允许myuser从ip为10.22.4.45的主机连接到mysql服务器的test数igRANT ALL PRIVILEGES ON test.\* TO 'myuser'@'10.22.4.4 FLUSH PRIVILEGES;

### 参见

#### **Error 1172**

1172 - Result consisted of more than one row 错误原因:使用select aaa into bbb,而select出来的记录个数为0没有关系。

怎么解决?

对于每一条记录的Id都要处理,使用游标。

### **Error\_1265**

1265 - Data truncated for column xxx 错误原因:字面意思是数据发生截断,也就是说插入或者修改记录的时候

比如: mysql> desc t1; +	<b>.</b> 4		<b>.</b>	<b>.</b>			
Field   Type	Null	Key	Default	Extra			
c1	NO		0   NULL   NULL	   			
3 rows in set							
mysql> insert into t1(c1,c2,c3) value(1,'hhh','hello' 1265 - Data truncated for column 'c2' at row 1							
c2是double,插入'hhh'							

#### **Error 1418**

ERROR 1418 (HY000): This function has none of DETERMI in its declaration and binary logging is enabled (you log\_bin\_trust\_function\_creators variable) 错误的原因:

Mysql配置了复制,复制功能也就意味着Master数据的变化,会同步到解决办法:

- 1、创建函数的时候,明确说明我不会修改数据。创建函数的时候有5个ì
  - a、DETERMINISTIC 确定性的
  - b、NO SQL 没有包含SQ1语句,当然也不会修改数据
  - c、READS SQL DATA 只是读取数据,当然也不会修改数据
  - d、MODIFIES SQL DATA 要修改数据
  - e、CONTAINS SQL 包含了SQL语句

怎么理解DETERMINISTIC确定性的?

可以认为输入相同,方法执行过程,输出也是相同的。也就是方法的可引方法内使用了静态的数据,使用了malloc和free,使用了标准I/0函数创建函数必须明确指出a、b、c三个选项中的一个,才能执行成功。如下mysql> show variables like 'log\_bin%';

```
| log bin
                                     ON
    | log_bin_trust_function_creators | OFF
    | log bin trust routine creators | OFF
   3 rows in set
   mysql> create function fun1 (num int(9)) returns
   begin
   return 1;
   end;
   1418 - This function has none of DETERMINISTIC, N
   in its declaration and binary logging is enabled
   log bin trust function creators variable)
   mysql> create function fun1 (num int(9)) returns
   begin
   return 1;
   end;
   Query OK, 0 rows affected
   mysql> drop function fun1;
   Query OK, 0 rows affected
   mysql> create function fun1 (num int(9)) returns
   begin
   return 1;
   end;
   Query OK, 0 rows affected
2、告诉Mysql,信任我,方法不会修改数据,Mysql不再检查,即使修
   mysql> drop function fun1;
   Query OK, 0 rows affected
   mysql> set global log_bin_trust_function_creators
   Query OK, 0 rows affected
   mysql> show variables like 'log_bin%';
    +-----+
    | Variable name
                                   | Value |
   +----+
    | log bin
                                     ON
    | log bin trust function creators | ON
    | log_bin_trust_routine_creators | ON
```

+-----+
3 rows in set
mysql> create function fun1 (num int(9)) returns
begin
return 1;
end;
Query OK, 0 rows affected

3、要想mysql服务重启之后,还起作用,需要在[mysqld]增加配置项log\_bin\_trust\_function\_creators=on //注意后面注:低版本的mysql可能不支持on的语法,需要使用log\_bin\_tr

参见

### **Error 2002**

ERROR 2002 (HY000): mysql Can't connect to local MyS(

错误原因:/var/lib/mysql目录中socket文件不存在。 连接mysql服务器有两种方式:tcp连接,通过socket文件连接。通过 生成的sock文件默认放在 --datadir=/var/lib/mysql,mysql默

### 解决办法:

- 1、看看/var/lib/mysql/mysql 有没有mysql.sock文件
- 2、没有mysql.sock,重启mysql服务,看看有没有。
- 3、没有的话, ps aux|grep mysql|grep -v 'grep' 查看mysql
- 4、如果mysql服务生成在其他目录,解决办法有:

方法一、修改mysql服务生成的目录,在my.cnf 中[mysqld] socke 方法二、mysql从/var/lib/mysql/读取sock文件,建立一个软连接 方法三、修改mysql读取的目录,在my.cnf 中[mysql] socket 方法四、启动指定socket,mysql -uroot -p123456 --socket='

### 注意:

- 1、通过socket,只能在本地机器上连接。
- 2、使用-hlocalhost就是经过socket,不经过网络,通过netstat
- 3、使用-h127.0.0.1就是经过tcp,通过netstat 可以看到连接。

### navicat执行utf8格式的sql脚本第一行报错

- 1、navicat执行utf-8格式的sql脚本,报错1064-You have an e
- 2、错误原因:utf-8格式的文件开头有三个字符,这三个字符是不可见 这三个字符不可见,navicat无法识别这三个字符。
- 3、怎么解决?

使用notepad++,格式-->转为UTF-8无BOM 编码格式,可以看到注意:转为UTF-8编码格式,大小比实际长度多出三个字节。

### 执行SQL文件DELIMITER报错

```
1、运行SQL文件报错,详细信息如下:
    [Err] 1064 - You have an error in your SQL syntax
    [Err] DELIMITER ;
    [Msg] Finished - Unsuccessfully
2、执行的SOL文件内容是:
    -- Procedure structure for `vms_getchncode_bydevc
   DROP PROCEDURE IF EXISTS `vms_getchncode_bydevcoc
   DELIMITER ;;
   CREATE DEFINER=`root`@`localhost` PROCEDURE `vms
   BEGIN
           declare vDevId int default 0;
           select devid into vDevId from device when
           select device.domainid, device.devicecode
   END
   DELIMITER;
3、错误原因是:最后多了一个DELIMITER;
   去掉DELIMITER;就可以了。
```

# Andy Niu Help 1.0.0.0

章 模块 **复制有关的错误** MySQL » Mysql常见错误

# 重变

Error\_1067\_进程意外终止

### 详细描述

## 变量说明

#### Error 1067 进程意外终止

- 1、在Slave端配置Master的信息,启动Mysql,报错:错误:1067 进
- 2、出现错误一定要去看mysql的错误日志,通过 show variables ] 找到数据存放的目录,打开文件 xxx.err文件
- 3、看到错误 unknown variable 'master-host=192.168.0.22
- 4、错误原因是,从5.1.7以后,在mysql配置文件中,不支持使用maschange master to 通过help change master to 可以看到语法。
- 5、从5.1.7以后,在mysql配置文件中,不支持使用master-host选I slave-skip-errors=all replicate-do-db=rep replicate-ignore-db=mysql
- 6、另外一点, mysql复制必须要给mysql一个唯一的标识,如下: server-id=6580

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 Mysql常见问题 MysQL

# 变量

Mysql内外连接
MySQL的Embedded模式
mysql下划线和中划线
Mysql的两种连接方式
Mysql默认的数据库
Mysql有时候为什么要flush
Mysql的有关日志
mysql表的复制
根据已有的记录增加新的记录
执行sql文件,添加数据乱码
查看字符串的16进制取值
Sql_Mode
Sql注入
分割求总数
分割后求第n个元素
删除重复记录_保存Id最小的一条

## 变量说明

#### mysql下划线和中划线

```
1、通过select或者show variables 可以看到系统变量单词之间的分
  mysql> show variables like 'event%';
   +----+
   | Variable_name | Value |
   +----+
   | event_scheduler | ON
  +----+
  1 row in set (0.00 sec)
  mysql> select @@event_scheduler;
   +----+
   | @@event_scheduler |
   +----+
   +----+
  1 row in set (0.00 sec)
2、在配置文件 my.cnf中,使用下划线或者中划线都是可以的。如下:
  lower case table names=1
  default-character-set=utf8
  event-scheduler=on
3、另外对于开关变量,比如event-scheduler,使用on或者1也都是
  event-scheduler=1
  或者 event-scheduler=on
4、但是在执行命令,使用选项参数的时候,必须使用中划线。
  两个中划线是全写方式,选项和取值之间必须有=号或者空格,但是
  单中划线是简写方式,选项和取值之间不能使用=号,可以有空格也
```

```
如下:
mysql --user=root --password=123456
mysql --user root --password=123456
mysql -u root -p123456
mysql -uroot -p123456
```

#### Mysql内外连接

```
mysql> desc student;
+----+-----
| Field | Type
               | Null | Key | Default
| PRI |
                      | NULL
     | varchar(16) | YES
l NAME
| NULL
3 rows in set
mysql> desc sc;
| Field | Type | Null | Key | Default | Extra
| auto incr∈
| CID | int(11) | YES | MUL | NULL
| SCORE | int(11) | YES
                 | | NULL
4 rows in set
mysql> select * from student;
+----+
          | AGE |
     | NAME
 ID
| 10001 | Andy
             | 26
| 10002 | Bill | 27
| 10003 | Caroline | 34
 10004 | David | 46
```

```
+----+
4 rows in set
mysql> select * from sc;
| ID | SID | CID | SCORE |
+---+
| 1 | 10001 | 101 | 78
| 2 | 10001 | 102 | 67
| 3 | 10008 | 103 | 100
+---+
3 rows in set
内连接
mysql> select student.*, sc.* from student inner joir
+----+
+----+
| 10001 | Andy | 26 | 1 | 10001 | 101 | 78
| 10001 | Andy | 26 | 2 | 10001 | 102 | 67
+----+
2 rows in set
内连接等价于我们平时的自然连接,也就是:
mysql> select student.*, sc.* from student, sc where s
+----+
   | NAME | AGE | ID | SID | CID | SCORE |
+----+---+----+
| 10001 | Andy | 26 | 1 | 10001 | 101 | 78
| 10001 | Andy | 26 | 2 | 10001 | 102 | 67
+----+---+----+
2 rows in set
左连接
```

考虑下面的需求,我想列出所有学生对应的成绩,一个学生可能多个成绩使用左连接,student表 left join sc表,如下:

ID	NAME	+   AGI	•	:D	SID	CID	SC
   10001	+   Andy	26	+   1	+ .	 10001	+   101	+   78
10001	Andy	26	j 2	: j	10001	102	67
10002	Bill	27	N	IULL	NULL	NULL	NU
10003	Carolin	e   34		IULL	NULL	NULL	NU
10004	0004   David		46   NULL		NULL	NULL	NU
吏用右连接	需求,我想 ,student	表 righ	nt jo	in sc	表,如下:		
ysql> se	elect stu	dent.*,				_	t jo
TD	++-   ΝΔΜΕ	•	•		+ CTD	•	+ =
ID	++-   NAME	•	•		CID	•	+ =   +
ID   	++-	•	•	SID  1000	CID	•	+ =   +
10001   10001	   Andy     Andy	AGE   + 26   26	ID   + 1   2	SID  1000 1000	CID	SCORI   78   67	+ =   + 
10001	   Andy     Andy	AGE   +	ID   + 1	SID  1000	CID	SCORI	+ =
10001   10001   NULL	+-   Andy     Andy     NULL    +-	AGE   + 26   26	ID   + 1   2	SID  1000 1000	CID	SCORI   78   67	+ =   +     +
10001   10001   NULL   rows ir	   Andy     Andy     NULL    +-	AGE   + 26   26   NULL   +	ID   + 1   2   3   +	SID 1000 1000 1000	CID   CID   101   102   103	SCORI   78   67   100	+ =   +     +
10001   10001   NULL   rows in	+-   Andy     Andy     NULL    +-	AGE   + 26   26   NULL   +	ID   + 1   2   3   +	SID 1000 1000 1000	CID   CID   101   102   103	SCORI   78   67   100	+ =   +       +
10001   10001   NULL   rows ir	   Andy     Andy     NULL    +-	AGE   + 26   26   NULL   +	ID   + 1   2   3   +	SID 1000 1000 1000	CID   CID   101   102   103	SCORI   78   67   100	+ =   +     +
10001   10001   NULL   rows ir UBM rows ir	Andy   Andy   Andy   NULL   Set A left	AGE   + 26   26   NULL   +	ID   1   2   3   +	SID 1006 1006 1006	CID   101   102   103   103   103	SCORI	<del>-</del>       +
10001   10001   NULL   rows ir R据对称性 	   Andy     Andy     NULL    +-	AGE   + 26   26   NULL   + join B	ID   + 1   2   3   + 等价-	SID 1000 1000 1000 	CID   101   102   103   103   103   103   103	SCORI   78   67   100 -+	+ + +
10001   10001   NULL   rows ir 表据对称性 	Andy     Andy     Andy     NULL     H+-   Set   A left   Set   Eyith (     Eyith (     Eyith (	AGE 26 26 NULL	ID	SID 1006 1006 1006 1006 	CID   101   102   103   103   103   103   103   105   105	SCORI   78   67   100   1 00   +	+
10001   10001   NULL   NULL   rows ir B接接地 连接下 mysonysql> se	Andy     Andy     Andy     NULL     Helen set   A left   A left   Eyy	AGE 26 26 NULL	ID 等 接学决,ID 等 接学决。SC	SID  1000 1000  B r  以应使 fro	CID   101   102   103   103   103   103   103   105   105	SCORI   78   67   100   + in A   2   4   2   4   5   5   6   7   7   8   7   8   9   1   9   1   1   1   1   1   1   1   1   1   1	+ 
10001   10001   NULL   rows ir BE接下 连接下 等度,myso ysql> se	Andy   Andy   Andy   NULL   Set A left	AGE 26 26 NULL	ID 等 接学决, 分 第 接学决, 分 , 对法:	SID 1000 1000 1000 	CID   101   102   103   103   103   103   103   103   104   105   105	SCORI  -+   78   67   100  -+   如全   如全   如子   如子   本子	+
10001   10001   NULL   rows ir B 接接下 连接下 等的,mysc ysql> se ID	Andy     Andy     Andy     NULL     Helen set   A left   A left   Eyy	AGE+ 26 26 NULL	I I I I I I I I I I I I I I I I I I I	SID 1000 1000 1000 1000 1000 1000 1000 10	CID   101   102   103   103   103   103   103   103   103   103   104   105   105	SCORI  -+   78   67   100  -+   1	+

```
| 10002 | Bill | 27
             | NULL | NULL | NULL | NUL
| 10003 | Caroline | 34
             | NULL | NULL | NULL | NUL
| 10004 | David
        | 46 | NULL | NULL | NULL | NUL
         | NULL | 3 | 10008 | 103 | 100
I NULL I NULL
6 rows in set
注意:这里的union自动去除了重复行,如果不想去除重复行,使用uni
还有一点需要注意:就是on 之后的条件,如下:
mysql> select student.*, sc.* from student left join
+----+---+----+----
       | AGE | ID | SID | CID | SCOF
I ID | NAME
| 10003 | Caroline | 34 | NULL | NULL | NULL | NULL
| 10004 | David | 46 | NULL | NULL
                    | NULL | NULL
5 rows in set
mysql> select student.*, sc.* from student left join
+----+---+-----
   l ID
                 | SID | CID | SCOF
| 10003 | Caroline | 34 | NULL | NULL
                    | NULL | NULL
| 10004 | David | 46 | NULL | NULL
                    | NULL | NULL
+----+---+----+----
4 rows in set
mysql> select student.*, sc.* from student left join
+----+---+----+
```

```
+----+---+----+
| 10001 | Andy | 26 | 1 | 10001 | 101 | 78
+----+---+----+
1 row in set
这里看出第二个查询和第三个查询的区别,换一种写法就很清楚了。
mysql> select student.*, sc.* from student left join
| NAME | AGE | ID | SID
                     | CID
+----+---+----+----
| 10003 | Caroline | 34 | NULL | NULL | NULL | NULL
        | 46 | NULL | NULL | NULL | NULL
| 10004 | David
+----+---+----
4 rows in set
mysql> select student.*, sc.* from student left join
+----+
+----+---+----+
| 10001 | Andy | 26 | 1 | 10001 | 101 | 78
+----+---+----+
1 row in set
mysql> select student.*, sc.* from student left join
1064 - You have an error in your SQL syntax; check th
select student.*, sc.* from student left join sc on (
1、内部连接
mysql> select student.*, sc.* from student inner joir
+----+
+----+---+----+
| 10001 | Andy | 26 | 1 | 10001 | 101 | 78
```

```
| 10001 | Andy | 26 | 2 | 10001 | 102 | 67 |
+----+
2 rows in set
2、选择出sc.cid=101,再进行左连接,没有成绩的使用NULL填充
select student.*, sc.* from student left join sc on (
1、左连接
mysql> select student.*, sc.* from student left join
     I NAME
            | AGE | ID | SID
                         | CID | SCOF
| 10003 | Caroline | 34 | NULL | NULL | NULL | NULL
| 10004 | David | 46 | NULL | NULL | NULL | NULL
5 rows in set
2、再选出 sc.cid=101
```

#### 参见

### Mysql有时候为什么要flush

- 1、为了性能考虑,mysq1服务启动的时候,会把常用的信息从数据库中 这样以后在查询或者操作的时候,直接在内存中操作就行了,大大 dmu的做法也是这样的。
- 2、对于GRANT、REVOKE、SET PASSWORD操作,mysql会更新数据库 但是对于手工修改表的数据,比如insert、update,mysql内存 客户端的请求,mysql还是在内存中进行,表现为insert,upda
- 3、怎么解决这个问题?

第一种方法是:重启mysql

第二种方法是:进行相应的flush,比如flush privileges,将

4、测试如下:

mysql> drop user niu;

```
Query OK, 0 rows affected (0.00 sec)
mysql> create user niu;
Ouery OK, 0 rows affected (0.00 sec)
mysql> update user set password=password('123') v
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
登录mysql -uniu -p123 报错,如下:
[root@localhost ~]# mysql -uniu -p123
Warning: Using a password on the command line int
ERROR 1045 (28000): Access denied for user 'niu'(
进行flush privileges
登录mysql -uniu -p123成功
使用
mysql> set password for niu=password('456');
Query OK, 0 rows affected (0.00 sec)
登录mysql -uniu -p456成功
```

#### MySQL的Embedded模式

1、一般情况下,我们通过CS模式,调用MySQL客户端mysqlclient和2、但是这种C/S的数据服务不适合于一些轻量级的应用,可以使用MySQ3、这两种方式的区别是:
 CS模式,客户端链接mysqlclient库,调用mysqlclient的接口 Embedded模式,客户端链接mysqld库,调用mysqld的接口,都4、CS模式 sudo apt-get install libmysqlclient-dev g++ -o Embed Embed.cc `mysql\_config --include --]
5、Embedded模式 sudo apt install libmysqld-dev g++ -o Embed Embed.cc `mysql\_config --include --]
6、可以看一下,mysql\_config的输出,如下:
 niuzibin@ubuntu:~/work/CPP\_2/Embed\$ mysql\_config

```
Usage: /usr/bin/mysql_config [OPTIONS]
Options:
                         [-I/usr/include/mysql -[
        --cflags
                         [-I/usr/include/mysql]
        --include
        --libs
                         [-L/usr/lib/x86 64-linux
        --libs_r
                         [-L/usr/lib/x86 64-linux
        --plugindir
                         [/usr/lib/mysql/plugin]
        --socket
                         [/var/run/mysqld/mysqld.
                         [0]
        --port
        --version
                         [5.5.59]
        --libmysqld-libs [-L/usr/lib/x86 64-linux
        --variable=VAR
                        VAR is one of:
                pkgincludedir [/usr/include/mysq]
                pkglibdir [/usr/lib/x86_64-li
                plugindir [/usr/lib/mysql/plu
```

#### Mysql的两种连接方式

- 1、mysql是典型的的CS架构,连接mysql服务有两种方式: TCP/IP套接字和Unix域套接字
- 2、TCP/IP套接字是MySQL在任何平台都提供的一种连接方式,也是网线这种方式通过TCP/IP建立网络连接,一般情况下客户端在一台服务mysql -h127.0.0.1 -uroot -p123456

使用netstat可以看到TCP连接,在Linux使用tcpdump抓包可以

3、如果mysql客户端和数据库实例在同一台服务器上,可以使用Unix域注:Unix域套接字其实不是网络协议,mysql客户端和数据库实例 查看socket文件,如下:

```
mysql -hlocalhost -uroot -p123456
4、通过TCP/IP套接字连接,抓包可以获取交互信息
a、抓包获取的内容如下:
    ".....select serverid,title from serversmana ....P....
    def.ibp_niu1.serversmanagement.serversmanagen def.ibp_niu1.serversmanagement.serversmanagen ".....10.VTDU.....11.DMU.....18.VRU......"
b、具体内容,请求是select serverid,title from server 回复包括槽位的信息,以及返回的数据集合。
c、注意:登录的时候,密码字段是加密的。
```

#### Mysql的有关日志

```
1、普通日志
   mysql> show variables like 'general%';
    +------
    | general log | ON
    | general_log_file | /var/lib/mysql/localhost.log
    +-----
   2 rows in set (0.00 \text{ sec})
2、注意:普通日志,不仅记录select查询语句,还记录insert,upd
    [root@localhost ~]# tail /var/lib/mysql/localhost
                   89 Query show tables
                   89 Field List
                   89 Field List
                                        b
                   89 Field List
                                        t1
                   89 Field List
                                        t2
   160625 11:28:28 89 Query create table t1(1 160625 11:28:33 89 Query create table t100
   160625 11:37:25 89 Query insert t100 value 160625 11:37:42 89 Query show variables li 160625 11:38:09 89 Query update t100 set i
3、慢查询日志
   mysql> show global variables like '%_query%';
```

```
Variable name
                               | Value
      ------+--
   | binlog_rows_query_log_events | OFF
   | ft_query_expansion_limit
                            | 20
   | have_query_cache
                               I YES
   | long_query_time
                               1 4.000000
   | slow_query_log
                               ON
   | slow_query_log_file
                              | /var/lib/mysql/]
   6 rows in set (0.01 sec)
   mysql> set global long_query_time=3;
   Query OK, 0 rows affected (0.00 sec)
   mysql> show global variables like '%_query%';
   | Variable_name
   | binlog_rows_query_log_events | OFF
   | ft_query_expansion_limit
                            | 20
   | have_query_cache
                              | YES
                             3.00000
   | long_query_time
   | slow_query_log
   | slow_query_log_file
                              | /var/lib/mysql/]
   +-----
   6 rows in set (0.00 sec)
   mysql> select sleep(5);
   \mid sleep(5) \mid
   +----+
   1 row in set (5.00 sec)
4、查看慢查询日志:
   [root@localhost ~]# tail /var/lib/mysql/localhost
```

```
Time Id Command Argument
# Time: 160625 12:22:23
# User@Host: root[root] @ localhost [] Id: 89
# Query_time: 5.002054 Lock_time: 0.000000 Rows_
use niu1;
SET timestamp=1466828543;
select sleep(5);
/usr/sbin/mysqld, Version: 5.6.15 (MySQL Communit
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql
Time Id Command Argument
```

#### mysql表的复制

```
1、create table xxx as select xxx,创建新表,没有原表的完
mysql> desc stu;
+----+-
| | NULL
| updatetime | datetime | YES | NULL
4 rows in set
mysql> select * from stu;
+---+
| Id | Name | Age | updatetime
+---+
| 1 | Andy | 28 | 2015-03-19 15:42:09 |
+---+
1 row in set
mysql> create table stu2 as select * from stu;
Query OK, 1 row affected
Records: 1 Duplicates: 0 Warnings: 0
```

```
mysql> desc stu2;
| Field | Type | Null | Key | Default |
l Id
                ,
NULL
l Name
| updatetime | datetime | YES | NULL
4 rows in set
mysql> select * from stu2;
+---+
Id | Name | Age | updatetime
+---+
| 1 | Andy | 28 | 2015-03-19 15:42:09 |
+---+
1 row in set
2、create table xxx like xxx,创建新表,约束和原表相同,只
mysql> desc stu;
+----+-
| Field | Type | Null | Key | Default |
+----+-
   | int(9)
          | NO | PRI | NULL
4 rows in set
mysql> select * from stu;
+---+
| Id | Name | Age | updatetime
+---+
| 1 | Andy | 28 | 2015-03-19 15:42:09 |
```

```
1 row in set
mysql> create table stu3 like stu;
Query OK, 0 rows affected
mysql> desc stu3;
Id | int(9) | NO | PRI | NULL
Name | varchar(100) | NO | | NULL
                    | | NULL
| Name
| updatetime | datetime | YES | NULL
4 rows in set
mysql> select * from stu3;
Empty set
3、如果我想拷贝表的结构(约束和原表相同),同时拷贝表的数据,怎
先create table xxx like xxx, 创建表结构, 再insert into >
mysql> desc stu;
+----+-
| Field | Type | Null | Key | Default |
| updatetime | datetime | YES | | NULL
4 rows in set
mysql> select * from stu;
+---+
| Id | Name | Age | updatetime
+---+---+---+
1 | Andy | 28 | 2015-03-19 15:42:09 |
```

```
+---+
1 row in set
mysql> create table stu4 like stu;
Query OK, 0 rows affected
mysql> desc stu4;
+----+-
4 rows in set
mysql> select * from stu4;
Empty set
mysql> insert into stu4(name,age,updatetime) select r
Query OK, 1 row affected
Records: 1 Duplicates: 0 Warnings: 0
mysql> select * from stu4;
+---+
| Id | Name | Age | updatetime
+---+
| 1 | Andy | 28 | 2015-03-19 15:42:09 |
+---+---+---+
1 row in set
```

### Mysql默认的数据库

- 1、数据库mysql,保存的信息有:
  用户,帮助信息,普通日志,慢查询日志,时区有关。
- 2、数据库information\_schema,保存的信息有:

schemata, tables, columns, funcs, procedures, even 全局和会话的status和variables, processlist, innodb的資注意:有些表并没有实际的物理文件,而是对应着内存中的一些信息。

3、数据库performance\_schema,保存的信息有: 性能统计和分析

#### Sql\_Mode

```
1、Sql Mode(Sql模式), mysql可以运行在不同的Sql Mode下, 不
2、举例来说,
  mysql> select @@session.sql_mode;
   | @@session.sql_mode
  | REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNOF
  1 row in set
  mysql> select * from user;
  +---+
  | ID | NAME | PASSWORD | UPDATETIME |
  +---+
  | 1 | Andy | 123456 | NULL
  +---+----+
  1 row in set
  mysql> update user set updatetime='2015-04-31 15:
  Database changed
  Rows matched: 1 Changed: 1 Warnings: 1
  mysql> show warnings;
  +-----
   | Level | Code | Message
  | Warning | 1264 | Out of range value for column
  1 row in set
```

```
mysql> select * from user;
   +---+
   | ID | NAME | PASSWORD | UPDATETIME
   +---+
   1 | Andy | 123456 | 0000-00-00 00:00:00 |
   +---+
   1 row in set
   在ansi模式下,更新一个无效的datetime, mysql给出一个war
3、Sql模式设置为 strict_trans_tables,如下:
   mysql> set @@session.sql mode='strict trans table
   Query OK, 0 rows affected
   mysql> update user set updatetime='2015-04-31 15:
   1292 - Incorrect datetime value: '2015-04-31 15:(
   直接报错, error, 不允许更新
   考虑事务的情况,如下:
   mysql> start transaction;
   Query OK, 0 rows affected
   mysql> update user set name='Bill' where id=1;
   Database changed
   Rows matched: 1 Changed: 1 Warnings: 0
   mysql> update user set updatetime='2015-04-31 15:
   1292 - Incorrect datetime value: '2015-04-31 15:0
   mysql> commit;
   Query OK, 0 rows affected
   mysql> select * from user;
   +---+
   | ID | NAME | PASSWORD | UPDATETIME
   +---+----
   | 1 | Bill | 123456 | 0000-00-00 00:00:00 |
   +---+
   1 row in set
   出现错误,不影响前面的操作,提交有效的操作。
```

```
4、strict_trans_tables比ansi严格,traditional比strict_t
5、需要说明的, ansi, strict_trans_tables, traditional是-
   mysql> set @@session.sql_mode='ansi';
   Query OK, 0 rows affected
   mysql> select @@session.sql_mode;
   | @@session.sql_mode
   | REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNOF
   1 row in set
   mysql> set @@session.sql_mode='strict_trans_table
   Query OK, 0 rows affected
   mysql> select @@session.sql_mode;
   | @@session.sql_mode |
   +----+
   | STRICT_TRANS_TABLES |
   +----+
   1 row in set
   mysql> set @@session.sql_mode='traditional';
   Query OK, 0 rows affected
   mysql> select @@session.sql_mode;
   | @@session.sql_mode
   | STRICT TRANS TABLES, STRICT ALL TABLES, NO ZERO ]
   1 row in set
6、mysql默认的sql mode是STRICT TRANS TABLES再加上两个原子
   mysql> select @@global.sql_mode;
```

#### Sql注入

```
int userId = -1;
      stream>>userId;
   这种情况,在只有用户名的情况就可以登入系统。
   原因是拼接的sql语句,绕过了正常的条件检查。通过给name和pa
   在没有用户名的情况下,也可以登入系统。
3、sql注入产生的原因是:对用户的输入没有严格检查,导致拼接的sq
4、方法1:不要拼接sql语句,而是使用绑定变量。
   这种方法相当于,发给mysql一个框架,把空留着,MySQL解析,
   不能再表达其他的语义。如下:
   void RefuseSqlInject(otl connect& otlConn)
   {
      // 使用or
      string name="Andy";
      string password="1' or 1='1";
      // 使用注释
      //string name="Andy'/*";
      //string password="*/'";
      char sql[1024] = \{0\};
      sprintf(sql, "select id from user "
      "where name=:Name<char[100]> and password=:Pa
      otl_stream stream(1, sql, otlConn);
      int userId = -1;
      stream<<name<<password;
      stream>>userId;
5、检查用户的输入,就是检查name, password是否包含/* or 这-
   可以使用mysql自带的检查方法,也可以自己实现方法,更针对性。
```

#### 分割后求第n个元素

1、考虑下面的需求,类似Andy;Bill;Caroline,用分号分割,求出

- 2、思路:使用substring\_index, substring\_index(src,delim 对于Andy;Bill;Caroline,使用substring\_index(src,'; reverse,为lliB;ydnA 对于lliB;ydnA,使用substring\_index(src,';',1),取出 再reverse,为Bill
- 3、总的调用, select reverse(substring\_index(reverse(substr

#### 参见

#### 分割求总数

- 1、考虑下面的需求,类似Andy;Bill;Caroline,用分号分割,求出。
- 2、简单的思路,求出原字符串长度,将分号替换为空,求出二者的长度 select length('Andy;Bill;Caroline') length(rep] 注:加1是因为,三个空,会有4个栏杆。

#### 参见

#### 删除重复记录 保存Id最小的一条

```
方法1:
1、创建一个临时表,选取需要的数据。
2、清空原表。
3、临时表数据导入到原表。
4、删除临时表。
mysql> select * from student;
+---+--+
| ID | NAME |
+---+--+
| 11 | aa |
| 12 | aa |
| 13 | bb |
| 14 | bb |
| 15 | bb |
| 16 | cc |
+---+---+
```

```
6 rows in set
mysql> create temporary table temp as select min(id),
Query OK, 3 rows affected
Records: 3 Duplicates: 0 Warnings: 0
mysql> truncate table student;
Query OK, 0 rows affected
mysql> insert into student select * from temp;
Query OK, 3 rows affected
Records: 3 Duplicates: 0 Warnings: 0
mysql> select * from student;
+---+
+---+
| 11 | aa |
| 13 | bb |
| 16 | cc |
+---+
3 rows in set
mysql> drop temporary table temp;
Query OK, 0 rows affected
这个方法,显然存在效率问题。
方法2:按name分组,把最小的id保存到临时表,删除id不在最小id集
mysql> create temporary table temp as select min(id)
Query OK, 3 rows affected
Records: 3 Duplicates: 0 Warnings: 0
mysql> delete from student where id not in (select mi
Query OK, 3 rows affected
mysql> select * from student;
+---+
```

```
| ID | NAME |
| 11 | aa |
| 13 | bb |
| 16 | cc |
+---+
3 rows in set
方法3:直接在原表上操作,容易想到的sql语句如下:
mysql> delete from student where id not in (select mi
执行报错:1093 - You can't specify target table 'stude
原因是:更新数据时使用了查询,而查询的数据又做了更新的条件,mys
怎么规避这个问题?
再加一层封装,如下:
mysql> delete from student where id not in (select mi
Query OK, 3 rows affected
mysql> select * from student;
+---+
+---+
| 11 | aa |
| 13 | bb |
| 16 | cc |
+---+
3 rows in set
```

#### 执行sql文件,添加数据乱码

- 1、使用navicat执行sql文件,中文没有乱码。
- 2、直接在linux下使用mysql客户端执行,如下:
  mysql -uroot -p123456 --database=niu1 <t1.sql
  插入的记录显示乱码
- 3、怎么解决? 在sql文件的开头加上 /\*!40101 SET NAMES utf8 \*/;

#### 查看字符串的16进制取值

```
1、使用hex可以查看字符串的16进制取值,如下:
   mysql> select username from user where userid=5;
   l username l
   Ⅰ 吕欣盈   Ⅰ
   +---+
   1 row in set
   mysql> select hex(username) from user where useri
   +----+
   | hex(username) |
   | C2C0D0C0D3AF |
   1 row in set
   可以看到存储格式是qb2312,因为每个中文字符占用2个字节。
2、将username字段的存储格式修改为utf8,查询16进制取值,如下:
   mysql> select hex(username) from user where useri
   +----+
   | hex(username)
   | E59095E6ACA3E79B88 |
   1 row in set
   可以看到是utf8编码格式,对于utf8编码格式,一个中文字符占。
   另外:中文字符的utf8编码前4bit的取值都是E,这是因为在uni
```

#### 根据已有的记录增加新的记录

```
1、考虑下面的需求,根据已有的记录,增加新的记录,但是有些字段取2、示例如下:
mysql> select * from t2;
+---+---+
```

```
| c1 | c2 | c3 | time
  1 | 1 | 1 | 2016-04-19 20:14:45 |
+---+---
1 row in set
mysql> insert into t2(c1,c2,c3,time) select 5555,
Query OK, 1 row affected
Records: 1 Duplicates: 0 Warnings: 0
mysql> select * from t2;
+----+
+----+
   1 | 1 | 1 | 2016-04-19 20:14:45 |
      1 | 1 | 2016-04-19 20:14:45 |
| 5555 |
+----+
2 rows in set
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

<sup>首页</sup>模块 **Mysql数据类型** MysQL

# 重变

# double类型

int\_11

### 详细描述

## 变量说明

#### double类型

- 1、double(5,2) 表示总的位数是5,小数点后面是2位。
- 2、插入数据,整数部分超出,报错: 1264 - Out of range value for column 'v3' at row 小数部分超出,不会报错,四舍五入只取两位

#### int 11

```
| Andy11 | NULL |
| 11
| 101 | Andy101 | NULL |
mysql> alter table student modify id int(2);
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> desc student;
+----+---+----+----
+----+---+----+----
| NAME | varchar(16) | YES | MUL | NULL
+----+----+----
3 \text{ rows in set } (0.01 \text{ sec})
mysql> select * from student;
+----+
| id | NAME | AGE |
+----+
| 11 | Andy11 | NULL |
| 101 | Andy101 | NULL |
+----+
3 \text{ rows in set } (0.00 \text{ sec})
mysql> alter table student modify id int(2) zero1
Query OK, 3 rows affected (0.04 sec)
Records: 3 Duplicates: 0 Warnings: 0
mysql> desc student;
| Field | Type
                    | Null | Key |
```

```
| int(2) unsigned zerofill | NO
                                      PRI
         | varchar(16)
                                YES
   | NAME
                                      MUL
         |
| int(11)
   | AGE
                                YES
   3 rows in set (0.01 sec)
   mysql> select * from student;
        | NAME
    id
                | AGE
   | 101 | Andy101 | NULL |
   +----+
   3 \text{ rows in set } (0.00 \text{ sec})
4、得出结果:
   a、int(N) 必须与zerofill 结合,才能有效果。
   b、只有宽度不够的,才补充0,宽度超过的,显示实际宽度。
   c、使用mysql客户端有效果,使用navicat等第三方软件可能没有
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

	substring	
	substring_index	
	last_insert_id	
	found_rows	
	日期时间操作有关	
	常用的系统信息函数	
	批量删除表	

## 变量说明

#### found rows

```
1、用法如下:
    mysql> select sql_calc_found_rows c1 from t1 limi
    +---+
    | c1 |
    +---+
    | 2 |
    +---+
    1 row in set

mysql> select found_rows();
+------+
    | found_rows() |
    +-----+
    | 3 |
    +-----+
    1 row in set

2、注意: found_rows()必须紧接着 sql_calc_found_rows执行,
    用户变量设置为found_rows(),再select用户变量
```

#### last\_insert\_id

```
1、last_insert_id()返回最后插入的Id,用法如下:
    select last_insert_id();
2、last_insert_id只与连接有关,不同的连接之间互不相关.
3、last_insert_id有一些约束条件:
```

```
a、id是自增的
b、使用insert,id是系统自己产生的,而不是用户填写的,如下
mysql> insert into t1(c2,c3) values(100,100);
Query OK, 1 row affected
mysql> select * from t1;
+---+
| c1 | c2 | c3 |
+---+
| 1 | 100 | 100 |
| 2 | 100 | 100 |
+---+
2 rows in set
mysql> select last_insert_id();
| last_insert_id() |
1 row in set
// 用户填写id,返回还是上一次的insert
mysql> insert into t1(c1,c2,c3) values(45,100,100
Query OK, 1 row affected
mysql> select last_insert_id();
| last_insert_id() |
+----+
1 row in set
c、单个insert插入多个记录 , last insert id返回第一个记录
mysql> insert into t1(c2,c3) values(100,100);
Query OK, 1 row affected
```

```
mysql> select * from t1;
+---+
| c1 | c2 | c3
+----+
| 1 | 100 | 100 |
+---+
1 row in set
mysql> insert into t1(c2,c3) values(100,100),(100)
Query OK, 2 rows affected
Records: 2 Duplicates: 0 Warnings: 0
mysql> select * from t1;
+---+
| c1 | c2 | c3
+----+
| 1 | 100 | 100 |
 2 | 100 | 100 |
  3 | 100 | 100 |
+---+
3 rows in set
mysql> select last_insert_id();
+----+
| last_insert_id() |
+----+
1 row in set
```

#### substring

```
ba
1 row in set
mysql> select substring('football' from 5 for 2);
| substring('football' from 5 for 2) |
 ba
1 row in set
mysql> select substring('football', -6, 4);
| substring('football', -6, 4) |
1 row in set
mysql> select substring('football' from -6 for 4)
| substring('football' from -6 for 4) |
otba
+------
1 row in set
注:负数从右向左
```

#### substring\_index

#### 常用的系统信息函数

```
mysql> select version();
+----+
| version() |
+---+
5.5.18
+---+
1 row in set
mysql> select user();
+----+
| user()
| root@pvs256971713 |
1 row in set
mysql> select database();
+----+
| database() |
+----+
| ibp
1 row in set
```

#### 批量删除表

```
-- Procedure structure for `clear`
-- Procedure structure for `clear`

DROP PROCEDURE IF EXISTS `clear`;

DELIMITER ;;

CREATE DEFINER=`mysql`@`%` PROCEDURE `clear`()

BEGIN

declare vTableName varchar(32) default '';

declare vDynamicSql text default '';

declare vIsOver int default 0;

declare vCursorTableName cursor for select ta

declare continue handler for sqlstate '02000'

open vCursorTableName;

fetch vCursorTableName into vTableName;

while(vIsOver = 0) do
```

```
select vTableName;
                -- drop table vTableName;
                set vDynamicSql = concat('drop table
                -- select vDynamicSql;
                call exe_dynamic_sql(vDynamicSql);
        fetch vCursorTableName into vTableName;
    end while;
    close vCursorTableName;
        truncate table source_video_info;
        truncate table video_synopsis_summary;
        show tables;
        select * from source_video_info;
        select * from video_synopsis_summary;
END
;;
DELIMITER ;
-- Procedure structure for `exe_dynamic_sql`
DROP PROCEDURE IF EXISTS `exe_dynamic_sql`;
DELIMITER ;;
CREATE DEFINER=`mysql`@`%` PROCEDURE `exe_dynamic_sql
BFGTN
    set @exeSql = iDynamicSql;
    prepare stmt from @exeSql;
    execute stmt;
    deallocate prepare stmt;
END
DELIMITER ;
```

```
1、日期转int
   mysql> select to_days('2016-4-12');
   | to_days('2016-4-12') |
   +----+
              736431 l
   +----+
   1 row in set
2、int转日期
   mysql> select from_days(736431);
   | from_days(736431) |
   +----+
   | 2016-04-12
   1 row in set
3、unix时间戳(unix timestamp)表示从unix的开始时间【1970红
   为有符号int类型,最大能表示到2038年。
4、当前日期转化为unix时间戳,如下:
   mysql> select now();
   +----+
   | now()
   +----+
   | 2016-09-06 14:18:16 |
   1 row in set
   mysql> select unix_timestamp('2016-09-06 14:18:16
   +-----+
   | unix_timestamp('2016-09-06 14:18:16') |
   1 row in set
5、unix时间戳转化为日期
   mysql> select unix_timestamp();
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 Mysql有关文件 MysQL

# 重变

文件介绍

文件结构

mysql数据文件

### 变量说明

#### mysql数据文件

- 1、mysql的数据文件存放在data目录,每个数据库有一个对应的目录。
- 2、对于MyISAM引擎,每个表有三个文件:
  - .frm 是元数据文件,保存表结构信息
  - .myd 【my data】是表数据文件,保存表的数据,也就是一
  - .myi 【my index】是表索引文件,保存表的索引信息,一个
- 3、对于INNODB引擎,每个表也都有一个对应的.frm文件,用于保存表 默认情况下,INNODB所有表的数据和索引都保存在 ibdata1 文 这会导致文件ibdata1非常大,怎么解决?
- 4、可以设置每个表使用一个独立的文件,来保存自身的数据和索引,只innodb\_file\_per\_table可以看到每个表都有一个对应的.ibd文件
- 5、对于已经存在比较大的ibdata1,怎么办?

先把数据导出来,删除ibdata1文件,配置成innodb\_file\_per

- 6、对于INNODB引擎,还需要记录事务,因此有事务日志。
  - INNODB有若干个 事务日志组,一个事务日志组至少有两个事务日文件的内存都是二进制0,INNODB会轮巡写事务日志文件。

默认情况下,只有一个事务日志组,事务日志组有两个文件,分别。 初始的时候,ib\_logfile0 和 ib\_logfile1文件的内容都是二

- 7、注意:事务日志和二进制日志的区别。
  - 事务日志是INNODB引擎记录事务,为了事务的前滚和后滚。
  - 二进制日志(binlog)所有的引擎都会记录,可以认为记录mysq.
- 8、.err是错误日志文件,记录mysql的出错信息
- 9、.pid是保存进程ID,停止mysql服务,这个文件就会自动删除。
- 10、对于查询日志,需要配置
  - log=C:\ProgramData\MySQL\MySQL Server 5.5\data\lc
- 11、对于慢查询日志,需要配置

Variable_name	12、	long_query_time=0.1 log-slow-queries=C:\ProgramData\MySQL\MySQL 特别注意:\s会被转义,因此需要使用C:\ProgramData 或者C:\ProgramData\MySQL\MySQL Server 5.5\d 通过show variables like '%log%'; 可以看到各种  mysql> show variables like '%log%'; +	\MySQL ata\\s.
binlog_cache_size			Value
max_binlog_size   10737		binlog_cache_size   binlog_direct_non_transactional_updates   binlog_format   binlog_stmt_cache_size   expire_logs_days   general_log   general_log_file   innodb_flush_log_at_trx_commit   innodb_locks_unsafe_for_binlog   innodb_log_buffer_size   innodb_log_file_size   innodb_log_files_in_group   innodb_log_group_home_dir   innodb_mirrored_log_groups   log   log_bin   log_bin_trust_function_creators   log_error   log_output   log_queries_not_using_indexes   log_slave_updates   log_slow_queries   log_warnings   max_binlog_cache_size   max_binlog_size   max_binlog_stmt_cache_size	32768   OFF   STATE   32768   O   ON   C:\Pr   1   OFF   20971   55572   2   .\   1   ON   OFF   OFF

```
relay_log
 relay_log_index
| relay_log_info_file
                                         relay
relay_log_purge
                                         ON
| relay_log_recovery
                                         OFF
| relay_log_space_limit
                                         0
 slow_query_log
                                         ON
| slow_query_log_file
                                       I C:\Pr
| sql_log_bin
                                         ON
 sql_log_off
                                         OFF
| sync binlog
                                         0
sync_relay_log
                                         0
 sync_relay_log_info
以及慢查询的时间设置
mysql> show variables like '%long%';
| Variable_name
| long_query_time
 max_long_data_size
 performance_schema_events_waits_history_long_si
+------
```

#### 文件介绍

- 1、在linux上安装好mysql之后,存在很多mysql有关的文件。
- 2、/etc/rc.d/init.d/mysql是mysql服务的启动脚本,是对mysqlmysqlmysqld\_safe是一个脚本,是对可执行文件mysqld的封装。
- 3、/usr/bin/mysql 是mysql客户端命令行工具,在同目录下还有其注意:mysqld\_safe也在这个目录,但是mysqld不在这个目录,/usr/sbin/目录下面放了很多服务,s认为是服务的意思。
- 4、/usr/lib/mysql 是一个目录,里面放着mysql有关的库文件。
- 5、/usr/include/mysql 是一个目录,里面放在很多的头文件,其中
- 6、/usr/share/mysql 下面放着一些mysql脚本文件,比如 mysql
- 7、/var/lib/mysql 是mysql数据库的存放目录,每个数据库有一个a、auto.cnf是mysql server的uuid

- b、ib\_logfile0是日志文件,还有一个ib\_logfile1,大小相同
- c、localhost.localdomain.err是错误日志
- d、localhost.localdomain.pid是mysql服务的进程ID
- 8、通过show variables like '%dir%' ,可以查看相应的目录文

#### 文件结构

1、初始化选项文件

window下在mysql的安装目录,文件为 my.ini Linux下/etc/my.cnf

2、错误日志文件

错误日志文件和数据库的数据文件放在一起,通过选项datadir型mysql的错误日志文件是排查错误的重要手段。

3、查询日志文件

分为慢查询日志和普通查询日志

4、二进制日志文件

这个文件在数据库复制的时候非常重要,master在binlog记录下然后把事情重放一遍,保证数据同步。mysql的复制是slave端配master,请求二进制日志。这一点和oracle不同,oracle是mamysql的二进制日志文件前缀一般配置为 mysql-bin

5、中继日志和复制状态文件

relay-bin 中继日志,保存读取到的master二进制日志relay-log.info 复制状态日志,保存处理的进度和中继日志式

6、数据文件

每个数据库对应一个目录,目录中有一些表文件

7、其他文件

进程id文件 pid

sock文件

# Andy Niu Help 1.0.0.0

 首页
 模块

 查询操作

 MySQL

# 重变

汇总数据

分组数据

使用子查询

## 变量说明

#### 使用子查询

```
1、所谓子查询,就是嵌套在其他查询中查询,经常使用的场景,如下:
   mysql> select name, teaid from stu where teaid in
            | teaid |
     name
   | Andy
   | Bill
   | Caroline |
   | David
   l Eric
   | Andy
   | David
   | Eric
   8 rows in set
   mysql> select name, teaid from stu where teaid in
   +----+
             | teaid |
   | Andy
    | Bill
   | Caroline |
   | David
    Eric
    Andy
                   2
```

```
David | 1 |
    Eric |
   8 rows in set
2、作为计算字段使用子查询,统计每个老师的学生个数,如下:
   mysql> select name, (select count(1) from stu wher
   +----+
           | StuNum |
   l name
   l Mr Li
   | Mr Zhang |
   | Mrs Niu |
   3 rows in set
   执行过程如下:
   投影name,对于tea表的每一个id取值,传给子查询,统计学生个
   相对于第一种子查询,这种子查询叫做相关子查询,因为这种子查
3、对于group by,也可以使用相关子查询来实现,如下:
   mysql> select name, avg(age) from stu group by nam
            | avg(age)
    name
   | Andy | 15.0000
   | Bill | 17.0000
   | Caroline | 17.0000
   | David | 11.5000
           | 19.0000
   | Eric
   +-----+
   5 rows in set
   mysql> select distinct name, (select avg(age) from
        | AvgAge
   +-----+
          | 15.0000 |
   | Andy
   | Bill | 17.0000 |
   | Caroline | 17.0000 |
```

执行过程:对于每一个distinct name,传给相关子查询,用于约

#### 分组数据

1、汇总数据是对满足条件的所有记录进行聚集。考虑下面的需求,将记再对每个逻辑组进行聚集计算,怎么办?

2、使用group by 进行分组,如下:

select name, avg(age) from stu group by name;

3、考虑下面两个语句的区别:

select name from stu where id>2;

select name, avg(age) from stu where id>2 group by 前一个语句,先选择,再投影,select字段name

后一个语句,先选择,然后进行逻辑分组,得到多个逻辑分组,每对每个分组进行投影,select字段name,对其它字段进行聚集计特别注意:对于分组,select的内容(也就是投影的内容)一般能当然,也可以select不是分组的字段,语法没错。但是,这样做为进行逻辑分组之后,对于其他字段,一个分组中往往包含多个这个数据往往没有意义,对其它字段的一组数据进行聚集计算才有

4、如果过滤分组,比如检查age的平均值,怎么办?

使用 having,这个时候的过滤条件往往是 其它字段的聚集函数 select name, max(age) from stu group by name havir

5、select的字句顺序和执行顺序

select xxx from xxx where xxx group by xxx havinç 执行顺序:

- a、首先根据where进行行级选择
- b、group by 进行分组
- c、通过having 进行分组过滤
- d、然后order by xxx limit
- e、最后投影

思考:为什么投影是最后进行的操作?

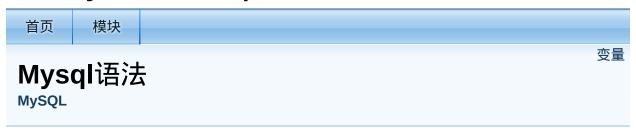
因为行过滤,分组过滤,有可能是根据其它的字段进行过滤,如果:

```
1、有时候我们需要汇总数据,而不是查询记录。Mysql提供5个聚集函数
   count
   max
   min
   avg
   sum
2、示例如下:
   mysql> select count(age) from stu;
   | count(age) |
   +---+
   1 row in set
   mysgl> select max(age) from stu;
   | max(age) |
     19 l
   +---+
   1 row in set
3、考虑下面的需求,某一个字段,对于不同的值进行聚集,怎么办?
   使用distinct,如下:
   select sum(distinct age) from stu;
   注意:函数max和min,可以对不同的值进行聚集,但这样做并没有
4、如果同时求出平均值和最大值,怎么办?
   使用组合聚集函数,也就是对多个聚集函数组合,如下:
   mysql> select count(age), avg(age), max(age) from s
   +----+
   | count(age) | avg(age) | max(age) |
   +----+
          5 | 15.8000 |
   +----+
   1 row in set
```

```
mysql> select count(age),avg(distinct age),max(aq+-----+
| count(age) | avg(distinct age) | max(age) |
+-----+
| 5 | 15.5000 | 19 |
+-----+
1 row in set
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

	Mysql语法注意事项	
	Mysql执行动态sql语句	
	Mysql游标示例	
	mysql循环控制	
	mysql的return语句	
	myoqii ji otarrivii 19	
	主键	
	索引	
	类型转换	

### 变量说明

#### mysql循环控制

```
1、使用while
    DROP PROCEDURE IF EXISTS `addstudent`;
    DELIMITER ;;
    CREATE PROCEDURE `addstudent`(iNum int)
    BEGIN
    declare vI int default 0;
    start transaction;
    while(vI < iNum) do
    insert into student(id, name) values(vI, concat('Ar
    set vI = vI+1;
    end while;
    commit;
    END
    DELIMITER;
2、使用repeat
    BEGIN
    declare vI int default 0;
    start transaction;
    repeat
    insert into student(id,name) values(vI,concat('Ar
```

```
set vI = vI+1;
    until vI >= iNum end repeat;
    commit;
    END
3、使用loop
    BEGIN
    declare vI int default 0;
    start transaction;
    label insert: LOOP
    insert into student(id, name) values(vi, concat('Ar
    set vI = vI+1;
    if(vI >= iNum) then
    leave label_insert;
    end if;
    end LOOP label_insert;
    commit;
    END
4、考虑下面的需求,在编程语言中循环控制中有continue,如何实现。
    使用 iterate label_insert; 对于LOOP如下:
    BEGIN
    declare vI int default 0;
    start transaction;
    label insert: LOOP
    if(vI=2) then
    set vI = vI+1;
    iterate label_insert;
    end if;
```

```
insert into student(id,name) values(vi,concat('Ar
    set vI = vI+1;
    if(vI >= iNum) then
    leave label_insert;
    end if;

end LOOP label_insert;

commit;
    END

5、注意:对于while和repeat也是同样的道理,但是需要在while和r
    label_insert: while
    label_insert: repeat
    mysql中暂时没有for循环。
```

#### Mysql执行动态sql语句

```
DROP PROCEDURE IF EXISTS `exe_dynamic_sql`;
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `exe_dynamic_sql`
BEGIN
set @exeSql = iDynamicSql;
prepare stmt from @exeSql;
execute stmt;
deallocate prepare stmt; -- 释放掉预处理段
END
;;
DELIMITER ;
```

#### Mysql游标示例

```
BEGIN

declare vPartitionName varchar(32) default '';
declare vIsOver int default 0;

declare vCursorPartitionName cursor for select pa
declare continue handler for sqlstate '02000' set
```

```
open vCursorPartitionName;
fetch vCursorPartitionName into vPartitionName;
while(vIsOver = 0) do
    -- do sth
    fetch vCursorPartitionName into vPartitionName end while;
close vCursorPartitionName;
END
```

参见

#### mysql的return语句

```
1、mysql存储过程没有return语句,怎么办?
2、使用变通的办法,在BEGIN处打一个标签,leave这个标签,就等价 label_1:BEGIN if(i > 5) then leave label_1; end if; select "aaa"; END
3、注意:标签一般只能打在begin和loop
```

#### Mysql语法注意事项

- 1、名称在前,类型在后
- 2、中划线与下划线有时候是等价的
- 3、value与values有时候是等价的

#### 主键

1、创建主键

```
alter table t1 add primary key(c1);
2、删除主键
   alter table t1 drop primary key;
3、查看主键
   show index from t1;
   或者
   desc t1;
```

#### 类型转换

```
1、类型转换有两种方法,如下:
   mysql> select cast('123' as unsigned integer);
   +----+
   | cast('123' as unsigned integer) |
   +----+
                          123 l
   +-----+
   1 row in set
   mysql> select convert('123',unsigned integer);
   | convert('123',unsigned integer) |
   1 row in set
2、需要注意的是,目标类型只能是下面的类型。
   BINARY[(N)]
   CHAR[(N)]
   DATE
   DATETIME
   DECIMAL
   SIGNED [INTEGER]
   TIME
   UNSIGNED [INTEGER]
```

3、注意:在Mysql中float,double是非标准的数据类型,在数据库F 而decimal是标准的数据类型,Mysql内部用字符串保存decima]

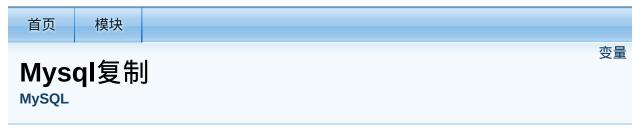
```
4、在mysql中,数值作为字符串,字符串作为数值,不需要特殊转化,
   mysql> select 12.1+13.4;
   +----+
   | 12.1+13.4 |
   | 25.5 |
   +---+
   1 row in set
   mysql> select '12.1'+'13.4';
   | '12.1'+'13.4' |
       25.5 |
   +----+
   1 row in set
   mysql> select concat('12.1','13.4');
   | concat('12.1','13.4') |
   | 12.113.4
   +----+
   1 row in set
   mysql> select concat(12.1,13.4);
   | concat(12.1,13.4) |
   +----+
   12.113.4
   +----+
   1 row in set
   注意:字符串转换为数值失败,当做0来处理,如下:
   mysql> select '12.1'+'aaa';
   | '12.1'+'aaa' |
```

```
12.1 |
  +----+
  1 row in set
5、数值作为字符串,字符串作为数值,不需要特殊转化,如下:
  mysql> desc t1;
  | c2 | double | YES
                   | NULL
  | c3 | varchar(100) | YES | | NULL
  3 rows in set
  mysql> select * from t1;
  +---+
  | c1 | c2 | c3
  +---+
  | 1 | 12.35 | 12345 |
  +---+
  1 row in set
  // 下面的update语句,数值作为字符串,字符串作为数值,没有
  mysql> update t1 set c2='25.6', c3=100 where c1='1
  Ouery OK, 1 row affected
  Rows matched: 1 Changed: 1 Warnings: 0
  mysql> select * from t1;
  +----+
  | c1 | c2 | c3 |
  +---+
  | 1 | 25.6 | 100 |
  +---+
  1 row in set
```

```
1、创建索引
alter table t1 add index index_c2(c2);
或者
create index index_c2 on t1(c2);
2、删除索引
alter table t1 drop index index_c2;
或者
drop index index_c2 on t1;
3、查看索引
show index from t1;
或者
desc t1;
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

主主复制的配置流程

show\_slave\_status

Mysql复制错误

Reconnecting\_after\_a\_failed\_master\_event\_read

Client\_requested\_master\_to\_start\_replication\_from\_impossible\_p

半同步复制

查看二进制日志

### 变量说明

#### Client\_requested\_master\_to\_start\_replication\_from\_impossible\_po

```
1、错误如下:
   mysql> show slave status\G
   Slave_IO_State:
                  Master Host: 192.168.1.176
                  Master_User: root
                  Master_Port: 3306
                  Connect Retry: 60
               Master_Log_File: mysql-bin.000010
           Read_Master_Log_Pos: 994941638
               Relay_Log_File: mysqld-relay-bin.0002
                  Relay_Log_Pos: 4
           Relay_Master_Log_File: mysql-bin.000010
               Slave_IO_Running: No
               Slave_SQL_Running: Yes
               Replicate Do DB: ibp
           Replicate_Ignore_DB: mysql
           Replicate_Do_Table:
       Replicate_Ignore_Table:
       Replicate_Wild_Do_Table:
   Replicate_Wild_Ignore_Table:
                  Last_Errno: 0
                  Last Error:
                  Skip_Counter: 0
           Exec_Master_Log_Pos: 994941638
               Relay_Log_Space: 106
```

```
Until Condition: None
               Until_Log_File:
                   Until_Log_Pos: 0
           Master SSL Allowed: No
           Master SSL CA File:
           Master SSL CA Path:
               Master_SSL_Cert:
               Master_SSL_Cipher:
               Master_SSL_Key:
           Seconds Behind Master: NULL
   Master SSL Verify Server Cert: No
                   Last IO Errno: 1236
                   Last IO Error: Got fatal error 12
               Last SQL Errno: 0
               Last_SQL_Error:
   1 row in set (0.00 sec)
2、从对方去复制一个不可能的位置,查看Master端的二进制日志,如<sup>-</sup>
    [root@localhost mysql]# 11 |grep mysql-bin.00001@
    -rw-rw---- 1 mysql mysql 994934249 2015-12-03 16
3、也就是说,Slave去复制位置994941638,而Master端的位置只有
4、查看Master端的错误日志,如下:
   InnoDB: Log scan progressed past the checkpoint ]
   151204 9:12:26
                    InnoDB: Database was not shut do
    InnoDB: Starting crash recovery.
   InnoDB: Reading tablespace information from the .
   InnoDB: Restoring possible half-written data page
    InnoDB: buffer...
   InnoDB: Doing recovery: scanned up to log sequence
   151204 9:12:26 InnoDB: Starting an apply batch
   InnoDB: Progress in percents: 7 8 9 10 11 12 13 1
   40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 5
   1 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97
   InnoDB: Apply batch completed
   InnoDB: Last MySQL binlog file position 0 9949416
           9:12:27 InnoDB: Started; log sequence nu
   151204
   也就是说, Master端异常关闭, 重启之后, 进行崩溃恢复, 而且也
   但是恢复之后,位置只有994934249,通过show binlog even
```

- 5、也就是说, mysql异常关闭(直接断电), 重启恢复二进制日志的时这个时候, 两边的数据还是一致的。因为Master也写入数据库了, 只不过Master端恢复出现错误,同时Slave复制的位置没有了。
- 6、解决办法是,手工调整复制的文件和位置,从下一个文件的位置0开撑
- 7、异常关闭mysql(直接断电),除了会导致上面的问题。还会导致下对于增删改操作,mysql记录二进制日志不会实时刷新到磁盘。通为0,mysql不控制binlog的刷新,靠文件系统自己控制。大于0,mysql调用文件系统的刷新操作,刷新到磁盘。sync\_bir

大于0, mysql调用文件系统的刷新操作,刷新到磁盘。sync\_bir没有很好的理由, mysql建议不要修改sync\_binlog的值,靠文作因为设置小了,导致频繁刷新到磁盘,性能很差。设置大了,刷新用的现来了:

异常关闭mysql,可能出现数据已经修改了,但是二进制日志没有。对于这个问题,可以通过半同步复制来避免,但是这个要使用mysc

8、也就是说,根据mysql复制的实现原理,异常关闭mysql会导致一些如果出现了异常关闭,人工介入,进行调整。

#### Mysql复制错误

#### 主备切换的流程:

- 1、hau彼此通信,初始化,认为当前设备是备机,收不到对方的消息,
- 2、hau认为当前设备是备机,调用rpu sdk的0nSlave,hau认为当前
- 3、OnSlave做的事情:
  - a、调用黄彪的脚本,禁用外网卡,删除外网卡的网关,同时把网身
  - b、把cmu, uac, uas, virgo等服务的可用状态设置为0,因为6而dmu, vtdu要负载均衡,两边都可用, mysql, hau两边也
  - c、然后关闭cmu, uac, uas, virgo, 虽然guard是活着的,但:
- 4、OnMaster做的事情:
  - a、调用黄彪的脚本,激活外网卡,设置外网卡的网关,免费ARP,设置路由规则,对于出去的修改ip地址,发到外部,对于进来
  - b、所有服务的可用状态设置为1,guard会把它们启动。

### 经过反复测试,基本可以确定的是:

- 1、mysql正常的启动关闭,复制是不存在问题的。
- 2、对于目前的处理流程,正常的操作,复制是没有问题的。

#### 问题是:

这两次出现的错误,错误码都是1236,就是Slave端的IO线程,从Mas 1、第一次的错误信息是,Slave从Master读取一个不存在的问题 2、第二次的错误信息是, Master端的二进制日志被截断。

网上查了,基本上是出现这种问题,怎么解决? 出现这种情况,需要stop slave,需要使用change master to,修

为什么出现这种情况?如何避免出现?

- 1、第一次的错误信息是,Slave从Master读取一个不存在的问题原因是:安装有问题,主主复制一端卸载又安装。比如A,B目前复A安装的时候清除二进制日志,新的位置从0开始,而B还认为从85°
- 2、第二次的错误信息是, Master端的二进制日志被截断。 原因是:主库意外死机,导致binlog有问题。

注意:二进制日志,使用二进制格式保存,如果查看?

需要使用程序 mysqlbinlog

#### Reconnecting\_after\_a\_failed\_master\_event\_read

1、rpu2000T 测试环境, mysql复制出现问题,如下: mysql> show slave status\G Slave\_IO\_State: Reconnecting after a Master\_Host: 192.168.1.176 Master User: root Master Port: 3306 Connect\_Retry: 60 Master\_Log\_File: mysql-bin.000003 Read\_Master\_Log\_Pos: 49136994 Relay\_Log\_File: mysqld-relay-bin.0000 Relay\_Log\_Pos: 48636785 Relay\_Master\_Log\_File: mysql-bin.000003 Slave\_IO\_Running: No Slave\_SQL\_Running: Yes Replicate\_Do\_DB: ibp Replicate\_Ignore\_DB: mysql Replicate\_Do\_Table: Replicate Ignore Table: Replicate Wild Do Table:

```
Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                   Skip Counter: 0
           Exec Master Log Pos: 49136994
               Relay_Log_Space: 48637086
               Until Condition: None
               Until Log File:
                   Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master SSL CA File:
           Master SSL CA Path:
               Master SSL Cert:
               Master_SSL_Cipher:
               Master_SSL_Key:
           Seconds Behind Master: NULL
   Master_SSL_Verify_Server_Cert: No
                   Last_IO_Errno: 2013
                   Last_IO_Error: error reconnecting
               Last_SQL_Errno: 0
               Last_SQL_Error:
   1 row in set (0.00 sec)
2、出现这种情况的原因是,对端作为master的mysql服务器关闭了,
   重连间隔是60秒,重连86400次。
   注:在配置文件中, master-connect-retry重连间隔时间, ma
3、当对端的Master重新启动之后,slave重连成功,会接着读取二进制
   这个场景,我已经模拟测试多次,是没有问题的。
4、现在的问题是,出现了一直重连不成功,我看到slave端的mysql错
   151124 15:05:05 [Note] Slave: received end packet
   151124 15:05:05 [Note] Slave I/O thread: Failed r
   37827
   151124 15:05:05 [ERROR] Slave I/O: error reconnect
   rror code: 2013
   151124 15:06:05 [Note] Slave: connected to master
   position 37827
   151125 9:35:26 [Note] Slave: received end packet
   151125 9:35:26 [Note] Slave I/O thread: Failed r
```

```
6366231
151125 9:35:26 [ERROR] Slave I/O: error reconnect
rror_code: 2013
151203 10:10:35 [Note] Error reading relay log ex
在151124 15:05:05 对端的master关闭了,后面有重新连接成
但是在151125 9:35:26 ,对端的master关闭了,后面没有重新
5、出现了这种情况,我通过在slave端执行 stop slave; 然后star
```

#### show slave status

```
Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.1.1.1
                  Master_User: rep_user
                  Master_Port: 3306
                Connect_Retry: 10
              Master_Log_File: binlog.000026
          Read_Master_Log_Pos: 446
               Relay_Log_File: relay.000008
                Relay_Log_Pos: 589
        Relay Master Log File: binlog.000026
             Slave_IO_Running: Yes
            Slave SQL Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate Wild Do Table:
  Replicate Wild Ignore Table:
                   Last Errno: 0
                   Last Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 446
              Relay_Log_Space: 878
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
```

Master\_SSL\_CA\_File:

Master\_SSL\_CA\_Path:

Master\_SSL\_Cert:

Master\_SSL\_Cipher:

Master\_SSL\_Key:

Seconds\_Behind\_Master: 0

Master\_SSL\_Verify\_Server\_Cert: No

Last\_IO\_Errno: 0

Last\_IO\_Error:

Last\_SQL\_Errno: 0

Last\_SQL\_Error:

Replicate\_Ignore\_Server\_Ids:

Master\_Server\_Id: 2211

#### 可以这样理解:

- 1、从库I/0线程到主库中读取二进制文件,记录读取哪个二进制文件,
- 2、从库I/0线程把读取到二进制内容,放入中继日志。
- 3、从库SQL线程重放中继日志,记录重放哪个中继日志,以及重放到中等中继日志重放之后,产生一个位置,这个位置对应主库二进制的位于中继日志的位置,和重放之后产生的位置(对应主库的二进制位置)。
- 4、这是典型的生产者/消费者模式,中继日志就是共享区,I/0往里面加

#### Slave有两个线程:

- 1、I/O线程从Master接收二进制日志文件,写入本地的中继日志
- 2、SQL线程,读取中继日志,重放。典型的生产者/消费者模型

#### 中继日志有三个位置:

- 1、中继日志的结尾, I/O从Master读到了哪个位置
- 2、SQL执行中继日志,执行到了哪个位置
- 3、重放之后,产生一个位置,这个位置对应Master上面的二进制文件(

Read\_Master\_Log\_Pos与Master上面的show master status 位: Read\_Master\_Log\_Pos和Exec\_Master\_Log\_Pos相等,说明没有需要完全同步,满足两个条件:

- 1、Read\_Master\_Log\_Pos与Master上面的show master status
- 2、Read\_Master\_Log\_Pos和Exec\_Master\_Log\_Pos相等,说明没证

特别注意:Seconds\_Behind\_Master不是表示Slave比Master落后。Seconds\_Behind\_Master=0 不表示Slave与Master完全同步,而是要判断是否完全同步,还需要判断Master记录到了哪个位置和Slave读

#### 注:

- 1、二进制日志中,不包括数据查询语句。因为只是select,不会修改数
- 2、在Master端,可以并行执行增删改操作,串行记录二进制日志。 Slave端串行接收二进制日志,对于接收的二进制写入到中继日志。 对于写入的中继日志,Slave只有一个SOL线程重放中继日志,不能

#### 主主复制的配置流程

- 1、先关闭B,把A的数据导出来,mysqldump -hlocalhost -uroot 注意:如果只导出表结构,使用 -d, --no-data No re
- 2、关闭A,启动B,进入mysql建立一个新的数据库 create databas
- 3、导入数据库 mysql -hlocalhost -uroot -p123456 <ibprpu 或者进入mysql,通过命令 source /root/ibprpu.sql 导入
- 4、修改A和B的配置文件, my.cnf, 重启A和B
- 5、注意:有些版本my.cnf中的复制属性配置不起作用,先关闭slave
- 6, stop slave;
  - change master to master\_host='192.168.1.101', mast 必要的时候还需要指定master log file 和 master log po
- 7、Last\_IO\_Errno: 1236 错误的原因是 从master读取一个无效的 在备机上,关闭slave, change master 重新指定master\_log 如果还有错误, stop slave; set global sql\_slave\_skip
- 8、主机要记录二进制日志,配置如下:

+---+

#### 半同步复制

- 1、mysql的复制是异步的,也就是说,Master提交事务后,写入二进制并不等待二进制日志传送给Slave。
- 2、考虑下面的极端情况,Master提交事务,写入二进制日志(可能还) 导致这个事务丢失,Slave损失这个事务,主从不一致。
- 3、怎么解决这个问题?

Master提交事务,写入二进制日志时,并不及时回复客户端成功,这样就保证了两份日志记录,一份在Master的二进制日志,一份在

- 4、为什么叫半同步复制?
  - 如果要完全同步复制,需要Master提交事务,写入二进制日志,后但是这里,只是成功发给了Slave, Slave还没有执行,数据还不
- 5、注意:二进制日志, mysql不会删除, 因为可能好几天之后在配置Sl需要手工处理二进制日志。对于中继日志, mysql会定时清除,它

#### 查看二进制日志

- 1、使用mysqlbinlog程序, mysqlbinlog --help 或者 mysqlbin注意:如果是binlog格式是ROW,使用mysqlbinlog查看会是乱。
- 2、mysql登录之后,使用语法 show binlog events in 'mysqlhelp show binlog events; 查看帮助文档
- 3、注意:使用mysqlbinlog,--start-position必须是二进制日志 [root@localhost mysql]# mysqlbinlog mysql-bin.000 /\*!40019 SET @@session.max\_insert\_delayed\_threads /\*!50003 SET @OLD\_COMPLETION\_TYPE=@@COMPLETION\_TYPE=DELIMITER /\*!\*/;

# at 994934222

#151203 3:03:39 server id 213 end\_log\_pos 99493 COMMIT/\*!\*/;

DELIMITER;

# End of log file

ROLLBACK /\* added by mysqlbinlog \*/;

/\*!50003 SET COMPLETION\_TYPE=@OLD\_COMPLETION\_TYPE 如果使用了没有存在的位置,报错如下:

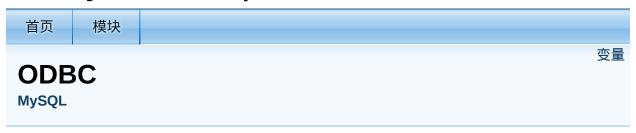
[root@localhost mysql]# mysqlbinlog mysql-bin.00@
/\*!40019 SET @@session.max\_insert\_delayed\_threads
/\*!50003 SET @OLD\_COMPLETION\_TYPE=@@COMPLETION\_TYPE=###
DELIMITER /\*!\*/;

```
ERROR: Error in Log_event::read_log_event(): 'rea
   DELIMITER;
   # End of log file
   ROLLBACK /* added by mysqlbinlog */;
   /*!50003 SET COMPLETION TYPE=@OLD COMPLETION TYPE
4、那么问题来了,如果不知道位置怎么办?
5、可以先使用show binlog events找到位置, show binlog events
   mysql> show binlog events in 'mysql-bin.000010' ]
   Log_name: mysql-bin.000010
          Pos: 994933417
   Event type: Query
   Server id: 213
   End_log_pos: 994933547
      Info: use `ibp`; delete from dev_accesory whe
   Log_name: mysql-bin.000010
          Pos: 994933547
   Event_type: Xid
   Server id: 213
   End_log_pos: 994933574
      Info: COMMIT /* xid=4154883 */
   Log_name: mysql-bin.000010
          Pos: 994933574
   Event_type: Query
   Server id: 213
   End_log_pos: 994933646
      Info: BEGIN
   ************************ 4. row **********
   Log_name: mysql-bin.000010
          Pos: 994933646
   Event_type: Query
   Server id: 213
   End_log_pos: 994934222
      Info: use `ibp`; insert into dev_accesory(DE\
                 values( NAME_CONST('_devid',1867)
```

```
Log_name: mysql-bin.000010
          Pos: 994934222
   Event type: Xid
   Server id: 213
   End log pos: 994934249
       Info: COMMIT /* xid=4154885 */
   5 rows in set (4.81 sec)
   注意:一个增删改操作或者事物,分为3个操作,一个BEGIN,一个
6、二进制日志中的位置就是文件大小,多少个字节数,需要注意的是:
   [root@localhost mysql]# 11 |grep mysql-bin.00001(
   -rw-rw---- 1 mysql mysql 994934249 2015-12-03 16
   [root@localhost mysql]# mysqlbinlog mysql-bin.00@
   /*!40019 SET @@session.max insert delayed threads
   /*!50003 SET @OLD COMPLETION TYPE=@@COMPLETION TY
   DELIMITER /*!*/;
   # at 994934222
   #151203 3:03:39 server id 213 end_log_pos 99493
   COMMIT/*!*/;
   DELIMITER;
   # End of log file
   ROLLBACK /* added by mysqlbinlog */;
   /*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

## Win7\_64位使用MysqlOdbc

### 详细描述

## 变量说明

#### Win7\_64位使用MysqlOdbc

- 1、首先不能安装Mysql Odbc 64位,因为我们的Mysql是32位,使用报错:驱动程序与应用程序之间的体系结构不匹配。
- 2、要安装Mysql Odbc 32位,安装好之后,在数据源中会找不到Mysc 实际上是安装好了的,运行 C:\Windows\SysWOW64\odbcad32
- 3、SysWOW64 是System Windows32 On Windows64,一个Windows

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 模块

## OTL常见错误

## 变量

otl绑定变量

otl多线程设置和自动提交

otl中文的有关问题

## 变量说明

#### otl中文的有关问题

```
1、示例代码如下:
   void Test_Xutao(otl_connect& otlConn)
   {
       char* sql ="select name from nbs_program_org
       string name;
       try
       {
           otl_stream stream(1,sql,otlConn);
           stream>>name;
       }
       catch (otl_exception& ex)
       {
           printf("ExecuteSql Error, ErrorMsg[%s], 5
              ex.msg,
              ex.stm_text);
       }
   鍒嗙粍1 是utf8格式的中文【分组1】,因为vs使用gb来解释ut1
   上面的查询,不能查询回来结果。
2、通过抓包到3306的端口,发现与数据库的交互,并不是utf8格式。
   怎么解决这个问题?
3、使用占位符的方式,如下:
   void Test_Xutao(otl_connect& otlConn)
   {
       char* sql ="select name from nbs_program_org
       string name;
```

```
try
        {
            otl_stream stream(1,sql,otlConn);
            stream<<"鍒嗙粍1";
            stream>>name;
        catch (otl_exception& ex)
            printf("ExecuteSql Error, ErrorMsg[%s], 5
                ex.msg,
                ex.stm_text);
        }
4、上面的问题,在windows下才有,linux两种方式都是可以的。如下
    void Test_Xutao111(otl_connect& otlConn)
    {
            char* sql ="select name from nbs_program_
            string name;
            try
            {
                    otl_stream stream(1,sql,otlConn);
                    stream>>name;
            catch (otl_exception& ex)
                    printf("ExecuteSql Error, ErrorMs
                            ex.msg,
                            ex.stm_text);
            printf("Test_Xutao111[%s]\n\n", name.c_s
    }
    void Test_Xutao222(otl_connect& otlConn)
    {
            char* sql ="select name from nbs_program_
            string name;
```

#### otl多线程设置和自动提交

1、多线程设置如下:
 otl\_connect::otl\_initialize(1);
 方法原型是:static int otl\_initialize(const int thr
2、自动提交设置如下:
 \_dbConnect.rlogon(szConStr,1);
 方法原型是:void rlogon(const char\* connect\_str, cc
3、没有设置自动提交,需要手动设置执行commit,如果没有执行comm 如果没有执行commit,相当于当前start一个事务,这个事务的所 其他事务修改了这条记录,再次读取还是老的记录,这就是问题所 也就是说,在当前事务的执行过程中,其他事务的修改,在当前事

#### otl绑定变量

```
1、示例代码如下:
    void Test3(otl_connect& otlConn)
    {
        char* sql ="insert into stu(name,age) value(:
        try
        {
```

```
otl_stream stream(1,sql,otlConn);
stream<="aaa"<<29;"
}
catch (otl_exception& ex)
{
    printf("ExecuteSql Error, ErrorMsg[%s], sex.msg, ex.stm_text);
}
}
2、需要注意的是:
a、对于string的输入,绑定变量不能是varchar,必须是charb、不管数据库中字段的类型是char,varchar,或者text,绑定、对于char必须指定长度,可以认为是分配Buf的大小,如下:Nid、注意:长度必须是文本数字,不能变量或者表达式,因为解析的e、如果输入的string长度,大于指定的char长度,报错 Input
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块

OTL常见错误
MySQL » OTL

## 变量

Error\_32019

使用otl的一个注意事项

重连数据库失败

流操作数据类型不匹配

stream\_flush

mysql\_Commands\_out\_of\_sync

使用绑定变量otl对类型检查非常严格

otl查询结果与预期不一致

### 变量说明

#### **Error 32019**

```
otl_stream::operator>>() should have been called befored 1、错误原因如下:
    把otl的输出写入到const int或者const对象的int字段。 错误已经说得很明白,>>要在int()之前调用,而这里是const int result =0; const int result =0; const int result dd = result;

sprintf(sql,"call test1(:in_Age<int>)"); otl_stream stream1(1, sql, otlConn,otl_implicit_stream1<<101; stream1>>dd;
```

参见

#### mysql\_Commands\_out\_of\_sync

```
1、代码如下:
void TestCache(otl_connect& otlConn)
{
    try
{
```

```
char sql[1024] = \{0\};
           sprintf(sql, "call test1(1)");
           otl_stream stream(100, sql, otlConn,otl_i
           int id;
           while(!stream.eof())
               stream>>id;
               char sql2[1024] = \{0\};
               sprintf(sql2, "call test2(:Id<int>)");
               otl stream stream2(100, sql2, otlConr
               stream2<<id;
               int ff = 0;
               while(!stream2.eof())
               {
                   stream2>>ff;
               }
           }
       }
       catch(otl_exception& ex)
       {
           printf("ExecuteSql Error, ErrorMsg[%s], 5
               ex.msg,
               ex.stm_text);
       }
2、执行otl_stream stream2(100, sql2, otlConn,otl_impli
   Commands out of sync; you can't run this command
   特别注意:如果test1 只返回1条或者0条记录,不会导致这个异?
3、错误原因:mysql上一次的查询没有将结果集释放掉,又进行下一次
4、otl:在第一个stream读取期间,第二个stream使用了绑定变量,
5、解决办法:
   a、第二个stream不使用绑定变量,如下:
   void TestCache(otl_connect& otlConn)
       try
```

```
char sql[1024] = \{0\};
        sprintf(sql, "call test1(1)");
        otl_stream stream(100, sql, otlConn,otl_i
        int id;
        while(!stream.eof())
            stream>>id;
            char sql2[1024] = \{0\};
            sprintf(sql2, "call test2(%d)",id);
            otl_stream stream2(100, sql2, otlConr
            int ff = 0;
            while(!stream2.eof())
            {
                stream2>>ff;
            }
        }
    }
    catch(otl_exception& ex)
    {
        printf("ExecuteSql Error, ErrorMsg[%s], 5
            ex.msg,
            ex.stm_text);
    }
b、先把第一个stream读取完,再进行第二个stream,如下:
void TestCache(otl_connect& otlConn)
{
    try
    {
        char sql[1024] = \{0\};
        sprintf(sql, "call test1(1)");
        otl_stream stream(100, sql, otlConn,otl_i
        vector<int> intVec;
```

```
int id;
            while(!stream.eof())
            {
                stream>>id;
                intVec.push_back(id);
            }
            for(vector<int>::iterator iter = intVec.t
                iter != intVec.end(); ++iter)
            {
                char sql2[1024] = \{0\};
                sprintf(sql2, "call test2(:Id<int>)");
                otl_stream stream2(100, sql2, otlConr
                stream2<<id;
                int ff = 0;
                while(!stream2.eof())
                    stream>>ff;
                }
            }
        catch(otl_exception& ex)
            printf("ExecuteSql Error, ErrorMsg[%s], S
                ex.msg,
                ex.stm_text);
        }
6、特别注意:使用存储过程,存储过程内部使用游标,并且存储过程有
```

必须修改实现,使用等效的方法处理。

参见

otl查询结果与预期不一致

- 1、问题表现是:执行查询得到记录,在其它地方修改这条记录,再次查
- 2、思考为什么?

原因是当前没有设置自动提交,相当于在当前start一个事务,这读取一条记录,其它事务修改了这条记录,再次读取还是老的记录也就是说,在当前事务的执行过程中,其它事务的修改,在当前事

3、事务的可重复读是如何实现的?

使用多版本并发控制, MVCC (MultiVersion Concurrency Concurrency

但是,当一个Session连接在事务操作过程中,修改了数据,就为修改的只是他自己的版本,其它的Session连接是看不到的。

对于修改了数据的Session连接,提交之后,合并到主干版本,其 4、怎么解决?

- a、设置当前连接session为自动提交,每次查询都是一条独立的
- b、修改当前连接session的事务隔离级别。但是这样会存在其它 隔离级别设置为提交读,导致问题,两次读取中间,其它事务 隔离级别设置为串行化,会严重影响访问性能。

#### stream\_flush

```
/*
在Windows下面,加上 stream.flush(); 会明显影响效率,递但是,奇怪的是,在Linux环境下,加上 stream.flush(); 能够完整插入1000条,而且效率特别高,违反常识。
*/
#ifdef WIN32
#define FLUSH(s)
#else
#define FLUSH(s) s.flush()
#endif
```

#### 参见

#### 使用otl的一个注意事项

- 1、存在问题:服务使用otl与mysql连接,如果长时间没有交互,再去
- 2、因为没有sql断开连接的通知,为了解决这个问题,有以下几种策略
  - a、每次直接执行sql语句,出现异常,重新连接mysql,再去执行
  - b、类似tcp的保活机制,与mysql的连接,定时执行一个简单的si
  - c、每次执行sql语句之前,先执行一个简单测试语句,测试连接是
- 3、在dmu和vmu中,使用了第一种方式,对执行sql语句进行了封装,有但是这里有个地方需要注意,如下:

```
try
{
    wall_cfg_list wall_list;
    otl stream walls;
    // 在executeSql检查是否有异常
    executeSql("{call vms_query_wall(:id<int>)}",
    walls << -1;
    while (!walls.eof())
    {
        wall_cfg cfg;
        walls >> cfg.id >> cfg.state >> cfg.domai
        wall_list.wall_list.push_back(cfg);
    }
catch (otl_exception &e)
{
    LogErr(VMULOG, "dealtvWallQueryAll failed, er
    return IBP_Err_DBQuery_Fail;
return IBP_Err_OK;
```

假如在executeSql中出现异常,executeSql并不能捕获到,要也就是说,输入绑定变量之后,otl才能判断出异常。一旦出现这个4、怎么解决上面的问题?

使用绑定变量,可以解决两个问题:

- a、批量操作,提高性能,减少sql的解析过程,循环传入参数。
- b、预防sql注入。

如果不存在上面的两种情况,解决办法就是不使用绑定变量。如下

```
char sql[1024]={0};
sprintf(sql,"call vms_query_wal(-1)");

wall_cfg_list wall_list;
otl_stream walls;
executeSql(sql,walls,1,otl_implicit_select);
5、但是有些情况,必须使用绑定变量,比如批量操作或者防止sql注入。这个时候,方法executeSql封装不起作用,因为输入绑定变量才是因为操作数据库的方法很多,每个方法不一样,导致每个封装。6、怎么办?
需要结合第二种方式或者第三种方式来处理,这就导致增加了额外,哪种方式更好?
取决于本身执行sql语句的频率,如果频率很高,采用第二种方式
```

#### 参见

#### 使用绑定变量otl对类型检查非常严格

```
1、举例来看,如下:
  mysql> desc t3;
  +----+---+----
  +----+----+----
  2 rows in set
2、示例代码
  void Test4(otl_connect& otlConn)
  {
    char* sql ="insert into t3(c1,c2) value(:C1<i</pre>
    unsigned int aaa = 1001;
    string name = "Andy";
    try
    {
      otl_stream stream(1,sql,otlConn);
```

```
stream<<aaa<<name;
     }
     catch (otl_exception& ex)
     {
        printf("ExecuteSql Error, ErrorMsq[%s], 5
           ex.msg,
           ex.stm text);
     }
  执行 stream<<aaa<<name; 报错 Incompatible data typ
  也就是 流操作存在不兼容的数据类型
3、原因是:
  绑定变量的时候,告诉它是int类型,但是实际上传输的是unsigr
4、解决办法是:
  修改绑定变量的类型,如下:
  char* sql ="insert into t3(c1,c2) value(:C1<unsiq</pre>
  或者修改传输变量的类型,如下:
  int aaa = 1001;
5、特别需要注意的是:mysql本身是弱类型的。
  也就是说,int可以当成varchar来使用,varchar也可以当成ir
  mysql> desc t3;
  +----+---+----
  2 rows in set
  mysql> select * from t3;
  Empty set
  mysql> insert into t3(c1,c2) value('1001',456);
  Query OK, 1 row affected
  mysql> show warnings;
  Empty set
```

```
mysql> select * from t3;
  +----+
   | c1 | c2 |
   | 1001 | 456 |
   +----+
  1 row in set
  也就是insert into t3(c1,c2) value('1001',456); 可以
6、现在考虑,把varchar当成int来使用,但是不是有效的int值,如
  mysql> insert into t3(c1,c2) value('a1001',456);
  Query OK, 1 row affected
  mysql> show warnings;
  | Level | Code | Message
  +------
   | Warning | 1366 | Incorrect integer value: 'a100
  +-----
  1 row in set
  mysql> select * from t3;
  +----+
   | c1 | c2 |
  +---+
   | 1001 | 456 |
      0 | 456 |
   +----+
  2 rows in set
  可以看到,产生了告警,但是记录还是插入进去了,数据进行强制:
7、如果希望这种情况报错,怎么办?
  设置sql模式为更严格的检查,如下:
  mysql> select @@session.sql_mode;
  +----+
   | @@session.sql_mode |
  +----+
```

#### 流操作数据类型不匹配

```
1、报错 Incompatible data types in stream operation,说
2、错误代码
    void Test2(otl_connect& otlConn)
    {
        char* sql ="select now()";
        try
        {
            otl_stream stream(1,sql,otlConn);
            string aa;
            stream>>aa;
            int hh = 0;
        }
```

```
catch (otl_exception& ex)
{
    printf("ExecuteSql Error, ErrorMsg[%s], sex.msg,
    ex.stm_text);
}

错误原因是:select出来now(),类型是datetime,而aa是str3、怎么解决?
进行类型转化,如下:
char* sql ="select cast(now() as char(32))";
char* sql ="select convert(now(), char(32))";
```

#### 重连数据库失败

```
/*

千万不要定义宏OTL_ODBC_MYSQL,这玩意是针对MyODBC 2.5,如果定义了宏OTL_ODBC_MYSQL,会导致下面的情况:
mysql服务关闭
应用程序连接失败,sleep,然后重新连接,一直循环
mysql服务启动,
这个时候,重新连接mysql服务不能连接成功

另外还有一点,otl_connect& otlConn不要使用动态分配。
/*
#ifndef OTL_ODBC_MYSQL
#define OTL_ODBC_MYSQL // 错误
#endif

#ifndef OTL_ODBC
#define OTL_ODBC
#define OTL_ODBC // 正确,Compile OTL 4/ODBC
#endif
```

#### 参见

# Andy Niu Help 1.0.0.0

事件调度器 MySQL

## 重变

事件调度器

### 变量说明

#### 事件调度器

```
1、事件调度器(Event Scheduler),和计划任务(Scheduled Ta
2、语法如下:
    mysql> help create event;
    Name: 'CREATE EVENT'
    Description:
    Syntax:
    CREATE
        [DEFINER = { user | CURRENT_USER }]
        EVENT
        [IF NOT EXISTS]
        event name
        ON SCHEDULE schedule
        [ON COMPLETION [NOT] PRESERVE]
        [ENABLE | DISABLE | DISABLE ON SLAVE]
        [COMMENT 'comment']
        DO event_body;
    schedule:
        AT timestamp [+ INTERVAL interval] ...
    | EVERY interval
        [STARTS timestamp [+ INTERVAL interval] ...]
        [ENDS timestamp [+ INTERVAL interval] ...]
    interval:
        quantity {YEAR | QUARTER | MONTH | DAY | HOUF
                WEEK | SECOND | YEAR_MONTH | DAY_HOUF
```

### DAY\_SECOND | HOUR\_MINUTE | HOUR\_SECON



# 重变

执行计划

### 详细描述

## 变量说明

#### 执行计划

```
1、执行计划就是在sql语句之前加上explain,使用desc 也可以。
2、desc有两个选项extended和partitions, desc extended 将原
  desc partitions 可以查看使用分区表的信息。
3、比如:
mysql> desc select * from student;
| id | select_type | table | type | possible_keys |
| student | ALL | NULL
id 是执行计划的编号,可以理解为方法的调用堆栈,调用堆栈先进后出
mysql> desc select * from student where id>1 union se
| id | select type | table | type | possible
| NULL | UNION RESULT | <union1,2> | ALL | NULL
mysql> desc select * from student where id in (select
+---+---+----
| 1 | PRIMARY | student | ALL | NULL
```

2   DEPENDENT SUBQUERY   student   AL					ID +		
	2 rows in set ################## select_type 查询类型 1、simple,只是简单的2 mysql> desc select	,主要的取 查询。如下	值如下:	######## +	########## 		
	id   select_type	table	type   possible_key		ole_keys		
	1   SIMPLE	studen	t   ALL	+   NULL			
		imary/union/union result ,组合查询 > desc select * from student where id>1 union se +					
	id   select_type	e   tab	le	type	possible +		
	1	stu	dent    dent    ion1,2>	range	PRIMARY		
	3、primary/dependent mysql> desc select '	•	•		in (select		
id   select_type							
	1   PRIMARY   2   DEPENDENT SUBQUERY		student sc	ALL     ALL	NULL   NULL		
4、primary/derived, 查询的目标不是物理表,也就是使mysql> desc select min(id) from (select id fr							
	id   select_type ++	table	ty <sub> </sub>	pe   pos	ssible_key		

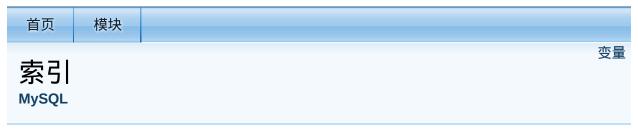
```
| 1 | PRIMARY | <derived2> | ALL | NULL
| 2 | DERIVED | student | ref | stu_index
table 引用的表
type是指访问类型,主要的取值有 all,index,range,ref,eq
1、all 遍历全表
mysql> desc select name from student;
+---+
| id | select_type | table | type | possible_keys |
+---+-----
| 1 | SIMPLE | student | ALL | NULL
2、index 使用索引,遍历索引树
mysql> create index index_name on student(name);
Query OK, 0 rows affected
Records: 0 Duplicates: 0 Warnings: 0
mysql> show index from student;
+-----+
| Table | Non_unique | Key_name | Seq_in_index |
student | 0 | PRIMARY | 1
+----+-
2 rows in set
mysql> desc select name from student;
| id | select_type | table | type | possible_keys
+---+----
| 1 | SIMPLE | student | index | NULL
```

```
3、range 使用索引,扫描范围
mysql> desc select name from student where name in ('
| id | select_type | table | type | possible_keys
| 1 | SIMPLE | student | range | index_name
4、ref 可以认为使用辅助索引,可能会有多个记录匹配。
  注意ref 与eg ref的区别, eg ref是使用主键索引,而ref是使
会有多个name为Andy的记录
mysql> desc select name from student where name = 'Anc
  | id | select_type | table | type | possible_keys |
| 1 | SIMPLE | student | ref | index_name |
id为2的查询,对于teacher的一个name,从student找到找到所有na
mysql> desc select * from student, teacher where stude
+---+-------
| id | select_type | table | type | possible_keys
| 1 | SIMPLE | teacher | index | index_name
| 1 | SIMPLE | student | ref | index_name
+---+----
5、eq_ref,使用主键索引,只有一个记录匹配,如下:
因为id是student主键, student只有一条记录匹配teacher的id。
mysql> desc select * from student, teacher where stude
+---+----
| id | select_type | table | type | possible_keys
| 1 | SIMPLE | teacher | index | PRIMARY
| 1 | SIMPLE | student | eq_ref | PRIMARY
6、const mysql对查询优化,转化为常量
mysql> desc select * from student where student.id=10
```

```
| id | select_type | table | type | possible_keys
| 1 | SIMPLE | student | const | PRIMARY
1 row in set
7、system 是const的一个特例,查询的记录只有一条
mysql> desc select id from (select * from student whe
+---+----+-----
| 1 | PRIMARY | <derived2> | system | NULL
| 2 | DERIVED | student | const | PRIMARY
2 rows in set
8、null 没有记录,都不需要执行查询
mysql> desc select 1;
| id | select_type | table | type | possible_keys | }
| 1 | SIMPLE | NULL | NULL | NULL
1 row in set
possible_keys 可能用到的索引
kev 用到的索引
key len 用到的索引长度,和三个因素有关:
是否允许为null,允许为null 多出一个字节标识是不是null
是否变长,变长多出两个字节表示长度
字符编码,不同字符编码,同一个字符占用的内存不一样latin1[1],
ref 匹配的条件
mysql> desc select * from student, teacher where stude
| id | select_type | table | type | possible_keys
```

| teacher | index | index\_name | SIMPLE | student | ref | | index name | SIMPLE rows 估算扫描的行数 Extra 额外信息,主要有 覆盖索引,从索引中就能找到所需要的信息 Using index Using where Using temporary 无法利用索引完成的排序 Using filesort MvS0L执行计划的局限性: 不考虑触发器、存储过程的信息或用户自定义函数对查询的影响情况 不考虑各种Cache 不能显示MySQL在执行查询时所作的优化工作 部分统计信息是估算的,并非精确值 只能解释SELECT操作,其他操作要重写为SELECT后查看。

### 参见



# 重变

MylSAM与InnoDB的索引实现

索引标识PRI\_UNI\_MUL

### 详细描述

## 变量说明

### MyISAM与InnoDB的索引实现

- 1、MyISAM 使用B+Tree 作为索引结构,叶子节点的data存放指针,
- 2、InnoDB 也使用B+Tree作为索引结构,也别需要注意的是,对于主题 InnoDB的数据文件本身就是就是索引文件。而MyISAM,主键索引
- 3、InnoDB数据文件,要按主键聚集索引,这就要求InnoDB的表必须要 InnoDB会自动选择一个可以唯一标识记录的字段作为主键,比如a InnoDB会自动生成一个隐含字段作为主键,这个隐含字段6个字节
- 4、对于InnoDB的辅助索引,叶子节点的data存放的是主键的值。这就 首先使用辅助索引找到主键的值,根据主键的值,使用主键索引找
- 5、InnoDB的辅助索引为什么要这样设计? 如果辅助索引data存放的行指针,当行移动或者数据页分裂时,需 data存在主键的值,就没有这个问题。行移动和数据页分裂,主键 相当于增加一个间接层。这个间接层对性能的影响也很小,因为通 和辅助索引字段的增删改,辅助索引还是要更新的。
- 6、了解了innoDB的索引实现,有几个地方需要注意: 不要使用过长的字段作为主键,因为辅助索引都要使用主键索引定 使用单调的字段作为主键,特别是insert的时候,如果是非单调的
- 7、这就能很好解释,隔离级别 repeatable-read, 不使用索引锁住
- 8、在查询执行计划中,有一个字段 type, eq\_ref 表示使用主键索而ref 表示先使用辅助索引,找到主键的值,再使用主键索引定位

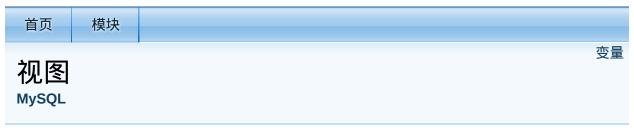
### 参见

### 索引标识PRI\_UNI\_MUL

1、通过desc 描述表结构, key字段显示有 PRI, UNI, MUL, 如下:

```
mysql> desc student;
   | Null | Key | Default | E
    Field
          | Type
          | int(11) | NO
   | id
                           | PRI | NULL
   | SCHOOL | varchar(64) | YES
                                I NULL
         | varchar(16) | YES
    NAME
                           | UNI | NULL
          | int(11) | NO
   l AGE
                           | MUL | NULL
2、表示的意思如下:
   PRI: 主键
   UNI:唯一键索引
   MUL:非唯一键索引
3、特别注意:对于多列索引,也就是多个字段组成的索引,往往只显示
   而且显示的规则有些复杂,可以通过show index查看具体的信息。
   mysql> create index index_name_age_school on stuc
   Query OK, 0 rows affected
   Records: 0 Duplicates: 0 Warnings: 0
   mysql> desc student;
                     | Null | Key | Default | E
    Field
          | Type
      PRI | NULL
   | SCHOOL | varchar(64) | YES
                                | NULL
   | NAME
          | varchar(16) | YES
                           | UNI | NULL
          | int(11) | NO
   l AGE
                           | MUL | NULL
   4 rows in set
   mysql> show index from student;
   +-----+-
    Table | Non_unique | Key_name
   l student l
                    0 | PRIMARY
   | student |
                   0 | index name
   | student | 1 | index age
```

### 参见



# 重变

视图

### 详细描述

### 变量说明

#### 视图

- 1、视图是一个虚拟表,可以认为对原表封装了一下,一般情况下,可以
- 2、视图的实现由两种策略:临时表算法与合并算法。临时表算法:把视
- 3、举例来说, mysql> create view bigAgeStuView as select 对于查询 mysql> select \* from bigAgeStuView where 如果采用临时表算法, Mysql会把select \* from stu where 如果采用合并算法, Mysql会把对视图的查询转化为select \* f显然, 临时表算法会存在严重的性能问题。
- 4、对于create view bigAgeStuView as select \* from stu mysql> create algorithm=temptable view bigAgeStuView Query OK, 0 rows affected

id   select_type   table   type   possib	=					
1   SIMPLE   stu   ALL   NULL						
1 row in set merge 执行计划显示,没有使用派生表						
<pre>mysql&gt; create algorithm=undefined view bigAgeStu\ Query OK, 0 rows affected</pre>						
mysql> desc extended select * from bigAgeS	StuView;					
id   select_type   table   type   possib	_					
1   SIMPLE   stu   ALL   NULL						
1 row in set undefined(由Mysql决定使用合适的算法)执行计划显示,没有						
需要注意的是,临时表算法存在性能问题,因此我们倾向于使用合并 mysql> create algorithm=merge view bigAgeStuView Query OK, 0 rows affected						
mysql> show warnings;						
Level	Code   Message					
Warning   1354   View merge algorithm ca						
1 row in set						
mysql> desc extended select * from bigAgeS	•					
id   select_type   table   type	possik					
1   PRIMARY   <derived2>   system     2   DERIVED   stu   ALL  </derived2>	NULL					

6、使用临时表算法,没有办法更新原表的记录,这个很好理解,因为临



# 模块

Python知识点

Python常用功能

# 详细描述



# 重变

Python导入导出
理解import
Python_Swap
Python的设计哲学
Python命令行查看帮助文档
查看帮助信息
pyc文件和pyo文件
name
元祖不可变

## 变量说明

\_\_name\_\_\_

```
1、考虑下面的需求,写了一个模块,里面有测试代码,运行的时候,执
   但是,当这个模块被import的时候,我们不期望执行测试代码,怎
2、这就需要使用,__name__,如下:
   if __name__ == '__main__':
      print "this is animal"
      raw_input()
    __name___可以认为是条件编译,当运行的时候,取值为'___main_
   被import的时候,取值为 模块名称
3、测试如下:
   class Animal(object):
   name=''
   age=0
   print ___name___
   if name _ == '__main__':
      print "this is animal"
      raw_input()
```

### pyc文件和pyo文件

1、pyc文件,是python编译后的字节码(bytecode)文件。只要你运一个对应的pyc字节码文件。这个pyc字节码文件,经过python解(这也是为什么pyc文件可以跨平台部署,类似于java的跨平台,下次调用直接调用pyc,而不调用py文件。直到你这个py文件有改

python解释器会检查pyc文件中的生成时间,对比py文件的修改图 2、pyo文件,是python编译优化后的字节码文件。pyo文件在大小上,如果想得到某个py文件的pyo文件,可以这样:
python -0 -m py\_compile xxxx.py
python文档是这样描述的:这个优化没有多大作用,只是移除了图 3、至于速度,运行几乎一样,加载pyc和pyo稍占优势。

### Python\_Swap

```
1、在C++中有值语义和引用语义,而Python中只有引用的语义。
2、示例如下:
>>> a=1
>>> a
1
>>> id(a)
19310280
>>> b=a
>>> b
>>> b=2
>>> b
2
>>> id(2)
19310268
解释如下:
b=a,b和a指向同一个对象,对象的引用计数为2,
b=2,修改b的指向,指向2,a的引用计数减1,为1
3、因此,对于swap的实现,
def swap(a,b):
   t=a
   a=b
   b=t
   return
是错误的,这里这是修改了形参的指向,实参的指向,并没有改变。解决
def swap(a,b):
   return b, a
调用如下:
```

```
a,b=swap(a,b)
```

### Python命令行查看帮助文档

### Python导入导出

### Python的设计哲学

```
Python的设计哲学是优雅、明确、简单。
Perl语言,同一件事,提供多种方法来做。
```

而Python,同一件事最好只有一种方法来做。 Python这样做有什么好处?

这会减轻程序员的心智负担,举例来说,C++中的string,计算字符串程序员就会想这两种方法有什么区别。孟岩在博客中写道,相对于C,C+影响很严重,语言提供的相似接口越多,参数越多,特性越多,程序员定:这样固然可以work,但恐怕还有更好的方案吧?会是什么呢? 使用机,使用编译时多态,还是运行时多态,内存谁来回收呢?使用智能指针,太复杂了,还是能够work就算了。而C程序员没有这些负担,C语言提供

#### 语言新特性也有同样的问题:

- 1、新特性加重了编译器的编译负担。
- 2、新特性加重了运行时的运行负担。
- 3、新特性加重了程序员的心智负担。

#### 元祖不可变

1、Python彻底分离了指针和内容,每次修改内容,都相当于,修改指针

2、需要注意的是, Python使用了共享内存, 如果内容相同, 指向同一均

>>> a='hello'

>>> b='hello'

>>> id(a)

43861376

>>> id(b)

43861376

可以这样理解,a='hello'分配一块内存,内容为"hello",afb='hello'先检查有没有这样的一块内存,内容为"hello",刚从内容到地址,可以使用哈希表,内容映射到地址。每次修改内容新的内容,如果存在,直接返回地址。如果不存在,new出一个,iC++静态存储区也是同样的道理。

3、查看下面的情况:

>>> a=123

>>> b=a

>>> b=456

>>> a

123

>>> b

```
456
   这和C++中的写时拷贝(copy-on-write), 道理一样。b=a是指
   b=456修改b的时候,b指向新的内容,但是a的指向不变,这个时间
4、元祖的元素也是指针,元祖的不可变指的是,元素不能指向其他地方
   但是,元祖本身是可以变的,可以指向其他的内容。如下:
   >>> aTuple=(1,2)
                 // 元素不能修改指向
   >>> aTuple[0]=5
   Traceback (most recent call last):
   File "<pyshell#34>", line 1, in <module>
       aTuple[0]=5
   TypeError: 'tuple' object does not support item a
   >>> aTuple=(3,4) // 元祖本身可以修改指向
   >>> aTuple
   (3, 4)
5、list与tuple之间的转化,使用list方法和tuple方法,返回一个
   >>> aList=list(aTuple)
   >>> aList
   [3, 4]
   >>> aList[1]=100
   >>> aList
   [3, 100]
   >>> bTuple=tuple(aList)
   >>> bTuple
   (3, 100)
```

#### 查看帮助信息

```
1, help
>>> help(Animal)
Help on class Animal in module animal:

class Animal(__builtin__.object)
   | Methods defined here:
   |
   | say(self)
   |
```

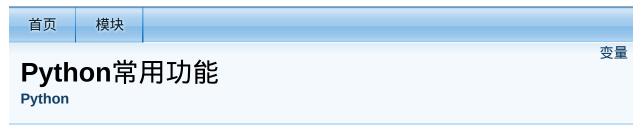
```
Data descriptors defined here:
    dict
        dictionary for instance variables (if defined
   __weakref
        list of weak references to the object (if def
    Data and other attributes defined here:
   age = 0
    name = ''
2、dir
>>> dir(Animal)
['__class__', '__delattr__', '__dict__', '__doc__', '
3、查询父类
>>> Animal.__base__
<type 'object'>
4、是否有某个属性
>>> hasattr(Animal, 'name')
True
5、方法的参数
>>> inspect.getargspec(Animal.say)
ArgSpec(args=['self'], varargs=None, keywords=None, c
```

### 理解import

```
1、import一个模块,如下:
>>> import c
>>> c1=C();

Traceback (most recent call last):
File "<pyshell#1>", line 1, in <module>
c1=C();
```

```
NameError: name 'C' is not defined
   >>> c1=c.C();
   >>> c1.say();
   I am c
   需要注意的是,import一个模块,使用模块中的对象时,需要在外
2、怎么解决上面的问题?
   使用import,从模块中import对象,如下:
   >>> from c import C
   >>> c1=C();
   >>> c1.say();
   I am c
   这种情况下,可以直接使用对象。
3、如何import子目录中的模块?
   考虑当前有 ./a/a.py, 如何import a.py
   在./a目录创建一个空的文件 __init__.py 即可
       >>> import a.a
   Traceback (most recent call last):
   File "<pyshell#0>", line 1, in <module>
       import a.a
   ImportError: No module named a.a
   >>> import a.a
4、如何引用父目录中模块?
   考虑当前目录为./a/ 父目录下有./b/b.py, 如何import a.p
   使用 sys增加搜索路径
   >>> import sys;
   >>> sys.path.append('../b');
   >>> import b;
   对于import子目录中的模块,也可以使用这种方式。
```



# 重变

批量重命名文件

Python之道

Python命名空间

### 变量说明

### Python之道

import this

The Zen of Python, by Tim Peters

Python之道

Beautiful is better than ugly.

美观胜于丑陋。

Explicit is better than implicit.

显示胜于隐式。

Simple is better than complex.

简单胜于复杂。

Complex is better than complicated.

复杂胜于过度复杂。

Flat is better than nested.

平面胜于嵌套。

Sparse is better than dense.

稀少胜于稠密。

Readability counts.

可读性需要考虑。

Special cases aren't special enough to break the rule 即使情况特殊,也不应打破原则,

Although practicality beats purity.

尽管实用胜于纯净。

Errors should never pass silently.

错误不应悄无声息的通过,

Unless explicitly silenced.

除非特意这么做。

In the face of ambiguity, refuse the temptation to gu

当有混淆时,拒绝猜测(深入的搞明白问题)。

There should be one-- and preferably only one --obvic 总有一个,且(理想情况下)只有一个,明显的方法来处理问题。

Although that way may not be obvious at first unless 尽管那个方法可能并不明显,除非你是荷兰人。(Python的作者Guido Now is better than never.

现在开始胜过永远不开始,

Although never is often better than \*right\* now. 尽管永远不开始经常比仓促立即开始好。

If the implementation is hard to explain, it's a bad 如果程序实现很难解释,那么它是个坏主意。

If the implementation is easy to explain, it may be a 如果程序实现很容易解释,那么它可能是个好主意。

Namespaces are one honking great idea -- let's do mor 命名空间是个绝好的主意,让我们多利用它。

"Python之道"强调美观、简单、可读和实用,拒绝复杂或模糊。

### Python命名空间

- 1、python彻底分离了变量名称和变量值,名称可以指向任何值。
- 2、定义一个名称的时候,必须赋值,否则报错:

```
>>> a
```

Traceback (most recent call last):
File "<pyshell#2>", line 1, in <module>
 a

NameError: name 'a' is not defined

原因是:没有对a进行赋值,python解释器认为是引用a,但是在记也就是说,定义一个名称的时候,没有赋值,语义就变了,不是定

3、赋值的时候,也就是定义了一个名称。这一点特别重要。如下:

func(); 报错如下:

```
Traceback (most recent call last):
   File "D:\Temp\NiuzbRepository\trunk\Python\自学教
      func();
   File "D:\Temp\NiuzbRepository\trunk\Python\自学教科
      i=i+1
   UnboundLocalError: local variable 'i' referenced
   原因是:进入方法func内,i=i+1是赋值,相当于在局部定义了-
   这个i不是全局的i,而是新定义了一个名称i,但是在赋值的右边
   怎么解决这个问题?
   就是在赋值之前,不要定义名称i,而是从全局引用i,如下:
   i=1
   def func():
      qlobal i
      i=i+1
      print i
   func();
   print i
4、特别注意:下面表示的不同语义:
   i=100 // 定义一个名称
         // 引用一个名称
   i
5、命名空间是一个字典(dictionary),它的键就是变量名,它的
6、一个名称的顺序是 LEGB,如下:
   L(local)方法
   E(enclosing function) 嵌套方法
   G(global) 全局,也就是一个模块
   B(builtin)内置命名空间
7、命名空间的访问
   locals();
   globals();
```

### 批量重命名文件

考虑下面的需求,对于目录下,dmu1.h,dmu1.cpp,dmu2.h,dmu2 重命名为vms1.h,vms1.cpp,vms2.h,vms2.cpp,...,代码如下

```
import os
```

```
target_path="D:/test"
print target_path
fileList = os.listdir(target_path)
print fileList

for a in fileList:
    postfix=a.split(".")[1]

    if postfix == "h" or postfix == "cpp":
        os.rename(target_path+"/"+a, target_path+"/"+a
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 **Redis** 

### 变量

Redis简单示例

Redis持久化

Redis安全

Redis复制

DENIED\_Redis\_is\_running\_in\_protected\_mode

Redis接口的调用

#### 变量说明

#### DENIED\_Redis\_is\_running\_in\_protected\_mode

- 1、开源软件往往做的非常出色,错误的时候,会告诉你原因,以及怎么 [root@localhost src]# ./redis-cli -h 172.16.2.16 172.16.2.16:6379> get name
  - (error) DENIED Redis is running in protected mode no authentication password is requested to client If you want to connect from external computers to
  - 1) Just disable protected mode sending the commar
  - 2) Alternatively you can just disable the protect
  - 3) If you started the server manually just for  $t\varepsilon$
  - 4) Setup a bind address or an authentication pass 已经告诉错误原因:默认情况下redis运行在保护模式(这种模式按照第二种方式修改。
  - # are explicitly listed using the "bind" directive protected-mode no
- 2、重新启动redis服务,要想配置文件起效,启动的时候,必须指定配 [root@localhost src]# ./redis-server ../redis.cor 但是,这个时候从外部连接,报错:
  - [root@localhost src]# ./redis-cli -h 172.16.2.16
    Could not connect to Redis at 172.16.2.16:6379: (
- 3、查看监听信息,如下:

  - 原因是配置文件redis.conf配置了只在127.0.0.1上绑定监听, # IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTE # JUST COMMENT THE FOLLOWING LINE.

#bind 127.0.0.1

4、重启服务即可。

#### Redis复制

- 1、现在有两个redis, 172.16.2.16为master, 10.65.200.202为
- 2、在slave端修改配置项 salveof,如下:

slaveof 172.16.2.16 6379

- 3、在这种情况下, master端的修改, 会同步到slave端。
- 4、但是slave端不能执行写操作,如下:

127.0.0.1:6379> get name

"andy"

127.0.0.1:6379> set name bill

(error) READONLY You can't write against a read (原因是:如果slave可以写操作,就会导致主从数据不一致。因此

5、当然,也可以配置写操作,如下:

slave-read-only no

但是, slave端的修改, 不会同步到master端。

- 6、mysql是支持主主复制的,也就是两个设备互为主从。如果把两个relis版本不支持互为主从。如果两个redis配置成互为主 Master does not support PSYNC or is in error stat
- 7、也就是说, redis要实现主主复制, 需要应用层管理。

A仅仅作为master, B设置为A的salve。

在某个条件下,B不再作为A的slave,仅仅设置为master,同时/ 注意:这里redis主主复制是指任一时刻只有一个方向的复制,并

#### Redis安全

- 1、默认情况下,访问redis服务是不需要密码的,如果在不信任的网络
- 2、查询当前密码,为空(也就是不需要密码),如下:

172.16.2.16:6379> config get requirepass

- 1) "requirepass"
- 2) ""
- 3、如何设置密码?

修改配置文件的requirepass项,如下:

requirepass 123456

然后重启redis服务

```
4、不重启服务的情况下,怎么修改密码?
   172.16.2.16:6379> config get requirepass
   1) "requirepass"
   2)
   172.16.2.16:6379> config set requirepass 123456
   0K
   172.16.2.16:6379> config get requirepass
   (error) NOAUTH Authentication required.
   172.16.2.16:6379> auth 123456
   0K
   172.16.2.16:6379> config get requirepass
   1) "requirepass"
   2) "123456"
   注意:一旦设置了密码,接下来的操作,首先需要认证。
5、如何登陆有密码的redis服务?
   a、登录时认证
   [root@localhost src]# ./redis-cli -h 172.16.2.16
   b、先登录,再认证
   [root@localhost src]# ./redis-cli -h 172.16.2.16
   172.16.2.16:6379> get name
   (error) NOAUTH Authentication required.
   172.16.2.16:6379> auth 123456
   0K
6、主从复制的时候, master有密码, slave怎么设置?
   # masterauth <master-password>
   修改为
   masterauth 123456
```

#### Redis持久化

- 1、redis持久化有两种方式:分别是RDB(Redis DataBase)和AOF RDB,简而言之,就是在不同的时间点,将redis存储的数据生成 AOF,则是换了一个角度来实现持久化,那就是将redis执行过的F在下次redis重新启动时,只要把这些写指令从前到后再重复执行
- 2、其实RDB和AOF两种方式也可以同时使用,在这种情况下,如果redi 这是因为AOF方式的数据恢复完整度更高。如果你没有数据持久化能 这样的话,redis将变成一个纯内存数据库,就像memcache一样。

3、不进行持久化,需要取消rdb方式和和aof方式,分别如下:

注销rdb的保存策略

#save 900 1

#save 300 10

#save 60 10000

取消aof方式

appendonly no

启动redis服务,指定配置文件,如下:

[root@localhost src]# ./redis-server ../redis.cor 注意:在这种方式下,设置一些key,当重启redis服务后,之前ì

4、使用RDB方式持久化,开启保存策略,如下:

save 900 1 至少有1个key被更改时,900秒后保存

save 300 10 至少有10个key被更改时,300秒后保存 save 60 10000 至少有10000个key被更改时,60秒后保存 这些保存策略是或的关系,有一个满足就执行。

RDB的保存策略是异步的,有很大延迟,很导致一部分数据丢失。

5、现在设置以上的保存策略,先删除当前目录的dump.rdb,运行如下

a、启动redis服务

[root@localhost src]# ./redis-server ../redis

- b、设置一个key,查询当前目录,并没有dump.rdb文件,说明没
- c、这个时候异常关闭, kill -9杀掉redis服务, 会导致数据丢线 特别注意:使用ctrl+c方式, redis会立即进行持久化。 这是因为redis可以捕获到ctrl+c的signal,刷新到文件, sigaction不能捕捉到kill -9和kill -19,其他都能捕
- 6、使用AOF方式持久化,设置如下:

appendonly yes

会在当前目录下,生成文件 appendonly yes

刷新策略有:

# appendfsync always

# appendfsync everysec

# appendfsync no

7、如果两种持久化方式都开启了,会优先使用AOF方式。

#### Redis接口的调用

- 1、hiredis是redis数据库的C接口,目录为/redis-3.2.6/deps/t
- 2、示例代码如下:

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdarg.h>
#include <string.h>
#include <assert.h>
#include "hiredis.h"
int main()
{
    //连接redis
    redisContext* c = redisConnect("127.0.0.1", {
    if (c->err)
    {
        redisFree(c);
        printf("Connect to redisServer faile\n");
        return -1;
    printf("Connect to redisServer Success\n");
    const char* setCommand = "set name andy";
    redisReply* r = (redisReply*)redisCommand(c,
    if( NULL == r)
    {
        printf("Execut setCommand failure\n");
        redisFree(c);
        return -1;
    if( !(r->type == REDIS_REPLY_STATUS && strcas
    {
        printf("Failed to execute command[%s]\n",
        freeReplyObject(r);
        redisFree(c);
        return -1;
    freeReplyObject(r);
```

```
printf("Succeed to execute command[%s]\n", se
        const char* getCommand = "get name";
        r = (redisReply*)redisCommand(c, getCommand);
        if ( r->type != REDIS REPLY STRING)
        {
            printf("Failed to execute command[%s]\n",
            freeReplyObject(r);
            redisFree(c);
            return -1;
        }
        printf("Succeed to execute command[%s]\n", ge
        printf("The value of 'name' is %s\n", r->str)
        freeReplyObject(r);
        redisFree(c);
        return 0;
3、编译,运行如下:
    [root@localhost hiredis]# g++ -o main main.cpp li
    [root@localhost hiredis]# ./main
    Connect to redisServer Success
    Succeed to execute command[set name andy]
    Succeed to execute command[get name]
    The value of 'name' is andy
```

#### Redis简单示例

```
1、Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存;从2010年3月15日起,Redis的开发工作由VMware主持。从2013 2、下载 redis-3.2.6.tar.gz,解压,进入目录redis-3.2.6,然 3、在src目录下,生成一些二进制可执行文件。如下:
[root@localhost src]# ll|grep -v "\."
total 33384
-rwxrwxrwx 1 root root 8173 Dec 6 16:38 Make1
-rwxr-xr-x 1 root root 4560528 Feb 10 09:38 redis
```

```
-rwxr-xr-x 1 root root 16880 Feb 10 09:38 redis
    -rwxr-xr-x 1 root root 6401156 Feb 10 09:38 redis
    -rwxr-xr-x 1 root root 4661462 Feb 10 09:38 redis
    -rwxr-xr-x 1 root root 6401156 Feb 10 09:38 redis
    -rwxr-xr-x 1 root root 6401156 Feb 10 09:38 redis
   文件介绍如下:
    ./redis-benchmark
                       //用于进行redis性能测试的工具
    ./redis-check-dump //用于修复出问题的dump.rdb文件
                     //redis的客户端
    ./redis-cli
    ./redis-server //redis的服务端
./redis-check-aof //用于修复出问题的AOF文件
    ./redis-sentinel //用于集群管理
4、启动redis服务,如下:
    [root@localhost src]# ./redis-server
   监听端口是6379,如下:
    [root@localhost ~]# netstat -anp|grep 6379
    tcp
              0
                    0 0.0.0.0:6379
                                                  6
    tcp
              0
                     0:::6379
5、启动redis客户端,如下:
    [root@localhost src]# ./redis-cli --help
   查看帮助信息
6、设置,获取,删除,最简单的例子如下:
    [root@localhost src]# ./redis-cli
   127.0.0.1:6379> get name
    (nil)
   127.0.0.1:6379> set name andy
   0K
   127.0.0.1:6379> get name
    "andv"
   127.0.0.1:6379> del name
    (integer) 1
   127.0.0.1:6379> get name
    (nil)
```

# Andy Niu Help 1.0.0.0

首页 模块 模块 | 变量

# 模块

### Shell脚本常用功能

特殊字符

### 变量

设置获取MySQL复制信息

shell的交互式和登录

profile和bashrc

#### 变量说明

#### profile和bashrc

- 1、/etc/profile、/etc/bashrc、~/.bash\_profile、~/.bash
- 2、/etc/profile: 用来设置系统环境参数,比如\$PATH. 这里面的引
- 3、/etc/bashrc: 这个文件设置系统bash shell相关的东西,对系只要用户运行bash命令,那么这里面的东西就在起作用。
- 4、~/.bash\_profile: 用来设置一些环境变量,功能和/etc/profi也就是说,你在/home/user1/.bash\_profile 中设定了环境引
- 5、~/.bashrc: 作用类似于/etc/bashrc, 只是针对用户自己而言,
- 6、另外/etc/profile中设定的变量(全局)的可以作用于任何用户, 而~/.bashrc等中设定的变量(局部)只能继承/etc/profile中的
- 7、~/.bash\_profile 是交互式、login 方式进入 bash 运行的 ,: ~/.bashrc 是交互式 non-login 方式进入 bash 运行的 ,用

#### shell的交互式和登录

1、交互式bash与非交互式bash

交互式模式就是在终端上执行, shell等待你的输入,并且立即执定的模式被称作交互式是因为shell与用户进行交互。这种模式也当你退出后, shell也终止了。

shell也可以运行在另外一种模式:非交互式模式,以shell scr在这种模式 下,shell不与你进行交互,而是读取存放在文件中的区别:读取的配置文件不同,交互式bash必然读取~/.bashrc文化非交互式bash读取的是环境变量BASH\_ENV(通常情况下)所指定

2、登录bash与非登录bash

登陆shell是指:

(1)用户登陆时,输入用户名和密码后启动的shell(例如使用s

- (2)通过带--login参数的命令:bash --login而启动的shel非登录bash:登录以后所打开的bash,比如通过Ctrl+Alt+T组;对于用户来说,登陆shell和非登陆shell的主要区别是:启动sh登陆shell执行startup文件为:/etc/profile、~/.bash\_pr而非登陆shell执行的startup文件仅为:~/.bashrc。
- 3、bash中解析命令的次序

在bash中输入一个命令,首先会从which所指定的路径中进行命令然后从配置文件的命令别名中寻找。注意,配置文件的修改不会立即

#### 设置获取MySQL复制信息

```
my.cnf内容:
server-id=111
auto increment offset=222
auto_increment_increment=2
log-bin=mysql-bin
master-host=333
master-user=444
master-password=555
master-port=666
log-slave-updates=1
slave-skip-errors=all
replicate-do-db=777
replicate-ignore-db=mysql
master-connect-retry=60
opt_my_cnf:
#! /bin/sh
set -x
mysqlcnf=./mysql.cnf
set_item()
{
    sed -r "s/(\$\{1\}=)(.*)/\1\$\{2\}/" -i $mysqlcnf
}
```

```
get_item()
{
    echo `cat mysqlcnf|grep "${1}"|sed -r "s/("${1}=
}
case $1 in
    "set")
    set item server-id $2
    set_item auto_increment_offset $3
    set item master-host $4
    set item master-user $5
    set_item master-password $6
    set_item master-port $7
    set_item replicate-do-db $8
    ;;
    "get")
    get_item $2
    ;;
    * )
    echo "param error:$1"
esac
设置:
opt_my_cnf set 111 222 333 444 555 666 777
获取:
opt_my_cnf get server-id
```

参见

# Andy Niu Help 1.0.0.0

To 模块

Shell脚本常用功能
Shell脚本

### 变量

脚本交互自动输入密码

间接引用变量

删除一部分文件

linux递归删除txt文件

shell数值运算

#### 变量说明

#### linux递归删除txt文件

```
1、先递归找出所有的txt文件,如下:
   [root@localhost d1]# 11
   total 4
   -rw-r--r-- 1 root root 0 2016-11-02 17:51 111.txt
   drwxr-xr-x 2 root root 0 2016-11-02 17:51 aaa
   drwxr-xr-x 3 root root 0 2016-11-02 17:51 bbb
   [root@localhost d1]# find ./ -iname "*.txt"
   ./bbb/ccc/c2.txt
   ./bbb/ccc/c1.txt
    ./bbb/b2.txt
    ./bbb/b1.txt
    ./aaa/a2.txt
    ./aaa/a1.txt
    ./111.txt
2、特别注意: find ./ -iname *.txt只能找到当前目录下的txt文
   原因是:*作为通配符,必须处理一下,当成字符串的一部分。
   [root@localhost d1]# find ./ -iname *.txt
    ./111.txt
   还可以使用单引号或者转义字符,如下:
   [root@localhost d1]# find ./ -iname '*.txt'
    ./bbb/ccc/c2.txt
    ./bbb/ccc/c1.txt
    ./bbb/b2.txt
    ./bbb/b1.txt
    ./aaa/a2.txt
```

```
./aaa/a1.txt
./111.txt

[root@localhost d1]# find ./ -iname \*.txt
./bbb/ccc/c2.txt
./bbb/ccc/c1.txt
./bbb/b2.txt
./bbb/b1.txt
./aaa/a2.txt
./aaa/a1.txt
./111.txt

3、找到这些文件,删除如下:
find ./ -iname "*.txt"|xargs rm -f
```

#### shell数值运算

```
1、考虑下面的情况,对var=1,对var加1,如下:
   [root@localhost tmp]# var=1
   [root@localhost tmp]# echo $var+1
   1+1
   这显然不是预期的行为。因为shell默认把var当成字符串处理。《
2、使用let
   [root@localhost tmp]# var=1
   [root@localhost tmp]# let var=var+1
   [root@localhost tmp]# echo $var
3、使用(())
    [root@localhost tmp]# var=1
   [root@localhost tmp]# ((var=var+1))
   [root@localhost tmp]# echo $var
4、使用$[]
   [root@localhost tmp]# var=1
   [root@localhost tmp]# var=$[$var+1]
   [root@localhost tmp]# echo $var
   $[]的执行过程是:对中括号中的表达式作为数字运算,$取出运算
```

```
5、使用expr
    [root@localhost tmp]# var=1
    [root@localhost tmp]# var=`expr $var + 1`
    [root@localhost tmp]# echo $var
    特别注意:使用expr,运算符左右要有空格
参见
```

```
删除一部分文件
1、当前目录有如下的文件:
    [root@localhost tmp]# 11
    total 0
    -rw-r--r-- 1 root root 0 2015-10-28 19:22 aaa
    -rw-r--r-- 1 root root 0 2015-10-28 19:22 bbb
    -rw-r--r-- 1 root root 0 2015-10-28 19:20 file1
    -rw-r--r-- 1 root root 0 2015-10-28 19:20 file2
    -rw-r--r-- 1 root root 0 2015-10-28 19:21 log1
    -rw-r--r-- 1 root root 0 2015-10-28 19:21 log2
2、删除file开头的文件,如下:
    [root@localhost tmp]# ll|grep -v total|cut -d ' '
    [root@localhost tmp]# 11
    total 0
    -rw-r--r-- 1 root root 0 2015-10-28 19:22 aaa
    -rw-r--r-- 1 root root 0 2015-10-28 19:22 bbb
    -rw-r--r-- 1 root root 0 2015-10-28 19:21 log1
    -rw-r--r-- 1 root root 0 2015-10-28 19:21 log2
    注意:这里必须使用xargs, xargs把前面的输出作为一个文件,
3、有没有其他的办法?
    使用命令替换,直接使用一个命令的返回值,如下:
    rm -f $(ll|grep -v total|cut -d ' ' -f 8|grep fi]
    等价干
    rm -f `ll|grep -v total|cut -d ' ' -f 8|grep file
4、删除除了file开头的文件,怎么办?
    使用grep -v反向选择。
```

- 5、在windows下面用法类似,如下: 使用xargs把前一个命令的输出,作为后一个命令的参数 dir|findstr file|cut -d ' ' -f 21 -s|xargs rm
- 6、考虑另一种方式,当前命令跟着另一个命令的输出,但是这存在问题在Windows下面,把一个命令的输出赋值给变量,或者作为其他命没有Linux类似的语法(Linux使用倒引号或者美元符号+小括号)
- 7、在windows只能使用for循环,如下:

```
echo on
cd D:/test
setlocal enabledelayedexpansion
for /f %%i in ('dir /b /a:-d') do (
    echo %%i|findstr "^file">nul
    if !errorlevel! equ 0 (rm %%i)
)
pause
```

- 8、特别需要注意的是:
  - a、因为在for是复合语句,需要使用延迟变量, setlocal enab
  - b、window下面的黑洞设备是 nul,而linux下面的黑洞设备是

#### 参见

#### 脚本交互自动输入密码

1、以sudo为例说明

#! /bin/bash -x

echo ruanxiaopang|sudo -S ifconfig eth0 192.168.2 sudo route add default gw 192.168.29.2

注意,这里对于sudo命令,必须加上 -S 选项,有些情况可能不 -S, --stdin read password from st

#### 间接引用变量

1、考虑下面的需求,如下:

```
[niu_zibin@localhost ~]$ andy100=hhh
[niu_zibin@localhost ~]$ name=andy
[niu_zibin@localhost ~]$ num=100
```

```
变量name和变量num组成andy100,我该如何引用呢?
   尝试如下:
   [niu_zibin@localhost ~]$ echo $name$num
   andy100
   [niu zibin@localhost ~]$ echo $($name$num)
   -bash: andy100: command not found
2、怎么解决这个问题,也就是说,如何对变量的取值再一次解引用。
3、这需要间接引用变量的语法,也就是两次解引用。如下:
   [niu_zibin@localhost ~]$ tmp="$name$num"
   [niu_zibin@localhost ~]$ echo ${tmp}
   andy100
   [niu_zibin@localhost ~]$ echo ${!tmp}
   hhh
   还有另外一种办法:
   [niu_zibin@localhost ~]$ eval aaa=\$${tmp}
   [niu_zibin@localhost ~]$ echo $aaa
   hhh
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 特殊字符 Shell脚本

### 变量

美元符

分号

变量引用和命令替换

调用命令\_\_命令替换\_\_命令返回值

倒引号\_\_单引号\_\_双引号

EOF

#### 变量说明

#### **EOF**

```
1、考虑下面的需求,在主shell执行命令,进入其他的命令,后面的输
2、使用<<EOF,告诉主shell,后续的输入,是其他命令或者子shell
3、这里的EOF只是分界符,使用其他的字符也可以。
4、比如cat,不使用EOF,如下:
[root@localhost ~]# cat >111.txt
abcd
1234
[root@localhost ~]# more 111.txt
abcd
1234
使用EOF
[root@localhost ~]# cat >111.txt<<EOF</pre>
> aaaa
> bbbb
> E0F
[root@localhost ~]# more 111.txt
aaaa
bbbb
5、mysql安装后之后,忘记密码,可使用说下面的脚本,如下:
/etc/init.d/mysqld stop
service mysqld start --skip-grant-tables
sleep 4
mysql -hlocalhost << EOF
update mysql.user set password=password('123456') whe
```

```
grant all privileges on *.* to 'root'@'%' identified
flush privileges;
EOF
/etc/init.d/mysqld restart
```

#### 参见

#### 倒引号 单引号 双引号

```
1、倒引号表示命令,用于命令替换,获取命令的返回结果。
echo now is `date` 或者 echo now is $(date)

2、单引号
name=Andy 没有问题,
如果想 name=Andy Niu,报错:bash:niu:command not fo 如何解决这个问题?
使用单引号 name='Andy Niu'
注:字符串中包含等号,也需要使用单引号引起来。

3、双引号
desc='I am $name'
echo $desc
I am $name',我想显示的是 I am Andy Niu。怎么办?
使用双引号,双引号对于其中的 $、\(反斜杠)、·(倒引号)特
```

#### 参见

#### 分号

```
1、分号是语句分隔符,当一行中有多个语句时,需要使用分号进行分割
2、换行符也可以作为语句分隔符,如果一行中只有一个语句,结尾不需
3、最后一个语句不需要分号
4、示例:
    #! /bin/sh -x
    var=5
    if [ $var == 8 ]; then
        echo 'equal to 8';
    elif [ $var -gt 8 ]
```

then

echo 'bigger than 8'

else

echo 'smaller than 8'

fi

#### 需要注意的几点:

- a、then是一个语句,如果和if在同一行,需要加上分号。否则sl
- b、一行只有一个语句,结尾不需要加上分号,最后一个语句不需要
- c、else 后面没有then
- d、== 可以表示 -eq,但是 >= 不能表示 -ge,这可能是因为

#### 参见

#### 变量引用和命令替换

1、变量引用

name=Andy

echo \$name 或 echo \${name}

2、命令替换

把命令的执行结果赋值给变量,使用倒引号或者\$()

APP\_HOME=`pwd` 或 APP\_HOME=\$(pwd)

注意:bash 与makefile的语法不一样,在makefile中,变量引

#### 参见

#### 美元符

\$num \$# \$* \$@	num为第几个参数,其中\$0是当前脚本的文件名 传入脚本的参数的个数 所有的位置参数(作为单个字符串) 所有的位置参数(每个都作为独立的字符串)。
\$?	当前shell进程中,前一个命令的返回值,执行成功则\$?值为
\$\$ \$!	当前shell进程的pid 后台运行的最后一个进程的pid
\$-	显示shell使用的当前选项
\$_	前一个命令的最后一个参数

#### 调用命令\_\_命令替换\_\_命令返回值

```
1、考虑下面的shell脚本
   #! /bin/sh
   #set -x
   fun1()
   {
           cd aa
   fun2()
   {
           pwd
           fun1
           pwd
   }
   fun2
2、在fun2直接调用fun1,执行结果如下:
    [root@localhost tmp]# ./test.sh
   /tmp
   /tmp/aa
3、在fun2中对fun1进行命令替换,如下:
   #! /bin/sh
   #set -x
   fun1()
    {
           cd aa
   fun2()
   {
           pwd
           `fun1`
           pwd
```

```
fun2
   执行结果如下:
   [root@localhost tmp]# ./test.sh
   /tmp
   /tmp
4、也就是说,对于命令替换,不会改变当前的执行目录。
5、命令替换是获取命令中echo的内容,如下:
   #! /bin/sh
   #set -x
   fun1()
   {
          cd aa
          echo "hello"
   fun2()
   {
          pwd
          EchoFromFun1=$(fun1)
          echo $EchoFromFun1
          pwd
   }
   fun2
   执行结果,如下:
   [root@localhost tmp]# ./test.sh
   /tmp
   hello
   /tmp
6、考虑下面的需求, fun1对输入的正数, 加上100, 负数不处理, 然后
   对于fun1,存在两个输出值。一个是否成功,另一个是处理后的数
   当然也可以使用一个返回值表示,一个特殊的值表示失败,其他的
   但是,尽量不使用这种做法。因为这样的话,一个返回值表示表示
   正确的做法如下:
   #! /bin/sh
   #set -x
   SUCCESS=0
```

```
FAILURE=1
    fun1()
    {
           if [ $1 -gt 5 ]; then
                   let aa=$1+100
                   echo $aa
                   return $SUCCESS
           fi
           return $FAILURE
    fun2()
    {
           EchoFromFun1=$(fun1 4)
           if [ $? == $SUCCESS ]; then
                   let aa=$EchoFromFun1\*2
                   echo $aa
           else
                   echo 'error'
           fi
   }
    fun2
7、特别注意,$?是上一个命令的返回结果,EchoFromFun1=$(fun1
    EchoFromFun1=$(fun1 4)
    echo $?
    if [ $? == $SUCCESS ]; then
    if中的$?,是上一个命令echo $?的回传值,一定是0
```

参见

# Andy Niu Help 1.0.0.0

首页	模块	
STL		模块   变量

# 模块

Effective\_\_STL

String有关

# 重变

理解迭代器
map的元素重载小于符号
迭代器失效
迭代器失效的测试
理解函数对象
容器删除迭代器
容器删除元素
对序列容器的元素排序
集合保存对象和保存指针

#### 变量说明

#### map的元素重载小于符号

1、map是用红黑树实现的,要比较大小,要求元素重载了操作符< 方法<的两个参数都是const引用,因此如果是是用成员方法重载< 也就是在方法之后加上const

#### 容器删除元素

```
1、vector删除元素
   int main(int argc, char* argv[])
   {
       vector<int> intArray;
       intArray.push_back(1);
       intArray.push_back(2);
       intArray.push_back(3);
       intArray.push_back(2);
       intArray.push_back(4);
       intArray.push_back(5);
       intArray.push_back(2);
       /*
          vector删除元素
          remove不能真正删除元素,因为:
              要删除元素,必须知道是哪一种容器,调用容器的
              但是remove只接收一对迭代器,不知道是哪一种和
          remove做的事情是:
              一种压缩,被删除的值挖掉,后面保留的值补充上
```

```
remove返回新的逻辑终点。
       * /
       intArray.erase(remove(intArray.begin(),intArr
       return 0;
2、list删除元素,也可以使用上面的方法,但是有更好的方法,如下:
   intArray.remove(2);
3、map直接erase关键字,如下:
   int main(int argc, char* argv[])
   {
       map<int, string> aMap;
       aMap[1] = "Andy";
       aMap[2] = "Bill";
       aMap[3] = "Caroline";
       aMap.erase(2);
       return 0;
   }
```

#### 容器删除迭代器

```
1、先看vector,示例代码如下:
    int main(int argc, char* argv[])
    {
        vector<int> intVec;
        intVec.push_back(1);
        intVec.push_back(2);
        intVec.push_back(3);
        intVec.push_back(4);

        for(vector<int>::iterator iter = intVec.begir {
            if(*iter > 2)
            {
                intVec.erase(iter);
        }
```

```
else
           {
               ++iter;
           }
       }
       getchar();
       return 0;
2、特别注意:intVec.erase(iter); 操作导致iter失效,正确的体
3、对于map存在同样的问题,因此解决办法也是一样,如下:
   int main(int argc, char* argv[])
   {
       map<int, string> intStrMap;
       intStrMap[1] = "Andy";
       intStrMap[2] = "Bill";
       intStrMap[3] = "Caroline";
       intStrMap[4] = "David";
       for(map<int,string>::iterator iter = intStrMa

       {
           if(iter->first > 2)
               iter = intStrMap.erase(iter);
           else
           {
               ++iter;
       }
       getchar();
       return 0;
4、需要注意的是,在linux平台,map的erase方法并不返回下一个迭
   因此,跨平台的解决办法是 intStrMap.erase(iter++); 但是
```

```
5、总结如下:
对于vector,必须使用 iter = vector.erase(iter);
对于map, 必须使用 map.erase(iter++);
对于list, 上面的两种方式都可以
```

#### 对序列容器的元素排序

```
示例代码如下:
#include <string>
#include <vector>
using namespace std;
struct Person
public:
    int _Age;
    Person(int age)
    {
        _{Age} = age;
    }
};
int main(int argc,char* argv[])
{
    Person p1(5);
    Person p2(2);
    Person p3(4);
    Person p4(1);
    Person p5(3);
    vector<Person> pVec;
    pVec.push_back(p1);
    pVec.push_back(p2);
    pVec.push_back(p3);
    pVec.push_back(p4);
    pVec.push_back(p5);
```

```
return 0;
}
第一种办法,添加重载<操作符的成员方法,调用sort(first,last)
struct Person
{
public:
    int _Age;
    Person(int age)
    {
        _{Age} = age;
    }
    bool operator< (const Person& rhs)</pre>
    {
        return this->_Age < rhs._Age;
    }
};
int main(int argc,char* argv[])
{
    Person p1(5);
    Person p2(2);
    Person p3(4);
    Person p4(1);
    Person p5(3);
    vector<Person> pVec;
    pVec.push_back(p1);
    pVec.push_back(p2);
    pVec.push_back(p3);
    pVec.push_back(p4);
    pVec.push_back(p5);
    sort(pVec.begin(), pVec.end());
```

```
return 0;
}
第二种办法,添加重载<操作符的普通方法,调用sort(first,last)
struct Person
{
public:
    int _Age;
    Person(int age)
    {
        _{Age} = age;
};
bool operator< (const Person& lhs,const Person& rhs)</pre>
{
    return lhs._Age < rhs._Age;</pre>
}
int main(int argc,char* argv[])
{
    Person p1(5);
    Person p2(2);
    Person p3(4);
    Person p4(1);
    Person p5(3);
    vector<Person> pVec;
    pVec.push_back(p1);
    pVec.push_back(p2);
    pVec.push_back(p3);
    pVec.push_back(p4);
    pVec.push_back(p5);
    sort(pVec.begin(), pVec.end());
```

```
return 0;
}
第三种办法,添加普通方法,比较大小,调用sort(first,last,pred
struct Person
{
public:
    int _Age;
    Person(int age)
    {
        _{Age} = age;
};
bool PersonLess(const Person& lhs,const Person& rhs)
{
    return lhs._Age < rhs._Age;</pre>
}
int main(int argc,char* argv[])
{
    Person p1(5);
    Person p2(2);
    Person p3(4);
    Person p4(1);
    Person p5(3);
    vector<Person> pVec;
    pVec.push_back(p1);
    pVec.push_back(p2);
    pVec.push_back(p3);
    pVec.push_back(p4);
    pVec.push_back(p5);
    sort(pVec.begin(), pVec.end(), PersonLess);
```

```
return 0;
}
第四种办法,使用函数对象,比较大小,调用sort(first,last,pre
struct Person
{
public:
    int _Age;
    Person(int age)
    {
        _{Age} = age;
};
class PersonCompare
{
public:
    bool operator()(const Person& lhs,const Person& r
    {
        return lhs._Age < rhs._Age;</pre>
};
int main(int argc,char* argv[])
{
    Person p1(5);
    Person p2(2);
    Person p3(4);
    Person p4(1);
    Person p5(3);
    vector<Person> pVec;
    pVec.push_back(p1);
    pVec.push_back(p2);
    pVec.push_back(p3);
```

```
pVec.push_back(p4);
   pVec.push_back(p5);
   sort(pVec.begin(), pVec.end(), PersonCompare());
   return 0;
}
特别注意:上述的解决方法都是在windows下使用的,在linux也有对L
但是在linux下,用于比较大小的函数,两个参数要求都是const引用。
因此,在linux下,第一种方法对应的代码实现分别为:
struct Person
{
public:
   int _Age;
   Person(int age)
   {
       _{Age} = age;
   }
   bool operator< (const Person& rhs) const
       return this->_Age < rhs._Age;
可以认为linux的要求更加严格,也非常合理,为了兼容,都应该使用c
如果是要求按年龄逆序,只需要修改一下比较的地方就好了。如下:
return this->_Age > rhs._Age;
```

#### 理解函数对象

```
1、考虑下面的需求,函数接受一个int参数,返回这个参数加上100,
int add(int a)
{
    return a+100;
}
```

```
int main()
   {
       int b = add(1);
       return 0;
2、这里add是一个函数,但是sum也可以是一个对象,如下:
   class Add
   {
   public:
       int operator() (int a)
       {
          return a+100;
   };
   int main()
   {
       Add add;
       int b = add(1);
       return 0;
3、在上面的示例中,add是一个对象。可以把add当成一个函数来使用
   Add类重载了小括号,小括号就是一个方法名称,换个方式调用,有
   int main()
   {
       Add add;
       int b = add.operator()(1);
       return 0;
   operator()就是一个方法名,只不过可以省略(连同点号),如
4、当然也可以把一个匿名对象当成函数使用,如下:
   int main()
   {
       int b = Add()(1);
       return 0;
   }
```

```
5、再扩展一下,可以使用模板类,如下:
   template<typename T>
   class Add
   {
   public:
      T operator() (T a)
          return a+100;
      }
   };
   int main()
   {
      int b = Add < int > ()(1);
      double d = Add < double > ()(1.0);
      return 0;
6、那么问题来了,使用函数挺好的,也可以创建模板函数,为什么还要
   考虑下面的需求,有另一个客户,要求对函数的参数加上200,而[
   因此,不能修改函数的实现,必须增加一个函数。
   问题的根源在于:函数只能使用形参,而这里的只有一个形参。
   而对象却很容易解决这个问题,因为对象可以包含字段。如下:
   class Add
   public:
      Add(int delta):_Delta(delta)
      {
      }
      int operator() (int a)
      {
          return a+_Delta;
      }
   public:
      int _Delta;
```

```
};

int main()
{
    int clientA = Add(100)(1);
    int clientB = Add(200)(1);

    return 0;
}
7、还有一点很重要,函数对象还可以适配。
```

#### 理解迭代器

```
1、迭代器是对指针的封装,具备指针语义
2、迭代器内部关联一个指针ptr,指向Item,*与->操作符实现如下:
    Item& operator*() const {return *ptr;}
    Item* operator->() const {return ptr;}
    特别注意:空指针是判断 ptr == NULL ,而迭代器不行,迭代
```

#### 迭代器失效

```
1、示例代码如下:
    int main()
    {
        vector<int> aVec;
        aVec.push_back(1);
        aVec.push_back(2);
        aVec.push_back(3);

        vector<int>::iterator iter = aVec.begin();
        aVec.erase(iter);

        if(iter == aVec.end())
        {
        }
}
```

```
return 0;
   }
   if(iter == aVec.end())运行崩溃,错误信息Expression:
2、示例代码如下:
   int main()
   {
      vector<int> aVec;
      aVec.push_back(1);
      aVec.push_back(2);
      aVec.push_back(3);
      vector<int>::iterator iter = aVec.begin();
      aVec.erase(iter);
      int aa = *iter;
      return 0;
   int aa = *iter;运行崩溃,错误信息Expression: vector
3、对于map和set也是同样的问题,错误原因是:迭代器失效。
4、思考, 迭代器失效到底是什么意思?
   无论是基于连续内存的容器,还是基于节点的容器,保存的元素都:
   迭代器是对指针的封装,这个指针就是指向容器元素的内存地址。
   当容器删除一个元素,或者元素发生移动,容器会对这个元素进行。
5、但是对于分配的内存空间呢?
   详见 迭代器失效的测试
6、注意:如果容器保存的元素是指针,必须手动对指针进行delete,是
   因为容器是对元素析构,就是对指针析构,而对指针析构是不做任何
```

#### 迭代器失效的测试

1、当对迭代器删除或者移动的时候,会导致迭代器失效,迭代器失效到 迭代器是对指针的封装,取指针值的方法是: vector<int>::iterator iter = aContainer.begin(); 当对迭代器删除或者移动的时候,会对迭代器关联的对象进行析构

2、测试场景可以分为如下:

int

```
连续内存
                   WinDebug
                    WinRelease
                    Linux
            基干节点
                   WinDebug
                    WinRelease
                    Linux
    Person
           连续内存
                   WinDebug
                    WinRelease
                    Linux
            基干节点
                   WinDebug
                   WinRelease
                    Linux
3、先看 int 连续内存,测试代码如下:
    #include <stdio.h>
    #include <string>
    #include <vector>
    #include <list>
    using namespace std;
    int main()
    {
        vector<int> aContainer;
        aContainer.push_back(1);
        aContainer.push_back(2);
        aContainer.push_back(3);
        vector<int>::iterator iter = aContainer.begir
        int*pp = &*iter;
        int aa = *iter;
        printf("Int addr[%p] value[%d]\n",
            pp,
            aa);
```

```
aContainer.erase(iter);
       pp = &*iter;
       aa = *iter;
       printf("Int addr[%p] value[%d]\n",
          pp,
          aa);
       getchar();
       return 0;
   a、WinDebug下 pp = &*iter; 会崩溃,调试可以发现,原因扩
       迭代器会关联容器的指针,在WinDebug下,会检查迭代器关
       在删除迭代器之后, 迭代器关联容器指针为空, 导致断言失败
   b、WinRelease下,打印如下:
       Int addr[007C2478] value[1]
       Int addr[007C2478] value[2]
       原因是:release正常,因为没有断言进行检查。
       删除迭代器, 迭代器关联的元素指针不变, 对int值1的内存机
       但是基于连续内存,删除一个元素,后面的元素会逐一补上来
   c、Linux下,和WinRelease结果一样。
4、看 int 基于节点,测试代码如下:
   #include <stdio.h>
   #include <string>
   #include <vector>
   #include <list>
   using namespace std;
   int main()
   {
       list<int> aContainer;
       aContainer.push back(1);
       aContainer.push back(2);
       aContainer.push back(3);
       list<int>::iterator iter = aContainer.begin()
```

```
int* pp = &*iter;
       int aa = *iter;
       printf("Int addr[%p] value[%d]\n",
          pp,
          aa);
      aContainer.erase(iter);
      pp = &*iter;
       aa = *iter;
       printf("Int addr[%p] value[%d]\n",
          pp,
          aa);
       getchar();
       return 0;
   }
   a、WinDebug下,和基于连续内存一样,会崩溃。
   b、WinRelease下,打印如下:
       Int addr[00552468] value[1]
      Int addr[00552468] value[-17891602]
      原因是:release正常,因为没有断言进行检查。
      删除迭代器, 迭代器关联的元素指针不变, 对int值1的内存机
      基于节点,后面的元素并不移动。
   c、Linux下,打印如下:
       [root@localhost niu]# ./main
       Int addr[0x9c16010] value[1]
       Int addr[0x9c16010] value[1]
       为什么?
       可以这样认为,windows析构,会对这块内存取值重置为无效
      但是标识这块内存可以被使用,在运行一段时间后,这块内存
       也就是说,没有被析构,这块内存不能被使用。当析构时, w.
      而linux对原来的取值不处理,仅仅标识这块内存可用。不知
5、再看 Person 连续内存,测试代码如下:
   #include <stdio.h>
   #include <string>
   #include <vector>
```

```
#include <list>
using namespace std;
struct Person
{
    int
            _Age;
    string _Name;
    Person()
    {
    }
    Person(int age, string name)
    {
        _Age = age;
        _Name = name;
    }
    Person(const Person& p)
    {
        _Age = p._Age;
        _Name = p._Name;
    }
};
int main()
{
    list<Person> aContainer;
    aContainer.push_back(Person(1, "Andy"));
    aContainer.push_back(Person(2, "Bill"));
    aContainer.push_back(Person(3, "Caroline"));
    list<Person>::iterator iter = aContainer.begi
    Person* pp = &*iter;
    Person aa = *iter;
```

```
printf("Person addr[%p] value[%d:%s]\n",
           pp,
           aa._Age,
          aa. Name.c str());
       aContainer.erase(iter);
       pp = &*iter;
       aa = *iter;
       printf("Person addr[%p] value[%d:%s]\n",
           pp,
          aa._Age,
          aa._Name.c_str());
       getchar();
       return 0;
   测试结果,道理和int 连续内存一样。
6、再看 Person 连续节点。
   测试结果,道理和int 连续节点一样。
7、无论集合中的元素是基本类型还是用户定义的类型,集合都会对动态
```

#### 集合保存对象和保存指针

1、考虑下面的需求,在vector集合中保存一组Dog对象,可以使用vec 二者的区别是:

但是如果集合元素是指针本身,需要手动管理。

vector<Dog>在vector分配的内存上,直接存放Dog对象。

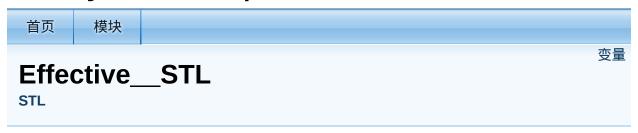
vector<Dog\*>在vector分配的内存上,存放Dog指针,Dog对象

- 2、优先考虑使用vector<Dog>?
  - a、vector<Dog>更节省内存, vector分配的内存也在动态内存。
  - b、vector<Dog>管理起来更简单,vector存放的是对象的副本 (注:这个时候对应的内存是一堆垃圾,还指向这块内存的设 但是对于指针需要手动管理,这增加内存管理的负担。没有释
- 3、但是有些场景下,需要使用vector<Dog\*>
- 4、第一种情况,对象的copy构造成本很大,或者不具备copy构造的语
- 5、第二种情况,包含一组类型不同但是相关联的对象,也就是继承关系

这个时候必须使用vector<Dog\*>, 否则保存副本,对象切割,不6、使用vector<Dog\*>增加内存管理的负担,解决办法是:使用代理,也就是智能指针。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

【S01】慎重选择容器类型
【S02】不要试图编写独立于容器类型的代码
【S03】确保容器中的对象拷贝正确而高效
【S04】使用empty而不是判断size是否为0
【S06】当心CPP编译器最烦人的分析机制
【S13】vector和string优先于动态分配的内存
【S16】了解如何把vector和string数据传给旧的API
【S17】使用swap技巧除去多余的容量
【S46】考虑使用函数对象而不是函数作为STL算法的参数
【S47】避免产生直写型的代码

#### 变量说明

#### 【S01】慎重选择容器类型

- 1、标准STL序列容器:vector、string、list和deque
- 2、标准STL关联容器:set、multiset、map和multimap
- 3、vector和string是基于连续内存,list、set和map是基于节点。 而deque是二者的结合, deque内部是一块一块的, 块与块之间是
- 4、对比vector, deque是双向开口。

#### 【S02】不要试图编写独立于容器类型的代码

- 1、为什么?
  - 从逻辑结构上,容器分为序列容器和关联容器。 从物理结构上,容器分为基于连续内存的容器和基于节点的容器。
- 2、对于不同的容器,支持的方法不一样(也就是暴露出来的接口不一样 相同方法名内部做的事情是不一样,迭代器、指针和引用无效的规则

#### 【S03】确保容器中的对象拷贝正确而高效

- 1、容器保存的对象,并不是你提供给容器的对象,而是这些对象的副本
- 2、保存对象的时候,调用copy构造,创建出副本。 这里特别注意:子类对象保存到父类容器中会存在对象切割。
- 3、因为是整体拷贝,为了高效,可以使用指针。但是指针带来问题,需 但是千万不能是auto\_ptr,因为这玩意的拥有权会转移。
- 4、为了避免不必要的拷贝,对于vector可以使用reserve提前把内存。

#### 【S04】使用emptv而不是判断size是否为0

- 1、二者的作用是一样的,结果也是等价的。就是判断集合是否为空。
- 2、二者是等价的,为什么强调使用empty,因为empty效率更高。

#### 【S06】当心CPP编译器最烦人的分析机制

```
1、考虑一个包含int的文件,复制到list,如下:
    ifstream dataFile("ints.bat");
    list<int> data(istream_iterator<int>(dataFile),is
2、上面的代码不是预期的行为。
3、先从最简单开始,声明方法 int f(double d); 等价的写法有 i 也就是说,形参名称可以使用括号括起来,形参名称也可以省略,4、考虑int g(double (*pf) ()); 形参是一个方法指针,等价的写 我们省略形参名称,就变成了 int g(double ());
5、现在考虑list<int> data(istream_iterator<int>(dataFile)
```

5、现在考虑list<int> data(istream\_iterator<int>(dataFi 第一个形参是:形参类型是istream\_iterator<int>,形参名和 第二个形参是:形参类型是一个方法指针,指向的方法是返回istr

6、C++中有一条规律,语句优先解释成 方法声明。当这个解释失败,尽 Student s; // OK

```
Print(s);
Student s = Student(); // OK
Print(s);

Print(Student()); // OK
Student* s = new Student(); // OK
Print(*s);

Student* s = new Student; // OK
Print(*s);
```

```
Student s(); // Error
Print(s);
因为C++会把Student s();当成一个方法声明。
7、怎么解决上面的问题?
两种办法:
办法一,对于方法调用,实参可以使用括号括起来,而对于方法声因此可以如下:
list<int> data( (istream_iterator<int>(dataFile) 办法二:不使用匿名对象,使用具名对象,如下:
istream_iterator<int> begin(dataFile);
istream_iterator<int> end;
list<int> data(begin,end);
```

#### 【S13】vector和string优先于动态分配的内存

- 1、使用new动态分配内存,必须承担如下责任:
  - a、使用delete释放内存
  - b、确保使用了正确的形式, delete与new的形式要匹配
  - c、不能重复delete
- 2、使用vector和string可以消除以上的负担。每当要动态分配一个数如果元素是字符char,使用string。否则使用vector。注意:看
- 3、vector和string的元素分配在堆上,它们内部维护一个指针,指向
- 4、vector和string,它们自己管理内存,内存会自动增长,当它们析
- 5、vector和string是功能完全的STL序列容器,可以使用很多STL功 没有begin, end, size这样的成员方法,也没有iterator这样
- 6、为了支持旧的代码,将vector和string转化为数组很简单。
- 7、有一种特殊情况,需要考虑。string是如此常用,它的使用效率很重 这在多线程中,会出现冲突问题。如果string不是引用计数,而是
- 8、对于基于引用计数的string,又运行在多线程环境下,有三种可行的
  - a、禁止引用计数,这种做法不可移植
  - b、寻找一个不使用引用计数的string
  - c、考虑vector<char>,代替string

注意: VS2010中的STL, string没有使用引用计数。

#### 【S16】了解如何把vector和string数据传给旧的API

1、尽量使用vector和string替换数组,但是老的代码还是使用数组。

需要把vector和string,暴露出数组接口,也就是第一个元素的 2、考虑方法DoSomething(const int\* pInt,size\_t size),对 DoSomething(&vec[0], v.size());

这里有个问题, vec的大小可能为0, 更安全的做法是:

```
if(!vec.empty())
{
    DoSomething(&vec[0], v.size());
```

- 3、考虑,能不能使用begin()替换&vec[0]? 我们知道,begin返回迭代器,是对指针的封装,类似于指针。 但是,不能把迭代器当成指针使用,可以使用&\*begin(),这种方
- 4、考虑DoSomething(const char\* pa); 对于string str,调用 DoSomething(str.c\_str()); c\_str返回一个char指针,指向字符串值,尾部再加一个空字符。:

但是, DoSomething的处理是以第一个空字符作为结束。

- 5、vector暴露出指针,调用端修改元素的值,通常没有问题。但是不能因为在外部增加新元素,vector不知道,不去更新size,size产注意:对于排序的vector,修改vector的元素值,也会产生问题
- 6、考虑C API数组初始化vector,很简单。如果C API数组初始化st可以先初始化vector<char>,vector<char>再去初始化strir先把C API数组写到vector,再把vector写到期望的STL容器中。也就是说,vector是一个适配器。

#### 【S17】使用swap技巧除去多余的容量

- 1、考虑下面的需求,对于vec开始的时候有1000个元素,后来只有10个后面的990个内存单元,没有使用,但是还被vec霸占着。如何释放
- 2、我们知道, vector进行copy构造的时候, 根据rhs 的size进行分ivector<int>(vec). swap(vec);

vector<int>(vec) 是个临时对象,可认为capacity为10,而

- 3、这里需要注意两点:
  - a、临时对象的capacity有可能还是大于10,不能保证容量最小,b、对于vector的swap方法,内部实现只是交换了彼此的begin这个很好理解,对于资源管理类,也就是内含指针的类,交换的时始个例子:甲住501,乙住502,现在甲乙想换房子。只要换一下针
- 4、考虑一个特殊情况,我想清空一个容器,并释放所有内存,该怎么办首先,clear方法是不行的,因为它只是把元素清空,内存还被霸

也就是:vector<int>().swap(vec);

#### 【S46】考虑使用函数对象而不是函数作为STL算法的参数

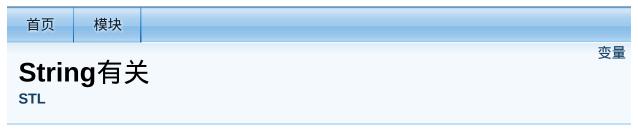
- 1、高级语言有个缺点,抽象程度提高了,但是所生成的代码效率降低了
- 2、操作一个只包含double类型的对象比直接操作double效率要低。
- 3、但是将函数对象传递给STL算法往往比传递函数效率要高。这是为什
- 4、第一个原因, C++并不能真正地将一个函数作为参数传递给另一个函 这就意味着,主调函数通过指针发出调用,编译器不会进行内联优 也就是说,函数指针参数抑制了内联机制。而函数对象是可以进行 因此,C++的sort算法就性能而言总是优于C语言的qsort
- 5、第二个原因,使用函数,有时候编译器会拒绝合法的代码,编译失败
- 6、第三个原因,使用函数对象有助于避免一些微妙的、语言本身的缺陷 也就是说,STL算法对于函数对象有更好的支持。

#### 【S47】避免产生直写型的代码

- 1、一条语句做多件事情,使用复杂的函数嵌套,会导致代码难以阅读和
- 2、软件工程的一条真理:代码被阅读的次数远远大于他被编写的次数。
- 3、更好的方法是:把做的多件事情分成几步来做。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

## Andy Niu Help 1.0.0.0



## 重变

string的常用方法

string中的特殊字符

#### 变量说明

#### string中的特殊字符

```
1、考虑如下的代码:
   #include <stdio.h>
   #include <string>
   using std::string;
   int main(int argc, char* argv[])
   {
       string aa = "abcde";
       aa[2] = 0;
       string bb(aa.c_str());
       string cc(aa.c_str(), aa.size());
       printf("aa size[%d] value[%s]\n", aa.size(),
       printf("bb size[%d] value[%s]\n", bb.size(),
       printf("cc size[%d] value[%s]\n", cc.size(),
       return 0;
2、输出结果为:
   aa size[5] value[ab]
   bb size[2] value[ab]
   cc size[5] value[ab]
3、得出结论:
   a、string中可以包含取值为0的字符,甚至可以是任意char类型
   b、这里和预期不一致,关键的问题是,对于c_str()方法,仅仅i
   c、在处理c_str()的时候,由于不知道处理的长度,只能是遇到\
```

而string cc(aa.c\_str(), aa.size()); 能够正确处于d、也就是说,使用char\*构造string的时候,要特别注意,如果

参见

#### string的常用方法

```
#include <algorithm>
// 是否匹配前缀
bool IsMatchPrefix(const string& str,const string& pr
{
    return strncmp(str.c_str(),prefix.c_str(),prefix.
}
// 是否包含字符串
bool IsContainsSubStr(const string& str,const string&
{
    return str.find(subStr,0) != string::npos;
}
// 用bbb替换aaa
void StringReplace(string& str,const string& aaa,cons
{
    size_t pos = 0;
    size_t aSize = aaa.size();
    size_t bSize = bbb.size();
    while((pos = str.find(aaa,pos)) != string::npos)
    {
        str.replace(pos, aSize, bbb);
        pos+=bSize;
    }
}
// 使用delim作为分隔符
```

```
void StringSplit(const string& str,const string& deli
{
    size_t begin = 0;
    size_t end = str.find(delim, begin);
    while(end!=string::npos)
    {
        strVec.push_back(str.substr(begin,end-begin))
        begin = end+strlen(delim.c_str());
        end = str.find(delim, begin);
    }
    if(end>begin)
        strVec.push_back(str.substr(begin,end-begin))
    }
    if(isRemoveEmpty)
    {
        strVec.erase(remove(strVec.begin(), strVec.enc
}
// 删除最后一个元素
// 注意:对于vector和list,删除最后一个元素,只能使用
// erase(迭代器) 和 resize(size-1)
void RemoveLastElement(string& str)
{
    if(str.size() == 0)
    {
        return;
    //str = str.substr(0, str.size()-1);
    //str.erase(str.size()-1);
    //str.erase(--str.end());
    str.resize(str.size()-1);
```

```
// 使最后一个字符无效,达到删除最后一个字符的目的
// 但是这种方式有问题,因为src的size没有变化
//str[str.size()-1]=0;
}
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

## Andy Niu Help 1.0.0.0



## 模块

传输层		
<b></b>		
应用层		
DAT.		
术语		
网络层		
链路层		

### 详细描述

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

## Andy Niu Help 1.0.0.0

## 模块

EffectiveTCPIP	
TCP	
UDP	

### 重变

测试udp打洞

端口复用

为什么对称性nat不能打洞成功

为什么网络穿透使用udp,不使用tcp

proc\_sys\_net\_ipv4下各项的意义

监听端口4个0

## 变量说明

#### proc\_sys\_net\_ipv4下各项的意义

```
/proc/sys/net/ipv4/icmp_timeexceed_rate
这个在traceroute时导致著名的"Solaris middle star"。这个文
/proc/sys/net/ipv4/igmp_max_memberships
主机上最多有多少个igmp (多播)套接字进行监听。
/proc/sys/net/ipv4/inet_peer_gc_maxtime
求助: Add a little explanation about the inet peer s
/proc/sys/net/ipv4/inet_peer_gc_mintime
每一遍碎片收集之间的最小时间间隔。当内存压力比较大的时候,调整证
/proc/sys/net/ipv4/inet_peer_maxttl
entries的最大生存期。在pool没有内存压力的情况下(比如, pool中
/proc/sys/net/ipv4/inet_peer_minttl
entries的最小生存期。应该不小于汇聚端分片的生存期。当pool的大
/proc/sys/net/ipv4/inet_peer_threshold
The approximate size of the INET peer storage. Starti
/proc/sys/net/ipv4/ip_autoconfig
这个文件里面写着一个数字,表示主机是否通过RARP、B00TP、DHCP或
/proc/sys/net/ipv4/ip_default_ttl
数据包的生存期。设置为64是安全的。如果你的网络规模巨大就提高这个
```

/proc/sys/net/ipv4/ip\_dynaddr/proc/sys/net/ipv4/icmp\_如果你有一个动态地址的自动拨号接口,就得设置它。当你的自动拨号接

/proc/sys/net/ipv4/ip\_forward 内核是否转发数据包。缺省禁止。

/proc/sys/net/ipv4/ip\_local\_port\_range 用于向外连接的端口范围。缺省情况下其实很小:1024到4999。

/proc/sys/net/ipv4/ip\_no\_pmtu\_disc 如果你想禁止"沿途MTU发现"就设置它。"沿途MTU发现"是一种技术,同

/proc/sys/net/ipv4/ipfrag\_high\_thresh
用 于IP分片汇聚的最大内存用量。分配了这么多字节的内存后,一旦用

/proc/sys/net/ipv4/ip\_nonlocal\_bind 如果你希望你的应用程序能够绑定到不属于本地网卡的地址上时,设置证

/proc/sys/net/ipv4/ipfrag\_low\_thresh 用于IP分片汇聚的最小内存用量。

/proc/sys/net/ipv4/ipfrag\_time IP分片在内存中的保留时间(秒数)。

/proc/sys/net/ipv4/tcp\_abort\_on\_overflow 一个布尔类型的标志,控制着当有很多的连接请求时内核的行为。启用的

/proc/sys/net/ipv4/tcp\_fin\_timeout 如果套接字由本端要求关闭,这个参数决定了它保持在FIN-WAIT-2状

/proc/sys/net/ipv4/tcp\_keepalive\_time 当keepalive起用的时候,TCP发送keepalive消息的频度。缺省是2/

/proc/sys/net/ipv4/tcp\_keepalive\_intvl 当探测没有确认时,重新发送探测的频度。缺省是75秒。 /proc/sys/net/ipv4/tcp\_keepalive\_probes 在认定连接失效之前,发送多少个TCP的keepalive探测包。缺省值是(

/proc/sys/net/ipv4/tcp\_max\_orphans 系 统中最多有多少个TCP套接字不被关联到任何一个用户文件句柄上。

/proc/sys/net/ipv4/tcp\_orphan\_retries 本端试图关闭TCP连接之前重试多少次。缺省值是7,相当于50秒~16分

/proc/sys/net/ipv4/tcp\_max\_syn\_backlog 记 录的那些尚未收到客户端确认信息的连接请求的最大值。对于有128

/proc/sys/net/ipv4/tcp\_max\_tw\_buckets 系 统同时保持timewait套接字的最大数量。如果超过这个数字,time

/proc/sys/net/ipv4/tcp\_retrans\_collapse 为兼容某些糟糕的打印机设置的"将错就错"选项。再次发送时,把数据f /proc/sys/net/ipv4/tcp\_retries1 在认定出错并向网络层提交错误报告之前,重试多少次。缺省设置为RF(

/proc/sys/net/ipv4/tcp\_retries2 在杀死一个活动的TCP连接之前重试多少次。RFC 1122规定这个限制应

/proc/sys/net/ipv4/tcp\_rfc1337 这个开关可以启动对于在RFC1337中描述的"tcp的time-wait暗杀危机

/proc/sys/net/ipv4/tcp\_sack 特别针对丢失的数据包使用选择性ACK,这样有助于快速恢复。

/proc/sys/net/ipv4/tcp\_stdurg 使用TCP紧急指针的主机需求解释。因为绝大多数主机采用BSD解释,所 /proc/sys/net/ipv4/tcp\_syn\_retries 在内核放弃建立连接之前发送SYN包的数量。

/proc/sys/net/ipv4/tcp\_synack\_retries 为了打开对端的连接,内核需要发送一个SYN并附带一个回应前面一个S' /proc/sys/net/ipv4/tcp\_timestamps 时间戳可以避免序列号的卷绕。一个1Gbps的链路肯定会遇到以前用过的

/proc/sys/net/ipv4/tcp\_tw\_recycle 能够更快地回收TIME-WAIT套接字。缺省值是1。除非有技术专家的建议

/proc/sys/net/ipv4/tcp\_window\_scaling

一般来说TCP/IP允许窗口尺寸达到65535字节。对于速度确实很高的网一旦内核认为它无法发包,就会丢弃这个包,并向发包的主机发送ICMP

/proc/sys/net/ipv4/icmp\_echo\_ignore\_all 根本不要响应echo包。请不要设置为缺省,它可能在你正被利用成为Do /proc/sys/net/ipv4/icmp\_echo\_ignore\_broadcasts [Usefu如果你ping子网的子网地址,所有的机器都应该予以回应。这可能成为:/proc/sys/net/ipv4/icmp\_echoreply\_rate 设置了向任意主机回应echo请求的比率。

/proc/sys/net/ipv4/icmp\_ignore\_bogus\_error\_responses 设置它之后,可以忽略由网络中的那些声称回应地址是广播地址的主机组

/proc/sys/net/ipv4/icmp\_paramprob\_rate
一个相对不很明确的ICMP消息,用来回应IP头或TCP头损坏的异常数据。

tcp\_syn\_retries : INTEGER 默认值是5

对于一个新建连接,内核要发送多少个 SYN 连接请求才决定放弃。不匠

tcp\_synack\_retries : INTEGER

默认值是5

对于远端的连接请求SYN,内核会发送SYN + ACK数据报,以确认收到

tcp\_keepalive\_time :INTEGER

默认值是7200(2小时)

当keepalive打开的情况下,TCP发送keepalive消息的频率。(由于

tcp\_keepalive\_probes:INTEGER

默认值是9

TCP发送keepalive探测以确定该连接已经断开的次数。(注意:保持连

tcp\_keepalive\_intvl:INTEGER

默认值为75

探测消息发送的频率,乘以tcp\_keepalive\_probes就得到对于从开始

tcp\_retries1 : INTEGER

默认值是3

放弃回应一个TCP连接请求前,需要进行多少次重试。RFC 规定最低的数

tcp\_retries2 : INTEGER

默认值为15

在丢弃激活(已建立通讯状况)的TCP连接之前,需要进行多少次重试。默

tcp\_orphan\_retries : INTEGER

默认值是7

在近端丢弃TCP连接之前,要进行多少次重试。默认值是7个,相当于 50科

tcp\_fin\_timeout : INTEGER

默认值是 60

对于本端断开的socket连接,TCP保持在FIN-WAIT-2状态的时间。对

tcp\_max\_tw\_buckets : INTEGER

默认值是180000

系 统在同时所处理的最大 timewait sockets 数目。如果超过此数

tcp\_tw\_recycle : BOOLEAN

默认值是⊙

打开快速 TIME-WAIT sockets 回收。除非得到技术专家的建议或要 tcp\_tw\_reuse: BOOLEAN

默认值是⊙

该文件表示是否允许重新应用处于TIME-WAIT状态的socket用于新的T

tcp\_max\_orphans : INTEGER

缺省值是8192

系统所能处理不属于任何进程的TCP sockets最大数量。假如超过这个

tcp\_abort\_on\_overflow : BOOLEAN

缺省值是0

当守护进程太忙而不能接受新的连接,就象对方发送reset消息,默认(i

tcp\_syncookies : BOOLEAN

默认值是⊙

只有在内核编译时选择了CONFIG\_SYNCOOKIES时才会发生作用。当出于注意:该选项千万不能用于那些没有收到攻击的高负载服务器,如果在Etcp\_max\_syn\_backlog

tcp\_synack\_retries

tcp\_abort\_on\_overflow

syncookie严重的违背TCP协议,不允许使用TCP扩展,可能对某些服务

tcp\_stdurg : BOOLEAN

默认值为⊙

使用 TCP urg pointer 字段中的主机请求解释功能。大部份的主机社tcp\_max\_syn\_backlog :INTEGER

对于那些依然还未获得客户端确认的连接请求,需要保存在队列中最大数

tcp\_window\_scaling :INTEGER

缺省值为1

该 文件表示设置tcp/ip会话的滑动窗口大小是否可变。参数值为布尔(

tcp\_timestamps : BOOLEAN

缺省值为1

Timestamps 用在其它一些东西中,可以防范那些伪造的 sequence 5

tcp\_sack : BOOLEAN

缺省值为1

使用 Selective ACK,它可以用来查找特定的遗失的数据报--- 因此

tcp\_fack : BOOLEAN

缺省值为1

打开FACK拥塞避免和快速重传功能。(注意,当tcp\_sack设置为0的时间

tcp\_dsack : BOOLEAN

缺省值为1

允许TCP发送"两个完全相同"的SACK。

tcp\_ecn : BOOLEAN

缺省值为⊙

打开TCP的直接拥塞通告功能。

tcp\_reordering : INTEGER

默认值是3

TCP流中重排序的数据报最大数量 。 (一般有看到推荐把这个数值略微

tcp\_retrans\_collapse : BOOLEAN

缺省值为1

对于某些有bug的打印机提供针对其bug的兼容性。(一般不需要这个支持

tcp\_wmem(3个INTEGER变量): min, default, max

min:为TCP socket预留用于发送缓冲的内存最小值。每个tcp sock

default:为TCP socket预留用于发送缓冲的内存数量,默认情况下i

max: 用于TCP socket发送缓冲的内存最大值。该值不会影响net.co

tcp\_rmem (3个INTEGER变量): min, default, max

min:为TCP socket预留用于接收缓冲的内存数量,即使在内存出现紧

default:为TCP socket预留用于接收缓冲的内存数量,默认情况下ì

max:用于TCP socket接收缓冲的内存最大值。该值不会影响 net.c

tcp\_mem(3个INTEGER变量):low, pressure, high

low: 当TCP使用了低于该值的内存页面数时, TCP不会考虑释放内存。

pressure: 当TCP使用了超过该值的内存页面数量时, TCP试图稳定其

high:允许所有tcp sockets用于排队缓冲数据报的页面量。(如果超

一般情况下这些值是在系统启动时根据系统内存数量计算得到的。

tcp\_app\_win : INTEGER

默认值是31

保留max(window/2^tcp\_app\_win, mss)数量的窗口由于应用缓冲。

tcp\_adv\_win\_scale : INTEGER

默认值为2

计算缓冲开销bytes/2^tcp\_adv\_win\_scale(如果tcp\_adv\_win\_stcp\_rfc1337 :B00LEAN

缺省值为0

这个开关可以启动对于在RFC1337中描述的"tcp 的time-wait暗杀危tcp\_low\_latency : BOOLEAN

缺省值为⊙

允许 TCP/IP 栈适应在高吞吐量情况下低延时的情况;这个选项一般情tcp\_westwood:BOOLEAN

缺省值为0

启用发送者端的拥塞控制算法,它可以维护对吞吐量的评估,并试图对常tcp\_bic:B00LEAN

缺省值为0

为快速长距离网络启用 Binary Increase Congestion;这样可以!

\$ /proc/sys/net/core/wmem\_max

最大socket写buffer,可参考的优化值:873200

\$ /proc/sys/net/core/rmem\_max

最大socket读buffer,可参考的优化值:873200

\$ /proc/sys/net/ipv4/tcp\_wmem

TCP写buffer,可参考的优化值: 8192 436600 873200

\$ /proc/sys/net/ipv4/tcp\_rmem

TCP 读buffer,可参考的优化值: 32768 436600 873200

\$ /proc/sys/net/ipv4/tcp\_mem

同样有3个值, 意思是:

net.ipv4.tcp\_mem[0]: 低于此值,TCP没有内存压力.

net.ipv4.tcp\_mem[1]:在此值下,进入内存压力阶段.

net.ipv4.tcp\_mem[2]: 高于此值,TCP拒绝分配socket.

上述内存单位是页,而不是字节.可参考的优化值是:786432 1048576

\$ /proc/sys/net/core/netdev\_max\_backlog

进入包的最大设备队列.默认是300,对重负载服务器而言,该值太低,可ì

```
$ /proc/sys/net/core/somaxconn
listen()的默认参数,挂起请求的最大数量. 默认是128.对繁忙的服务
$ /proc/sys/net/core/optmem_max
socket buffer的最大初始化值,默认10K.
$ /proc/sys/net/ipv4/tcp_max_syn_backlog
进入SYN包的最大请求队列. 默认1024.对重负载服务器,增加该值显然
$ /proc/sys/net/ipv4/tcp_retries2
TCP失败重传次数,默认值15,意味着重传 15次才彻底放弃.可减少到5,
$ /proc/sys/net/ipv4/tcp_keepalive_time
$ /proc/sys/net/ipv4/tcp_keepalive_intvl
$ /proc/sys/net/ipv4/tcp_keepalive_probes
这3个参数与TCP KeepAlive有关.默认值是:
tcp_keepalive_time = 7200 seconds (2 hours)
tcp_keepalive_probes = 9
tcp keepalive intvl = 75 seconds
意思是如果某个TCP连接在idle 2个小时后,内核才发起probe.如果pr
/proc/sys/net/ipv4/tcp_keepalive_time 1800
/proc/sys/net/ipv4/tcp_keepalive_intvl 30
/proc/sys/net/ipv4/tcp_keepalive_probes 3
$ proc/sys/net/ipv4/ip_local_port_range
指定端口范围的一个配置,默认是32768 61000,已够大.
net.ipv4.tcp_syncookies = 1
表示开启SYN Cookies。当出现SYN等待队列溢出时,启用cookies来
net.ipv4.tcp_tw_reuse = 1
表示开启重用。允许将 TIME-WAIT sockets重新用于新的TCP连接,
net.ipv4.tcp_tw_recycle = 1
表示开启TCP连接中TIME-WAIT sockets的快速回收,默认为0,1表,
net.ipv4.tcp_fin_timeout = 30
表示如果套接字由本端要求关闭,这个参数决定了它保持在FIN-WAIT-
net.ipv4.tcp_keepalive_time = 1200
表示当keepalive起用的时候,TCP发送keepalive消息的频度。缺省
net.ipv4.ip local port range = 1024 65000
表示用于向外连接的端口范围。缺省情况下很小:32768到61000,改为
net.ipv4.tcp_max_syn_backlog = 8192
表示SYN队列的长度,默认为1024,加大队列长度为8192,可以容纳更
net.ipv4.tcp_max_tw_buckets = 5000
表示系统同时保持TIME_WAIT套接字的最大数量,如果超过这个数字,
```

```
# 以下一段为抵抗syn flood攻击,平时建议关闭
sysctl -w net.ipv4.tcp_syncookies=1
                                               # tc
sysctl -w net.ipv4.tcp_max_syn_backlog=1280
                                            # syn队
sysctl -w net.ipv4.tcp_synack_retries=2
sysctl -w net.ipv4.tcp_syn_retries=2
# 以下一段为应对tcp connect连接耗尽攻击,如果开启iptables (
# 有严重连接性能影响和不稳定因素,慎用
sysctl -w tcp_tw_recycle=1
sysctl -w tcp_tw_reuse=1
sysctl -w tcp_keepalive_intvl=60
                                                # 9
sysctl -w tcp_keepalive_probes=3
                                               # 默
sysctl -w tcp_keepalive_time=1800
sysctl -w tcp_fin_timeout=30
#sysctl -w net.ipv4.tcp_retries1=2
#sysctl -w net.ipv4.tcp_retries2=8
sysctl -w net.ipv4.ip_conntrack_max=65535
                                                 #
```

#### 为什么对称性nat不能打洞成功

### 1、nat的类型分为:

- (1) 完全锥型NAT(FULL CONE NAT) 完全锥型NAT在内网用户A(Private Endpoint)首次向外部 并为A分配一个公网地址和端口(Public Endpoint),以后 此后,任何外部主机想要与A通信,只要将数据包发送到Pub]
- (2) 限制锥型NAT (RESTRICT CONE NAT)
  R制锥型NAT在内网用户A(Private Endpoint)首次向外部并为A分配一个公网地址和端口(Public Endpoint),以后此后,如果某个外部主机(Endpoint IP:PORT)想要与A证保证A曾用当前与NAT的会话向该外部主机的IP地址发送过数:
- (3) 端口限制锥型NAT(PORT RESTRICT CONE NAT) 端口限制锥型在内网用户A(Private Endpoint)首次向外部并为A分配一个公网地址和端口(Public Endpoint),以后此后,如果某个外部主机(Endpoint IP:PORT)想要与A证保证A曾用当前与NAT的会话向该外部主机的Endpoint发送证
- (4) 对称型NAT(SYMMETRIC NAT)

对称型NAT是一种比较特殊的NAT。内网用户A(Private En并为A分配一个公网地址和端口(Public Endpoint1),以为如果之后A用同一个Socket向外部主机S2发送数据包,这时分并为A分配一个新的公网地址和端口对(Public Endpoint)对称型NAT规定Public Endpoint1和Public Endpoint2此外,如果任何外部主机想要发送数据给A,那么它首先应该一否则即使他知道内网主机的Public Endpoint也不能发送数这种NAT可以通过端口猜测等方法进行穿透,但是效果并不是

- 2、锥形和对称形的区别是:对于锥形,B使用端口5555连接S的端口66B使用端口5555连接A的端口4444,在NAT\_B转换为6200到4444对于对称形,前一次是6200,后一次可能是6201,反正两次不一样
- 3、连接都是udp,现在考虑内网主机A和另一个内网主机B都连接了外网内网是消息发过来的,外网通过udp连接可以获取。
- 4、现在A去连接B,A该如何连接B呢?

A和S说,我要去连接B,于是S告诉A有关B的内网和外网ip和port去连接B的内网,不能成功。去连接B的外网,因为没有在NAT\_B设

5、如何不让NAT\_B丢弃呢?

需要B从里面打个洞出来,这个洞怎么打呢?

S告诉B,A要来连接你,你向A那边在NAT\_B上打个洞,这是A的内型,如果是锥形,在外网上的端口和之前B连接S是一样的,和S为6200由于之前S告诉A,B的外网端口就是6200,那么从A的内网去连接

6、但是如果是对称形,B向A打洞,端口是6201,那么从A的内网去连边。 也就是说,A期望去连接B的6200(这个6200就是B连接S的端口)

#### 为什么网络穿透使用udp,不使用tcp

- 1、考虑外网访问内网主机A,内网网关为NAT\_A,从外网访问不了内网。 从内网出来的请求,经过NAT\_A转换后,对外的端口是一样的,假是
- 2、根据从外面请求的传入能力,将NAT\_A分为全锥型,限制性锥型,端 全锥型:外网都能传输进来

限制性锥型:打洞对应的的IP才能传输进来

端口限制性锥型:打洞对应的IP和Port才能传输进来。

3、对于udp,同一个端口既可以向外连接,又能够接受来自外部的连接但是对于tcp,套接字通过调用connect()函数来建立向外的连接而且tcp的套接字通常仅允许建立1对1的响应。

即应用程序在将一个套接字绑定到本地的一个端口以后,任何试图:

1、正常使用场景是:主机A位于NAT\_A后面,主机B位于NAT\_B后面,NAA向B进行UDP连接,需要B向A方向,在NAT\_B上打个洞,要求NAT\_2、这个场景不好搭建,搭建一个更简单的场景进行测试。如下: 主机A是192.168.1.160,udp绑定端口4444 NAT\_A是10.22.4.201 192.168.1.164 服务S是10.22.4.145,udp绑定端口6666

从A[192.168.1.160:4444]去连接S[10.22.4.145:6666]没一但是如果从S[10.22.4.145:6666]去连接A的内网4444,显然不行。连接A的外网4444,因为没有在NA为什么不在NAT\_A对4444设置端口映射,因为往往不知道A的程序。

3、从S[10.22.4.145:6666]去连接A[10.22.4.201:4444] 在A进行udp绑定,如下:

[root@localhost ~]# nc -uv -l -p4444
listening on [any] 4444 ...

#### 在S去连接A,如下:

[niu\_zibin@localhost sbin]\$ nc -uv -p6666 10.22.4 10.22.4.201: inverse host lookup failed: : No st (UNKNOWN) [10.22.4.201] 4444 (?) open open

[niu\_zibin@localhost sbin]\$ 输入open直接报错

4、解决办法是先从A到S打个洞。

在S进行udp绑定,如下:

[niu\_zibin@localhost sbin]\$ nc -uv -l -p6666 listening on [any] 6666 ...

#### 在A去连接S,如下:

[root@localhost ~]# nc -uv -p4444 10.22.4.145 666 10.22.4.145: inverse host lookup failed: : No st (UNKNOWN) [10.22.4.145] 6666 (?) open open

hhh 可以与S[10.22.4.145 6666]交互 注意:在A看到的是A的内网到S的连接,在S看到的是S到A的外网设 5、现在关闭A的S的主动连接,之前A的绑定4444端口,可以与S进行交 也就是说,A和S都有绑定udp端口,S去连接A失败,A向S打个洞( 这个时候S与这个洞可以交互,然后把这个洞关闭,S就可以与A的约

#### 监听端口4个0

- 1、tcp或者udp作为服务端,监听的时候,监听地址为 0.0.0.0,这是
- 2、正常情况下,使用bind系统调用,监听的时候,使用本机ip地址和-
- 3、考虑下面的情况,本地机器上的ip地址是变化的,本地机器也可能有特别是在多个网卡的情况下。比如有两个ip地址,如下:

```
10.36.65.80
192.168.1.80
我监听端口9820,期望连接10.36.65.80和192.168.1.80的98
那么ip地址该怎么填写呢?
```

4、指定地址为0.0.0.0,表示本机监听的地址不确定,因为有多个ip划

#### 端口复用

```
1、无论是udp还是tcp,一个socket不能绑定两个端口。测试代码如下
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include <unistd.h>
   #include <sys/socket.h>
   #include <netinet/in.h>
   #include <arpa/inet.h>
   int main(int argc, char* argv[])
   {
       char server_ip[30] = "10.22.4.145";
       // 初始化本地网络信息
       struct sockaddr_in my_addr;
       bzero(&my_addr, sizeof(my_addr));
       my addr.sin family = AF INET;
       my_addr.sin_port = htons(8000);
```

```
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int sockfd;
    sockfd = socket(AF_INET, SOCK_DGRAM, 0); //创
    if(sockfd < 0)
    {
            perror("socket");
            exit(-1);
    }
    // 第一次绑定端口8000
    int err log;
    err_log = bind(sockfd, (struct sockaddr*)&my_
    if(err_log != 0)
    {
            perror("bind 8000");
            close(sockfd);
            exit(-1);
    }
    // 又一次绑定别的端口9000, 会绑定失败
    my_addr.sin_port = htons(9000);
    err_log = bind(sockfd, (struct sockaddr*)&my_
    if(err_log != 0)
    {
            perror("bind 9000");
            close(sockfd);
            exit(-1);
    close(sockfd);
    return 0;
}
[niu_zibin@localhost socket]$ g++ -o main main.cr
[niu zibin@localhost socket]$ ./main
bind 9000: Invalid argument
```

```
改成tcp,也就是 sockfd = socket(AF_INET, SOCK_STRE/
2、udp和tcp是传输层的两个协议,他们之间的端口不搭嘎。也就是ud
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include <unistd.h>
   #include <sys/socket.h>
   #include <netinet/in.h>
   #include <arpa/inet.h>
   int main(int argc, char* argv[])
    {
       char server_ip[30] = "10.22.4.145";
       // 初始化本地网络信息
       struct sockaddr_in my_addr;
       bzero(&my_addr, sizeof(my_addr));
       my_addr.sin_family = AF_INET;
       my_addr.sin_port = htons(8000);
       my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
       int sockfd;
       sockfd = socket(AF_INET, SOCK_DGRAM, 0); //创
       if(sockfd < 0)
       {
               perror("socket");
               exit(-1);
       }
       // UDP绑定端口8000
       int err_log;
       err log = bind(sockfd, (struct sockaddr*)&my
       if(err log != 0)
       {
               perror("bind 8000");
               close(sockfd);
               exit(-1);
```

```
// TCP绑定8000
    my_addr.sin_port = htons(8000);
    int sockfd 2;
    sockfd_2 = socket(AF_INET, SOCK_STREAM, 0); /
    if(sockfd < 0)
    {
            perror("socket");
            exit(-1);
    }
    err_log = bind(sockfd_2, (struct sockaddr*)&n
    if(err log != 0)
    {
            perror("bind 8000");
            close(sockfd);
            exit(-1);
    }
    // 注意:TCP和UDP不同,TCP绑定之后,还必须要listen
    listen(sockfd_2,10);
    getchar();
    close(sockfd);
    close(sockfd_2);
    return 0;
}
运行
[niu_zibin@localhost 2]$ g++ -o main main.cpp
[niu_zibin@localhost 2]$ ./main
查看绑定信息
[niu_zibin@localhost ~]$ netstat -anp|grep 8000
(Not all processes could be identified, non-owned
will not be shown, you would have to be root to s
                  0 0.0.0.0:8000
tcp
           0
```

```
0 0.0.0.0:8000
   udp
3、默认情况下,对于udp,两个socket不能绑定同一个端口,对于tcr
   对于udp,使用SO_REUSEADDR,两个socket可以重复绑定同一个
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include <unistd.h>
   #include <sys/socket.h>
   #include <netinet/in.h>
   #include <arpa/inet.h>
   int main(int argc, char* argv[])
   {
       char server_ip[30] = "10.22.4.145";
       // 初始化本地网络信息
       struct sockaddr_in my_addr;
       bzero(&my_addr, sizeof(my_addr));
       my_addr.sin_family = AF_INET;
       my_addr.sin_port = htons(8000);
       my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
       int sockfd;
       sockfd = socket(AF_INET, SOCK_DGRAM, 0); //创
       if(sockfd < 0)
       {
               perror("socket");
               exit(-1);
       }
       int err log;
       // 在sockfd绑定bind之前,设置其端口复用
       // sockfd绑定端口8000
       int opt = 1;
       setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (cc
       err_log = bind(sockfd, (struct sockaddr*)&my_
       if(err log != 0)
```

```
perror("bind 8000");
           close(sockfd);
           exit(-1);
   }
   int sockfd 2;
   sockfd_2 = socket(AF_INET, SOCK_DGRAM, 0); //
   if(sockfd\ 2<0)
   {
           perror("socket");
           exit(-1);
   }
   // 在sockfd绑定bind之前,设置其端口复用
   // sockfd 2绑定端口8000
   setsockopt(sockfd_2,SOL_SOCKET,SO_REUSEADDR,(
   err_log = bind(sockfd_2, (struct sockaddr*)&n
   if(err_log != 0)
   {
           perror("bind 8000");
           close(sockfd_2);
           exit(-1);
   getchar();
   close(sockfd);
   close(sockfd_2);
   return 0;
注意:每个socket绑定之前,都要设置SO_REUSEADDR。
   如果第一个socket绑定端口8000之前,没有设置端口复用,
   bind 8000: Address already in use
可以看到进程对于udp的8000端口,绑定了两次,如下:
[niu_zibin@localhost ~]$ netstat -anp|grep 8000
(Not all processes could be identified, non-owned
will not be shown, you would have to be root to s
```

```
udp
                     0 0.0.0.0:8000
                                                    (
   udp
               0
                     0 0.0.0.0:8000
4、但是对于tcp,即使设置了SO_REUSEADDR,也不能复用端口。
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   #include <unistd.h>
   #include <sys/socket.h>
   #include <netinet/in.h>
   #include <arpa/inet.h>
   int main(int argc, char* argv[])
    {
       char server_ip[30] = "10.22.4.145";
        // 初始化本地网络信息
        struct sockaddr_in my_addr;
        bzero(&my_addr, sizeof(my_addr));
       my_addr.sin_family = AF_INET;
       my_addr.sin_port = htons(8000);
       my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        int sockfd;
        sockfd = socket(AF_INET, SOCK_STREAM, 0); //{
       if(sockfd < 0)
        {
           perror("socket");
           exit(-1);
       }
        int err log;
       // 在sockfd绑定bind之前,设置其端口复用
        // sockfd绑定端口8000
        int opt = 1;
        setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (cc
       err_log = bind(sockfd, (struct sockaddr*)&my_
        if(err log != 0)
```

```
{
        perror("bind 8000");
        close(sockfd);
        exit(-1);
    listen(sockfd, 10);
    int sockfd_2;
    sockfd_2 = socket(AF_INET, SOCK_STREAM, 0); /
    if(sockfd_2 < 0)
    {
        perror("socket");
        exit(-1);
    }
    // 在sockfd绑定bind之前,设置其端口复用
    // sockfd_2绑定端口8000
    setsockopt(sockfd_2,SOL_SOCKET,SO_REUSEADDR,(
    err_log = bind(sockfd_2, (struct sockaddr*)&n
    if(err_log != 0)
    {
        perror("bind 8000");
        close(sockfd_2);
        exit(-1);
    listen(sockfd_2,10);
    getchar();
    close(sockfd);
    close(sockfd_2);
    return 0;
}
报错如下:
[niu_zibin@localhost 4]$ g++ -o main main.cpp
[niu_zibin@localhost 4]$ ./main
bind 8000: Address already in use
```

- 5、需要注意的是,对于udp,绑定端口就好了。但是对于tcp,只是绑定 绑定之后,还必须要listen,才能看到tcp的监听。
- 6、对于udp,端口可以复用,绑定同一个端口。但是对于tcp,不能在F 那么端口复用对于tcp有什么意义呢?
- 7、首先,对于相同的端口,可以在不同的ip地址上监听。它的意义在于 或者一个物理网卡配置了多个逻辑ip,或者表现形式不同.

比如当前主机为10.22.4.145,tcp可以分别在10.22.4.145, 使用网络助手NetAssist也可以测试。

```
测试代码如下:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int main(int argc, char* argv[])
{
    char server_ip[30] = "10.22.4.145";
    // 初始化本地网络信息
    struct sockaddr_in my_addr;
    bzero(&my_addr, sizeof(my_addr));
    my addr.sin_family = AF_INET;
    my_addr.sin_port = htons(8000);
    my_addr.sin_addr.s_addr = inet_addr("10.22.4.
    int sockfd;
    sockfd = socket(AF_INET, SOCK_STREAM, 0); //{
    if(sockfd < 0)
    {
        perror("socket");
        exit(-1);
    }
    int err_log;
```

```
// 在sockfd绑定bind之前,设置其端口复用
// sockfd绑定端口8000
int opt = 1;
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (cc
err_log = bind(sockfd, (struct sockaddr*)&my_
if(err log != 0)
{
    perror("bind 8000");
    close(sockfd);
    exit(-1);
listen(sockfd, 10);
int sockfd 2;
sockfd_2 = socket(AF_INET, SOCK_STREAM, 0); /
if(sockfd_2 < 0)
{
    perror("socket");
    exit(-1);
}
// 在sockfd绑定bind之前,设置其端口复用
// sockfd 2绑定端口8000
my_addr.sin_addr.s_addr = inet_addr("127.0.0.
setsockopt(sockfd_2,SOL_SOCKET,SO_REUSEADDR,(
err_log = bind(sockfd_2, (struct sockaddr*)&n
if(err_log != 0)
{
    perror("bind 8000");
    close(sockfd_2);
    exit(-1);
listen(sockfd_2,10);
getchar();
close(sockfd);
close(sockfd_2);
```

	return 0;
	}
	查看监听情况,如下: [niu_zibin@localhost ~]\$ netstat -anp grep 8000 (Not all processes could be identified, non-owned
	will not be shown, you would have to be root to s
	tcp 0 0 127.0.0.1:8000 (
	tcp 0 0 10.22.4.145:8000 (
	注意:这种情况,不设置SO_REUSEADDR,也可以在不同ip上的框
8、	tcp还有一种情况,相同的tcp连接四元组,比如当前连接被动断开ì
	这时候设置S0_REUSEADDR,可以在这个端口上监听。
	当一个socketA被动断开连接,处于time_wait状态,另一个soc
	注意:如果不想经历TIME_WAIT的过程,设置如下:
	BOOL bDontLinger = FALSE;
	setsockopt(s, SOL_SOCKET, SO_DONTLINGER, (const cha
9、	对于udp,设置了SO_REUSEADDR,相同IP和port,可以重复绑定。
·	如果外部来的数据报目的地址是单播,则只有最后一个socket能以
	如果外部来的数据报目的地址是多播,则多个socket都能收到相同
	也就是说,相同ip和port的绑定,用于udp的多播。对于udp,相
10	、linux上,tcp不能在相同的ip和port监听。对于freebsd(类u
TO.	·
	可以让tcp的socket绑定相同的ip和port,只有第一个socket能

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块

Effective\_\_TCP\_IP

TCP\_IP » 传输层

## 重变

- 【T01】理解面向连接和无连接协议之间的区别
- 【T02】理解子网和CIDR的概念
- 【T03】理解私有地址和NAT
- 【T04】开发并使用应用程序框架
- 【T05】套接字接口比XTI TLI更好用
- 【T06】记住TCP是一种流协议
  - 【T07】不要低估tcp的性能
  - 【T08】避免重新编写TCP
  - 【T09】要认识到TCP是一个可靠的,但不是绝对可靠的协议
  - 【T10】记住,TCP\_\_IP不是轮询的
  - 【T11】提防对等实体的不友好动作

## 变量说明

#### 【T01】理解面向连接和无连接协议之间的区别

- 1、面向连接和无连接指的是协议,本质区别在于:对于无连接协议来说而对于面向连接的协议,协议实现维护了当前分组与后继分组有关。
- 2、无连接就是指udp,分组称为数据报(datagram),每个分组都独立但是不保证数据报不丢失,不延迟,不错序。

注意:对于udp,可以在应用层维护数据报的状态。

- 3、面向连接就是tcp,tcp的分组称为段(segment),协议维护了分
- 4、一个标准的类比就是:udp就像寄明信片,tcp就像打电话。
- 5、udp有这么多缺点,为什么还要使用这种协议呢?
  - a、面向连接只支持一对一通信,无连接很方便地支持一对多和多对
  - b、更重要的是, 无连接是面向连接的基础, udp和tcp在ip协议之 ip就是不可靠的无连接服务。
- 6、tcp是怎么提供可靠的传输?

tcp在ip协议上增加了3项功能:

- a、为tcp段中的数据提供校验和,在接收端检查数据是否损坏
- b、tcp为每个字节提供序列号,即使在接收端错序了,可以根据性注意:tcp并没有为每一个字节附加一个序列号,而是在tcp. 这样就隐含了段中其它字节的序列号
- c、tcp提供了确认重传机制
- 7、确认重传机制是如何工作的?

tcp是全双工的,每一端既是接收端又是发送端。

8、站在接收端的角度,接收端维护了接收窗口,左边界是期望接收的下接收端通过告诉发送端自己的接收窗口,可以用于流量控制,防止的对于接收的数据(一组字节,会覆盖接收窗口的一个区间),如果如果在右边界之后,说明是没有对应的缓冲区存储,丢弃。

如果不是所期望的下一个字节,说明是错序的,需要等待前面缺少如果是所期望的下一个字节,通知应用程序有数据可读。这时候所能

- 窗口向右滑动。同时向发送端发送ack,告知所期望接收的下一个59、站在发送端的角度,发送端维护发送窗口,窗口划分成两部分,已发对于已发送但还未被确认的字节,会启动RTO(Retransmission注意:RTO超时并不意味着原来的数据没有到达目的地,有可能是适重传不会导致什么问题,因为接收端会丢弃重新收到的数据。
- 10、udp在ip协议上增加了2项功能,分别如下:
  - a、提供了可选的校验和用来检测数据是否损坏。ip也有校验和,f mudp和tcp提供的校验和用于保护他们自己的首部和数据。
  - b、udp增加了端口的概念(当然tcp也是),ip地址对应主机, 峁

#### 【T02】理解子网和CIDR的概念

1、IP地址分为5类,A、B、C、D、E,它们的前缀分别是:

A:0 网络个数2^7, 主机个数2^24, 大概1千6百万

B:10 网络个数2<sup>14</sup>,大概1万6千,主机个数2<sup>16</sup>,対

C: 110 网络个数2^21,大概2百万,主机个数2^8,254

D:1110 用于多播编址 E:1111 留作未来使用

注意:对于主机号,全0和全1特殊用处,因此可用的地址要减去2

- 2、子网划分,IP地址分为网络ID和主机ID,考虑B类的主机ID,可以图 因此在内部(比如局域网)可以从主机ID中,划出一部分作为子网
- 3、广播地址分为以下情况:

有限广播为255.255.255.255,路由器从不转发这种数据报,局际网络直接广播为指定网络ID,主机ID全部设为1,传送到目标网络F对直接广播为指定网络ID和子网ID,子网内的所有主机收到数据

- 4、对于C类地址,能够容纳主机个数太少,需要网络ID减少,这可以认 子网掩码是加长网络ID,而CIDR使其变短,同时指定网络掩码。 虽然B类地址,加上子网掩码,也类似C类地址缩短网络号,但是B CIDR不按照传统的IP地址分类,而是指定网络掩码,使得IP地址
- 5、为什么CIDR使得IP地址的分配更加有效?

举例来说,现在有3个1000G的硬盘,使用传统的方式,只能进行510个100G的分区,100个10G的分区,1000个1G的分区,分别对近但是在实际使用中,可能根本不需要10个100G的分区,但是没有过而CIDR彻底废弃了这种方法,无分类就是不考虑IP地址所属的类别根据实际的需求进行分区,比如第一个1000G的硬盘,可以分成25

6、更重要的一点是,CIDR这种组网技术可以将一组较小的无类别网络测减少Internet路由域中路由表条目的数量。

#### 【T03】理解私有地址和NAT

1、私有地址包括三块:

10.0.0.0 到 10.255.255.255

172.16.0.0 到 172.31.0.0

192.168.0.0 到 192.168.255.255

2、私有地址接入公网,需要进行NAT(Network Address Translat 出去进行SNAT,在NAT服务器修改Source 进来进行DNAT,在NAT服务器根据端口,修改Destination,也是

#### 【T04】开发并使用应用程序框架

1、TCP/IP应用程序分为

TCP服务器

TCP客户端

UDP服务器

UDP客户端

2、构建框架库是比较简单的一件事,主要就是对socket编程。

#### 【T05】套接字接口比XTI TLI更好用

1、用于网络编程的API接口有两种:

Berkeley套接字

XTL

- 2、套接字是加州大学伯克利分校为其Unix操作系统版本开发的,TLI是
- 3、二者通信能力没有差别,但是语义差别很大,套接字是更好的选择。

#### 【T06】记住TCP是一种流协议

- 1、TCP是一种流协议(stream protocol),这意味着数据是以字节接收端读取tcp数据,无法预知在这一次读操作中会返回多少个字量
- 2、假设主机A向主机B发送两条报文M1和M2,调用两次send发送两条独在发送端,send操作只是将数据复制到主机A的TCP/IP协议栈,目决定过程很复杂,取决于很多因素。如:发送窗口,拥塞窗口,路也就是说,主机A真正发送数据的情况有很多种:

M1和M2分开发送,M1和M2合并起来发送,M1先发送一部分,剩下 注意:发送窗口由对端控制,防止发送过多数据,导致接收端的缓 拥塞窗口自己控制,tcp要提供可靠的传输服务,因此具备超时重信 考虑,现在网络比较拥堵,超时没有收到回复,认为丢失,重传。 tcp发送策略的一个主要目标是:尽可能高效地利用网络带宽。因此 更好的策略是:不要再发送数据了,等到网络缓解的时候在发送。 拥塞窗口就是,记录当前网络能够容纳的流量。当前tcp能够发送能 也就是说,发送窗口允许发送很多数据,但是网络环境比较差,你

- 3、因此,接收端主机B不能对读取数据做任何假设,需要读几次,每次ì
- 4、对于定长报文来说,接收端读取到一定长度就认为是一个完整报文,
- 5、解决办法有两个:

办法一:报文的前面加一个首部,在固定位置标识这个报文的长度办法二:在报文中增加结束标记来分割报文,这个结束标记是一个规则的现象,如何表示结束标记本身呢?

需要使用转义字符。

#### 【T07】不要低估tcp的性能

- 1、tcp在ip的基础上增加了校验和、可靠性和流量控制的功能,而udp. 但事实不是这样,有时候tcp比udp的性能还要好。
- 2、思考,在什么情况下udp的性能要好于tcp,以及为什么?
- 3、这两种协议的cpu开销主要是进行数据拷贝和校验,二者区别不大。
- 4、tcp为了提供可靠的传输,会发送ack包进行确认,但是这个工作量之 其次,tcp没有必要对每个段都进行ack,可以间隔一段时间对多个
- 5、相对于ucp, tcp在建立连接三次握手,拆除连接四次握手,如果连挂连接的建立和拆除被分摊下来,对整个的性能影响很小。如果应用程序只是进行简短的请求应答回话,udp的性能好。如果应用程序持续时间比较长,并且进行大量的数据传输,tcp的情况。
- 6、客户端和服务端都是本地,tcp的性能要远远好于udp,这个奇怪的现在,这个奇怪的现在,这是不是环路udp的MTU是1500,对于环路的tcp的MTU可以达到而是直接放入接收队列。

#### 【T08】避免重新编写TCP

- 1、有时候为了所谓的性能,我们倾向于使用udp,但是我们又期望数据
- 2、可靠、健壮的udp必须提供:
  - a、在合理的时间内没有收到回复,进行重传

- b、保证应答和请求的匹配
- 3、基于udp协议,在应用层提供可靠性,需要做很多事情,比如:滑动。
- 4、因此,如果要可靠的网络传输,就直接使用tcp,而不是使用udp,?
  - a、应用程序实现的tcp功能肯定比不上真正的tcp协议,tcp协议
  - b、tcp协议的实现在内核中执行,性能更高。
  - c、有时候在应用层超时重传并不是用户期望的,比如客户端发送一 导致一次转账请求,在服务端处理了两次,这不是我们所期望 在应用层只收到一次请求。

因此,使用udp,在应用层提供可靠性并没有什么意义。

- 5、因此,udp的使用场景是:对可靠性的要求不是很高,允许丢包,丢 tcp的使用场景是:对可靠性的要求很高。
- 6、如果既想拥有tcp的可靠性,同时减少建立连接和拆除连接的开销,避免了建立连接的三次握手,同时缩短拆除连接时的TIME\_WAIT阶

#### 【T09】要认识到TCP是一个可靠的,但不是绝对可靠的协议

- 1、稍微想一下就知道,TCP不是绝对可靠的协议,比如:网络断开,主
- 2、考虑应用程序A向应用程序B发送数据的TCP流程,数据流从应用程序通过应用程序B所在主机的TCP/IP栈向上传输,最后到达应用程序特别注意:中间的路由器(即使拥有TCP/IP协议栈)并不涉及TCF也就是说,不会在路由器的TCP/IP协议栈向上传输,然后在向下停
- 3、考虑一种极端的情况,TCP确认了收到的数据,但是应用程序还没有 这种情况下,发送端认为接收端已经准确无误地收到了数据,但是 除此之外,还有其他一些的情况,
- 4、网络中断,如果没有备用的路径,数据传输一直失败,知道传输路径
- 5、对等实体崩溃,也就是对端的应用程序崩溃, 注意:对等实体崩溃与对等实体调用close(windows下调用close) 这种情况下,对等实体会发送fin包,表明没有办法再发送数据了。
- 6、对等实体的主机崩溃,这和对等实体崩溃不同,主机崩溃了,没法发

### 【T10】记住,TCP\_\_IP不是轮询的

- 1、TCP/IP协议栈无法将连接的丢失立即通知应用程序.
  TCP为什么不提供这种通知机制,不这么做的优点和缺点,应用程序。
- 2、TCP为什么不提供连接丢失即时通知的功能?
  - a、其他通信协议,比如SNA和X.25,在连接丢失的时候会通知应是他们的策略是轮询发送显示报文"有东西要发给我吗?"或者

这意味着要消耗一定的网络带宽。这是原因之一。

- b、还有哲学方面的考虑,上层协议不应该对下层协议做任何假设, 应用程序根据需求,来决定是否检测连接的丢失。
- c、还有一个重要的原因,和TCP/IP的主要设计目标有关:出现网TCP/IP的起源是美国国防部要求,出现战争或者自然灾害等也就是说,网络故障往往是暂时的,路由器会重新找到一条路这种暂时的连接丢失,再应用程序还没有意识到的时候就已经
- 3、如何检测连接的丢失呢?
- 4、TCP的保活机制,是为了检测长时间没有交互的死连接,并且丢弃这是TCP/IP协议栈运行在系统内核,独立于应用程序。如果对等应用程表明我不再向对端发送数据了。

如果对等应用程序所在的主机崩溃,运行在内核中的TCP/IP协议机如果到达对等主机,但是应用程序没有运行,内核中的TCP/IP协议

- 5、TCP的保活机制涉及到时间间隔,要求是至少2个小时的默认空闲时间 这就意味着TCP的保活机制要2个多小时以后才能检测到连接丢失。 这两个时间间隔可以修改,但是这种修改是全局的,会影响到所有如何时间间隔设置太短,就违背了它清除长时间死连接的最初目标。 另外,TCP保活机制不仅检测死连接,还丢弃这些连接,这往往不是
- 6、那么问题来了,应用程序如何检测连接的丢失呢?

在应用程序添加心跳,设置心跳的频率,已经多久收不到心跳信号

- 7、还有另外一种办法,思路是使用一条新的连接来发送心跳信号,也就
- 8、TCP没有提供连接丢失即时通知应用程序的功能,但是在应用程序可

#### 【T11】提防对等实体的不友好动作

- 1、好的软件应该能够处理各种可能的错误,也就是防御式编程。
- 2、作为服务端,应该检测客户端的终止,客户端的无效输入。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

 首页
 模块

 TCP
 TCP\_IP » 传输层

## 模块

拥塞控制

## 重变

测试tcp端口
TCP是一种流协议
TCP的标志位
糊涂窗口综合征以及解决办法
TCP的定时器
TCP知识点
出现RST的几种情况
TCP头部
TCP状态机
长连接和短连接

## 变量说明

#### TCP头部

- 1、前2个字节是源端口,接下来2字节是目的端口
- 2、接下来4字节表示发送序号,标识发送数据的第一个字节的序号。第一比如当前序号为7,并发送了4个字节,下一次发送的序号为11。 注意:这个序号,不同于IP头部中的报文唯一标识。另外,对于T( 但是,知道了第一个syn的发送序号,其它报文的相对序号也都可L
- 3、接下来4字节表示确认序号,告诉对端,已经接收到了第几个字节,身确认序号,由对端的发送序号和发送的数据长度确定。
- 比如:对端的发送序号为3,并且发送数据长度为4,我的确认序号4、接下来4个bit,表示TCP头部长度(单位是32bit),这个字节取值
- 4、接下来4个01t,表示TCP头部长度(单位是3201t),这个字节取值 注意TCP头部的最大长度是60个字节,,并且是4字节的倍数,为作 因为只有4个bit表示TCP头部长度,最大取值为15,也就是15\*4= 接下来6个bit是预留位,接下来6个bit是标志字段,分别为UAPR 单位是字节,从对端收到数据,缓冲区变小,应用程序取走数据, 当取值为0,表示缓冲区满了,对端不能再发送数据,否则接收数据。
- 5、接下来16个bit是TCP校验和,TCP校验和覆盖TCP头部和TCP数据,TCP校验和是必须的。接下来16个bit表示紧急指针,只有当URG标表示紧急数据最后一个字节的序号。紧急字段的使用场景是,当有接收端就可以优先处理紧急数据,紧急数据后面的数据是普通数据。
- 6、接下来是TCP的可选项。这部分最长40字节,因为TCP头部最长60字因为TCP头部中只有4个bit表示TCP头部长度(单位是32bit,也)常用的可选项如下:

kind=0

kind=1

kind=2length=4MSS取值(2字节)kind=3length=3移位数(1字节)

kind=4 length=2

kind=5 length=8\*N+2 第一块左边沿和右边沿 第二块左kind=8 length=10 时间戳(4字节) 时间戳回显

kind=0是选项表结束

kind=1是空操作(nop)选项,没有特殊意义,只是为了填充,使kind=2是MSS选项,只在syn包设置,三次握手双方协商MSS。

发送syn报的时候告诉对方,我期望接收的最大段长度是你别发太长,短一些没有问题。MSS一般是1460,因为而TCP最小的头部长度为20字节,因此1500-20(IP头UDP有效数据的最大长度是1500-20-8=1472字节TCP报文段在传输过程中,如果经过网络(路由器)的M

kind=3是窗口缩放因子,只在syn包设置。解决什么问题?

作为接收端,我告诉对方我的接收缓冲区还有多大,如紧这就是TCP头部中的接收窗口。但是这个字段是16bit,这个值很快就递减为0,导致对方不能发送数据,出现卡实际上我的接收缓冲区很大,甚至1个G,可以不停地接发送syn报的时候告诉对方,这个缩放因子(取值0~14在发数据交互的时候,我发送的接收通告窗口是64,那也就是256,也就是实际的接收窗口要扩大一下。

作为发送端,我通告的时候,要缩小一下,实际上我的技术,从,

- kind=4是否支持选择性确认选项,只在syn包设置。解决什么问题 比如我发了A、B、C、D四个报文段,你收了A、C、D三 按道理,你必须按顺序确认A、B、C、D,因此只会确认 超时,我没有收到B的确认,认为B、C、D都丢失了,B( 设置这个字段,告诉对方你可以选择性的确认。这样的i 那么我就只需要发送B就好了。
- kind=5是选择性确认,和kind=4结合使用。在交互的过程中,前可以一次性的选择确认多个,确认一个报文段需要8字节因此一次最多确认4个
- kind=8是时间戳选项,时间戳选项的作用有:

序号回绕,计算RTT。按道理数据报的序号是单调递增的我发送的时候带上当前时间,你Ack的时候,回显一下,注意:这不是没有个报文段的RTT,而是每个Ack的RTT另外,时间戳的时间单位可以是1毫秒到1秒,回显的一点

- 1、TCP是一种流协议,发送者以字节流的方式发给接收者,发送者的写 接收者无法预先得知在一次读操作中返回多少字节,接收的数据没
- 2、那么接收者如何识别收到一个完整的报文?

#### 两种解决办法:

- a、报文结尾增加结束标记,用于分割记录,需要注意的是:要表示
- b、报文头部增加字段,表示报文长度,读取到一定长度就认为是-

#### TCP状态机

#### 使用网络助手观察tcp状态机

8、客户端的网络连接

```
场景一:客户端主动断开连接
1、服务端使用12345端口监听,客户端去连接。
2、服务端的网络连接
   C:\Users\niu>netstat -ano|findstr 12345
        10.65.200.168:123450.0.0.0:010.65.200.168:1234510.36.65.80:34814
   TCP
   TCP
3、客户端的网络连接
   C:\Users\25697>netstat -ano|findstr 12345
         4、注意:netstat的输出最后一个字段是进程Id,前面一个端口是当前
   对于在一台机器上的输出,很有帮助,如下在Linux下:
   [root@localhost IBP]# netstat -anp|grep 9932
               0 0.0.0.0:9932
   tcp
            0
                                           (
                 0 172.16.2.102:9932
   tcp
   tcp 0 0 172.16.2.102:9932
         0 0 172.16.2.102:43061
0 0 172.16.2.102:43008
                  0 172.16.2.102:43061
   tcp
   tcp
5、这个时候抓包可以看到连接的三次握手。
6、然后客户端主动断开连接。
7、服务端的网络连接
   C:\Users\niu>netstat -ano|findstr 12345
   TCP
        10.65.200.168:12345  0.0.0.0:0
   TCP 10.65.200.168:12345 10.36.65.80:34814
```

C:\Users\25697>netstat -ano|findstr 12345

TCP 10.36.65.80:34814 10.65.200.168:12345

9、分析如下:

客户端主动断开连接,先发送fin包,进入FIN\_WAIT\_1状态,服制同时服务端进入CLOSE\_WAIT状态

按道理,这种情况下,服务端应该接着发送fin包,进入LAST\_ACI客户端收到fin包,回复ack(服务端收到ack进入CLOSED状态)

10、但是服务端并没有发送第二个fin包,这时候状态一直保持。 过了两分钟之后,处于半连接的客户端意识到这个问题,于是发送 注意:还有一种情况发送rst包,就是去连接没有监听的端口,对算

11、为什么服务端没有发送第二个fin包?

推测:发送fin包意味着我没有数据发给对方了,作为被动关闭的E

场景二:服务端主动断开连接

- 1、服务端主动断开连接,可以看到断开连接的四次握手。
- 2、服务端发送fin包,客户端回复ack,然后发送fin包,服务端回复a

#### TCP的定时器

1、tcp有四种定时器,分别解决不同的问题。

超时重传定时器

坚持定时器

保活定时器

2MSL定时器

2、超时重传定时器,也就是RTO(Retransmission Time Out)定时 我发送的数据,如果一段时间内没有收到对方的确认,我就重传。 注意:这个RTO是根据RTT(Round Trip Time),往返时间计算

3、坚持定时器,解决什么问题?

解决0窗口的问题,考虑接收端缓冲区满了,发送一个0窗口,发送现在应用程序从接收端的缓冲区读取了一部分数据,接收端就发送特别注意:这个确认是不包含数据的(窗口大小在tcp首部中),t那么问题来了,假如这个ack丢失了,接收端认为我已经告诉对方对我不能发送数据,就等着对方告诉新的窗口大小。由于这个ack不管怎么解决这个问题?

理论上,发送方认为对方的缓冲区不可能是一直满的。当收到一个时期,以上的时间没有告诉我新的窗口大小,我就主动向你去查询。 这个报文就是 探测报文段。

4、保活定时器,解决什么问题?

考虑tcp连接建好之后,处于长时间的空闲,假设客户端崩溃了, (注意:tcp不是轮询的,无法将连接的丢失立即通知应用程序) 解决办法就是:

定时发送心跳包,一段时间内没有收到确认,就认为连接出问题了注意:tcp保活机制的心跳包间隔比较大,检测到连接丢失延迟比较5、2MSL定时器,解决什么问题?

主动断开连接的一方,发送ack之后,如果立即进入closed状态。 会重新发送fin包,但是我已经closed了,无法再对这个fin进行 怎么解决这个问题?

站在我的角度考虑,发送ack之后,如果对方没有收到的ack,肯定则证明对方肯定收到了我的ack。

因此,发送ack之后,开启一个定时器,时间为2MSL(从我发送ac说明对方肯定收到了我的ack,我就可以放心地进入closed状态了

#### TCP的标志位

1,	使用wireshark抓包可以看到三次握手: syn syn ack ack
2、	使用tcpdump直接抓包查看,如下:     [root@localhost ~]# tcpdump -i any -s 0 tcp port tcpdump: WARNING: Promiscuous mode not supported tcpdump: verbose output suppressed, use -v or -vv listening on any, link-type LINUX_SLL (Linux cool 16:34:47.340784 IP 10.36.65.60.40343 > 10.36.65.8 E <ce@.@.@. \$a<="" \$ap091<="" td=""></ce@.@.@.>
	16:34:47.341712 IP 10.36.65.80.italk > 10.36.65.65.65.65.65.89  \$AP \$A<09P1

\$A< \$AP..091...P.....\8P..... .....? 可以看到第三次的标志有个点. 这个点表示syn fin rst psh i

#### TCP知识点

- 1、syn、fin占用一个序列号, ack不占用序列号
- 2、mss和窗口大小,是告诉对端,我能接收到的最大报文长度和还能接好 对端发送的报文长度超过mss,我不接收。对端发送的数据大小超证
- 3、发送fin,意味着我不再向你发送数据,但是我还可以接收你发过来
- 4、一个进程可以重新使用仍然处于2MSL等待的接口(设置套接字S0\_RI但是tcp不允许一个新的连接建立在相同的套接字对上。 比如服务端主动断开连接,处于2MSL的TIME\_WAIT状态,客户端记 但是新建立的连接,套接字对不同于上一次的套接字对。
- 5、ack的传输并不可靠,tcp只对那些包含有数据的ack报文段进行确认思考为什么? 如果对于没有数据的ack报文段也进行确认,那就变成了死循环,是 在建立连接的时候,三次握手中的最后一次ack,不需要对方确认。
- 6、现在思考另一个问题,为什么ack不占用序列号? 假如ack也占用一个序列号,根据tcp的超时重传特性,我的ack需 注意:ack可以单独发送,但是为了效率,往往是发送数据的时候

#### 出现RST的几种情况

- 1、端口未打开, C向S发送SYN, 去连接S的端口9820, 但是S没有打开!
- 2、请求超时,C向S发送SYN,S回复ACK+SYN,如果C从发送SYN到收到 C发送RST,表示拒绝进一步发送数据。
- 3、提前关闭连接,S端tcp协议收到的数据,应用程序没有接收,S发送
- 4、S端socket调用close不愿再接收数据,但是还是从C收到数据,发
- 5、在现场环境中,遇到一种情况发送rst包,特别注意一下,测试一下。
  - a、M70解码器和主控板在一个网络内,M70从业务板卡vtdu获取证但是,一旦有数据交互,就发现M70发送rst包。
  - b、原因是:从M70抓包发现,TCP的三次握手都是直接和vtdu的户一旦有数据交互,vtdu回复ack,IP地址就成了主控板的外际经过主控板设置了SNAT,源IP转换为主控板的外网IP,导致
  - c、怎么修改?

#### 强制M70经过主控板,去连接内网的vtdu。

#### 测试tcp端口

1、测试网络通不通,使用ping测试,使用traceroute(windows下扩

2、测试tcp端口是否打开,使用telnet ip port,ctrl+]退出,进

3、在windows下,输入telnet,报错telnet不是内部或外部命令,怎 需要开启telnet客户端,操作:控制面板-->程序-->程序和功能 相同位置还有Telnet服务器,在命令输入 services.msc进入服

注意: services.msc 进入服务管理 远程桌面

mstsc

进入事件查看器 eventvwr

4、另外windows下如何开启telnet服务。win7默认不开启telnet服:

a、控制面板-->程序-->程序和功能-->打开或关闭Windows功能

b、重新进入 services.msc找到telnet,启动。

c、启动失败,通过属性,找到依存关系,把依赖的组件启动,然后

#### 糊涂窗口综合征以及解决办法

1、糊涂窗口综合症是指当发送端应用进程产生数据很慢、或接收端应用 就会使应用进程间传送的报文段很小,特别是有效载荷很小。 极端情况下,有效载荷可能只有1个字节,传输开销有40字节(205)

2、怎么解决糊涂窗口综合征?

从接收端来考虑,当前缓冲区已满,应用程序取走一个字节,接收! 这样发送端就会发送一个字节,导致糊涂窗口综合征。解决办法很过 缓冲区还是偏移0个字节。当应用程序取走比较多的字节,积攒到一 这样发送端可以一次性发送多个字节。

但是,这样会导致发送端窗口收缩的问题。正常情况下,发送端可 考虑下面的情况,接收端期望接收第10000个字节,缓冲区满了, 然后应用程序取走1025个字节,通知发送端[10000,11025],发 按道理应该通知发送端[11024,11025],但是为了防止发送端发ì 站在发送端的角度来看,上一次为[10000,11025],这一次为[1] 这就是发送端的发送窗口收缩。

3、注意:在实际情况下,为了解决糊涂窗口综合征,发送端也会进行相 等到积累到一定程度我再发送。

#### 长连接和短连接

- 1、长连接和短连接的使用场景不一样: 长连接建立连接后,开始频繁地交互。 短连接建立连接后,交互一下子,下一次交互不知道什么时候。
- 2、tcp连接一旦长时间不交互,连接就会断掉。那么,对于长连接怎么一使用tcp的保活机制,keep-alive, 当然也可以在应用层处理,定时发送心跳包。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

## 重变

## 拥塞窗口

Nagle算法

### 变量说明

#### Nagle算法

- 1、发送的数据都要加上协议头部,协议头部占用一部分带宽。为了有效 考虑极端的情况,每次发送一个字节,头部是40个字节,净负荷才
- 2、Nagle算法就是为了解决这个问题:当网络上存在一个没有被确认的
- 3、怎么理解这句话?
  - a、网络上存在没有被确认的大段,不需要考虑,因为大段本身就需
  - b、网络上所有的段都被确认了,可以直接发送。
  - c、当前是一个大段,可以直接发送,因为净负荷已经比较高了。
  - d、当前是一个fin包,立即发送
  - e、当前是一个小段,不发送,再积累一些,然后一起发送。

#### 拥塞窗口

- 1、考虑下面的场景,tcp传输中出现拥塞。因此tcp超时重传,越重传动 网络传输的原则是:尽量多地传输数据。如果不拥塞,就增加数据
- 2、如何解决这个问题?

发送端预设值一个大小值,表示当前网络的拥塞程度,这就是拥塞这个窗口(和发送窗口)影响发送端发送数据的数量,当前不拥塞

- 3、刚开始不知道网络的拥塞程度,怎么办呢?
  - 先假设一个拥塞窗口(慢启动门限),然后采用试探的方式。发1个通过这样的方式,发现合适的拥塞窗口。这就是慢启动算法,需要
- 4、当到达慢启动门限时,这个时候就不能指数增长了。而应该使用线性 一旦出现拥塞,设为原来的一半。

也就是说,当接近网络吞吐量的时候,一点一点增加。拥塞了,并相对于慢启动,这就是快恢复。

5、考虑另外一种场景:

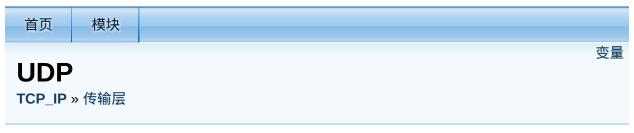
发送端发送了m1, m2, m3, m4四个报文, 而接收端收到了m1, m2但是发送端并不知道, 还会接着发送m5, m6, m7...

这种情况下,m3会重发,并且m3之后的m4, m5, m6, m7也要求重怎么解决这个问题?

6、接收端回复说,我才收到m2,我才收到m2,我才收到m2,重要的事ì

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



# 重变

## UDP头部

测试udp端口

### 变量说明

#### UDP头部

- 1、前16个bit,表示源端口,接下来16个bit表示目的端口
- 2、接下来16个bit,表示UDP数据长度,这个UDP数据长度,包括UDP的UDP有效数据长度,可以使用IP数据长度,减去IP头部(20字节)也可以使用UDP数据长度,减去UDP头部长度(8字节)。接下来16个bit是UDP的校验和,注意IP校验和只覆盖IP头部,并以及UCP的伪头部,也就是IP头部的前12个字节。注意:UDP的检验

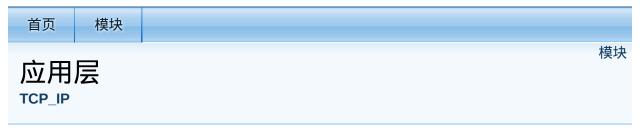
#### 测试udp端口

- 1、使用telnet可以测试tcp端口,不能测试udp端口,因为telnet是
- 2、linux下使用nc,如下:

nc -uv 10.36.65.80 8084 就可以和10.36.65.80,8084交互

- 3、如果只是测试 10.36.65.80 8084端口是否打开,可以使用nc -uzv 10.36.65.80 8084
  - 注:u[使用udp] z[不发送数据] v[verbose 详细信息]
- 4、需要注意的是,nc不仅能够测试udp端口,还可以测试tcp端口,选定nc-v 10.36.65.80 8084
- 5、windows下好像没有好的工具,测试对端的udp端口。

# Andy Niu Help 1.0.0.0



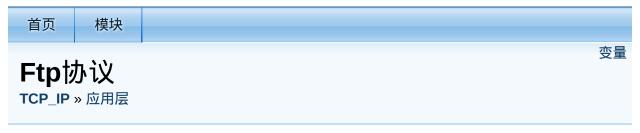
# 模块

## Ftp协议

## 详细描述

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 变量

<b>Windows</b>	下搭建ft	p服务
----------------	-------	-----

Linux下搭建ftp服务

使用ftp客户端命令

ftp协议抓包

访问ftp服务

ftp的客户端命令

### 变量说明

#### ftp协议抓包

```
1、在地址栏输入ftp服务,抓包可以发现交互过程。
2、ftp服务的监听端口是21,客户端连接端口21,走控制命令。
3、数据流的传输有两种方式,站在ftp服务端的角度来看,分为主动和
   主动:客户端开一个端口,ftp服务去连接(使用端口20),然后。
   被动:服务端开一个端口,客户端去连接,然后发送数据。
   注意:主动情况下,ftp服务使用端口20去连接客户端。
   主动连接的端口一般是随机生成的,但是也可以设置,只不过我们
4、被动方式的抓包如下:
   220 Microsoft FTP Service
   USER 25697
   331 Password required for 25697.
   PASS Dh @201569
   230 User 25697 logged in.
   SYST
   215 Windows NT
   PWD
   257 "/25697" is current directory.
   TYPE I
   200 Type set to I.
   PASV
   227 Entering Passive Mode (10, 30, 1, 74, 14, 198).
   SIZE /25697
   550 /25697: The system cannot find the file speci
   PASV
   227 Entering Passive Mode (10,30,1,74,14,199).
   CWD /25697/
```

```
LIST -l
    125 Data connection already open; Transfer starti
    226 Transfer complete.
    QUIT
    221
    客户端的监听端口是 256*14+199=3783 , 在3783端口可以看到
5、特别注意:这里ftp的信令连接和数据连接都是短连接,执行完之后
    使用tcp.flags.syn==1 or tcp.flags.fin==1 显示过滤,
ftp的客户端命令
1、连接如下:
    [root@localhost aaa]# ftp
    ftp> open 10.22.4.150
    Connected to 10.22.4.150 (10.22.4.150).
    220 Serv-U FTP Server v15.1 ready...
    Name (10.22.4.150:root): user
    331 User name okay, need password.
    Password:
    230 User logged in, proceed.
    Remote system type is UNIX.
    Using binary mode to transfer files.
2、文件列表
    ftp> ls
    227 Entering Passive Mode (10, 22, 4, 150, 14, 214)
    150 Opening ASCII mode data connection for /bin/]
                 1 user
    drwxrwxrwx
                                            0 Mar 30
                            group
                                            0 Jun
                                                  1
    drwxrwxrwx
                 1 user
                            group
                                           0 May 12
    drwxrwxrwx
                1 user
                            group
                                            0 Jan 20
                 1 user
    drwxrwxrwx
                            group
                                           0 Aug 15
    drwxrwxrwx
                1 user
                            group
                                           0 Mar 7
    drwxrwxrwx 1 user
                            group
    drwxrwxrwx
                 1 user
                                            0 Jan 26
                            group
    drwxrwxrwx
                 1 user
                            group
                                           0 Apr 15
```

0 Jan

250 CWD command successful.

drwxrwxrwx

3、下载文件,如下:

1 user

group

```
ftp> !ls
   111.txt
   ftp> get readme.txt
   local: readme.txt remote: readme.txt
   227 Entering Passive Mode (10, 22, 4, 150, 14, 216)
   150 Opening BINARY mode data connection for readm
   226 Transfer complete. 64 bytes transferred. 0.06
   64 bytes received in 0.000989 secs (64.71 Kbytes/
   ftp> !ls
   111.txt readme.txt
4、cd切换远程ftp服务器的路径,如果切换本机的路径怎么办?
   使用lcd
5、ftp客户端有些命令和linux中的命令冲突,比如ls
   1s是列出ftp服务器上面文件列表,如果列出本机的文件列表,使
   其他的命令 dir也是类似。
6、ftp帮助命令
   ftp> help
   Commands may be abbreviated. Commands are:
    ļ
                   debug
                                   mdir
   $
                   dir
                                   mget
                   disconnect
                                   mkdir
   account
   append
                   exit
                                   mls
   ascii
                   form
                                   mode
   bell
                                   modtime
                   get
   binary
                   glob
                                   mput
                   hash
   bye
                                   newer
                   help
   case
                                   nmap
                   idle
   cd
                                   nlist
   cdup
                   image
                                   ntrans
   chmod
                   lcd
                                   open
   close
                   1s
                                   prompt
                   macdef
                                   passive
   cr
   delete
                   mdelete
                                   proxy
   ftp> help get
```

get receive file

7、mget \* 用于获取多个文件,这个时候需要逐个确认。如果不想确认

#### Linux下搭建ftp服务

1、linux系统下常用的ftp软件是vsftp(Very Security File T [root@localhost vru]# rpm -qa|grep vsftpd vsftpd-2.2.2-11.el6\_3.1.i686 如果没有安装,使用yum安装,如下: yum install vsftpd -y

2、vsftpd作为服务,和其它类似,操作命令如下:
 [root@localhost vru]# service vsftpd
 Usage: /etc/init.d/vsftpd {start|stop|restart|try
 [root@localhost vru]# service vsftpd status
 vsftpd is stopped

- 3、vsftp提供3种远程的登录方式:
  - a、匿名登录方式

就是不需要用户名,密码。就能登录到服务器电脑里面

b、本地用户方式

需要帐户名和密码才能登录。而且,这个帐户名和密码,都是 c、虚拟用户方式

同样需要用户名和密码才能登录。但是和上面的区别就是,这 4、vsftpd使用黑名单策略,以邮箱为例,简单说就是:

白名单就是愿意接收的邮箱地址,黑名单就是拒绝接收的邮箱地址。

5、ftp的配置文件主要有三个,位于/etc/vsftpd/目录下,分别是:

ftpusers 指定哪些用户不能访问FTP服务器

user\_list 指定的用户是否可以访问ftp服务器由vsftr

默认\_userlist\_deny=YES (default),

vsftpd.conf 主配置文件

6、vsftpd启动之后,在windows资源管理访问ftp失败,抓包看到报知 not found: directory given in 'secure\_chroot\_dir'解决办法:

创建目录 /usr/share/empty

同时在vsftpd.conf中,增加配置选项 secure\_chroot\_dir=. 然后重启vsftpd

7、在Windows资源管理器中打开FTP站点,ftp://172.16.2.31:18 然后输入用户名和密码,就是登录linux的用户名和密码。 这个时候进入root的用户目录,也就是/root

8、在linux的/root目录下增加一个软连接,连接到 /usr/local/definition [root@localhost ~]# ln -fs /usr/local/datafs/IBP/[root@localhost ~]# ll total 4

-rw----- 1 root root 1867 2015-01-09 22:53 anaclrwxrwxrwx 1 root root 26 2016-08-15 16:33 ivss

9、在Windows资源管理器访问ivss目录 ftp://172.16.2.31:180 就可以管理ivss目录中的文件

#### Windows下搭建ftp服务

- 1、创建一个用户,用于访问ftp服务。 我的电脑右键->管理->本地用户和组->用户->"右键"新建用户niu niu
- 2、创建ftp服务的目录, E:\FtpDirectory 共享目录E:\FtpDirectory,使用户niu具备读写权限。 注:如果不共享,ftp客户端niu登录不能访问。
- 3、打开ftp服务的功能 控制面板(快捷键 ctrl+r 输入control) -->程序-->程序和功 选中FTP服务,然后确定。
- 4、打开IIS管理控制台,用于配置ftp服务器 控制面板(快捷键 ctrl+r 输入control)-->程序-->程序和I 选中IIS管理控制台,然后确定
- 5、配置ftp服务器 控制面板-->系统和安全-->管理工具,找到 Internet 信息服务 双击打开,找到左侧的网站,右击,添加FTP站点 输入站点名称,填写内容目录的物理路径 E:\FtpDirectory 下一步 SSL选无
- 下一步 选上匿名 基本 允许访问:所有用户 权限:读取 写入 6、访问 ftp://10.36.65.80/,输入用户名和密码

#### 使用ftp客户端命令

C:\Users\25697>ftp ftp> open 10.30.1.74 连接到 10.30.1.74。

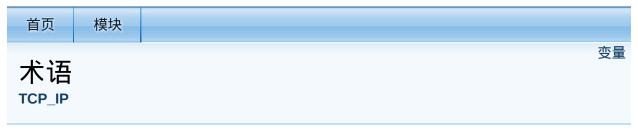
```
220 Microsoft FTP Service
用户(10.30.1.74:(none)): 25697
331 Password required for 25697.
密码:
230 User 25697 logged in.
ftp> help
                命令为:
命令可能是缩写的。
Ţ
               delete
                              literal
                                             promp
?
               debug
                              ls
                                             put
               dir
                              mdelete
append
                                             pwd
ascii
               disconnect
                              mdir
                                             quit
bell
                              mget
               get
                                             quot€
binary
               qlob
                              mkdir
                                             recv
               hash
                              mls
bye
                                             remot
               help
cd
                              mput
                                             renan
close
               lcd
                              open
                                             rmdir
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
229_250808_General_NetSDK_Chn_Win32_IS_V3.43.8.R.1516
ddd.pcap
python socket编程详细介绍 - Just for you - 博客频道 - CS
python 多线程就这么简单 - 虫师 - 博客园.mht
Python补充06 Python之道 - Vamei - 博客园.mht
Python命名空间的本质 - windlaughing - 博客园.mht
Python命名空间详解_python_脚本之家.mht
python中pyc和pyo的作用 - 常城的专栏 - 博客频道 - CSDN_NET
在路由器后面, traceroute返回一系列原因分析 - 向东博客 专注WE
构架之美,在于尽态极妍 应用之美,在于药到病除 - 赢在IT,Play
226 Transfer complete.
ftp: 收到 596 字节,用时 0.00秒 149.00千字节/秒。
ftp> quit
221
```

#### 访问ftp服务

- 1、浏览器本身是支持ftp协议的,可以作为ftp的客户端使用。
- 2、当我们使用IE浏览器登陆ftp服务的时候,会显示下面的内容: 若要在 Windows 资源管理器中查看此 FTP 站点,请单击"页面"
- 3、也就是说,我们可以 在Windows资源管理器中访问ftp服务的文件 资源管理器对应的就是上面的Windows地址栏,下面的叫任务栏,
- 4、从IE浏览器转到地址栏访问ftp服务器,需要设置如下:
  - a、工具-->Internet选项-->高级-->启动ftp文件夹视图(在Ib、按照页面提示 若要在 Windows 资源管理器中查看此 FTP 或点击页面,然后点击在 Windows 资源管理器中打开 FTP 即可在资源管理器(也就是地址栏)中查看ftp站点。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

TTL			
PDU			
MTU			
MSS			
MSL			

### 详细描述

### 变量说明

#### **MSL**

MSL(Maximum Segment Lifetime):最大报文段生存时间,TCP当主动断开连接的是C,被动断开连接的是S,假设C回一个ack立即进入cl站在C的角度,如果S没有收到ack,那么S肯定会再次发fin,从C发sck可以认为S肯定收到了我的ack,那么就closed MSL的时间一般为2分钟。

参见

#### **MSS**

MSS(Maximum Segment Size):最大分段的大小,这个是TCP连接约定好最大的段大小,在发送的时候,不超过这个段大小,就是MSS。这确定一个更合理的值。

参见

#### **MTU**

MTU(Maximum Transmission Unit):最大传输单元,链路层以为1、为什么最小是46个字节?

以太网的错误主要是发生碰撞,是指两台主机同时检测到网络空闲在发送数据的时候,收到碰撞信号,重传数据。如果数据已经发送数据还没有发送完毕,这就要求数据必须超过一定的长度。考虑两再减去4个字节(以太网帧尾部),也就是46个字节。

2、对于TCP,最大的净负荷是1500-20(IP头部)-20(TCP头部)=13、对于UDP,最大的净负荷是1500-20(IP头部)-8 (UDP头部)=1

参见

#### **PDU**

PDU(Protocol Data Unit):协议数据单元,在物理层PDU是数据在应用层PDU是报文(message)

参见

#### TTL

TTL(Time To Live):生存时间,数据包存活时间,定义是以秒为单位

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

## 模块

ICMP

## 重变

_	
	各种Ip地址
	IP选路
	路由器有关
	路由打印
	IP头部

### 变量说明

#### IP头部

- 1、第1个字节是45,前面4个bit取值为4,表示IPv4,后面4个bit取1 特别注意:这里的单位是32bit,也就是说单位是4字节,5个单位 第2个字节是服务类型字段,3-4字节表示总长度,可表示的最大长 减去IP头长度[20字节],减去TCP头长度[20字节固定长度+可选]
- 2、接下来是16位标识,用于标识一个报文,可以看到是增长上去的。接
- 3、接下来8位生存时间TTL,8位协议字段,常用的协议有 1[ICMP]Internet控制报文协议 6[TCP] 17[UDP] 45[IDRP] 接下来16位表示IP头部校验和,注意:这里检验和只是对IP头部边
- 4、接下来32位(4个字节)表示源IP地址
- 5、接下来32位(4个字节)表示目的IP地址,总共20字节。

#### IP选路

#### 1、IP选路原理

- a、搜索路由表,寻找能与目的IP地址完全匹配的表目(网络号和:如果找到,则把报文发送给该表目指定的下一站路由器或直接
- b、搜索路由表,寻找能与目的网络号相匹配的表目。 如果找到,则把报文发送给该表目指定的下一站路由器或直接 目的网络上的所有主机都可以通过这个表目来处置。例如,-
- c、搜索路由表,寻找标为"默认( d e f a u l t)"的表目。
- d、如果上面这些步骤都没有成功,那么该数据报就不能被传送。 如果不能传送的数据报来自本机,那么一般会向生成数据报的 注意:可以认为查找的顺序是从小范围到大范围。
- 2、对于一个给定的路由器,可以打印出五种不同的标志(flag) U 该路由可以使用

- G 该路由是到一个网关(路由器)。如果没有设置该标志,说明目
- H 该路由是到一个主机,也就是说,目的地址是一个完整的主机地如果没有设置该标志,说明该路由是到一个网络,而目的地址
- D 该路由是由重定向报文创建的
- M 该路由已被重定向报文修改

#### 3、注意:

G是区分下一跳的,没有G是直接路由,有G是间接路由。直接路由i间接路由说明下一跳是路由器,分组中包含最终的目标IP地址和下H是区分目标地址完整程度的,没有H说明目标是一个网络,有H说FG和H二者是不互斥的,可以各种组合。

默认路由,全部为0,也就是0.0.0,0,这个路由将匹配剩余所有£4、在不同的协议层,对于传输的数据,叫法不一样。

在链路层以太网 称为帧

在网络层IP 称为IP数据报 在传输层UDP 称为UDP数据报 在传输层TCP 称为TCP报文段

计算机网络不能一次性传输任意长的数据,需要把数据分割成小块

#### 各种Ip地址

#### 127.x.x.x

- 1、127.x.x.x是环回接口(Loopback interface),包括从127.6 ping 127.0.0.1到127.255.255.254结果都是一样的,它们是
- 2、环回接口的原理是:

正常的数据包发送接收流程:发送数据包从传输层到网络IP层,再然后从物理层收到数据包,传给数据链路层,再传给Ip层,放入IF而向环回接口(也就是127.x.x.x)发送数据包,在IP层短路,质直接放入IP的接收队列中,然后接收数据包就是,从IP的接收队列

- 3、也就是说,环回接口发送数据包,在IP层短路,直接放入IP的接收队
- 4、除了127.x.x.x.x.,还有两种情况,数据包的发送直接放入IP的接收 a、UDP的广播和多播,因为广播和多播的定义包含主机本身,对于 同时向下发给数据链路层,广播或者多播出去。
  - b、目的地址是本机地址,比如本机地址是10.36.65.80,向10.
- 5、对于环回接口,在Linux上使用tcpdump可以抓到相应的包,在Win 这是因为Linux系统的TCP/IP栈了实现环回接口,而Windows系约 怎么解决在Windows上使用wireshark抓不到数据包?
- 6、在Windows上使用wireshark抓不到数据包,是因为直接放入IP的

对于目标地址127.x.x.x,肯定不能抓到。

而对于目标地址10.36.65.80, IP层会有一个判断过程, 到达10如果没有设置, 就是本机地址, 直接放入IP的接收队列中。

因此,解决办法就是,到达10.36.65.80强制指定一条路线,经过route add 10.36.65.80 mask 255.255.255.255 10.36. (注意:对于127.0.0.2使用这种方式设置,也是抓不到包的,医这样的话,IP层就必须按照指定的路线将数据包向下发出去。

流程是:数据包发出去,经过路由器,再转发回来。也就是说,在 这样就能抓到数据包了。

0.0.0.0

1、解决什么问题?

主机可能有多个IP地址(含有多个网卡的情况),IP地址可能会发客户端可能在远端,通过主机IP地址10.36.65.80来连接,也有可能

2、那么问题来了?

当前应用程序在哪个IP地址监听呢?

- 3、如果固定下来IP地址,就会导致其他的情况连接失败。
  - 0.0.0.0就是为了解决这个问题,表示监听主机的所有IP地址(不
- 4、同一个端口可以监听多次,只要使用的IP地址不同,比如,当前主机 10.36.65.80:8080

127.0.0.1:8080

0.0.0.0:8080

#### 私网地址

1、私网地址如下:

10.0.0.0/8: 10.0.0.0~10.255.255.255 A类和 172.16.0.0/12: 172.16.0.0~172.31.255.255

192.168.0.0/16: 192.168.0.0~192.168.255.255 C类利

2、特别注意:私网的ABC类地址,与互联网的ABC类地址,子网掩码表证 互联网的ABC类地址子网掩码分别是8,16,24,而私网的ABC类地址 它们的子网掩码不存在必然的关系。可以这样理解:

互联网的ABC类地址是分别从0.0.0.0-255.255.255.255中挖出对于挖出来的一块空间(前缀标识),使用不同的子网掩码,标识而私网地址,从互联网的ABC类地址空间中,分别挖出一块,专门作品。

3、另外, 169.254.x.x 则主要是分配给DHCP服务使用的

#### 广播地址

1、255.255.255.255是全局广播地址

- 2、主机号全部为1的地址是子网广播地址,如:192.168.1.255
- 3、主机号全部为0的地址是子网的网络地址,代表子网本身,而不是一台

#### 路由器有关

- 1、一个网络内的设备通过交换机连接在一起,直接通信,交换机在内存 (注意:集线器是转发给所有的设备,因此交换机更有效率)
- 2、交换机只能对同一个网络的数据包进行转发,当与另一个网络进行通
- 3、需要一个路由器。这个路由器一般有两个网卡,一个网卡连接当前网 在路由器上根据Ip地址设置网卡的转发功能,这就实现了一个网络
- 4、要完成当前网络到另一个网络的通信,还需要对当前网络的设备设置把默认网关设置为当前网络上的路由器,注意:设置的路由必须是对于同一个网络的通信,不需要设置路由。因为一个网络接口,也是
- 5、在两个网络中间搭建一个路由器,就可以实现两个网络的通信。通过加上一个路由器就是直接的通信,加上多个路由器就是间接的通信。 当然路由器也可以提供3个或者以上的网卡,分别连接多个网络,标
- 6、路由功能可以在硬件实现,比如思科,华为,也可以在软件实现,Li Linux内核通过route命令添加路由,用于数据包的转发。另外,是

#### 路由打印

1、路由打印如下:

[root@localhost lib]# route -n

Kernel IP routing table

= -:						
Destination	Gateway	Genmask	F			
10.22.4.0	0.0.0.0	255.255.255.0	l			
169.254.0.0	0.0.0.0	255.255.0.0	l			
0.0.0.0	10.22.4.1	0.0.0.0	Į			

2、前面三列很好理解,Genmask是子网掩码。

IP选路的策略是:先匹配主机,再匹配网络;先找小范围,再找大

- 3、Flags的含义如下:
  - U该路由可以使用
  - G 该路由是到一个网关(路由器)。如果没有设置该标志,说明目
  - H 该路由是到一个主机,也就是说,目的地址是一个完整的主机地如果没有设置该标志,说明该路由是到一个网络,而目的地址
  - D 该路由是由重定向报文创建的
  - M 该路由已被重定向报文修改

注意:D在什么情况下会产生?

主机向路由器A,发送消息,路由器A通过ICMP重定向差错,告诉34、Metric的含义:

表示一组参数,包括带宽,跳数,负载等,可以理解为路由成本,

#### 5、Ref的含义:

正在使用该路由的进程个数。特别注意,使用route -n看不到结影比如,新建一个telnet连接,可以看到Ref的值增加1,如下:

[root@localhost lib]# route -nC|grep -v lo

Kernel IP routing cache

Source	Destination	Gateway
10.22.4.151	10.36.65.80	10.22.4.1
10.22.4.151	10.36.65.80	10.22.4.1
10.22.4.151	10.36.65.84	10.22.4.1
10.22.4.151	10.36.65.95	10.22.4.1

#### 6、Use的含义:

通过该路由发送的分组个数,使用ping命令,可以看到Use值的增[root@localhost lib]# route -nC|grep -v lo

Kernel IP routing cache

Source	Destination	Gateway
10.22.4.151	10.36.65.80	10.22.4.1
10.22.4.151	10.36.65.80	10.22.4.1

### 7、Windows下的路由打印,如下:

IPv4 路由表

#### 活动路由:

各	目标	网络	<b>播码</b>		网关	接口	跃点数
		0.0.0.0		0.0.	0.0	10.36	.65.1
	10	.36.65.0	255.2	55.25	5.0	7	在链路上
	10.	36.65.80	255.255	.255.	255	10.36	.65.1
	10.	36.65.80	255.255	.255.	255	10.36	.65.2
	10.3	6.65.255	255.255	.255.	255	7	在链路上
	1	27.0.0.0	2	55.0.	0.0	7	在链路上
	1	27.0.0.1	255.255	.255.	255	7	在链路上
12	27.255	.255.255	255.255	.255.	255	7	在链路上
	2	24.0.0.0	2	40.0.	0.0	7	在链路上
	2	24.0.0.0	2	40.0.	0.0	7	在链路上
25	55.255	.255.255	255.255	.255.	255	7	在链路上

255.255.255.255 255.255.255 在链路上 8、分别介绍如下: 缺省路由 0.0.0.0 直连网段的路由 10.36.65.0 10.36.65.80 本地主机路由 10.36.65.255 本地广播路由 本地环路 127.0.0.0 组播路由 224.0.0.0 广播路由 255.255.255.255 注意:组播路由和广播路由都是两条记录,是因为有环路127.0.0

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

## ICMP说明

重定向到更好的路由

### 详细描述

## 变量说明

#### ICMP说明

ICMP(Internet Control Message Protocol): Internet控制 ping 和 traceroute 是使用ICMP实现的,消息类型echo request

#### 参见

#### 重定向到更好的路由

使用ICMP可以为主机指向更好的路由,考虑A向B发数据,A根据子网掩码下这个时候只配置了R1,于是发给R1,R1检查自己的路由,发现下一跳于是R1发一个ICMP Redirect数据包(包含R2的IP地址)给A,告诉A

#### 注意:

- 1、假设A配置的多个路由,A查找的是顺序是从小范围到大范围。
- 2、路由转发过程中,在网络层,源IP和目标IP不变,在链路层,源MA(也就是说,在链路层,两个MAC地址连通,才能发送数据。路由器根据目标IP找到下一跳的IP地址,根据ARP找到下一跳的M/

#### 参见

# Andy Niu Help 1.0.0.0

## 量变

以太网头部

ARP

RARP

## 详细描述

## 变量说明

#### **ARP**

ARP(Address Resolution Protocol):地址解析协议,作用是根基

#### **RARP**

RARP(Reverse Address Resolution Protocol):反向地址解析

#### 以太网头部

- 1、以太网头部长度是14字节,前6个字节是目的mac地址,接着的6个字 注:以太网网头部是目的mac在前,源mac地址在后。这一点和IP IP头部和TCP头部,都是源在前面,目的在后面。
- 2、常用的协议类型有:

0x0800[IPv4] IP数据报长度[46~1500字节]

0x0806[ARP]请求回复[28字节]PAD[18字节]0x0835[RARP]请求回复[28字节]PAD[18字节]

注:ARP是地址解析协议,根据Ip地址获取mac地址,RARP是逆地

- 3、在windows下使用wireshark抓包,空负荷的TCP包长度为66字节 而在linux下,使用tcpdump抓包保存下来,可以看到空负荷的T( tcpdump多出来的两个字节在源mac地址之后,取值为00 00, 对干目的mac地址,主动连接的一方取值固定为00 00 00 01 00
- 4、为什么这样?

这是因为包是在linux中使用tcpdump抓包,指定参数-i any来打 它会把所有包的以太网头都换成linux cooked capture, wire

- 5、如果tcpdump使用 -i eth0来抓包,就会显示和windows一样。
- 6、以太网尾部还有4字节,是CRC,用于数据校验。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

Unix环境高级编程

## 重变

文件描述符FD

## 变量说明

#### 文件描述符FD

```
1、文件描述符是针对进程的,也就是说一个进程维护一组文件描述符FL
  因为按照惯例0,1,2分别是标准输入,标准输出,错误输出。
```

```
2、一个进程中的FD各不相同,每次打开一个文件(即使是同一个文件)
   代码:
   #include <stdio.h>
   #include <fcntl.h>
   int main(int argc, char* argv[])
   {
           int fd1 = open("./aaa.txt", O_RDONLY);
           int fd2 = open("./aaa.txt", 0 RDONLY);
           printf("fd1[%d], fd2[%d]\n", fd1, fd2);
           getchar();
           return 0;
   测试:
    [root@localhost FD]# g++ -o main main.cpp
    [root@localhost FD]# ./main
   fd1[3], fd2[4]
```

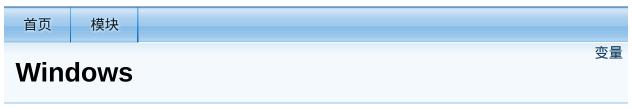
- 3、文件描述符是针对进程的,一个进程中FD不会重复,但是不同进程中
- 4、每个进程包含一个FD列表,每个FD包含文件描述符标志和一个指向了 目前FD标志只有一个close-on-exec标志,取值0表示fork子进 系统默认是0
- 5、内核维护一个文件表的列表,对应于所有打开的文件。文件表包含文 文件状态标志主要有读,写,添加,同步,阻塞等。
- 6、复制FD,使用dup方法。

dup复制文件描述符,返回一个新的FD,新FD与老FD的FD标志和了7、注意以下单词的区别:

duplicate和replicate都是复制的意思,但是二者的侧重点不du是double的意思,成双,复制一个一模一样的东西。re是redo的意思,重做,对于做过的事情重新做一遍。dump的原意是倾倒垃圾,在计算机中指把一个存储器中的内容转移保存到文件中。在mysql中dump是转存,也就是备份数据。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0



## 重变

粘贴变成灰色,原来是飞秋的问题
хсору
rename
dir
Windows和Linux下面的线程睡眠
批处理递归删除指定的目录或者文件
findstr
访问共享目录的注意事项
echo
延迟变量
批处理注释
批处理百分号
其它常用的批处理
批处理for循环
tasklist
taskkill

### 详细描述

## 变量说明

#### dir

```
1、查看帮助,如下:
   D:\test>dir /?
   显示目录中的文件和子目录列表。
   DIR [drive:][path][filename] [/A[[:]attributes]]
   [/0[[:]sortorder]] [/P] [/Q] [/R] [/S] [/T[[:]tin
   [drive:][path][filename]
            指定要列出的驱动器、目录和/或文件。
            显示具有指定属性的文件。
   /A
   属性
            D 目录
                             R 只读文件
            H 隐藏文件
                            A 准备存档的文件
            S 系统文件
                              无内容索引文件
              解析点
                              表示"否"的前缀
            使用空格式(没有标题信息或摘要)。
   /B
            在文件大小中显示千位数分隔符。这是默认值。用
   /C
            禁用分隔符显示。
            跟宽式相同,但文件是按栏分类列出的。
   /D
            用小写。
   /L
            新的长列表格式,其中文件名在最右边。
   /N
            用分类顺序列出文件。
   /0
   排列顺序
           N 按名称(字母顺序)
                            S 按大小(从小到大
            E 按扩展名(字母顺序) D 按日期/时间(从
              组目录优先
                            - 反转顺序的前缀
            在每个信息屏幕后暂停。
   /P
```

```
显示文件所有者。
   /Q
   /R
             显示文件的备用数据流。
             显示指定目录和所有子目录中的文件。
   /S
             控制显示或用来分类的时间字符域。
   /T
   时间段
            C 创建时间
             A 上次访问时间
             W 上次写入的时间
             用宽列表格式。
   /W
             显示为非 8.3 文件名产生的短名称。格式是 /N
   /X
             短名称插在长名称前面。如果没有短名称,在其位
             显示空白。
   /4
             用四位数字显示年
   可以在 DIRCMD 环境变量中预先设定开关。通过
2、列出所有目录和文件的名称,如下:
   D:\test>dir /b
   aaa
   bbb
   dmu1.cpp
   dmu2.cpp
   dmu2.h
   file1
   log1
   log2
   test.bat
3、只列出目录,如下:
   D:\test>dir /b /a:d
   aaa
   bbb
4、只列出文件,如下:
   D:\test>dir /b /a:-d
   dmu1.cpp
   dmu2.cpp
   dmu2.h
   file1
   log1
   log2
```

```
test.bat
5、只列出前缀为log的文件,如下:
D:\test>dir /b /a:-d log*
log1
log2
```

#### 参见

#### echo

```
1、echo回声,echo什么,返回什么。
2、echo hello, 结果是:
    echo hello
    hello
3、如果我不想显示echo hello,怎么办?使用:
    echo off
    echo hello
4、3的显示结果:
    echo off
    hello
    如果我不想显示echo off,怎么办?
    @echo off
    hello
    hello
    metho off
    hello
```

#### findstr

- 1、查找满足条件的行
  - . 匹配任意一个字符
  - \* 前面的字符出现0次或多次
  - ^ 匹配行头
  - \$ 匹配行尾
  - /i 不区分大小写

注意:findstr没有语法表示匹配一次或多次,使用替代的方法。 比如匹配27,7出现一次或多次,使用替代的方法 findstr 277

2、类似Linux下面的grep,查找文件的内容,文件内容:

```
AAA

222

aaa

333

aaa

444

bbb

555

3、执行如下:

d:\>findstr /in "aaa" test.txt

2:AAA

4:aaa

6:aaa
```

#### rename

```
rename .\aaa\111.txt 222.txt
```

参见

#### taskkill

#### 杀掉进程

im 进程映像名称,也就是进程名称

pid 进程id

注意:和findstr不同,taskkill不区分大小写,同时\*是通配符,匹另外,控制台程序 system("pause");会导致进程杀死了,但是界面

#### tasklist

列出进程,经常和findstr一起使用

#### Windows和Linux下面的线程睡眠

- 1、Windows下面是 Sleep(毫秒),注意是大写的S
- 2、Linux下面, sleep(秒), usleep(微秒)

```
3、比如睡眠100毫秒,实现为
#ifdef WIN32
::Sleep(100);
#else
::usleep(100*1000);
#endif
```

#### xcopy

1、考虑下面的需求,把aaa目录下面的111.txt 拷贝到 bbb,如下:echo on xcopy .\aaa\111.txt .\bbb\ /y pause 当bbb作为目录已经存在的情况下,上面等价于xcopy .\aaa\11 如果bbb目录不存在 , .\bbb\ 自动建立目录 ,并拷贝 .\bbb 会提示bbb是目录 ,还是文件。

- 2、注意这里表示路径要用右斜杠,因为左斜杠表示命名的选项,如tes
- 3、考虑下面的需求,把aaa目录下面的111.txt 拷贝到 bbb,并且重echo on

xcopy .\aaa\111.txt .\bbb\222.txt /y pause 执行过程,会提示目标 .\bbb\222.txt 是文件名还是目录名 (F = 文件, D = 目录)? 需要用户输入。

新安州广制八。 4 西田县 200027

4、原因是,xcopy不知道222.txt是个文件还是目录?怎么解决这个问这就需要用户,显示地告诉xcopy,222.txt是目录还是文件。如果希望222.txt是个目录,把111.txt拷贝到222.txt,修改为

如果希望222.txt是个文件,也就是copy并且重命名,使用下面的 echo f | xcopy .\aaa\111.txt .\bbb\222.txt /y

原理是通道的概念,前一个命令的标准输出作为后一个命令的标准echo f 输出f,xcopy 提示输入,前一个输出f作为xcopy的标注意:通道是将前一个命令的输出,作为字符串传递给下一个命令对于有些情况,需要使用xargs,xargs是把前一个命令的输出作

后一个命令是rm,rm要求将前一个命令的输出作为文件,而不是字 5、当然希望222.txt是个目录,也可以使用echo d,如下: echo d| xcopy .\aaa\111.txt .\bbb\222.txt /y

#### 参见

#### 其它常用的批处理

```
1、VS2005自动编译
echo on
echo test compile

set BUILD_EXE="C:\Program Files (x86)\Microsoft \
set BUILD_FLAG=rebuild
set BULID_OUT="debug"

%BUILD_EXE% TestCompileByBat.vcproj /project Test
pause
注意:编译之后,日志写入文件makefile.log,通过findstr。
2、SVN自动更新
"C:\Program Files\TortoiseSVN\bin\TortoiseProc.e>
```

#### 延迟变量

```
1、一对小括号中的多个语句,组成复合语句,考虑下面的复合语句:echo on for /l %%i in (1 1 3) do ( set var=%%i echo %var% ) pause 运行发现,赋值明明成功了,但是echo的结果为空。

2、修改脚本,如下:echo on set var=hello echo %var%
```

```
for /1 %%i in (1 1 3) do (
      set var=\%i
      echo %var%
   echo %var%
   pause
   运行发现,前面echo的结果都是hello,出了for循环,echo的结
3、思考为什么?
   这是因为对于复合语句,使用一个变量的时候,认为这个变量已经;
   还是引用原来的变量值。
4、怎么解决这个问题?
   原因是对于复合语句,批处理认为变量都是预先准备好的。解决办法
   使用延迟变量,有两个地方需要注意:
   a、加上setlocal enabledelayedexpansion
   b、同时使用!var!这种方式引用变量,而不是%var%
   如下:
   echo on
   setlocal enabledelayedexpansion
   for /l %%i in (1 1 3) do (
      set var=\%i
      echo !var!
   echo !var!
   pause
```

#### 批处理for循环

FOR这条命令基本上都被用来处理文本,但还有其他一些好用的功能! 看看他的基本格式(这里我引用的是批处理中的格式,直接在命令行只需要 FOR 参数 %%变量名 IN (相关文件或命令) DO 执行的命令 参数:FOR有4个参数 /d /l /r /f 他们的作用我在下面用例子解释 %%变量名 :这个变量名可以是小写a-z或者大写A-Z,他们区分大小写,FIN:命令的格式,照写就是了; (相关文件或命令) :FOR要把什么东西读取然后赋值给变量,看下面的例 do:命令的格式,照写就是了! 执行的命令:对每个变量的值要执行什么操作就写在这.可以在CMD输入for /?看系统提供的帮助!对照一下

FOR %%variable IN (set) DO command [command-parameter %%variable 指定一个单一字母可替换的参数。

(set) 指定一个或一组文件。可以使用通配符。

command 指定对每个文件执行的命令。

command-parameters

为特定命令指定参数或命令行开关。

#### 现在开始讲每个参数的意思

/d

仅为目录

如果 Set (也就是我上面写的 "相关文件或命令") 包含通配符(\* 和录(而不是指定目录中的文件组)执行指定的 Command。

系统帮助的格式: FOR /D %%variable IN (set) DO command 他主要用于目录搜索,不会搜索文件,看这样的例子

@echo off

for /d %%i in (\*) do @echo %%i

pause

把他保存放在C盘根目录执行,就会把C盘目录下的全部目录名字打印出来在来一个,比如我们要把当前路径下文件夹的名字只有1-3个字母的打出。@echo off

for /d %%i in (???) do @echo %%i pause

这样的话如果你当前目录下有目录名字只有1-3个字母的,就会显示出来

#### 思考题目:

@echo off

for /d %%i in (window?) do @echo %%i pause

保存到C盘下执行,会显示什么呢?自己看吧!

/D参数只能显示当前目录下的目录名字,这个大家要注意!

/R

递归

进入根目录树 [Drive:]Path,在树的每个目录中执行 for 语句。如当前目录。如果 Set 只是一个句点 (.),则只枚举目录树。

系统帮助的格式:FOR /R [[drive:]path] ‰variable IN (set 上面我们知道,/D只能显示当前路径下的目录名字,那么现在这个/R也是

#### /D强大多了!

他可以把当前或者你指定路径下的文件名字全部读取,注意是文件名字, @echo off

for /r c:/ %%i in (\*.exe) do @echo %%i pause

咋们把这个BAT保存到D盘随便哪里然后执行,我会就会看到,他把C盘根上的EXE文件都列出来了,这里的c:/就是目录了。

再来一个

@echo off

for /r %%i in (\*.exe) do @echo %%i pause

参数不一样了,这个命令前面没加那个C:/也就是搜索路径,这样他就会个BAT你把他防灾d:/test目录下执行,那么他就会把D:/test目录和他来!!!

#### /L

迭代数值范围

使用迭代变量设置起始值(Start#),然后逐步执行一组范围的值,直  $_{\circ}$  /L 将通过对 Start# 与 End# 进行比较来执行迭代变量。如果 S如果迭代变量超过 End#,则命令解释程序退出此循环。还可以使用负的行此范围内的值。例如,(1,1,5) 生成序列 1 2 3 4 5,而 (5,-1 系统帮助的格式:for /L %% Variable in (Start#, Step#, End# 例如:

@echo off

for /l %%i in (1,1,5) do @echo %%i pause

保存执行看效果,他会打印从1 2 3 4 5 这样5个数字 (1,1,5)这个参数也就是表示从1开始每次加1直到5终止! 再看这个例子

@echo off

for /l %%i in (1,1,5) do start cmd pause

执行后是不是吓了一跳,怎么多了5个CMD窗口,呵呵!如果把那个 (1,1,我先告诉大家,会打开65535个CMD窗口....这么多你不死机算你强! 当然我们也可以把那个start cmd改成md %%i 这样就会建立指定个目看完这个被我赋予破坏性质的参数后,我们来看最后一个参数 /f

含有/F的for详细说明

含有/F的for有很大的用处,在批处理中使用的最多,用法如下: 格式:

FOR /F ["options"] %%i IN (file) DO command

FOR /F ["options"] %%i IN ("string") DO command

FOR /F ["options"] %%i IN ('command') DO command

这个可能是最常用的,也是最强的命令,主要用来处理文件和一些命令的 file代表一个或多个文件

string 代表字符串

command代表命令

["options"] 可选

对于FOR /F %%i IN (file) DO command

file为文件名,按照官方的说法是,for会依次将file中的文件打开,假如文件a.txt中有如下内容:

第1行第1列 第1行第2列 第1行第3列 第2行第1列 第2行第2列 第2行第3列 第3行第1列 第3行第2列 第3行第3列

你想显示a.txt中的内容,会用什么命令呢?当然是type,type a.txfor也可以完成同样的命令:

for /f ‰i in (a.txt) do echo ‰i 还是先从括号执行,因为含有参数/f,所以for会先打开a.txt,然后读

{"第1行第1列 第1行第2列 第1行第3列", //第一个元素 "第2行第1列 第2行第2列 第2行第3列", //第二个元素 "第3行第1列 第3行第2列 第3行第3列"} //第三个元素

集合中只有3个元素,同样用‰i依次代替每个元素,然后执行do后面的 具体过程:

用‰i代替"第1行第1列 第1行第2列 第1行第3列",执行do后面的ecl用‰i代替"第2行第1列 第2行第2列 第2行第3列",执行echo ‰i,依次,直到每个元素都代替完为止。

为了加强理解/f的作用,请执行一下两个命令,对比即可明白:

for /f %%i in (a.txt) do echo %%i //这个会显示a.txt里面的内容。

for ‰i in (a.txt) do echo ‰i //而这个只会显示a.txt这个

通过上面的学习,我们发现for /f会默认以每一行来作为一个元素,但它们就是:delims和tokens

delims 用来告诉for每一行应该拿什么作为分隔符,默认的分隔符是到比如,还是上面的文件,我们执行下面的命令:

for /f "delims= " %%i in (a.txt) do echo %%i

#### 显示的结果是:

第1行第1列 第2行第1列

第3行第1列

为什么是这样的呢。因为这里有了delims这个参数,=后面有一个空格执行过程是:

将第一个元素"第1行第1列 第1行第2列 第1行第3列"分成三个元素:"

但是这样还是有局限的,如果我们想要每一行的第二列元素,那又如何则这时候,tokens跳出来说,我能做到。

它的作用就是当你通过delims将每一行分为更小的元素时,由它来控制还是上面的例子,执行如下命令:

for /f "tokens=2 delims= " %%i in (a.txt) do echo %%i

#### 执行结果:

第1行第2列

第2行第2列

第3行第2列

如果要显示第三列,那就换成tokens=3。

同时tokens支持通配符\*,以及限定范围。

如果要显示第二列和第三列,则换成tokens=2,3或tokens=2-3,如果此时的命令为:

for /f "tokens=2,3 delims= " %%i in (a.txt) do echo 9

怎么多出一个‰j?

这是因为你的tokens后面要取每一行的两列,用%%i来替换第二列,用注并且必须是按照英文字母顺序排列的,%%j不能换成%k,因为i后面是j执行结果为:

第1行第2列 第1行第3列

第2行第2列 第2行第3列

第3行第2列 第3行第3列

对以通配符\*,就是把这一行全部或者这一行的剩余部分当作一个元素了比如:

for /f "tokens=\* delims= " %%i in (a.txt) do echo %%i

#### 执行结果为:

第1行第1列 第1行第2列 第1行第3列

第2行第1列 第2行第2列 第2行第3列

第3行第1列 第3行第2列 第3行第3列

其实就跟for /f %%i in (a.txt) do echo %%i的执行结果是一样再如:

for /f "tokens=2,\* delims= " %%i in (a.txt) do echo 9 执行结果为:

第1行第2列 第1行第3列 第2行第2列 第2行第3列

第3行第2列 第3行第3列

用%%i代替第二列,用%%i代替剩余的所有

最后还有skip合eol,这俩个简单,skip就是要忽略文件的前多少行, 比如: for /f "skip=2 tokens=\*" %%i in (a.txt) do echo %%i 结果为:

第3行第1列 第3行第2列 第3行第3列 用skip来告诉for跳过前两行。 如果不加tokens=\*的话,执行结果为:

第3行第1列 不知道怎么回事。 再如,当a.txt内容变成:

.第1行第1列 第1行第2列 第1行第3列 .第2行第1列 第2行第2列 第2行第3列 第3行第1列 第3行第2列 第3行第3列 执行for /f "eol=. tokens=\*" %%i in (a.txt) do echo %%.

第3行第1列 第3行第2列 第3行第3列 用eol来告诉for忽略以"."开头的行。同样也必须加tokens=\*,否则只会显示"第3行第1列

#### 终极dos批处理循环命令详解

格式:FOR [参数] %%变量名 IN (相关文件或命令) DO 执行的命作用:对一个或一组文件,字符串或命令结果中的每一个对象执行特注意:在批处理文件中使用 FOR 命令时,指定变量请使用 %%var 关于:for命令可以带参数或不带参数,带参数时支持以下参数:/d下面分别解释一下

===

零:无参数时:

- - -

FOR %variable IN (set) DO command [command-parame %variable 指定一个单一字母可替换的参数。

(set) 指定一个或一组文件。可以使用通配符。

command 指定对每个文件执行的命令。

command-parameters

为特定命令指定参数或命令行开关。

#### TTT示例:

for ‰i in (t\*.\*) do echo ‰i --显示当前目录下与t\*.\* for ‰i in (d:/mydocuments/\*.doc) do @echo ‰i --

一、参数 /d (参数只能显示当前目录下的目录名字)

- - -

格式: FOR /D %variable IN (set) DO command [command 这个参数主要用于目录搜索,不会搜索文件,/D 参数只能显示当前目

#### TTT示例:

for /d %%i in (c:/\*) do echo %%i --显示c盘根目录下的 for /d %%i in (???) do echo %%i --显示当前目录下名字

#### ===

二、参数 /R (搜索指定路径及所有子目录中与set相符合的所有文

格式: FOR /R [[drive:]path] %variable IN (set) DO 心此命令会搜索指定路径及所有子目录中与set相符合的所有文件,注

1、set中的文件名如果含有通配符(?或\*),则列举/R参数指定的2、如果set中为具体文件名,不含通配符,则枚举该目录树(即列等例:for /r c:/ %%i in (\*.exe) do echo %%i --把C盘根

### TTT示例:

for /r c:/ %%i in (boot.ini) do echo %%i --枚举了c: for /r d:/backup %%i in (1) do echo %%i --枚举d/ba for /r c:/ %%i in (boot.ini) do if exist %%i echo

#### ===

三、参数 /L (该集表示以增量形式从开始到结束的一个数字序列。

格式:FOR /L %variable IN (start, step, end) DO commi 该集表示以增量形式从开始到结束的一个数字序列。可以使用负的

#### TTT示例:

for /l %%i in (1,1,5) do @echo %%i --输出1 2 3 4 5

for /l %%i in (1,2,10) do @echo %%i --输出1,3,5,7 for /l %%i in (100,-20,1) do @echo %%i --输出100,8 for /l %%i in (1,1,5) do start cmd --打开5个CMD窗口 for /l %%i in (1,1,5) do md %%i --建立从1~5共5个文件 for /l %%i in (1,1,5) do rd /g %%i --删除从1~5共5个

#### 四、参数 /F (使用文件解析来处理命令输出、字符串及文件内容。)

- - -

这个参数是最难的,参数又多,先简单的解释一下:for命令带这个以下是某高手的解释,感觉有点太专业了,自认为不太容易理解,也[迭代及文件解析--使用文件解析来处理命令输出、字符串及文件内请注意:在没有使用令牌选项时,/F 将只检查第一个令牌。文件解默认情况下,/F 传递每个文件每一行的第一个空白分隔符号。跳过+++

#### 格式:

FOR /F ["options"] %variable IN (file-set) DO com FOR /F ["options"] %variable IN ("string") DO com FOR /F ["options"] %variable IN ('command') DO com

#### 或者,如果有 usebackg 选项:

FOR /F ["options"] %variable IN (file-set) DO com FOR /F ["options"] %variable IN ("string") DO com FOR /F ["options"] %variable IN ('command') DO com

TTT说明:以上是WinXP系统中的帮助内容,你可以注意到,两者完FOR /F ["options"] %variable IN ("file-set") DO comFOR /F ["options"] %variable IN ('string') DO comFOR /F ["options"] %variable IN (`command`) DO com

+++

(TTT说明:下面是详细的解释,大部分是系统中的帮助内容,也有)

#### 1) OPTION关键字详解:

eol=c:指一个行注释字符的结尾(就一个)。例如:eol=; --忽断skip=n:指在文件开始时忽略的行数。例如:skip=2 --忽略2行delims=xxx:指分隔符集。这个替换了空格和跳格键的默认分隔符

tokens=x,y,m-n:指每行的哪一个符号被传递到每个迭代的 for 关于usebackq,不同版本的系统提示不同的帮助,不过都有助于理(1),usebackq:使用后引号(键盘上数字1左面的那个键`)。未(2),usebackq:指定新语法已在下类情况中使用:在作为命令执以上两条结合着看,其实已经可以明白了,我再说明一下:其实这个参数的目的就是为了处理带有空格的文件名。如果您要处理

- 2) file-set 为一个或多个文件名。继续到 file-set 中的下-
- 3) %i:专门在 for 语句中得到说明,%j 和 %k 是通过tokens (TTT补充说明:

一般在tokens后只指定第一个参数,如%%i或%a,在后面使用第二如果使用的变量名超过了%z或%Z,就无法使用了,曾经以为会循环如:for /f "tokens=1,2,3\* delims=-, " %%y in ("aa +++

#### 以下是系统提供的范例:

eol=; --忽略以分号打头的那些行;

tokens=2,3\* --将每行中的第二个和第三个符号传递给 for 程 delims= , --用逗号和/或空格定界符号。

%i --这个 for 程序体的语句引用 %i 来取得取得的首个字符串 (TTT说明:上述例子和说明中明显的错误,%i应该换为%%i(帮助中)

#### +++

TTT: 下面列我做的几个例子:

1,分析文件的例子

FOR /F "eol=; tokens=1,2\* delims=,- " %%i in (d:/2,分析字符串的例子:

for /f "tokens=1,2,3\* delims=-, " %%i in ("aa bb, 3,分析命令输出的例子:

FOR /F "tokens=1\* delims==" %%i IN ('set') DO @ec

如果使用了usebackq参数后,命令如下,结果与上面的完全相同。 1,分析文件的例子

FOR /F "usebackq eol=; tokens=1,2\* delims=,- " %%.2,分析字符串的例子:

for /f "usebackq tokens=1,2,3\* delims=-, " %%i in 3,分析命令输出的例子:(会枚举当前环境中的环境变量名称和值。FOR /F "usebackq tokens=1\* delims==" %%i IN (`set 结果大家可以试一下,很容易就明白的。

===

FOR命令中的变量

- - -

FOR 变量参照的替换已被增强。您现在可以使用下列选项语法:

~I - 删除任何引号("),扩充 %I

%~fI - 将 %I 扩充到一个完全合格的路径名

%~dI - 仅将 %I 扩充到一个驱动器号

%~pI - 仅将 %I 扩充到一个路径

%~nI - 仅将 %I 扩充到一个文件名

%~xI - 仅将 %I 扩充到一个文件扩展名

%~sI - 扩充的路径只含有短名

%~aI - 将 %I 扩充到文件的文件属性

%~tI - 将 %I 扩充到文件的日期/时间

%~zI - 将 %I 扩充到文件的大小

%~\$PATH: I 查找列在路径环境变量的目录(TTT提示:是环境此外,还可以组合修饰符来得到多重结果:

%~dpI - 仅将 %I 扩充到一个驱动器号和路径 %~nxI - 仅将 %I 扩充到一个文件名和扩展名

%~fsI - 仅将 %I 扩充到一个带有短名的完整路径名

%~dp\$PATH:i - 查找列在路径环境变量的目录,并将 %I 扩充到

%~ftzaI - 将 %I 扩充到类似输出线路的 DIR

在以上例子中,%I 和 PATH 可用其他有效数值代替。%~ 语法用-

#### (以上是系统帮助的内容)

我们可以看到每行都有一个大写字母"I",这个I其实就是我们在FOF FOR /F "usebackq eol=; tokens=1,2\* delims=,- " %% 这里我们就要把那个x,y,z改成%~fx,%~fy,%~fz。

+++

TTT特例:以下是我根据以上说明作的一个综合的例子,可以直接复@echo off

echo ---显示"dir c:/boot.ini /b /ah"

for /f "delims==" %%i in ('dir c:/boot.ini /b /ah

```
for /f "delims==" %%i in ('dir c:/boot.ini /b /ah
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
echo ---以下显示组合修饰符来得到多重结果---:
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                  %%i in ('dir c:/boot.ini /b /ah
for /f "delims=="
                 %%i in ('dir c:/boot.ini /b /ah
echo.
echo ---显示"dir C:/WINDOWS/system32/notepad.exe /
for /f "delims=="
                  %%i in ('dir C:/WINDOWS/system3:
for /f "delims==" %%i in ('dir C:/WINDOWS/system3:
for /f "delims==" %%i in ('dir C:/WINDOWS/system3)
for /f "delims=="
                  %%i in ('dir C:/WINDOWS/system3:
for /f "delims=="
                  %%i in ('dir C:/WINDOWS/system3:
for /f "delims=="
                  %%i in ('dir C:/WINDOWS/system3:
                  %%i in ('dir C:/WINDOWS/system3:
for /f "delims=="
for /f "delims=="
                  %%i in ('dir C:/WINDOWS/system3:
echo ---以下显示组合修饰符来得到多重结果---:
                  %%i in ('dir C:/WINDOWS/system3:
for /f "delims=="
for /f "delims=="
                  %%i in ('dir C:/WINDOWS/system3:
for /f "delims=="
                  %%i in ('dir C:/WINDOWS/system3:
for /f "delims=="
                  %%i in ('dir C:/WINDOWS/system3)
for /f "delims==" %%i in ('dir C:/WINDOWS/system3:
Pause
```

```
TTT说明:
 1,以上命令中, %%~fsI无法显示,估计是系统错误,因为%%~fIi
 2,以上命令如果保存在别的盘中,无法显示正确的驱动器和路径。
 3,如果想要%%~dp$PATH:i正常显示,要保证环境变量path中确等
下面依次说明一下:
 一、~I - 删除任何引号("),扩展 %I
 这个变量的作用就如他的说明, 删除引号!
 删除引号规则如下(BAT兄补充!):
 1、若字符串首尾同时存在引号,则删除首尾的引号;
 2、若字符串尾不存在引号,则删除字符串首的引号;
 3、如果字符串中间存在引号,或者只在尾部存在引号,则不删除。
 龙卷风补充:无头不删,有头连尾删。
 我们来看这个例子,首先建立临时文件temp.txt,内容如下
 "1111
 "2222"
 3333"
 "4444"44
 "55"55"55
 也可建立个BAT文件代码如下:
 @echo off
 echo ^"1111>temp.txt
 echo "2222">>temp.txt
 echo 3333^">>temp.txt
 echo "4444"44>>temp.txt
 echo ^"55"55"55>>temp.txt
 rem 上面建立临时文件,注意不成对的引号要加转义字符^,重定向
 pause
 del temp.txt
 执行后,我们看CMD的回显如下:
           #字符串前的引号被删除了
 1111
 2222
           #字符串首尾的引号都被删除了
```

3333" #字符串前无引号,后面的引号保留4444"44 #字符串前面的引号删除了,而写 #字符串前面的引号删除了,而中间的引号保留 55"55"55 #字符串前面的引号删除了,而中间的引号保留 请按仟意键继续. . . 结果和之前temp.txt中的内容对比一下,我们会发现第1、2、5行能 +++ \_、 %~fI - 将 %I 扩展到一个完全合格的路径名 示例: 把代码保存放在随便哪个地方,我这里就放桌面吧. FOR /F "delims==" %%i IN ('dir /b') DO @echo %% pause 执行后显示内容如下 C:/Documents and Settings/Administrator/桌面/test. C:/Documents and Settings/Administrator/桌面/test. 当我把代码中的 %%~fi直接改成%%i FOR /F "delims==" %%i IN ('dir /b') DO @echo %%. pause 执行后就会显示以下内容: test.bat test.vbs 通过对比,我们很容易就看出没有路径了,这就是"将 %I 扩展到一个 三、%~dI - 仅将 %I 扩展到一个驱动器号 看例子: 代码如下,我还是放到桌面执行! FOR /F "delims==" %%i IN ('dir /b') DO @echo %% pause 执行后我CMD里显示如下 C: C: 我桌面就两个文件test.bat,test.vbs,%%~di作用是,如果变量5 +++ %~pI - 仅将 %I 扩展到一个路径 四、

这个用法和上面一样,他只打印路径不打印文件名字

```
FOR /F "delims==" %%i IN ('dir /b') DO @echo %%
pause
我就不打结果了,大家自己复制代码看结果吧,下面几个都是这么个,
+++
万、 %~nI - 仅将 %I 扩展到一个文件名
只打印文件名字
FOR /F "delims==" %%i IN ('dir /b') DO @echo
                                      %%
pause
+++
六、 %~xI - 仅将 %I 扩展到一个文件扩展名
只打印文件的扩展名
FOR /F "delims==" %%i IN ('dir /b') DO @echo
                                      %%
pause
+++
七、 %~sI - 扩展的路径只含有短名
打印绝对短文件名
FOR /F "delims==" %%i IN ('dir /b') DO @echo
                                      %%
pause
+++
八、 %~aI - 将 %I 扩展到文件的文件属性
打印文件的属性
FOR /F "delims==" %%i IN ('dir /b') DO @echo
                                      %%
pause
+++
九、 %~tI - 将 %I 扩展到文件的日期/时间
打印文件建立的日期
FOR /F "delims==" %%i IN ('dir /b') DO @echo
                                      %%
pause
+++
十、 %~zI - 将 %I 扩展到文件的大小
打印文件的大小
```

龙卷风补充:上面例子中的"delims=="可以改为"delims=",即

+++

十一、 %~\$PATH: I - 查找列在路径环境变量的目录,并将 %I j 这是最后一个,和上面那些都不一样,我单独说说!

然后在把这些代码保存为批处理,放在桌面。

@echo off

FOR /F "delims=" %%i IN ("notepad.exe") DO echo pause

龙卷风补充:上面代码显示结果为C:/WINDOWS/system32/notele的意思就在PATH变量里指定的路径里搜索notepad.exe文件,如(TTT说明,保存到桌面上,运行显示结果为:系统找不到文件 "nc上面的命令应该写成:

FOR /F "delims=" %%i IN ("notepad.exe") DO echo
)

#### 批处理注释

echo on

::使用一个冒号,不能和已有的标签同名

:echo aaa

::使用两个冒号

::echo aaa

::使用rem,会打印出来,但是后面的命令不会执行rem echo aaa

::使用echo,输出到黑洞 echo echo aaa>nul

::使用%% %echo aaa%

```
::使用goto加上标签,跳过下面的内容
goto start
echo aaa
:start
echo bbb
pause
```

#### 批处理百分号

```
1、%1,%2...引用外部变量,只有9个
2、%aa%,引用变量aa,如下:
    set aa=123
    echo %aa%
3、在cmd窗口的for循环中,使用%i,如下:
    for %i in (1,2,3) do echo %i
4、注意:在批处理的文件中,要使用%%i,如下:
    for %%i in (1,2,3) do echo %%i
    为什么批处理文件中需要使用两个%?
    这是因为编译器解析的时候,屏蔽一个,可以认为前面一个%作为等
```

### 批处理递归删除指定的目录或者文件

```
1、考虑下面的需求,从svn服务器Checkout出来的内容,包含了.svn
注:这里只是为了举例子,实际情况,如果不想要.svn目录,应该
2、批处理脚本如下:
echo on
setlocal enabledelayedexpansion
call:DelFile "D:\Python"
pause
exit /b 0
:DelFile
cd %1
for /f %%i in ('dir /b /a') do (
echo %%i|findstr "^.svn$">nul
```

```
if !errorlevel! equ 0 (rmdir /q /s %%i)
       set str=%%~ai
       if "!str:~0,1!"=="d" (call:DelFile %%i)
   )
   cd ..
   goto :eof
3、有几个地方需要特别注意:
   a、函数的退出使用 goto :eof
   b、pause之后,必须调用exit /b 0,否者批处理继续向下执行
   c、删除目录使用 rmdir /q /s %%i
   d、!str:~0,1! 前后两个感叹号去变量的内容,str:~0,1从位
4、如果删除指定的文件,比如txt,如下:
   echo on
   setlocal enabledelayedexpansion
   call:DelFile "D:\Python"
   pause
   exit /b 0
   :DelFile
   cd %1
   for /f %%i in ('dir /b /a') do (
       echo %%i|findstr ".txt$">nul
       if !errorlevel! equ 0 (rm /f %%i)
       set str=%%~ai
       if "!str:~0,1!"=="d" (call:DelFile %%i)
   )
   cd ..
   goto :eof
5、需要注意的地方:
   a、使用 rm /f %%i删除文件
   b、匹配的条件, ^.svn$是匹配.svn目录,.txt$是匹配.txt后
```

#### 粘贴变成灰色,原来是飞秋的问题

- 1、检查当前飞秋是不是截图了,并且没有发送。
- 2、关闭飞秋,不要使用飞秋了,使用微信就很好。

#### 访问共享目录的注意事项

- 1、在windows的资源管理器,访问其他设备的共享目录,格式如下: \\10.30.6.201\Software 不能写成\\10.30.6.201/Software,否则报错: Windows 无法访问\\10.30.6.201/Software,请检查名称的
- 2、对于本地的目录,如下:E:\软件\Redis 可以写成 E:\软件/Redis, Windows会自动转化为 E:\软件\Re

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

**家庭** 模块 模块 模块 | 変量

# 模块

你就是孩子最好的玩具(我的小螃蟹)

## 重变

### 婆媳关系

### 变量说明

#### 婆媳关系

- 1、婆媳关系是个谎言,听上去是婆婆和媳妇的二元关系,却忽略了本质 三角关系的核心是儿子,如果儿子袖手旁观,所谓的婆媳关系很难。
- 2、这个三角关系,是中国传统的大家庭观念所导致的结果。媳妇的角色婆婆理所当然地认为这是我的家,家里的事情我做主。但是现代家。
- 3、传统中国家庭是失衡的,亲子关系是核心,夫妻关系是配角。在这种
- 4、如果你是长辈,请记住孩子不该是你的最爱,配偶才是你最重要的爱
- 5、夫妻关系是家庭中的第一位,这是健康家庭的第一定律。夫妻关系是如果亲子关系凌驾于夫妻关系之上,就会产生最常见的两个问题:
  - a、糟糕的婆媳关系;
  - b、严重的恋子情结。
- 6、这两点是相辅相成的。在新家庭中,如果有一个糟糕的婆媳关系,那也曾有一个糟糕的婆媳关系。而那个糟糕的婆媳关系,导致婆婆与。 当儿子建立一个自己的新家庭时,婆婆还想维持与儿子亲密的关系
- 7、相反,如果婆婆心目中最重要的人一直是丈夫而不是儿子,那么儿子 她会欣喜地看到,儿子找到了他最爱的人,他可以拥有他的家庭、
- 8、不健康的关系有:

烦丈夫,爱儿子 太愚孝,轻妻子

9、势必要分离的,不是最爱。

如果是儿子,就要对自己说,爸爸才是妈妈最爱的人,自己不是;如果是女儿,就要对自己说,妈妈才是爸爸最爱的人,自己不是;如果是父亲,就要对女儿说,我爱你,但妈妈才是能陪伴我一生的如果是母亲,就要对儿子说,我爱你,但爸爸才是能陪伴我一生的,这才是健康家庭之道。

- 10、最好的婆媳关系是:没有关系。公公婆婆的英语翻译很贴切,fath
- 11、有了小孩子之后,老人会帮忙来照顾孩子。但是,请不要忘了,

老人不是在自己的家里操持家务带孩子,而是在儿子/女儿的家里这两者在本质上是有非常大的区别的。

12、至此,这个千古难题终于有了答案:如果你老婆和你老妈同时掉水! 救老婆。

老妈呢?

我老爸去救啊!

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

你就是孩子最好的玩具 (我的小螃蟹) <sup>家庭</sup>

## 变量

### 不眠之夜

### 安抚毯具有象征意义

误区一:控制型父母与放任型父母

误区二:轻视、低估和否定孩子的感受

误区三:采用外部激励和奖赏的手段

误区四:以消极后果作为惩罚

情感引导的关键步骤

### 详细描述

### 变量说明

#### 不眠之夜

- 1、转折点概念,当孩子在某一领域发生飞跃的进步,往往会在其他方面
- 2、宝宝彻夜不眠,有可能有惊喜在早上等着你。

#### 安抚毯具有象征意义

- 1、孩子不善于调节自己的情绪,因此需要一个过渡性的物品来安慰他们
- 2、孩子意识到有时候会和父母分开,开始建立独立性。就有可能对某物 二者都可以提供平静和喜乐。

### 情感引导的关键步骤

1、播下种子

提前告诉孩子在特定的场景下会发生什么事,应该怎么做,只告诉

2、观察和判断

观察和判断的本质是关注,孩子会因为你的关注,去做你期望他做

3、聆听

蹲下来和孩子的视线齐平,以便更好的眼神交流。

4、体察并理解孩子的感受

承认并接受孩子的情绪,鼓励孩子把自己的感受说出来,而不是把

5、同情

当孩子感受到有人理解他的时候,孩子就更愿意与人交流。

### 误区一:控制型父母与放任型父母

1、控制型父母与放任型父母,都不会教会孩子如何尊重他人的需要和体

- 2、不要让孩子出于对你的恐惧而听从你的命令,而是应该出于对你的爱
- 3、如果家庭不能给孩子带来归属感和安全感,孩子就会到家以外的地方
- 4、正确的做法是:采用折中的方法,跟孩子沟通和交流。

误区三:采用外部激励和奖赏的手段

- 1、用奖励来操纵孩子的表现是错误的。
- 2、孩子最希望得到的不是什么玩具,而是你的关注。

误区二:轻视、低估和否定孩子的感受

- 1、孩子跌倒,不要说,不疼不疼。这会让孩子心里很难受。
- 2、父母很容易把自己的观点和感受凌驾于孩子之上,应该站在孩子的角
- 3、轻视、低估和否定孩子的感受,会导致孩子缺乏同情心,并以自我为
- 4、孩子难过的时候需要成年人的安慰,无论对社会还是他人的关怀,孩 我们怎样对待孩子,孩子就会怎样对待别人。

误区四:以消极后果作为惩罚

1、什么是消极后果?

孩子不听话,回来太晚,导致没饭吃饿肚子,父母很生气,说今天

- 2、孩子会困惑,为什么回来晚了,不能看电视。更好的策略是: 对不起,你回来晚了,妈妈也想给你吃的,可是没有了。我们明天下次不要再回来晚了,好吗?
- 3、不要对孩子抱有不切实际的期望,父母的很多要求会超出小家伙的能
- 4、包容孩子是种乐趣,做出符合现实的期望,照顾孩子就会是很享受的

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

工作有关

## 模块

### CryptDB

密码学有关

## 重变

备忘录

### 详细描述

### 变量说明

#### 备忘录

```
1、ftp信息
   ftp: 192.168.10.70
    user: uftp
    passwd: hzflk1234
2、无线
    hzflk789
3、邮箱
    ym.163.com niuzibin@flksec.com/123456/ruanxiaopar
4、git
    niuzibin@flksec.com ruanxiaopang
5、电脑登录
    niuzibin/ruanxiaopang
6、虚拟机
    ubuntu niuzibin/ruanxiaopang
    root/ruanxiaopang
7、svn
    niuzh/niuzh
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

首页 模块 **CryptDB** 工作有关

# 模块

## CryptDB流程

## 变量

DBMeta写入到embedded\_db

对于HOM反序列化,再序列化之后发生变化,为什么

运行环境有关

CryptDB原理介绍

QA常见问题解答

出现问题,怎么恢复

查看嵌入数据库embedded\_db

blowfish的秘钥key哪里来的

### 变量说明

### blowfish的秘钥key哪里来的

```
1、proxy启动后,第一次tlogin的时候
   bf(cinteger.getKey())
   const CryptedInteger cint = CryptedInteger::deser
   DBMeta::doFetchChildren 读取出来的,执行下面的语句
   " SELECT embedded_db.generic_prefix_MetaObject.se
2、得到三条记录,如下:
   67 DETJOIN DETJOIN_int 16_P F???g?鈳r?9_MYSQL_TYF
   67 DET DET_int 16_+DoL8; 颜*+攝壓19_MYSQL_TYPE_LON(
   67 RND RND_int 16_紅Ql6z 村?19_MYSQL_TYPE_LONGLON
   分别是child_serial_object, child_key, child_id
3、其中的child_serial_object, 67 RND RND_int 16_紅Ql6z
   构建出来 const CryptedInteger cint = CryptedIntege
   得到
   struct SerialLayer
       SECLEVEL 1;
                                  SECLEVEL::RND
       std::string name;
                                  RND int
       std::string layer_info;
                                  16_紅Ql6z 桁?19_M
4、再看create table的时候,怎么构建出来的?
   bool CreateDelta::apply 构建出来秘钥,然后insert一下
     INSERT INTO embedded_db.generic_prefix_Bleeding
   " INSERT INTO embedded_db.generic_prefix_Bleeding
5、serial_object是怎么来的?
   child serial
                      "67 RND RND_int 16_+\310\356$
```

```
esc_child_serial "67 RND RND_int 16_+\310\356\s esc_child_serial是对双引号转义一下
6、child_serial怎么来的?
    const std::string &child_serial = object.serializ
7、找到不同的加密方法
    std::string OnionMeta::serialize(const DBObject & {
        const std::string &serial = serialize_string(this->onionname) + serialize_string(std::to_string(this->uniq_cc serialize_string(TypeText<SECLEVEL>::toText(t return serial; }
```

### CryptDB原理介绍

- 1、Client<--->Proxy<--->Mysql
- 2、站在Client的角度,是和Mysql交互,不需要了解后面的机制。
- 3、站在Mysql的角度,内部存储的是加密信息,但是要支持同态运算。
- 4、这些工作,都是Proxy做的处理,接收客户端的正常sql请求,经过 特别注意,发给Mysql的信息,还必须要符合sql语法,只不过关键

### DBMeta写入到embedded db

- 1、这里分为两个阶段,
  - 第一个阶段写入到 generic\_prefix\_BleedingMetaObject 第二个阶段写入到 generic\_prefix\_MetaObject
- 2、需要保证这两个表的数据保持一致?
  - 先考虑 generic\_prefix\_BleedingMetaObject,是个树形结id是主键,并且自增,每次insert的时候,id的value写成0,m注意:如果不是写成0,比如当前的值为200,id的value写成300
- 3、因为要保持一致,所以要保存起来id,后面写入到 generic\_pref

### QA常见问题解答

- 1、一个字段为什么分成多个字段? 不能支持全同态加密,需要分成多个字段,支持不同模式的同态加速。
- 2、同态加密的字段为什么要多次加密? 同态加密字段按道理加密一次就够了,但是这样不够安全。因为同题 因此,增加一次RND,对同态加密的结果再执行一个加盐。需要保证

#### 出现问题,怎么恢复

- 1、删除mysql中的数据库cryptdb\_udf、remote\_db和yytest
- 2、删除 /home/niuzibin/work/heming/cryptdb/shadow目录中

### 对于HOM反序列化,再序列化之后发生变化,为什么

- 1、MjcgSE9NIEhPTSAxNiBIT00gSE9NIAko13oqF7NHoGBJCTcuj4 base64解码后为 27 HOM HOM 16\_XXX
- 2、然后构建HOM,如下:

return std::unique\_ptr<EncLayer>(new HOM(id, seri 这里存在一个问题,把整个 27 HOM HOM 16\_XXX 作为seed\_k

- 3、再次序列化的时候,前面一些固定的参数,加上seed\_key,如下: 38 HOM HOM 27 HOM HOM 16\_XXX
- 4、然后base64编码为

MzggSE9NIEhPTSAyNyBIT00gSE9NIDE2IEhPTSBIT00gCSjXe导致发生了变化

- 5、怎么解决这个问题? 就是再构建HOM的时候,使用16\_XXX作为seed\_key
- 6、这里引申出一个问题,HOM是同态加法,如果秘钥发生了变化,就没想那么原来写入数据库的时候,为什么没有问题?因为数据库只写入一次HOM的这条记录,以后都是读取。但是对于文件,涉及到读取,写入,重启又涉及到读取,写入。

### 查看嵌入数据库embedded db

1、准备工作

mysql> create database yytest;
Query OK, 1 row affected (0.01 sec)

```
mysql> use yytest;
   Database changed
   mysql>
   mysql>
   mysql> create table t1(c1 int);
   Query OK, 0 rows affected (0.01 sec)
   mysql> show tables;+----+
   | Tables_in_yytest |
    | t1
   +----+
   1 row in set (0.00 sec)
   mysql> insert into t1(c1) value(100);
   Query OK, 1 row affected (0.02 sec)
2、在shadow目录会多出来一些文件,如下:
   niuzibin@ubuntu:~/work/heming/cryptdb/shadow$ 11
   total 28688
   drwxrwxr-x 4 niuzibin niuzibin
                                      4096 Apr
                                               9 2
   drwxrwxr-x 15 niuzibin niuzibin 4096 Apr 3 2
                                     4096 Apr 9 2
   drwx----- 2 niuzibin niuzibin
   -rw-rw---- 1 niuzibin niuzibin 18874368 Apr 9 2
   -rw-rw---- 1 niuzibin niuzibin 5242880 Apr 9 2
-rw-rw---- 1 niuzibin niuzibin 5242880 Apr 9 2
   drwx----- 2 niuzibin niuzibin
                                      4096 Apr 9 2
3、关闭mysql和proxy
4、拷贝文件,到另一个mysql
   注意:不要拷贝到当前的这个mysql,否则会影响yytest
   sudo scp -r ./* niuzibin@192.168.29.128:~/aaa
5、再到目标mysql,拷贝一下,修改所有者
   cp /home/niuzibin/aaa/* ./ -r
   root@ubuntu:/var/lib/mysql# chown mysql ./ -R
6、重新启动mysql,然后连接mysql测试一把
还有另外一种办法:
1、set @cryptdb="show";
```

#### 运行环境有关

# Andy Niu Help 1.0.0.0

首页 模块 **CryptDB**流程 工作有关 » **CryptDB** 

# 重变

7	ξПІ	ua	台台	<b>₹</b> >	Ħ	公子	行
1	,HI	ua	ロリ	X	ユ	ハルイ	生

类的层次化结构

洋葱层调整测试

加解密元数据的读写

加解密元数据的读操作

加解密元数据的写操作

### 变量说明

#### 加解密元数据的写操作

```
1, CreateDelta::apply
   deltaOutputBeforeQuery insert到 generic_prefix_Bl
   deltaOutputAfterQuery insert到 generic_prefix_Me
2、这些可以怎么来的,在什么地方插入到数据库?
   序列化写入到数据库中
    --> " INSERT INTO " + table_name + "
                                            (serial
    --> esc_child_serial = escapeString(e_conn, child
    --> object.serialize(parent) // 序列化,看一下DBMet
    --> EncLayer::serialize
    --> return serial_pack(this->level(), this->name(
    --> doSerialize() 查看EncLayer 子类的doSerialize多流
   找两个例子
   DETJOIN int
        --> DET_abstract_integer::doSerialize()
        --> getCInteger_().serialize();
        --> CryptedInteger::serialize() 对以下字段序列作
           serializeStrings({key,TypeText<enum enum_</pre>
        --> DETJOIN int新增加了字段cinteger, 重写getCInt
           注:DETJOIN_int(unsigned int id, const Cr
   DETJOIN str
        --> std::string doSerialize() const {return r
       --> rawkey(prng_expand(seed_key, key_bytes))
    --> 都是找到seed kev
    --> DETJOINFactory::create(const Create_field &c1
    --> EncLayerFactory::encLayer(onion o, SECLEVEL 5
    --> const std::string key = m_key ? getLayerKey(n
```

```
--> OnionMeta::OnionMeta(onion o, std::vector<SE(
--> static bool init_onions_layout(const AES_KEY
--> FieldMeta::FieldMeta(const Create_field &fie)
--> a.getMasterKey().get()
--> createAndRewriteField(Analysis &a,xxx)
--> class CreateTableHandler::rewriteAndUpdate(Ar
--> AbstractQueryExecutor *DDLHandler::transformL
--> Rewriter::dispatchOnLex(Analysis &a, const st
--> Analysis analysis(default_db, schema, ps.getN
--> ps.getMasterKey()
--> Rewriter::rewrite(const std::string &g, const
--> return shared.getMasterKey();
--> const SharedProxyState &shared;
--> ProxyState *const ps = thread_ps = c_wrapper-
--> WrapperState *const c_wrapper = clients[clier
   clients[client]-->WrapperState-->ProxyState--
--> clients[client]->ps = std::unique_ptr<ProxySt
--> SharedProxyState::SharedProxyState(Connection
        : masterKey(std::unique_ptr<AES_KEY>(get)
          embed_dir(embed_dir),
          mysql_dummy(SharedProxyState::db_init(
--> getKey(master_key)获得 AES_KEY
--> const std::string &mkey
                                 = "113341234";
   shared_ps = new SharedProxyState(ci, embed_di
```

### 加解密元数据的读写

- 1、一个字段会扩展为多个字段,用于不同的同态加密。扩展后的一个字
- 2、每个洋葱层,需要确定加密算法和秘钥。
- 3、加密算法是由所支持的同态运算所决定的,比如 支持int的同态加密
- 4、秘钥是怎么来的?

秘钥是创建表的时候生成的,转化的步骤很多。参数的来源在哪里

5、秘钥需要持久化,保存在哪里? 在embedded\_db,递归调用读取。

### 加解密元数据的读操作

```
1、类图
   SharedProxyState<>--SchemaCache--->SchemaInfo<>---[
   SchemaInfo对应一个数据库的加密元数据
2、递归调用,load加密元数据
   std::function<DBMeta *(DBMeta *const)> loadChildr
   [&loadChildren, &e_conn](DBMeta *const parent) {
       auto kids = parent->fetchChildren(e conn);
       for (auto it : kids) {
          loadChildren(it);
       }
       return parent; /* lambda */
   };
   loadChildren(schema.get());
   堆栈可以看到,___lambda5::operator() rewrite_main.cc
   构建秘钥,从数据库读取,然后反序列化。
3、数据的crud操作(增删改查)涉及到加解密,那就需要用到秘钥,每
   需要把数据load到内存中,这就是涉及到同步,保持一致性。怎么
   a、SchemaCache有个字段no loads表示是否加载元数据,初始的
   b、同时【INSERT INTO embedded_db.generic_prefix_st
   c、每次判断是否需要重新读取,都是 【SELECT stale FROM ·
   d、在哪里更新stale?
       SchemaCache::updateStaleness,为了保证已经存在的
4、哪些情况判断是否读取?直接看调用图
   SchemaCache::getSchema
   rewrite
   OnionAdjustmentExecutor::nextImpl(const ResType &
   ShowTablesExecutor::nextImpl(const ResType &res,
   SpecialUpdateExecutor::nextImpl(const ResType &re
5、哪些情况更新stale值
   SchemaCache::updateStaleness
   AbstractQueryExecutor::next --> AbstractQueryExec
```

#### 和lua的交互流程

```
1、read_query
```

read\_query --> read\_query\_real --> next\_handler - 接收客户端的请求sql,进行转化,return proxy.PROXY\_SEND 改写后的sql语句,通过 proxy.queries:append 传递过去。

2 read\_query\_result

read\_query\_result --> read\_query\_result\_real --> 接收mysql的返回,进行处理,return proxy.PROXY\_SEND\_RD 处理后,如果有结果集需要返回,通过 proxy.response.result.

- 3、proxy与mysql交互,从mysql返回之后,必定进入 read\_query\_ 分为几种情况:
  - a、不需要处理,比如 show databases; 直接返回
  - b、只需要处理一次,比如 show tables; 解密一些表名即可
  - c、需要处理多次,比如create table;
- 4、分析show databases;的流程:
  - a、客户端发给proxy请求show databases, read\_query接口:
  - b、内部的next接口返回query-results,把skip置为true
  - b、proxy和mysql交互,从mysql收到响应包,在read\_query\_
- 5、分析show tables;的流程:
  - a、客户端发给proxy请求show tables, read\_query接口返回
  - b、内部的next接口返回again
  - c、proxy和mysql交互,从mysql收到响应包,在read\_query\_
  - d、next返回"results",解密返回,return proxy.PROXY\_5
- 6、分析create table;的流程:
  - a、客户端发给proxy请求create table, read\_query接口返[
  - b、内部的next接口返回again
  - c、proxy和mysql交互,从mysql收到响应包,在read\_query\_
  - d、next返回"again",告诉proxy忽略mysql的返回。
  - e、proxy和mysql交互,从mysql收到响应包,next返回"resul
- 7、也就是说,

第一次的改写返回PROXY\_SEND\_QUERY,中间过程的改写,返回PROXY\_IGNORE\_RESULT最后一步的处理,返回PROXY\_SEND\_RESULT

8、CryptDB.next的返回有三个值:

again 要求proxy和mysql交互,并且再次进来,复query-results mysql的响应,到lua之后,lua直接返回results 返回最后处理的结果,要求proxy转发给客户

9、proxy收到的回复有三个值:

PROXY\_SEND\_QUERY 向mysql发送请求

PROXY\_SEND\_RESULT 向客户端发送最后的结果 PROXY\_IGNORE\_RESULT 忽略中间过程的结果

10、总结一下,也就是说:

第一步是改写,要求proxy和mysql交互,中间步骤处理返回的结果,比如剥洋葱,或者插入到remote\_db,最后一步,改写mysql返回的结果集。

#### 洋葱层调整测试

- 1、剥洋葱,已经插入的数据,通过update更新掉,新插入的数据,加密
- 2、准备测试数据如下:

```
start transaction;
create table student(id int, name varchar(64));
insert into student(id ,name) value (1,"Andy");
insert into student(id ,name) value (2,"Bill");
insert into student(id ,name) value (3,"Caroline'
insert into student(id ,name) value (4,"David");
commit;
```

3、测试洋葱层调整

select \* from student where id =2;

- 4、按道理,洋葱层会少一层,在哪里处理的? 之前查回来的结果,和现在查回来的结果,对照一下。
- 5、在哪里删除RND记录的?

deltaOutputBeforeQuery-->bool DeleteDelta::apply
" DELETE embedded\_db.generic\_prefix\_BleedingMeta(

```
deltaOutputAfterQuery-->DeleteDelta::apply
" DELETE embedded_db.generic_prefix_MetaObject
```

### 类的层次化结构

- 1、DBMeta是数据库的元数据类,子类MappedDBMeta是类模板,字段
- 2、MappedDBMeta会实例化一些模板类,比如MappedDBMeta<Datab
- 3、包含一组DatabaseMeta的集合,其实也就是 SchemaInfo, Sche
- 4、DatabaseMeta、TableMeta、FieldMeta也是同样的道理。
- 5、TableMeta关注以下几个字段,分别为:

counter 因为一个表的字段,是有顺序的,counter记录当前的anon\_table\_name 是加密后的表名

注意:IdentityMetaKey 是由明文的表名,序列化出来的

tm->addChild(IdentityMetaKey(name), std::mc 明文的表名-->IdentityMetaKey-->TableMeta-->anno\_ta

6、FieldMeta关注以下几个字段,分别为:

fname 字段名称

salt name 前缀加上随机值

onion\_layout 是字段扩展后的洋葱,表示一个字段对应一组洋系

- 7、OnionMeta不同于上面的结构,不是继承MappedDBMeta,而是直挂 onionname是洋葱字段名称
- 8、EncLayer就是叶子结点,继承了LeafDBMeta,在LeafDBMeta中 EncLayer代表了一个加密层次的抽象

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

**宮の** 模块 変量 **密码学有关** 工作有关

## 变量

密码学概述

**Blowfish** 

MD5

SHA1

数据加密为什么要加盐

### 详细描述

### 变量说明

#### **Blowfish**

1、Blowfish算法

Blowfish算法是一个64位分组及可变密钥长度的对称密钥分组密 32位处理器诞生后,Blowfish算法因其在加密速度上超越了DESī Blowfish算法具有加密速度快、紧凑、密钥长度可变、可免费使

#### MD5

- 1、Message Digest Algorithm MD5(中文名为消息摘要算法第五
- 2、MD5算法具有以下特点:

压缩性:任意长度的数据,算出的MD5值长度都是固定的。

容易计算:从原数据计算出MD5值很容易。

抗修改性:对原数据进行任何改动,哪怕只修改1个字节,所得到的强抗碰撞:已知原数据和其MD5值,想找到一个具有相同MD5值的数

3、MD5是不可逆加密,不涉及到解密,也无法解密,因此没有秘钥的概定 MD5相同的明文,会得到相同的密文,存在风险,使用彩虹表可以证

#### SHA1

- 1、安全哈希算法(Secure Hash Algorithm)对于长度小于2^64位 在传输的过程中,数据很可能会发生变化,那么这时候就会产生不 不可以从消息摘要中复原信息;
  - 两个不同的消息不会产生同样的消息摘要,(但会有1x10 ^ 48分え
- 2、SHA1在许多安全协议中广为使用,包括TLS和SSL、PGP、SSH、S/I

### 密码学概述

- 1、对称密码,加解密的密钥相同,存在密钥配送问题。
- 2、解决密钥配送问题,使用非对称密码,公钥是公开的,用于加密,私
- 3、非对称密码,存在效率问题。因此,使用非对称的公钥,加密对称的
- 4、检查消息是否一致,也就是是否被篡改,使用单向散列函数,也叫消
- 5、单向散列函数可以辨别出"篡改",但是不能辨别出"伪装",需要确认
- 6、问题的原因是,单向散列函数,每个人计算的结果都一样,Bob无法 解决办法是Alice和Bob共享一个密钥,其他人不知道,计算hash 必须知道密钥,才能计算出相同的hash值,这就是消息认证码。
- 7、相对于单向散列函数,消息认证码多了一个共享的密钥。
- 8、消息认证码存在问题,一个是共享密钥的配送问题,还存在一个问题 Bob说Alice发了某个消息,Alice说没有发过,并且Alice说,E
- 9、如何解决这个问题?
  - 也就是说,只有Alice才能生成这个消息,其他人不能生成这个消息参考非对称加密,其他人用公钥加密,只有我能使用私钥解密。反识有我自己保存私钥,其他人能用公钥解密,就说明这个消息是我这就是数字签名。私钥加密生成签名,公钥解密验证签名。
- 10、正确使用数字签名,有一个前提,那就是验证签名的公钥,必须是〕
- 11、这就要解决公钥的合法性,公钥的获取需要一个可信任的第三方认证 第三方对Bob的公钥加上数字签名,发给Alice, Alice获取Bob的可信的第三方也就意味着,Alice已知Trent合法的公钥,通过Tr
- 12、具体流程是:
  - a、Bob向Trent注册自己的公钥,这个环节是人工保证可靠性,T
  - b、Alice已知Trent的公钥,向Trent请求Bob的公钥,Trent为
  - c、Bob的公钥,加上Trent的数字签名,就是证书。通过Trent初
  - d、有了Bob的公钥,二者就可以加密通信了。
  - 注意:这里传输的公钥,不需要加密,只是保证Alice能够获取合意

#### 数据加密为什么要加盐

- 1、一般的加密算法,使用同一个秘钥,相同的明文加密得到相同的密文 MD5等不可逆的hash算法,不涉及秘钥,相同的明文也会得到相同
- 2、相同明文得到相同密文,存在风险,特别是明文的集合空间比较小的
  - a、数据库保存用户的登录密码,密码是加密保存的,用户张三访证 推断李四的密码和自己的密码相同,然后就可以使用李四的账
  - b、考虑数据库保存员工的绩效考核,有优秀、良好、合格、不合析 推断出每个明文对应的密文。

- c、还有一种情况,只拿到密文,但是明文的集合空间比较小,通过 3、怎么解决上面的问题?
  - 加盐,也就是对明文添加一些随机值,然后再加密。
- 一方面,相同的明文得到不同的密文,解决a、b两种情况,另一方 4、对于我们的项目,需要权衡考虑。
  - 这里存在一种假设,那就是第三方只拿到加密后的数据库,并且不过这里涉及到几种场景:
  - a、预期支持同态运算,比如add、serach,不能加盐,也就是不
  - b、前期不确定,开始的时候,添加RND洋葱层,当涉及到同态运算
  - c、预期不支持同态运算,也就是不作为查询条件,添加RND洋葱层
- 5、现在考虑,是否足够安全,假设第三方只拿到加密后的数据库。 为了支持同态运算,不能加盐,也就是没有RND洋葱层,存在一定的 c情况彩虹表只是对于hash算法有效,对于有些算法不能破解,比

特别核心的数据,只是作为查询的结果,建议使用RND层。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

工具软件

# 模块

Clion

CRT

Notepad

SVN

VS2005

Wireshark

# 详细描述

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.



# 重变

## CLion操作的问题

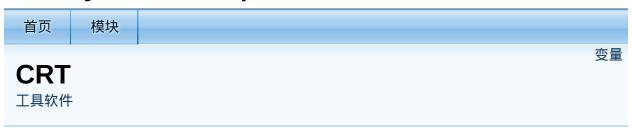
## 详细描述

## 变量说明

### CLion操作的问题

1、CLion操作完全不反应,这是因为选择了vim编辑,只要取消即可Tools --> Vim Emulator 取消

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.



# 重变

secureCRT修改终端外观

使用SFTP下载上传文件

### 详细描述

## 变量说明

#### secureCRT修改终端外观

- 1、点击选项,会话选项只是修改当前连接的属性。
- 2、需要设置全局选项,如下:常规-->默认会话-->编辑默认设置-->约

#### 使用SFTP下载上传文件

- 1、一般情况下,我们使用SecureCRT连接Linux主机,当需要从Linux SecureFX提供了一个UI界面,方便我们操作。
- 2、当我们打开一个SecureFX界面的时候,可以看到多出一个连接SSH2可以认为SecureFX使用FTP over SSH2,也是连接SSH2服务的

```
[root@localhost ~]# netstat -anp|grep :22
                  0 :::22
tcp
           0
                0 ::ffff:10.36.65.60:22
tcp
                52 ::ffff:10.36.65.60:22
tcp
           0
[root@localhost ~]# netstat -anp|grep :22
                0 :::22
tcp
           0
                 0 ::ffff:10.36.65.60:22
tcp
           0
tcp
           0
                 52 :: ffff: 10.36.65.60: 22
                  0 ::ffff:10.36.65.60:22
tcp
           0
```

- 3、当SecureFX不能使用的时候,我们可以使用SFTP(Secure FTP) 作为客户端FTP不需要新建一个TCP连接,而是直接使用当前Secu
- 4、具体操作如下:
  - a、右击当前SecureCRT的连接标签,选择 连接SFTP标签页,也 这个过程,不会新建TCP连接,而且交互是加密的。
  - b、然后作为FTP客户端,可以使用一些FTP的命令,也可以通过he
  - c、下载文件,如下:

```
sftp> get main
       Downloading main from /home/niu11/main
       100% 266KB
                    266KB/s 00:00:00
       /home/niu11/main: 272894 bytes transferred in
       这个时候,不知道下载到本地的哪个地方,使用1pwd,查看>
       sftp> lpwd
       //10.30.21.200/UserDesktop01/25697/MyDoc
   d、上传文件,如下:
       sftp> put desktop.ini
       Uploading desktop.ini to /home/niu11/desktop.
       100% 402 bytes 402 bytes/s 00:00:00
       //10.30.21.200/UserDesktop01/25697/MyDoc/desk
5、除此之外,还可以使用
   sz 下载
   rz 上传
   在会话选项-->终端-->X/Y/Zmodem看到,上传下载的目录。
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.



# 变量

常用快捷键

如何查看中文的utf8编码取值

## 变量说明

#### 如何查看中文的utf8编码取值

- 1、一般情况下,我们在notepad++中输入中文,拷贝出来,都是gb23:如何查看中文的utf8编码取值?
- 2、在notepad++中,选择格式,以utf8格式编码,然后输入中文,比这个时候拷贝到网络助手或者科鼎转换工具,转换为16进制,可以把utf8格式的中文,当成gb2312解释,选择格式,以ansi格式编把这3个中文,拷贝到网络助手或者科鼎转换工具,转换为16进制
- 3、道理是什么样的? 在utf8格式下,输入"广播",这个时候不能拷贝到科鼎转换工具, 而是采用ansi编码去解释utf8编码的"广播",因为utf8每个中文 当成ansi解析,会有三个乱码的中文字符,这时候再拷贝到科鼎转 注意:ascii是单字节编码,使用一个字节表示一个字符,而ansi 这里的ansi可以认为就是qb2312
- 4、明白了上面的道理,还有另外一种方式,查看中文字符的各种编码。新建一个txt文件,输入"a广播",使用UE打开该文件,使用16进然后另存为utf8,可以看到16进制为 EF BB BF 61 E5 B9 BF 然后另存为unicode,可以看到16进制为 FF FE 61 00 7F 5E 然后另存为unicode big endian,可以看到16进制为 FE FF 注意:大端是从前往后读,小端是从后往前读,广泛使用的是小端。这里可以看出,ansi和utf8的优点,对于ascii的英文字符,使是而unicode也就是utf6,固定使用两个字节表示一个字符,即使是

### 常用快捷键

- 1、列模式,按住alt,再操作鼠标
- 2、查找,下一个【F3】 上一个【shift+F3】

- 3、选中字符, Ctrl+u 转化为小写, Ctrl+Shift+u 转化为大写, Sh
- 4、查找的时候,一行出现多个结果,查找的结果列出多次,如何解决? 第一种办法,扩大搜索的关键字
  - 第一种办法,使用正则表达式如下:^(.\*abc.\*)+\$, 仔细思考abc 匹配最后一行中的最后一个abc,由于notepad 本来就是按
- 5、想把 AAA\_123 BBB\_456 CCC\_789 替换为 123:AAA 456:BBB 查找 ([A-Z]+)(\_)([0-9]+) 替换为 \$3:\$1
- 6、注意小括号与中括号的区别,小括号表示整体,中括号表示集合
- 7、删除包含abc的行

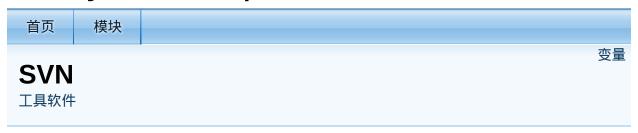
查找 ^.\*abc.\*\$ 替换为 空,发现包含abc的行,变成了空行, 需要把换行符一起换掉,查找 ^.\*abc.\*\$\r\n 替换为 空

- 8、删除空行
  - 查找 ^[ \t]\*\$\r\n 替换为 空,这种方式,匹配了行结尾的位
- 9、删除非e开头的行

查找 ^[^e].\*\$\r\n 替换为 空,这种方式,匹配了行结尾的位

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.



# 重变

svn图标显示不正常

## 详细描述

## 变量说明

#### svn图标显示不正常

- 1、在本地提交之后,文件还是显示红色感叹号,多次刷新之后还是红色2、这个时候,使用clean up一下就好了。如果当前文件没有clean u
  - Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.



# 重变

展开合并 引用变量 查看内存 错误查找 统计代码行数

## 详细描述

## 变量说明

#### 展开合并

右击 - - > 大纲显示 - - > 切换所有大纲显示

参见

#### 引用变量

使用 \$(var)

注意:在dos下引用变量使用 %var%

#### 查看内存

菜单-->调试-->窗口-->内存-->内存1

参见

### 统计代码行数

使用正则表达式:^[\t]\*[^#/].\*\$

行头 行尾

[\t]\* 开头有0到多个空格或者制表符

[^#/] 紧接着的字符不是 # / ,也就是排除 #include 和 // ネ .\* 任意字符,0到多个,不能使用.+,因为[^#/]已经匹配了一

注意:对于下面的情况,不能排除aa 和 \*/两行

```
/*
aa
*/
```

参见

## 错误查找

菜单-->工具-->错误查找

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.



# 变量

查找关键字

Wireshark抓取本地回路包

显示过滤

抓包过滤

## 详细描述

## 变量说明

#### Wireshark抓取本地回路包

- 1、cmd, ipconfig查看自己的ip地址。
- 2、执行命令:route add 10.36.65.89 mask 255.255.255.255 其中10.36.65.89是目的地,这里填写本机地址,表示到达本机地。 可以理解为到达目的地,经过几个路由器。
- 3、route print。查看结果。

#### 抓包过滤

只抓取自己关心的数据包。有时候抓取所有的包,然后显示过滤,导致排特别注意:抓包过滤和显示过滤的语法不一样。

显示过滤 tcp.port == 9845,而抓包过滤是 tcp port 9845。witcpdump -i any -s 0 tcp port 9845 -w test.pcap // 抓能tcpdump -i any -s 0 tcp port 9845 -A // 抓能抓包过滤设置:Capture-->Options-->Capture Filter,输入抓

语法:Protocol Direction Host Value 例子:tcp dst port 80

#### Protocol(协议):

可能的值: ether, fddi, ip, arp, rarp, decnet, lat, sca 如果没有特别指明是什么协议,则默认使用所有支持的协议。

#### Direction(方向):

可能的值: src, dst, src and dst, src or dst 如果没有特别指明来源或目的地,则默认使用 "src or dst" 作为关例如, "host 10.2.2.2"与"src or dst host 10.2.2.2"是一样

Host:

可能的值: net, port, host, portrange.

如果没有指定此值,则默认使用"host"关键字。

例如 , "src 10.1.1.1"与"src host 10.1.1.1"相同。

注意:Host的取值和协议有关,前面是ip,对应host,前面是tcp,u一个很实用的选项是端口范围,如下:tcp portrange 9530-9538

Logical Operations (逻辑运算):

可能的值:not, and, or.

否("not")具有最高的优先级。或("or")和与("and")具有相同的优势例如,

"not tcp port 3128 and tcp port 23"与"(not tcp port 3 "not tcp port 3128 and tcp port 23"与"not (tcp port 3

注意:可以使用&& ||! 代替 and or not

#### 显示过滤

- 1、在抓包结果中进行过滤,显示自己关心的内容。
- 2、显示过滤的语法比较简单,而且有提示。点击expression可以查看 比如输入tcp,可以定位到tcp,然后对字段取值。
- 3、ip过滤:
  - ip.addr == 192.168.1.101
  - ip.srcaddr == 192.168.1.101
  - ip.len > 0 // ip数据长度大于0
- 4、tcp过滤:
  - tcp.port == 9820
  - tcp.dstport == 9820
  - tcp.flags.fin == 1
  - tcp.flags.reset == 1 // 这个数据包显示红色
  - tcp.len > 0 // tcp数据长度大于0
- 5、逻辑操作可以使用and or not 也可以使用 && ||!
- 6、过滤后数据进行保存,选择文件-->导出特定的包
- 7、注意:telnet,Follow TCP Stream显示数据是重复的,但是抓

### 查找关键字

1、ctrl+f, by String , Search in Packet bytes , Filter \$ 2、下一个【ctrl+n】 上一个【ctrl+b】

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

心得体会 で量

# 重变

多读书
一人独处
工作报酬
项目债务
异化 异化
公司与员工的想法
职业倦怠
工作时间
彼得原理
老板和员工
跟我竞争的人
职业规划
<ul><li>纯粹的代码</li><li>—————————————————————————————————</li></ul>
员工离职的原因
好的管理是坏公司的结果
就业有两点需要考虑
自制节制

不要考验人性 自律自信自由 招人和买东西 皮格马利翁效应

## 详细描述

### 变量说明

#### 一人独处

一人独处时能超然物外,与人相处时能和蔼可亲 无所事事时能语陌沉静,处理事务时能雷厉风行 得意时能淡然坦荡,失意时能泰然处之

#### 不要考验人性

1、再亲密的人也会反目,再疏远的人也会靠近,只是要看在什么场景下

#### 公司与员工的想法

- 1、屁股决定脑袋,站在不同的立场,看问题得出的结果不一样,这不是
- 2、公司的想法:假定员工在公司长期工作,先保证公司的利益,公司发员工的想法:从契约关系看,我和公司是平等的合作关系,为什么要
- 3、员工和公司是阶段性的合作关系,在这个阶段,公司没有耐心等待员公司不想做,个人不会做。
- 4、对于职务和薪资,公司只能给出循序渐进式的发展,而员工更渴望跳

#### 参见

#### 员工离职的原因

有句话说,"不满意现在,原因却在过去。"

员工离职的原因是什么?

答案基本上是,家里有事,薪水太低。这些答案偏离了问题的本质。

而应该问:什么场景下,员工不会离职?

必须满足两个条件:员工觉得公司有发展,员工觉得自己有发展。

#### 多读书

1、培根说:

读史使人明智, 读诗使人灵秀, 演算使人精密, 哲理使人深刻, 伦理学使人庄重, 逻辑修辞使人善辩, 这就是读书的魅力。

2、为什么大家一起上班,一起吃饭,一起生活,但是你觉得你和他们不因为"你必须只有内心丰富,才能摆脱这些生活表面的相似"。

#### 好的管理是坏公司的结果

好的管理是坏公司的结果,好的公司往往没有好的管理。 为什么需要好的管理,因为基于对人的不信任,所以才要制定出各种各模

#### 就业有两点需要考虑

- 1、企业需要什么样的人
- 2、我是什么样的人
- 3、不要抱怨环境,想办法提高自己

#### 工作报酬

- 1、你的报酬不是和你的劳动成正比,而是和你劳动的不可替代性成正比
- 2、不要以为你辛苦,做的事情多,你就应该挣得多,你的不可替代性在
- 3、工资不是用来支付给技能的,不要以为技能越高、工资自然应该越高工资是用来支付给责任的,责任越大,工资越高。
  涨工资,是因为承担了更大的责任。

#### 工作时间

- 1、站在领导的角度,工作时间全部做本职工作,不停地给员工新的任务
- 2、其实这个管理有问题,站在员工的角度,我干什么做这么快,反正任

#### 参见

#### 异化

- 1、一个政策的出发点是好的,但是在实施的过程中会走向反面,因此不
- 2、举例来说,公司制定制度,要求员工一定要负责,提交给自己的问题如果问题不好解决,员工的做法不是想尽一切办法解决,而是想办。
- 3、再比如,公司限制员工的离职,你知道员工是怎么想的吗? 员工想,限制我的离职就是想长期剥削我,那我要未雨绸缪。即使

#### 参见

#### 彼得原理

- 1、在组织的等级制度中,员工向往更高的位置,由于某种特质或者特殊相反成为组织的障碍物或者负资产。
- 2、好的程序员为了高薪,选择往管理层升迁,变成管理层不需要写代码
- 3、另一方面,好的程序员不代表好的管理者,管理绩效会很差,但是回

#### 参见

#### 招人和买东西

- 1、公司招人和我们买东西类似,买东西要看自己的需求,是愿意买质量
- 2、价格贵的,除了价格贵,其他都是优点。价格便宜的,除了价格便宜
- 3、就像买手机,相对于苹果手机,小米手机该有的功能都有,性价比好

#### 皮格马利翁效应

- 1、皮格马利翁是古希腊神话中的一个雕塑家,雕塑了一个女神,假定她
- 2、皮格马利翁效应是说,你假定对方是什么样的人,这种假定会潜移默

这也可以说是,暗示的力量。

- 3、教育小孩子也是这样,你假定他什么都不会做,什么都要教,他慢慢
- 4、自由即责任,所以大家畏惧它。如果没有能力驾驭自由,自由便是负
- 5、要让小孩子知道,他想要什么,明白自己在做什么,为自己的行为负

#### 纯粹的代码

纯粹的代码没有意义,属于自娱自乐,解决现实问题才是软件存在的最大而现实问题来自各行各业,所以大部分的软件产品,是行业需求和软件技

#### 老板和员工

- 1、对很多老板来说,程序员就是水,没有的时候很痛苦。有了的时候没什么?水卖这么贵?不就是水而已吗?老板往往以为什么水都能解了 其实,水只是表面上的相似,成分有很大的不同。
- 2、老板总以为程序员是固定成本,而成本控制得越低越好,换成我是老老板想的是省钱,这就导致双方对立无解。一个理想的状态是,程规程序员当做潜力股,会收益更大的利益。

#### 参见

#### 职业倦怠

- 1、工作久了就会有职业倦怠,没有治疗职业倦怠的灵丹妙药。
- 2、转变心态,给别人打工,当成为自己积累资源和经验,这样以后的筹
- 3、职业倦怠是因为时间久了,自己感觉总在做着重复无趣,又耗费心力 而这个人工作久了,也会产生职业倦怠。
- 4、这就存在一个问题:

会做的人,由于职业倦怠,感到无趣,不想去做,会拖延。 想去做的人,由于不熟悉业务,不会做,做的过程,遇到层层障碍。 解决办法:

公司的解决办法:新人带旧人,衔接好。

5、感觉职业倦怠的重新找一个工作,点燃自己的工作热情。

#### 参见

#### 职业规划

- 1、具体的目标往往不靠谱,以我最近10多年的经历来说。高考的时候到结果给我调剂到医学。读大学时,接触到计算机,感觉这玩意有意。读研的时候,正是Java语言火的时候,学习Java,毕业之后,应即对着到大华,使用C++从事服务的开发。
- 2、因此,只有有个大致的目标就行了,过程中随时调整。比如说,你定如果你是死脑筋,就在这条路上等着。其实这个时候,需要调整路; 这间店关门了,那你这个目标还有意义吗?聪明的做法是调整目标
- 3、工作的选择是多方面权衡的结果,比如说工作轻松和薪资待遇。要看有时间陪家人和孩子。如果很缺钱,就会拼命工作挣钱。

#### 自制节制

- 1、做人要自制节制,君子,绅士,精英都是自制节制能力很强的人,能
- 2、很多事情可以由着性子去做,但是要清楚知道,必须承担这样做的后
- 3、心理学上有舒适区,学习区,恐慌区,舒适区是不会进步的,恐慌区 只有处于学习区,才能不断地进步。
- 4、有三个自己,别人眼中的自己,自己眼中的自己,上帝眼中的自己。自己认为遭受到的不公平,往往是因为自己眼中的自己与别人眼中
- 5、对待别人,把心放宽,学会理解别人,自己的格调要高一些。有时候

#### 自律自信自由

- 1、自律是对自己的控制,自信是对事情的控制。
- 2、厉害的人都是狠角色:虐他和自虐。吃的苦中苦,方为人上人。 科比的凌晨四点钟洛杉矶,雷阿伦从不喝咖啡,德约科维奇从不吃。
- 3、自律让你自信,自信让你自由。生活容易吗? 可以放任自己,也可以严格要求自己,甚至残酷要求自己。 选择必有代价。
- 4、生活工作学习,都是逆水行舟,不进则退。没人能够稳定在某个位置
- 5、人生的时间应该只花在两个地方:享受生活和为了享受生活而奋斗。
- 6、自律是来自自己内心的要求,而不是外部的要求。外部的要求往往不 好的小孩教不坏,坏的小孩教不好。
- 7、人到成年以后,很少有人抱怨说小时候父母对自己要求严格。 相反,很多人会说,如果父母当年对自己要求更严格,自己现在的

看了郎朗的锵锵三人行,心里特别感激小时候父母对自己近乎残酷 你可以想象,一个小孩子,每天被父母要求练习钢琴,而身边的小, 小孩子心里是多么的恨父母,想离家出走,想换个父母。

#### 跟我竞争的人

- 1、不要抱怨公司,市场经济,契约精神,不是公司跟你竞争,而是其他
- 2、站在公司的角度,你不干,有人干。
- 3、换个角度,公司也不应该抱怨员工的离职,你不给的薪资,有人给。

#### 参见

#### 项目债务

- 1、每次做项目,为了速度,不经过严格的测试,发到现场。
- 2、然后做下一个项目,做下一个项目的时候,修改前一个项目的Bug,
- 3、造成恶性循环,维护以前的项目,导致时间人力成本更大。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

<sup>首页 模块</sup> 架构和设计

# 变量

为什么new出来pe协议,打包的时候调用dhtp协议的serialize方法 理解消息模式的线程处理 网络数据的打包解析 以scs为例说明网络包的接收和发送 连接和登录的区别 底层库局部更新的办法 主备构架iptables设置 理解CComProcessorBase类 CComProcessorBase使用说明 一主多从 负载均衡 一主一从 主从切换 负载均衡 项目中大数据的处理 设备增删改的影响 发送消息的peer是怎么来的 服务sdk发送消息的session是怎么来的 rpu2000T有关

dmu\_sdk同步与异步

负载均衡的两种场景

理解rtsp的传输模式

消息交互异步存在的问题

原始请求消息的保存

如何确定业务服务器在哪个板卡上

# 变量说明

#### CComProcessorBase使用说明

1、start方式

start(0): PushMsg的时候,触发run()-->Heartbeat()-->

start(1):while死循环,定时触发

2、创建定时器

threadInit() 创建定时器,定时器属于当前线程 其它地方创建定时器,定时器属于主线程

3、在threadInit() 创建定时器,可以避免多线程竞争的问题,但是

#### 参见

# dmu\_sdk同步与异步

- 1、整个框架都是基于消息的,每个消息处理单元继承CComProcessor 启动线程有两种方式:
  - start(0) 基于事件, push进去一个消息,调用notify,然后回start(1) while死循环,处理消息队列中的消息,防止cpu占用
- 2、事件驱动, SendMsg-->PushMsg-->notify 回调上来 特别注意: SendMsg的时候,检查方法形参的接收者是否为空,为 然后接收者Push消息进入自己的消息队列。 当前线程SendMsg,不是Push给自己(Push给自己没有意义),
- 3、sdk异步的过程:

比如当前模块A关联outprocess,并设置outprocess的接收者是初始化设置异步回调,A模块线程SendMsg,关联outprocess,Foutprocess处理消息DealWithOneRequest,调用底层库底层库线程收到消息,然后解析,onPDU回调上来,给outprocess

Push给模块A,模块A回调上来,处理消息DealWithOneRes实际情况还要多一层关联,dmu\_session关联sdk\_manager,sc发请求,dmu\_session模块SendMsg直接调用m\_sdkManager模k如复,outprocess的OnPDU上来,SendMsg时Push给SdkMan

4、sdk同步的过程:

同步是对异步的封装,在发送消息和接收消息的两个口子,处理不 发送消息:使用P操作(检查超时)

接收消息:设置异步回调,异步回调上来,进行V操作。然后发送;注意:同步与异步,设置的异步回调不一样。同步设置的异步回调

5、SendMsg是对PushMsg的封装,对于请求消息,SendMsg的时候保证

# rpu2000T有关

- 1、rpu2000T对外(也可以理解对上)暴露口子uac,uac向上级uas注
- 2、uac下面是我们的IBP平台,包含cmu,dmu,vtdu等,uac登录IBF
- 3、IBP平台下面是uas,设备分为三种情况:

本级域设备,dmu直接登录

下级域设备,dmu通知uas去登录

还有一种设备,现在nvr上添加设备,然后nvr向uas注册,推送设4、请求rtsp的流程,要经过uac,获取到rtsp之后,根据rtsp的地址

# 一主一从\_\_主从切换

- 1、通过hau心跳服务器,彼此通过udp,监听端口,和向对端发送信息。如果一段时间没有收到对端的master报告,自己升级为master,这里存在一个极端的情况:两个服务同时启动,都收不到对端的ma然后同时升级为master,并向对端报告自己是master,这就出现
- 2、怎么解决这个问题?
- 3、增加一个比较项,也就是两个服务设置不同的等级。发送master报台 当服务自己是master,同时又收到对端的mster报告,比较自己的 等级低,自己降为slave。这样保证两个master,其中一个自动阶

# 参见

# 一主多从 负载均衡

有个中心管理器,连接一组dmu,决定哪个是master,其他的是s.根据回复的先后顺序,进行排序,第一个为master,第一个挂了,

参见

# 为什么new出来pe协议,打包的时候调用dhtp协议的serialize方法

```
1、new ibpSocket的时候,创建peParser,然后peParser创建dhF
2、dhParser在构造方法中,加载动态库new_dhtp_protocold.dll
   定位到方法 void* pFun = pMa->dllsym(fileName, "pro
   方法原型为 DLL_EXPORT ibpProtocol* protocol_factor
   放入集合vecICreate
3、发送数据,打包的时候调用dhParser::createCopyProtocol(i
   obj = ((i_protocol_factory)m_vecICreate[i])(proto
   根据协议的命令码和类型(请求、回复、通知)创建出dhtp协议
   m_vecICreate[i] 就是dhtpProtocol.cpp中的方法protocol
   强转为方法i_protocol_factory,然后调用,传入参数(proto
4、在方法protocol factory中,
   ibpProtocol* retPdu = NULL;
   const size t count = sizeof(DHCP ALL PROTOCOL UN]
   const DHCP_OPERATION_UNIT* opUnit;
   for(size_t i = 1; count > i; ++i)
    {
       opUnit = &DHCP_ALL_PROTOCOL_UNITS[i];
       if(opUnit-> opType == command)
       {
           if(eIbpCmdType_Request == etype)
               retPdu = opUnit->_reqFactory();
           else if(eIbpCmdType_Response == etype)
               retPdu = opUnit->_resFactory();
           else if(eIbpCmdType_Notify == etype)
               retPdu = opUnit->_notifyFactory();
           break;
       }
   在这里遍历数组DHCP_ALL_PROTOCOL_UNITS,数组DHCP_ALL_
```

```
struct DHCP_OPERATION_UNIT
    {
       enumIbpCmd
                                           _opType;
                                           _opStr;
       const char*
       FunDHCPProtocolRequestFatcory
                                           _reqFacto
       FunDHCPProtocolResponseFatcory
                                          _resFacto
       FunDHCPProtocolNotifyFatcory
                                          _notifyFa
   };
   后面三个字段都是模板方法,如下:
   template<class T>
   ibpProtocolRequset* dhcpProtocolRequestFatcory()
    {
       return new T;
   初始化DHCP_ALL_PROTOCOL_UNITS指定模板参数
   {eIbpCmd_HeartBeat, "HeartBeat",dhcpProtocolReque
    retPdu = opUnit->_regFactory(); 就可以new出dhtp协议
5、从网络上收到数据,是如何解析的?如下:
   t_int32 dhParser::onPacket(prtlParserI *parser, r
    {
       //版本
       //t_int32
                   packVersion = packet->getVersion(
       //命令
       stringI packMethod = "";
       packet->getMethod(packMethod);
       //命令类型 请求 应答 回复
       //t_int32 packType = packet->getPacketType();
       //包序号
       t_uint32 packSeq = 0;
       packet->getSequence(packSeq);
       //模块名称
       stringI packMode = "";
       packet->getModule(packMode);
       t uint32 packSession = 0;
       packet->getSession(packSession);
       //包内容
       int packBodyLen = 0;
```

```
char* packBody = packet->getBody(packBodyLen)
if (parser == NULL || peer == NULL)
{
    LogErr(IBP_UTILS, "parser or peer is null
    return E FAIL;
}
LogDebug(IBP_UTILS, "onPacket method=[%s] net
ibpProtocol* protocol = NULL;
try
{
    char* buffer = NULL;
    rapidxml::xml_document<> obj;
    if((t_uint32)packBodyLen > m_nMaxLenBody)
    {
        delete[] m_pBodyBuffer;
        m_pBodyBuffer = new char[packBodyLen+
        m_nMaxLenBody = packBodyLen+1;
    }
    t_{int32} nCmd = -1;
    t int32 nUnit = -1;
    for (size_t i = 0; i < m_vecINameToInt.si</pre>
    {
        if(0 == ((i_cmd_str_to_int)m_vecIName
            break;
        }
    for (size_t i = 0; i < m_vecINameToUnit.s</pre>
        if(0 == ((i_unit_str_to_int)m_vecINan
```

```
break;
    }
}
for (size_t i = 0; i < m_vecICreate.size(</pre>
{
    protocol = ((i_protocol_factory)m_vec
    if(protocol)
    {
        //多个命令,共享协议体时,需要再次赋值
        protocol-> cmd = nCmd;
        break;
if(protocol != NULL)
{
    protocol->setBody(packBody, packBodyl
    protocol->_sequence = packSeq;
    protocol->_unit = nUnit;
    protocol->_session = packSession;
    buffer = (char*)protocol->getBody();
    if (buffer != NULL && packBodyLen > 6
    {
        memset(m_pBodyBuffer, 0, m_nMaxLe)
        char* end = strstr(buffer + size
        if (end == NULL)
        {
            end = strstr(buffer + sizeof(
        }
        if (end == NULL)
        {
            end = strstr(buffer + sizeof(
        }
        if(end != NULL)
```

```
memcpy(m_pBodyBuffer, buffer,
    obj.parse<0>(m_pBodyBuffer);
    rapidxml::xml_node<>* pbody =
    if (pbody != NULL)
    {
        rapidxml::xml_attribute<>
        if(head != NULL)
        {
            protocol->_takeLen =
        }
    }
else
{
    memcpy(m_pBodyBuffer, buffer,
    //如果下面没有xml直接调用xml.par
    t_uint32 takeLen = 0;
    {
        char tag[64] = \{0\};
        sprintf(tag, "%s=", IBP_1
        char* pTakenLen = strstr(
        if(pTakenLen != NULL)
        {
            pTakenLen += strlen(t
            if(pTakenLen)
            {
                char* pStart = p1
                char* pEnd = strs
                char szLen[32] =
                if(pStart != NULL
                {
                     memcpy(szLen,
                     takeLen = atc
                }
            }
        }
```

```
if(takeLen == 0)
            obj.parse<0>(m_pBodyBuffe
        else
        {
            char* tmpBuffer = obj.al]
            memset(tmpBuffer, 0, pack
            memcpy(tmpBuffer, m_pBody
            obj.parse<0>(tmpBuffer);
        }
        rapidxml::xml_node<>* pbody =
        if (pbody != NULL)
        {
            rapidxml::xml_attribute<>
            if(head != NULL)
            {
                protocol->_takeLen =
            }
        if(takeLen != protocol->_take
        {
            protocol->release();
            protocol = NULL;
            onPacketError(parser, pee
            LogErr(IBP_UTILS, "protoc
            return E_FAIL;
        }
    }
}
```

```
int ret = protocol->deserialize(obj);
        if(ret < 0)
        {
            protocol->release();
            protocol = NULL;
            onPacketError(parser, peer, packet
            LogErr(IBP_UTILS, "protocol deser
            return E_FAIL;
        }
    }
    else
    {
        onPacketError(parser, peer, packet, a
        LogErr(IBP_UTILS, "create the protoco
    }
    if(m_pIbpProtocol != NULL && protocol !=
    {
        protocol->_peer = (t_uint32)peer;
        protocol->_transId = packet->getTrans
        protocol->addRef();
        m_pIbpProtocol->onPDU(parser, peer, r
        LogDebug(IBP_UTILS, "onPDU ok cmd=[%s
        protocol->release();
    obj.clear();
catch (...)
{
    if(protocol)
        protocol->release();
```

```
}
LogErr(IBP_UTILS, "exception occur! cmd=|
    return -1;
}
return E_OK;
}
```

6、收到数据,首先解析出来命令码,然后和包的类型(请求、回复、通 protocol = ((i\_protocol\_factory)m\_vecICreate[i])( 设置protocol->setBody(packBody, packBodyLen); 然后

# 主备构架iptables设置

- 1、考虑下面的需求,两台设备,部署两套服务,主服务器主备热切换,
- 2、每台设备都有两个网卡,内网网卡和外网网卡,设备之间通过内网通 也就是说,只有一台设备的外网网卡激活。
- 3、比如外网ip 10.65.200.71, A设备内网192.168.1.71,业务服: B设备内网192.168.1.72,业务服务监听地址为9822,考虑ipt:
- 4、现在假设A设备外网可用,外部Client通信地址10.65.200.100,A<-->Client,直接使用外网卡通信B--->Client,B的默认网关为192.168.1.71,A的iptablesisrc[10.65.200.71] dst[10.65.200.100] Client--->B,通过端口区分,A的iptables设置,src[10.65.200.71] dst[192.168.1.72]

# 参见

# 以scs为例说明网络包的接收和发送

```
1、dmu去监听scs,发送peLoginRequest,string dllName = "2. 适配器管理器加载动态库scs_deviced.dll,创建scs_device3、scs_device处理peLoginRequest,创建peOptSocketInfoRec4、outprocess处理peOptSocketInfoRequest,判断socket的fltypedef enum {
    eIbpNet_Invalidate = 0,
```

```
eIbp_TCPClient,
eIbp_TCPServer,
eIbp_UDPClient,
eIbp_UDPServer,
}enumIbpNetType;
创建ibpSocket *pSession = new ibpSocket
创建m_peParser = new peParser(this, dllNames, pro创建m_protocolParser = new dhParser(ipbProtocol,d.
```

- 5、outprocess关联一组ibpSocket,ibpSocket关联peParser,peParser创建prtlCallbackIbp,根据协议类型,如果内部协议 否者是裸的,动态加载方法create parser
- 6、ibpSocket关联ibpNet,ibpNet关联prtlCallbackI【peParsibpNet发送消息的时候,找到peParser,peParser找到SCSCm从framework底层库收到数据,peParser执行onData,找到SC
- 7、协议打包出现问题,有可能是ibp\_utils没有编译好。

#### 原始请求消息的保存

- 1、当前模块转发请求,orgReqMsg是原始请求,reqMsg是新创建的请msg中的协议体是复制指针,而msg是整体拷贝。为什么msg要整体
- 2、因为转发请求,src和dst要发生变化,不能直接修改原始请求信息 发送模块,发送一个请求,sequence+1,要保存原始请求,简单 SendMsg(reqMsg);
  - saveOrgReqMsg(reqMsg.\_moduleSeq, orgReqMsg); // 如 收到回复ackMsg, ackMsg的\_moduleSeq与reqMsg的\_module 根据ackMsg或者reqMsg的\_moduleSeq找到原始请求。
- 3、这里的设计不好,逻辑有点绕,把简单的问题搞的复杂。
  PushOriginalMsg(orgReqMsg) 保存原始请求,返回Id,根据这个Id赋值给reqMsg.\_moduleContext,

  收到回复ackMsg,ackMsg的\_moduleContext与reqMsg的\_moduleContext找到原始请求。
- 4、上面的第一种方法,发送成功再保存原始请求,而这里不行,要先保为什么? 先发送,会保存请求消息【用于处理回复的时候,调用DealWith(

这个时候,保存到vector起来,整体拷贝,\_moduleContext是后面再设置\_moduleContext,已经没有用了,因为vector中保证因此,要先保存原始请求。

5、但是这存在问题,如果发送失败,会造成资源泄露。怎么解决这个问 CheckOutOfDateRequest会定时清除。

# 发送消息的peer是怎么来的

```
1、msg发给outprocess,需要指定peer,outprocess再发出去。
2、outprocess首先根据peer找串口句柄,如果找到直接发出去,并r
3、然后getPeerBySeg根据peer找到realpeer,然后getSessionF
4、ibpSocket关联ibpNet, 在ibpNet调用 m_dhcpParser->seri
   m_dhcpParser是协议解析基类,ibpPacket是数据报
5、outprocess在onConnect的时候保存peer与realpeer的映射关系
6、outprocess在DealCreateSocket的时候,
   ibpSocket *pSession = new ibpSocket();
   SocketInfo info;
   m_mapSocketInfo[pSession] = info;
   在onConnect的时候,保存ibpSocket关联的一组realpeer
   it_map_SocketInfo it = m_mapSocketInfo.find(sessi
   if(it != m_mapSocketInfo.end())
   {
       it->second.insert(peer);
   }
```

# 如何确定业务服务器在哪个板卡上

```
1、以vmu为例说明,首先在主控板,查看cmu连接的vmu在哪个ip地址
   [root@localhost ~]# netstat -anp|grep 9930
                    0 192.168.1.164:59784
                                                1
   tcp
2、查看iptables的DNAT设置,确定端口的映射
   [root@localhost ~]# service iptables status
   Table: nat
   Chain PREROUTING (policy ACCEPT)
                  prot opt source
                                              des
        target
   num
                  tcp -- 0.0.0.0/0
   1
        DNAT
                                              10.
                                              10.
        DNAT
                  tcp -- 0.0.0.0/0
   2
3、连接10.22.4.201的1736端口,即可进入192.168.1.161
```

#### 底层库局部更新的办法

- 1、适配器管理器根据设备类型,创建不同的设备适配器,设备适配器对比如:dhdevice对dhsdk封装,hkdevice对hksdk封装。
- 2、这里存在问题,其他服务使用老的dhsdk,而vru需要使用新的dhsc 新的dhsdk,头文件和库的内容都发生了变化,会导致生成的dhde
- 3、不能影响老的dhdevice编译,因此增加编译配置选项为Ivss\_Debu
- 4、其他服务的编译不用修改,对于编译vru,需要修改的地方:
  - a、编译vru之前,先对老的dhsdk头文件和库,做备份,重命名为
  - b、并且已经生成dhdevice也要做备份,重命名为xxx\_bak,
  - c、然后拷贝新的dhsdk头文件和库,到对应目录,使用Ivss\_xxx
  - d、编译生成vru,将新生成的dhdevice拷贝vru目录中,
  - e、另外vru的启动脚本加载dh sdk中的dhsdk动态库,这里为了
  - f、上面的事情做完以后,还要恢复到原来的样子,也就是把xxx\_
- 5、对于我本地调试,我可以直接覆盖,使用的dhsdk头文件和库,生成

#### 服务sdk发送消息的session是怎么来的

- 1、以Dmu为例说明,DMU\_AC\_Connect异步连接,产生一个dmuSessi\_m\_sdkSessionSeq = SdkDataCenter::instance()->Crea
- 2、保存映射关系,如下:
  - DmuSession\* pDmuSs = new DmuSession(DmuSdkManager SdkDataCenter::instance()->AddSession(pDmuSs); / \*dmusession = pDmuSs->GetSdkSessionSeq(); / 以后客户端请求,使用dmuSession,在sdk中找到pDmuSs,然后
- 3、pDmuSs发送消息,需要指定peer,这个peer就是连接成功,outp protocol->\_peer = m\_peer;
- 5、客户端主动连接cmu之后,m\_session = pResponse->\_session 根据cmuSession获取pSs,然后获取session SdkSessionBase\* pSs = SdkDataCenter::instance()-> ss = pSs->GetCheckSession();
- 6、有一种情况,比如iis服务,不是主动连接cmu,而是cmu主动连接i //不使用CMU登陆时的异步连接专用 usersession 填写用户验i t\_int32 DMU\_A\_ConnectNotCmuHandle(t\_uint32 userse

7、上面的解决办法是投机的做法,更好的解决办法是: cmu连接iis之后,报告这个连接的session,然后iis连接dmu的

#### 消息交互异步存在的问题

#### 1、存在问题:

客户端在vru上配置录像计划,对两个通道录像。vru向vtdu请求 然后分别去连接rtsp,并且设置回调。连接两次,只收到一次回调

2、分析日志发现:

对于第二次的连接,在设置回调之前,就已经收到连接成功onCon

- 3、怎么解决这个问题?
- 4、先设置回调,再去连接rtsp。

但是这里不行,因为必须先连接rtsp,获取到一个句柄,再根据这

5、换一种方法:

在onConnect的时候,检查回调方法,不为NULL,直接回调。为N这还会存在问题,因为这里需要一个集合保存消息,为了同步,需要也是,网络线程onConnect上来,这时候没有设置回调,试图去保但是,但是,但是,这时候主线程设置了回调,尝试到集合中取出然后网络线程获取到锁,进去保存消息,但是这时候已经没有办法。

6、怎么解决上面的问题?

问题的根源是:onConnect在设置回调之前。根本的解决办法是:connect是一个假动作,把句柄返回出来,然后设置回调,设置回真正的connect操作放在设置回调之后。

# 理解CComProcessorBase类

- 1、CComProcessorBase是底层库的基类,对网络库的封装,处理收到 a、创建一个线程,while(1)循环,调用run(),sleep(5),ru b、创建一个线程,调用run(),run调用HeartBeat,下一次再i 注意:HeartBeat从缓冲区取数据处理。可以一次取一条数据,也
- 2、上面两种方式的使用场景不同:

对于while循序,不管什么情况,定时处理缓冲区收到的数据,缺对于第二种方式,使用场景是,触发一下,从缓冲区取数据。比如没有必要while循环去处理缓冲区数据,因为不可能有数据。解决过段时间,这个时候才调用run处理 缓冲区数据。

1、CComProcessorBase发送消息的时候,不是自己直接发送。策略是

- a、通过SetReciver设置一个默认的接收者
- b、SendMsg的时候,也指定一个接收者,如果没有指定接收者,自然后接收者直接PushMsg,放入自己的MsgPool
- 2、一个进程中,会有多个线程,也就是模块,模块之间会彼此发送消息 创建一个Manager,维护所有的模块,这些模块向Manager注册, 并且Manager定义为转发者,调用DispatchMsg。 这些模块发送消息,接收者是Manager,并且Manager是转发者, 而是根据Dst,找到对应的模块,这个模块把消息Push到自己的Ms
- 3、CComProcessorBase处理消息有两种方式:
  - a、定时处理, HeartBeat
  - b、事件触发
- 4、定时处理的流程:

启动线程threadfuncNoEvent-->While循环(内部run,然后s-->run-->Heartbeat-->从MsgPool取出消息-->DealMsg(m也就是:定时从pool取出消息处理,sleep一会,继续

5、事件触发的流程:

启动线程threadfunc-->m\_evnetNotify调用open-->然后设置每次PushMsg(把消息Push到自己的MsgPool),都会调用m\_ev-->run-->Heartbeat-->从MsgPool取出消息-->DealMsg(m也就是:每次Push一个消息(包括请求,回复,通知消息),通知回调的时候从pool取出消息处理,sleep一会,继续

- 6、事件触发的方式,为什么不直接处理?而要绕一个圈子:先放进去, 为了接口统一化,其它模块只需要Push到当前模块就好了,后面是
- 7、之前vmu与vms交互,发现回调上来的序号不对,原因是: 使用了事件触发的模式,但是想vms发送消息,并没有PushMsg的 重写了run,调用CComProcessorBase::Heartbeat(),并且ì

参见

# 理解rtsp的传输模式

```
1、传输码流方式有以下几种:
enum RtpType
{
    RTP_OVER_UDP,
    RTP_OVER_TCP,
```

```
RTP_OVER_RTSP,
      OTHERS_OVER_TCP,
      OTHERS_OVER_UDP,
   };
   RTP OVER UDP: 9836端口的tcp连接走rtsp流程,沟通udp端[
   RTP OVER TCP:9836端口的tcp连接走rtsp流程,沟通tcp端[
   RTP OVER RTSP: 9836端口的tcp连接走rtsp流程,同时码流也
   也就是说,rtsp信令(setup,play,teardown)走在一个tcr
   或者码流和信令走同一个连接。
   注:Others是指PS码流,PGPS是指对PS码流封装了一些头部信息
2、客户端StartVideo的时候,指定传输协议enumTransProto
   typedef enum
   {
      ST_TCP,
      ST_UDP,
      ST RTSP
   }enumTransProto;
   在创建RtspClientKit的时候,根据_transProto转化为RtpTy
3、rstp分为服务端和客户端, vru存在一个问题:
   在rtsp连接, onAccept以后,创建自增Id,然后new MediaSe
   这就导致一个问题,一个rtsp的9836连接,只能回放一路视频。
   而vtdu从vru回放视频的时候,只建立一个rtsp连接,用于多路视
4、怎么解决呢?
```

# 理解消息模式的线程处理

- 1、我们的程序是基于消息,内部流程都是异步的,只要把消息发给目标
- 2、什么要发消息给目标,而不是关联目标,直接调用目标的处理方法?如果使用后者,目标直接处理可能会比较耗时,也就是说卡在这里而发送消息给目标是异步的,当前模块只管发送请求给目标(不会也就是说当前模块不会卡主,但是目标收到的请求比较多,按顺序
- 3、这里还有一个问题,发送消息给目标,如果当前模块关联目标,就会解决办法是:使用中介者模式,每个模块向中介者注册,发送消息发给中介,中介进行转发。
  - 这是在进程中的多个模块之间进行处理,如果通过网络发给对端,一方面,消息接收者需要指定outprocess,同时需要指定消息中区之样才能通过网络发出去。

- 4、服务内部都是异步的,客户端有时候需要调用同步接口,怎么处理?使用PV操作。客户端调用同步接口,发送请求的时候,创建一个信另外有一个线程,处理回复,收到回复之后,把回复保存在map中在这个时候,上面P操作就可以进行下去,然后到map中找到回复,特备注意:这里的PV操作必须是两个线程,否者死锁。
- 如果是一个线程,当前的P操作卡住,等待V操作,而V操作代码在F 5、目标处理消息,要启动一个线程,启动线程有两种方式:
  - a、基于event,先设置回调,每次pushMsg的时候,调用notifyb、使用while死循环,处理消息。

这两种启动方式在CComProcessorBase体现

- // 启动线程,默认O启动EventManager 1启动while 调用 r virtual bool start(int iStartFlag = 0);
- 6、每个目标启动一个线程,存在问题:

目标很多,特别是存在设备的情况下,线程特别多。我们知道一个完全要的dows下线程栈默认是1M, Linux下线程栈默认是10M,可

7、怎么解决上面的问题?

不能每个目标启动一个线程,而是创建一个线程,管理一组目标,调用每个目标的heartbeat处理消息。也就是说,消息还是发给目也就是说,目标处理消息,有两种驱动方式:

- a、自身驱动,线程个数比较多
- b、外部驱动,一个线程管理一组目标。

这两种方式在CDeviceBase: public CComProcessorBase中 virtual t\_int32 startDriver(void)=0;

自身驱动在startDriver中调用 ibpThreadBase::start();

8、对于外部线程驱动的流程是:调用适配器管理器的时候,会初始化登添加一个设备,会分配给一个负载最小的登录线程。一个登录线程 对于登录成功的设备,从登录线程移除,放入到负载最小的工作线

# 参见

# 网络数据的打包解析

- 1、一个进程内的线程之间通信,没有经过网络传输数据,不需要打包解
- 2、如果通过网络传输数据,需要打包解析。TCP和UDP是一种流协议,注
- 3、应用程序发送数据和接收数据都是直接对结构体操作,结构体提供打 发送数据:应用程序把结构体传给底层库,底层库调用结构体的打

接收数据:底层库收到数据,对数据进行解析,得到一个结构体,

- 4、底层库负责发送和接收数据,但是不知道怎么打包和解析,因此会关 应用程序需要提供一个具体的解析器,让底层库来调用。
- 5、底层库关联peParser, peParser关联prtlCallbackIIbp\*
  而class dhParser: public prtlCallbackIIbp
  在与外部通信的时候,创建socket的时候,指定协议库pSocket]
  动态加载,根据协议库的dll名称,创建出m\_protocolParser:
- 6、收到数据,底层库处理Input 会执行 dhParser::onPacket ---发送数据,outprocess发送一个消息,ibpSocket::sendMsgF
- 1、ibpNet关联底层库,把peParser传递过去,peParser关联ptrl(
- 2、ptrlCallbackIIbp的子类有dhParser, ScsCmdParser, Pion 通过接口create\_parser 暴露出去
- 3、以ScsCmdParser为例,ScsCmdParser关联ibpProtocolI(也能 ibpProtocolI的子类有ibpSocket和ibpCom
- 4、outprocess关联一组ibpSocket,创建解析器的时候,把ibpSoc
- 5、底层库收到数据, onData找到peParser, 找到ScsCmdParser,
- 6、ibpSocket的onPDU回调给outprocess, outprocess把消息Pus
- 7、scsDevice继承CComProcessorBase,在虚方法DealWithOneRe
- 8、注意:在ScsCmdParser,对于收上来的数据,每个连接(Peer)1 当数据累积到一个完整的消息(消息头包含消息长度),解析,回

# 设备增删改的影响

# 1、存在问题:

在设备存在的情况下,配置了很多信息,比如上墙的设备,配置了是已经配置好的信息怎么办?

2、根据实际的需求来处理。有些情况,不允许删除设备,先解除关联,

# 参见

# 负载均衡

# 1、解决什么问题?

一个域(可以理解为区域)下面有很多设备,需要对这些设备管理问题是设备很多,一个dmu管理不过来,需要一组dmu分布式部署,master由cmu来指定,slave去连接master。

#### 存在两种情况:

- a、自上而下的控制命令,客户端发送控制命令给master,maste 类似LVS的NAT模式(Network Address Translation,
- b、自下而上的告警收集,客户端向master请求所有的slave,然 类似LVS的IP隧道模式(IP Tunneling,IP隧道模式)。 还有一种DR模式(Direct Routing,直接路由模式),和

还有一种DR模式(Direct Routing,直接路田模式),和 再考虑级联的情况,dmu除了直接管理的当前域设备,还要管理下约 举个现实的例子,一个省有一个省长,多个副省长。下面有多个市

对于当前域,负载均衡的输入是当前域的一组dmu,和总的设备。其和各个下级域的一组dmu服务以及对应管理的设备。这里有分为两利对下级域的所有设备拿过来分,还是把下级域的dmu服务作为一个证理论上应该是每个dmu负责50个。实际上没有必要,可以上级一个因为瓶颈在下级dmu,上面dmu负载均衡也没有特别的意义。

- 2、怎么解决?
- 3、不考虑权重,没有记忆功能,每个dmu应该负责的设备个数 (DevSi 比如当前两个dmu,60个设备,负载后,d1[1-30],d2[31-60] d1[1-20],d2[21-40],d3[41-60],但是这样会导致:
  - d1登出10个设备
  - d2登出20个设备,同时登入10个设备
  - d3登入20个设备
- 性能不好,理论上应该是d1[1-20],d2[31-50],d3[21-30,54、这就需要有记忆功能。思路是:第一个平均分配,以后每次负载都要a、设备增加,计算出每个dmu应该负责多少设备,遍历dmu,不够b、设备删除,去掉设备,计算出每个dmu应该负责多少设备,遍历dmu,逐个c、dmu增加,计算出每个dmu应该负责多少设备,遍历dmu,逐个
- d、dmu删除,负责的设备拿出来,再逐个补充。 5、上面没有考虑权重,有的设备只有一个通道,而有的设备可能有几十
- 思路:应该负责多少个,当前负责多少个,找出一个最接近的补充。 考虑极端的情况,权重分别为:100,4,3,2,1 3个dmu负载。理想 按照上面的思路,应该负责37,结果是d1[100], d2[4,3,2,1]
- 6、考虑权重,再考虑记忆功能,和上面的情况类似。
- 7、负载均衡,由master调用,好了之后,告诉每个slave负责的设备

#### 负载均衡的两种场景

- 1、考虑dmu的负载均衡,也就是M个dmu去分N个设备。Master进行负载分好之后,通知每个Slave,然后Slave对负责的设备进行登录。
- 2、问题来了,当设备增删改的时候,怎么办? 设备的增删要重新负载均衡,Master重新负载均衡之后,再次通知 计算出应该登录哪些设备和应该登出哪些设备,进行处理。设备的 如果登录信息发生变化(比如ip、port、登录名和密码)需要登出
- 3、考虑vru的负载均衡,vru负责录像和回放。开始的时候是不知道要i 处理策略是每次收到一个新的录像或者回放的请求,找出当前负载:
- 4、这里存在问题,负载均衡的时候,需要知道每个vru的负载信息。怎一种办法是:可以在Master端进行管理,发给对应Slave录像或者当Slave完成录像或者回放的时候,对应Slave的负载减1.
  - 另一种办法是:Slave端保存自己的负载信息,定时向Master汇扎

#### 连接和登录的区别

- 1、连接就是网络层建立tcp或者udp连接
- 2、登录是指在连接成功之后,输入用户名和密码

# 项目中大数据的处理

- 1、主要是实时数据和告警。以告警为例,自下而上收到的告警非常多,
- 2、从源头上过滤,一段时间的重复告警过滤掉,过滤策略可以针对告警比如移动侦测告警,小区门口特别多,仓库里就很少。小区门口,
- 3、按照客户端的订阅,来上报给客户端。减少网络流量,进入页面订阅
- 4、告警的写入,缓存起来,批量写入。
- 5、历史告警会查询,因为查询条件肯定带着时间段。因此按时间段建立
- 6、实时数据的平均值,比如温度,求一天每个小时的平均问题,一个月利用mysql的时间调度器,定时求出平均值放入平均值表中,以后

# 参见

# Andy Niu Help 1.0.0.0

直页 模块 模块 模块 | 变量

# 模块

常用正则表达式

# 重变

特殊字符和转义	
匹配优先和忽略优先	
括号的用处	
环视	
分组_捕获_反向引用	
分组_不捕获	
固化分组	
占有优先	

# 详细描述

# 变量说明

#### 分组 不捕获

- 1、在分组\_捕获\_反向引用一节中,我们对每个分组都进行了捕获,比如使用([a-zA-Z]+)(\d)(\1)(\2),(\1)(\2) 我们不需要使用引擎必须记住(\1)(\2)的内容,这会导致额外的开销。
- 2、怎么解决?

解决办法是使用分组不捕获,如下:

 $([a-zA-Z]+)(\d)\1\2$ 

注意:这个时候使用notepad++进行替换的时候,只捕获了两个分没有\$3和\$4,也可认为\$3和\$4对应为空。

- 3、考虑下面的需求,找出[a-zA-Z]+\d[a-zA-Z]+,其中[a-zA-Z]·([a-zA-Z]+)(?:\d)\1
- 4、在notepad++,匹配的时候使用\n反向引用第n个捕获到的分组,程 有些程序支持命名捕获分组,对捕获到的分组,取一个名字,后面
- 5、注意:\d等价于[0-9],还有一个常用的缩写\w,但是不推荐使用\ 有些等价于[a-zA-Z0-9],有些等价于[a-zA-Z0-9\_],有些等价 为了兼容,不推荐使用\w

#### 参见

分组 捕获 反向引用

# 分组\_捕获\_反向引用

1、考虑下面的需求,对于文本 Andy1Andy1 Bill3Bill4 Caroine5Caroine5 找出重复出现的[a-zA-Z]+\d,也就是说Andy1Andy1匹配成功,

2、使用分组\_捕获\_反向引用,如下:

 $([a-zA-Z]+)(\d)(\1)(\2)$ 

(\1)反向引用([a-zA-Z]+),然后本身又作为一个捕获分组。

- 3、这里捕获了4个分组,使用notepad++进行替换的时候,\$1对应([a \$3对应(\1),\$4对应(\2)
- 4、考虑下面的需求,找出连续出现3次[a-zA-Z]+\d的文本,如下: ([a-zA-Z]+)(\d)(\1)(\2)(\1)(\2),当然也可以使用 ([a

#### 参见

分组 不捕获

#### 匹配优先和忽略优先

- 1、考虑下面的需求 , "Tom" is short for "Thomas" 找出所有的
- 2、容易想到的办法:查找 ".\*" 发现找出的结果是 "Tom" is shor
- 3、这不是我们所期望的,为什么?因为.\* 是匹配优先,即贪婪模式,也就是说会尽量多吃。
- 4、怎么解决这个问题?

使用忽略优先,也就是尽量少吃,如下:".\*?"对于不支持忽略优先的程序,可以使用"[^"]\*",实现忽略优先。"[^"]\*"表示可以尽量多吃,但是不能吃",遇到第一个"就要停

# 参见

# 占有优先

- 1、占有优先和固化分组一样,解决同样的问题,吃下去,丢弃备用状态
- 2、占有优先的语法如下:

([a-zA-Z]++):

# 参见

# 固化分组

- 1、考虑下面的需求,使用[a-zA-Z]+:去匹配文本,对于文本Subjection
- 2、[a-zA-Z]+匹配优先,一直吃到t,发现冒号不能匹配,于是使用备 还是不能匹配,再吐出一个字符.....一直继续下去。
- 3、这里存在问题,引擎试图吐出一个字符匹配冒号,我们知道这是徒劳 如果正则表达式引擎足够聪明,意识到这一点,就不需要尝试吐出: 但是,一般情况下,引擎没有这么聪明,需要人为的协助,提示引擎
- 4、怎么提示引擎呢?

使用固化分组,如下

(?>[a-zA-Z]+): (?>[a-zA-Z]+)作为一个整体,吃下去文本

- 5、也就是说,使用固化分组可以提前报告匹配失败,不需要进行徒劳的
- 6、对于不支持固化分组的程序,可以使用占有优先和环视。

占有优先,如下 ([a-zA-Z]++):

环视匹配一个位置,有一个重要的特点,就是匹配尝试结束后,不 (?=([a-zA-Z]+))\1: (?=([a-zA-Z]+))匹配一个位置,右

#### 参见

#### 括号的用处

1、限定多选项的范围

gr(a|e)y, 查找 gray和grey, 等价于gr[ae]y

- 2、多个字符括住,合成一个单元,接受量词的限制 (th){2}, 查找th出现量词
- 3、分组,反向引用

表达式:([a-z]{3})([0-9]{2})\1\2 , 匹配下列文本, abc12def12 // 匹配失败 thy34thy34 // 匹配成功

4、环视功能

(?=\d) 顺序环视(右边包含一个数字)

(?<=\d) 逆序环视(左边包含一个数字) =换成! 变成否定环视,左边或右边不 变成否定环视,左边或右边不包含一个数字。

#### 参见

# 特殊字符和转义

- 1、正则表达式中有些字符用作特殊的意义,比如元字符\*表示匹配0个
- 2、这些字符表示特殊的意义,如何表示这些字符本身呢?使用转义字符是字符本身。比如: shell中,双引号用作框住字符串,要表现特别注意:不同程序,所包含的特殊字符不一样。同时是否转义的在notepad++中,+表示匹配1个或多个,\+表示+字符本身在vim中,+就是+字符本身,\+匹配一个或多个

参见

#### 环视

- 1、环视就是看周围的意思,环视匹配一个位置,这个位置的周围满足一 也就是这个位置的左右,包含某些内容,或者不包含某些内容。
- 2、考虑下面的需求,对于12,345,678,修改为 12,345,678。 这个需求要做的事情就是在数字中的一些位置添加逗号,这个位置 前面匹配一个数字,后面匹配三个数字的整数倍,结尾不是数字或:
- 3、解决办法:

查找 (?<=\d)(?=(\d\d\d)+([^\d]|\$)) 替换为逗号。也就是

4、匹配流程如下:

先匹配1,后面有7个数字,不是3的倍数,匹配失败。 再匹配2,后面有6个数字,是3的倍数,匹配成功,替换。 再匹配3,后面有5个数字,是3的倍数,匹配失败。

5、环视还有一个重要的特点,就是匹配尝试结束后,不会留下任何备用

# 参见

固化分组

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

常用正则表达式 正则表达式

# 重变

中文有关

# 详细描述

# 变量说明

#### 中文有关

- 1、对于中文字符,根据unicode的特点,使用[\u4e00-\u9fa5]匹配
- 2、但是在notepad++中,不能这样使用。需要: 菜单-->格式-->转为UTF-8

使用[\x{4e00}-\x{9fa5}]

- 3、[\x{4e00}-\x{9fa5}]匹配中文字符,不包括中文的标点符号, 使用[,。、:]
- 4、[\x{00}-\x{ff}]用于匹配英文单字节字符,那么[^\x{00}-\x{ 中括号的减号表示从xxx到xxx

[\x{00}-\x{ff}]也就是[\x{0000}-\x{00ff}],匹配第0个 [^\x{00}-\x{ff}]也就是[\x{0100}-\x{ffff}],匹配第\x

#### 参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

程序员修炼

# 变量

读取日志的技巧

解决问题的思路和办法
如何快速完成功能
如何定位内存泄露
如何当一个好老师

# 变量说明

### 如何定位内存泄露

- 1、首先分析内存泄露的特点,是缓慢式的,还是脉冲式的。
  - 缓慢式的:内存增长比较频繁,但是每次增长一点。比如每过10秒脉冲式的:平时内存消耗不变化,突然某个时间点内存增长很多(
- 2、如果内存增长比较频繁,而且每次增长很多,往往比较好定位。
- 3、缓慢式的内存泄露,场景往往是:频繁执行new操作,每次分配一点
- 4、脉冲式的内存泄露,场景往往是:经过长时间运行,走到某个分支, 又经过一段很长的时间(或者某个外部条件触发),再次走到这个
- 5、内存泄露的根本原因是:new出来的动态内存,没有释放。因此需要
- 6、开始的时候,定位内存泄露往往没有头绪。最好使用排除法。
- 7、排除法的步骤是:注释掉有嫌疑的代码,观察是否有内存泄露。如果还有内存泄露,说明这块代码没有问题。如果没有了内存泄露,说明就是这块代码的问题,然后采用同样的

# 如何当一个好老师

- 1、知识丰富,并不意味着能当一个好老师。
- 2、给非专业人士讲解概念的时候,按照水平高低,分为3个档次、
- 3、三流老师:讲一大堆细节,列举条例,追求大而全,不重视逻辑和因
- 4、二流老师:善用比喻,讲述幽默,吸引人。
- 5、一流老师:来龙去脉,一气贯通,逻辑缜密,当头一棒。
- 6、一流老师的讲述过程可以抽象为:目的--->产生什么问题--->解决。 因为本质上,人类的所有知识都是为了解决问题而存在的。
- 7、以足球越位为例来说明,很多人搞不清楚怎么回事。按照上面的思路
- 8、目的:足球运动的目的是把足球踢进对方的球门。
- 9、产生什么问题:为了进球,最有效的办法就是:前锋可劲往前钻,最

这样就有很大机会,而且很容易进球。大家都这么踢球,就没有什么 10、解决思路:怎么解决上面的问题?

为了防止这种猥琐的踢球方法,需要制定规则:当队友向前给你传送总得有个标准,标准就是:当队友向前给你传球的时候,除了对方达就是越位规则。

### 如何快速完成功能

- 1、有时候搭建一个测试环境,需要很长时间,在windows下调试比较麻
- 2、在具备条件的时候,可以把服务切换到本地进行调试。
- 3、如果不能切换到本地,就必须直接修改代码,编译出来,放到linux 使用gdb调试比较麻烦,特别是基于多线程的消息机制,更难调试。
- 4、这个时候,就依赖最重要的工具,日志。写日志需要很多技巧,如下
  - a、在重要的分支前,打印判断条件的取值
  - b、方法的返回值重要,多个地方调用,在方法返回之前打印出来
  - c、没有性能问题,日志级别使用Info
  - d、存在性能问题(也就是频繁地执行打印),但是取值又需要观察

### 解决问题的思路和办法

- 1、有时候会遇到莫名其妙的问题,解决起来完全没有思路,不知道从哪
- 2、采用排除法,把可能导致问题的代码注释掉,看看问题是否还存在。 如果问题不存在,就说明这部分代码有问题。通过这种方式逐渐缩。

# 读取日志的技巧

- 1、有时候日志文件太大,打开很慢,可以使用grep对日志文件操作,imore 2016-12-9.log|grep -n "Data For File\[" > 11 注意:-n显示行数,中括号[需要转义
- 2、windows下使用findstr,如下: findstr /i "setRecordId\_ForSlaveChannel" 2016-12-

# Andy Niu Help 1.0.0.0

章页 模块 模块 模块 | で量

# 模块

剑指Offer

智力题

# 变量

main之前之后做了什么

需要多少个桶

见面概率

一篇文档的所有单词倒序

面试过程沟通

# 变量说明

### main之前之后做了什么

- 1、main之前对全局对象进行初始化工作,会调用全局对象的构造方法。
  注意:C++不保证全局对象的初始化顺序,如果全局对象A依赖全局
- 2、main之后对全局对象进行清理工作,会调用全局对象的析构方法。
- 3、main作为程序的入口,只是一种惯例,可以修改,但是不推荐。修改 a、main方法名修改为main\_2,重新编译报错:error LNK2019 b、右击当前工程-->属性-->配置属性-->链接器-->高级-->入[注意:这样修改后,就不能保证main\_2之前对全局对象初始化,m
- 4、你可能奇怪,之前的入口点没有写,为什么还能找到main呢? 这是因为 C/C++-->预处理器-->预处理器定义中有:\_CONSOLE 可以认为 CONSOLE对应 main
- 5、对应关系:

main: Windows控制台程序的入口

WinMain: Windows下GUI程序的入口

wmain: Unicode版本的main,w是宽字符

\_tmain: 是个宏 , Unicode情况下是wmain , 否者是main

- 6、启动控制台程序,打上断点,在调用堆栈中 mainCRTStartup,双调用顺序:mainCRTStartup-->\_\_tmainCRTStartup-->\_in
- 7、也就是控制台程序main方法是由mainCRTStartup调用的,对应关

CONSOLE<--> mainCRTStartup <-->main

WINDOWS<--> WinMainCRTStartup <-->WinMain

DllMain<--> \_DllMainCRTStartup <-->必须使用\_\_stdc

8、如果想main结束后,还执行一个方法,使用atexit和exit,如下:atexit(DoSthAtExit); // 注册,在main结束之后调用,在f/exit(0); //这个调用不是必要的,加在return之前,程序提return 0;

### 一篇文档的所有单词倒序

- 1、一篇文档的所有单词倒序,要求线性时间常数空间。比如: I am Andy 转化为 Andy am I
- 2、解决办法:先对整个文档的所有字符进行倒序,再对每个单词倒序。 I am Andy -->ydnA ma I -->Andy am I
- 3、上面的方法等价于,先对每个单词倒序,再对整个文档的所有字符每 I am Andy -->I ma ydnA-->Andy am I
- 4、这使人想到循环移位,abcdef,向左循环移动两位是 cdefab,有前面2位反转,后面4位反转,再整个反转,如下:abcdef-->bacdef-->bafedc-->cdefab 注意:前面2位反转,后面4位反转,再整个反转,等价于先整个反abcdef-->fedcba-->cdefba-->cdefab

#### 见面概率

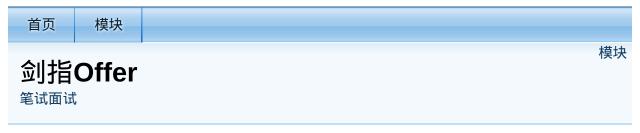
### 需要多少个桶

考虑有 y 升水,每个桶的容量是 x 升,需要多少个桶? 需要 y/x+(y%x==0?0:1); 有没有其它的计算方法? 有,y/x+1能够表示不整除情况,要想同时表示整除的情况,使用(y-1

### 面试过程沟通

- 1、面试官给出一个题目,往往需求不明确,场景范围太大,不要埋头去
- 2、查找一个数组中最大的K个数,这个题目的场景就太大,需要进一步A
- 3、最大的K个数要不要有序?
- 4、数组的规模,如果太大,可能内存都放不下。
- 5、K的大小相对于数组的大小,比如100个数中最大的5个数,还是100-如果是后者,可以筛选出最小的5个数,剩下的就是最大的95个数。
- 6、时间开始和空间开销,更关心哪一点?

# Andy Niu Help 1.0.0.0



# 模块

基础知识

# 详细描述

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

道页 模块 变量 基础知识 笔试面试》剑指Offer

# 变量

读取命令参数并排序输出

# 变量说明

### 读取命令参数并排序输出

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void BubbleSort(int* pInt,int len)
{
    if(pInt == NULL)
    {
        return;
    }
    for(int i=0;i<len;++i)</pre>
        for(int j=0; j<len-i-1; ++j)</pre>
         {
             if(pInt[j]>pInt[j+1])
             {
                 int tmp = pInt[j];
                 pInt[j] = pInt[j+1];
                 pInt[j+1] = tmp;
             }
        }
    }
}
int main(int argc, char* argv[])
```

```
{
    int len = argc-1;
    int* pInt = new int[len];
    memset(pInt,0,len*4);
    for(int i=0;i<len;++i)</pre>
    {
         pInt[i] = atoi(argv[i+1]);
    }
    BubbleSort(pInt,len);
    for(int i =0;i<len;++i)</pre>
    {
         printf("%d ",pInt[i]);
    }
    getchar();
    return 0;
}
```

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

 首页
 模块

 智力题

 笔试面试

# 变量

吃能量豆过桥

老张分牛

猜数字

游戏24点

# 详细描述

# 变量说明

#### 吃能量豆过桥

- 1、小明要过一个桥,桥长80,吃一个能量豆走一米,小明的包最多只能
- 2、逆向思维,要想过桥,必须在20米处,攒下60个豆子。怎么在20处扎
- 3、第一次带60个,在20米处放下20个,然后回来。第二次到20米处,
- 4、因此需要120个豆子。

### 游戏24点

```
4 4 10 10 ==> (10*10-4)/4

1 5 5 5 ==> (5-1/5)*5

3 3 7 7 ==> (3+3/7)*7

3 3 8 8 ==> 8/(3-8/3)

4 4 7 7 ==> (4-4/7)*7
```

# 猜数字

- 1, he\*he = she
  - e\*e末位还是e,只有1和5。

考虑1,h1\*h1结果是三位数的,只有11,21,31都不满足。 考虑5,h5\*h5结果是三位数的,只有15,25,25满足条件。

2、奇怪的数字7

ABCDEF\*2=CDEFAB, ABCDEF为142857 试一下1到6,每个数字除以7

3、一篮子鸡蛋,2个2个数,多出1个,3个3个数,多出1个,4个4个数最小公倍数+1

考虑,2个2个数,多出1个,3个3个数,多出2个,4个4个数,多长

### 最小公倍数-1

### 老张分牛

- 1、老张有三个儿子,7头牛,老张临死的时候立下遗嘱,老大分1/2,表
- 2、按照老张的遗嘱,不能分到整个牛,但是注意:三个儿子加到一起并
- 3、解决办法是:增加一头牛,参与分,但是这头牛是分不走的。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# Andy Niu Help 1.0.0.0

<sup>首页</sup>模块 **初频音频** 

# 变量

上墙有关的问题
理解采样率
理解码流有关的概念
播放抓包得到的码流数据
H264码流数据帧
vtdu打开视频的参数
码流有关的端口
作为客户端理解rtsp流程
理解rtsp的流程
理解rtsp_rtcp
理解我们的rtsp流程【服务端】
理解我们的rtsp流程【客户端】
rtsp有关类的结构
rtsp穿网包
浓缩视频回放的问题
OCX注册打开视频
H264结构

vru码流数据的发送

网页中嵌入播放器控件

H264\_NAUL头的解析

如何查看视频的编码类型

# 变量说明

### H264 NAUL头的解析

1、NAL全称Network Abstract Layer,即网络抽象层。 在H.264/AVC视频编码标准中,整个系统框架被分为了两个层面: 其中,前者负责有效表示视频数据的内容,而后者则负责格式化数:

NAL单元是NAL的基本语法结构,它包含一个字节的头信息和一系列

- 2、如果NALU对应的Slice为一帧的开始,则用4字节表示,即0x0000@NAL Header:forbidden\_bit,nal\_reference\_bit(优先)
- 3、标识NAL单元中的RBSP数据类型,其中,nal\_unit\_type为1, 2 其他类型的NAL单元为非VCL的NAL单元,如下:
  - 0:未规定
  - 1:非IDR图像中不采用数据划分的片段
  - 2:非IDR图像中A类数据划分片段
  - 3:非IDR图像中B类数据划分片段
  - 4:非IDR图像中C类数据划分片段
  - 5:IDR图像的片段
  - 6:补充增强信息(SEI)
  - 7:序列参数集(SPS)
  - 8:图像参数集(PPS)
  - 9:分割符
  - 10:序列结束符
  - 11:流结束符
  - 12:填充数据
  - 13:序列参数集扩展
  - 14:带前缀的NAL单元
  - 15: 子序列参数集
  - 16 18:保留
  - 19: 不采用数据划分的辅助编码图像片段

20:编码片段扩展 21 - 23:保留 24 - 31:未规定

### H264码流数据帧

- 1、H264是新一代的编码标准,以高压缩高质量和支持多种网络的流媒体参照一段时间内图像的统计结果表明,在相邻几幅图像画面中,一而色度差值的变化只有1%以内。所以对于一段变化不大图像画面,随后的B帧就不编码全部图像,只写入与A帧的差别,这样B帧的大规的可以继续以参考B的方式编码C帧,这样循环下去。这段图像我当某个图像与之前的图像变化很大,无法参考前面的帧来生成,那也就是对这个图像生成一个完整帧A1,随后的图像就参考A1生成,
- 2、在H264协议里定义了三种帧,完整编码的帧叫I帧,参考之前的I帧给还有一种参考前后的帧编码的帧叫B帧。H264采用的核心算法是帧帧间压缩是生成B帧和P帧的算法。
- 3、序列的说明

在H264中图像以序列为单位进行组织,一个序列是一段图像编码后一个序列的第一个图像叫做 IDR 图像(立即刷新图像),IDR 图 当解码器解码到 IDR 图像时,立即将参考帧队列清空,将已解码这样,如果前一个序列出现重大错误,在这里可以获得重新同步的一个序列就是一段内容差异不太大的图像编码后生成的一串数据流一个序列可以很长,因为运动变化少就代表图像画面的内容变动很当运动变化多时,可能一个序列就比较短了,比如就包含一个I帧和

- 4、三种帧的说明如下:
- 5、I帧: 帧内编码帧, I帧表示关键帧, 你可以理解为这一帧画面的完整 I帧特点:
  - 1. 它是一个全帧压缩编码帧。它将全帧图像信息进行JPEG压缩编码
  - 2.解码时仅用I帧的数据就可重构完整图像;
  - 3. I帧描述了图像背景和运动主体的详情;
  - 4. I帧不需要参考其他画面而生成;
  - 5. I帧是P帧和B帧的参考帧(其质量直接影响到同组中以后各帧的原
  - 6. I帧是帧组GOP的基础帧(第一帧), 在一组中只有一个I帧;
  - 7. I帧不需要考虑运动矢量;
  - 8. I帧所占数据的信息量比较大。
- 6、P帧:前向预测编码帧。P帧表示的是这一帧跟之前的一个关键帧(或)解码时需要用之前缓存的画面叠加上本帧定义的差别,生成最终画

(也就是差别帧,P帧没有完整画面数据,只有与前一帧的画面差别P帧的预测与重构:P帧是以I帧为参考帧,在I帧中找出P帧"某点"的在接收端根据运动矢量从I帧中找出P帧"某点"的预测值并与差值相P帧特点:

- 1.P帧是I帧后面相隔1~2帧的编码帧;
- 2.P帧采用运动补偿的方法传送它与前面的I或P帧的差值及运动矢
- 3.解码时必须将I帧中的预测值与预测误差求和后才能重构完整的F
- 4.P帧属于前向预测的帧间编码。它只参考前面最靠近它的I帧或PI
- 5.P帧可以是其后面P帧的参考帧,也可以是其前后的B帧的参考帧;
- 6.由于P帧是参考帧,它可能造成解码错误的扩散;
- 7. 由于是差值传送, P帧的压缩比较高。
- 7、B帧: 双向预测内插编码帧。B帧是双向差别帧,也就是B帧记录的是本 (具体比较复杂,有4种情况,但我这样说简单些),换言之,要制 通过前后画面的与本帧数据的叠加取得最终的画面。B帧压缩率高, B帧的预测与重构

B帧以前面的I或P帧和后面的P帧为参考帧,"找出"B帧"某点"的预接收端根据运动矢量在两个参考帧中"找出(算出)"预测值并与差值B帧特点

- 1.B帧是由前面的I或P帧和后面的P帧来进行预测的;
- 2.B帧传送的是它与前面的I或P帧和后面的P帧之间的预测误差及证
- 3.B帧是双向预测编码帧;
- 4.B帧压缩比最高,因为它只反映丙参考帧间运动主体的变化情况,
- 5.B帧不是参考帧,不会造成解码错误的扩散。

注: I、B、P各帧是根据压缩算法的需要,是人为定义的,它们都是一般来说,I帧的压缩率是7(跟JPG差不多),P帧是20,B帧可以可见使用B帧能节省大量空间,节省出来的空间可以用来保存多一些

# 8、压缩算法的说明

h264的压缩方法:

- 1. 分组: 把几帧图像分为一组(GOP, 也就是一个序列), 为防止运动
- 2. 定义帧: 将每组内各帧图像定义为三种类型, 即I帧、B帧和P帧;
- 3. 预测帧: 以I帧做为基础帧, 以I帧预测P帧, 再由I帧和P帧预测B帧
- 4. 数据传输: 最后将I帧数据与预测的差值信息进行存储和传输。
- 9、帧内(Intraframe)压缩也称为空间压缩(Spatial compress) 这实际上与静态图像压缩类似。帧内一般采用有损压缩算法,由于帧内压缩一般达不到很高的压缩,跟编码jpeg差不多。
- 10、帧间(Interframe)压缩的原理是:相邻几帧的数据有很大的相差 也即连续的视频其相邻帧之间具有冗余信息,根据这一特性,压缩机

- 帧间压缩也称为时间压缩(Temporal compression),它通过帧差值(Frame differencing)算法是一种典型的时间压缩法这样可以大大减少数据量。
- 11、顺便说下有损(Lossy )压缩和无损(Lossy less)压缩。无损 多数的无损压缩都采用RLE行程编码算法。有损压缩意味着解压缩质 在压缩的过程中要丢失一些人眼和人耳所不敏感的图像或音频信息 几乎所有高压缩的算法都采用有损压缩,这样才能达到低数据率的目解压缩后的效果一般越差。此外,某些有损压缩算法采用多次重复员

### H264结构

- 1、视频就是图片的连续切换,当一秒钟切换的图片个数大于25(帧率)
- 2、但是图片往往比较大,一段视频占用非常大的存储空间,因此需要进
- 3、H264编码格式就是为了解决这个问题。H264的功能分为两层: 视频编码层(VCL, Video Coding Layer)和网络提取层(NAL, 注意:VLC是多媒体播放器。
- 4、H264对一幅图片编码,切割成一个一个的NALU(NAL Unit),一个宏块是视频信息的主要承载者,它包含着每一个像素的亮度和色度<sup>4</sup>

### OCX注册打开视频

- 1、进程中关闭 iexplorer, regsvr32.exe \*32
- 2、加到可信任站点
- 3、安装VideoDisplay
- 4、手动运行regOcxAndSetIE.bat,和 regOcx.bat注册

# rtsp有关类的结构

- 1、现在考虑vru, vru从vtdu拉取码流,作为rtsp的客户端,使用vtc vru本身又要回放视频,作为rtsp的服务端,也就是vru\_slave
- 2、mediaNodeI对应连接的一端,rtsp\_client关联rtsp连接和rtp
- 3、rtsp\_server在端口9816监听,关联一组rtsp的连接,MediaSes在RTSPServer::onAccept接收到一个9816的rtsp连接,创建一个rtsp连接,支持回放多路视频,MediaSession关联一组med
- 4、那么问题来了,
  - vru使用vtdu的sdk,作为rtsp的客户端,vtdu\_sdk需要使用rt

vru\_slave作为rtsp的服务端,需要使用rtsp\_server

- 5、中间需要加一层,也就是rtsp\_client\_kit和rtsp\_server\_kit 他们是怎么关联的?
- 6、先考虑Client, vtdu\_realvideo关联rtsp\_client\_kit, rtsp\_client\_kit关联rtsp\_client, rtsp\_client关联IRts
- 7、再考虑Server, vru\_salve关联rtsp\_server\_kit, rtsp\_server rtsp\_server\_kit关联rtsp\_server, rtsp\_server关联IRt

### rtsp穿网包

1、为什么发穿网包?

考虑下面的场景,客户端在内网,服务端在外网,服务端向客户端。

2、解决办法有:

办法一:在客户端的NAT上设置DNAT,但是客户端接收码流的UDPI办法二:客户端向服务端发送一个穿网包,进行内网穿透,然后服

- 3、什么时候发穿网包?
  - a、客户端先向服务端请求rtsp地址(9810端口),获取回来的r
  - b、客户端连接rtsp地址,连接成功后。获取一个自增的udp端口通过setup告诉服务端我的端口20001。
  - c、服务端收到setup请求之后,先保存请求,获取一个自增的udp 然后去绑定udp端口30001,绑定成功之后,找到setup请求
  - d、客户端收到setup回复,然后绑定20001,绑定成功以后,发送这样,服务端从30001向客户端20001发送码流就能成功。
  - e、客户端端口绑定成功,发送穿网包之后,回调onRtpConnect\_ 等待接收码流。
  - f、停止播放,客户端发送teardown
- 4、怎么发送穿网包?

发送什么内容不重要,随便发几个字节都行,但是为了规范性,发示另外,这个穿网包使用udp,可能会丢包,因此要多发送几次。

# vru码流数据的发送

- 1、从cqfs读取的码流,带有dh头,帧的开头是DHAV,帧的结尾是dha
- 2、发送码流有两种方式:发给testclient和vlc
- 3、发给testclient,直接把带有dh头的数据,切割,发送出去,发送
- 4、发给vlc,去掉dh头,逐个读取naul,去掉naul头,把naul的内容如果naul的内容太长,要分成多次发送。

### vtdu打开视频的参数

```
1、vtdu打开视频的接口,参数有
   码流打包类型
   typedef enum
                                 //用对于的制造商sdk
       eSPT_RTP_SDK = 0 ,
       eSPT_RTP_H264_RAW ,
                                //rpt h264 裸码流
       eSPT_RTP_H264_NAL ,
                                 //rpt h264 nal打钅
                                  //pg pss流 (南方
       eSPT_PG_PSS,
   }enumStreamPackType;
   传输协议
   typedef enum
    {
       ST_TCP,
       ST_UDP,
       ST RTSP
   }enumTransProto;
2、调用打开视频的接口,设置peStartRVideoRequest的这些字段,
3、然后根据enumStreamPackType和enumTransProto,设置Rtsp(
   RtpType rtpType = RTP_OVER_UDP;
   if (m_transType == ST_TCP)
    {
       rtpType = RTP_OVER_TCP;
       if(m streamPackType == eSPT PG PSS)
       {
           rtpType = OTHERS_OVER_TCP;
       }
   else
       if(m_streamPackType == eSPT_PG_PSS)
       {
           rtpType = OTHERS_OVER_UDP;
```

```
m_pRtspClient = new RtspClientKit(m_streamPackTyr
4、RtspClientKit 根据m_streamPackType设置码流解析器,如下
   if (m streamPackType >= eSPT RTP SDK && m streamF
    {
        RtpStreamParseType nParseType = (RtpStreamPar
       RtpStreamParser *parser = new RtpStreamParser
        parser->SetVideoRtpType(100);
        parser->SetVideoStreamParseType(nParseType);
        parser->SetAudioRtpType(m_audioPayload);
       if (m_audioType == DH_AUDIO_G711U)
        {
            parser->SetAudioStreamParseType(STREAM_OF
        }
        else
        {
            parser->SetAudioStreamParseType(STREAM_OF
        }
        m_strmPrs = parser;
   else if(eSPT_PG_PSS == m_streamPackType)
       // [25697] 根据客户端的配置,设置StreamParser
        m_strmPrs = new PgStreamParser;
5、RTSPClient根据m_type,设置多媒体解析器,如下:
   mediaParserI *pParser = NULL;
   if (m_type == RTP_OVER_TCP || m_type == RTP_OVER_
    {
        pParser = new mediaRTPParserI;
       mediaRTPParserI *pR = (mediaRTPParserI *)pPar
        pR->setRTPTTL(2);
        pR->setExconnect(1, 1);
   else
```

### 参见

### 上墙有关的问题

1、视频轮巡,停留时间相同可以轮巡,停留时间不同不能轮巡。 原因是:添加解码器的时候,streammode不对,导致报错。时间 表示从平台拉码流,有两个地方:一个添加设备的时候streamMod

2、nvd0405不支持自由分割,由上墙任务的screenMode来体现。

screenMode

分割模式(1)是规则分割,top left width height参数忽略 开窗模式(2)是自由分割,必须填写正确的top left width h

3、融合的控制

配置墙的时候,按照融合的方式进行。

添加解码器的时候,表示是否支持融合,在任务上墙的时候,设置

4、解码器的登录信息

ssh的用户名和密码是 admin 7ujMko0admin 如果修改了密码,比如修改为 xxx,那么密码为7ujMko0xxx nvd0405不支持ssh,只有telnet,取日志必须使用SecureCRT

5、解决nvd0405上墙的问题

使用目前的vms版本 vms\_2.3X\_532950测试是可以的。 但是必须升级nvd0405。

### 作为客户端理解rtsp流程

- 1、作为client, rtsp连接成功之后,使用异步集成接口VTDU\_A\_Rts 发送setup,告诉server我的udp端口,从server收到setup的 这个时候client向server发送udp穿网包
- 2、作为client,发送穿网包之后,回调上来udp的onRtpConnect,材如果不是集成接口,需要client再发送play
- 3、那么什么时候发送穿网包呢?

rtsp\_kit工程引用rtsp\_rtp工程。

RtspServerKit关联RTSPServer

RtspClientKit关联RTSPClient

实时视频走rtsp协议,作为服务端关联RtspServerKit,作为客无论是客户端还是服务端,都继承RtspBase,通过RtspServerk

从server收到setup的回复,执行onMessage-->onHandleRes-->RTSPClient::setUp-->mediaNodeI::open发送穿网包

### 参见

## 如何查看视频的编码类型

- 1、使用SPDemo.exe,把视频文件拖进去,就能看到。
- 2、使用电子眼 eseye\_u.exe,也可以查看视频的编码格式。
- 3、常用的编码格式有 H.26X系列和MPEG系列,另外一些不常用的编码
- 4、常见的音频格式有G711A、G711U、PCM8、PCM16、IMA\_ADPCM等。

## 播放抓包得到的码流数据

情况一:TCP交互rtsp信令,通过UDP发送码流

- 1、找到码流的UDP数据报,右击选择Decode As,协议选择RTP,然后
- 2、然后Follow UDP Stream,选择C Arrays,然后Save As,保存
- 3、运行StreamData.exe,点击RTP(C Arrow),选择文件aaa,然后
- 4、运行Elecard StreamEye Tools目录中的eseye\_u.exe,将aaa
- 5、可以看到I帧、P帧。点击播放按钮即可

6、对于自己保存的码流文件也是一样。vru从vtdu获取到码流,保存到

情况二:TCP交互rtsp信令并且发送码流

- 1、找到tcp连接, Follow TCP Stream, 在选择raw的情况下, Save
- 2、运行TcpRtpParser.exe,将111拖进去,
- 3、然后播放111\_nortp.dav

注:软件做个备份,包括StreamData.exe, TcpRtpParser.exe 和 这些工具在软件目录中

### 浓缩视频回放的问题

- 1、考虑下面的问题,浓缩视频回放,定时读取一帧数据。开始的解决办第一次读取整个浓缩文件,把内容写到工具类,然后从里面读取数:
- 2、上面的方式是提前把数据帧都准备好。但是这样存在问题: 第一个问题,浓缩视频文件很大,读取整个文件耗时很多,导致获 第二个问题,回放的时候,会导致内存使用突然间变很大,然后一
- 3、怎么解决上面的问题?

解决思路是分摊读取浓缩视频文件的开销。理想的情况是:按需读但是这会增加代码的复杂度,有没有更好的解决办法?

4、每次获取一帧的时候,从浓缩视频文件中读取一定的数据,保证读取但是,一般情况下,每一帧的数据大小差别很大,I帧会有200多K 因为一个I帧后面往往跟着20个左右的P帧。

很极端的情况下,一个I帧有2M,每次都读取2M,还是导致问题,

5、怎么解决?

第一次读取,先读取16M缓冲起来,以后每次读取平均值(20K)的这样的话,一方面,避免了极端情况下,读取数据不够一帧的问题

6、示例代码如下:

```
// 防止极端情况,第一帧特别大,先读取16M缓冲起来
    int buffSize = 1024*1024*16;
    uint8 t* buff = new uint8 t[buffSize];
    memset(buff, 0, buffSize);
    int readed = fread(buff, 1, buffSize, fileHa
    VruLogInfo("Channel[%p] Read File[%s] Fil
        this,
        m_synopsisFilePath.c_str(),
        fileHandleForRead,
        readed);
    DhStream dhStream;
    dhStream.Write(buff, readed);
    dhStream.Read(frameListFromSynopsisFile);
    delete[] buff;
}
if(feof(fileHandleForRead) == false) // 一次词
{
    // SizeForOneRead = 1024*128
    memset(m_buffReadSynopsisFile, 0, SizeForOr
    int readed = fread(m_buffReadSynopsisFile
    VruLogDebug("Channel[%p] Read File[%s] Fi
        this,
        m_synopsisFilePath.c_str(),
        fileHandleForRead,
        readed);
        DhStream dhStream;
    dhStream.Write(m_buffReadSynopsisFile, re
    dhStream.Read(frameListFromSynopsisFile);
}
// 每次返回一帧
if(frameListFromSynopsisFile.size() > 0)
    DhFrame* dhFrame = frameListFromSynopsisF
```

```
frameListFromSynopsisFile.pop_front();
return dhFrame;
}

VruLogWarning("Get Frame Over");

// 返回所有的帧
this->m_rtspState = RTSP_INIT;
return NULL;
}

Channel析构的时候,进行fclose(fileHandleForRead);
```

## 理解rtsp\_rtp\_rtcp

RTP: Real-time Transport Protocol,实时传输协议,一般RTCP: RTP Control Protocol,实时传输控制协议,同RTP-RTSP: Real Time Streaming Protocol,实时流协议,用于RTP/RTCP相对于底层传输层,和RTSP,SIP等上层协议一起可以下tsp发起/终结流媒体(通过sdp)

rtp传输流媒体数据

rtcp对rtp进行控制,同步。注意:rtcp的端口一般是rtp的端口RTSP的请求主要有DESCRIBE, SETUP, PLAY, PAUSE, TEARDOWN, RTP/RTCP是实际传输数据的协议

RTP传输音频/视频数据,如果是PLAY,Server发送到Client端RTCP包括Sender Report和Receiver Report,用来进行音频RTSP的对话过程中SETUP可以确定RTP/RTCP使用的端口,PLAY/

# 理解rtsp的流程

- 1、客户端使用vru\_sdk,调用集成接口VRU\_AC\_StartVideo\_I,向\
- 2、VruSession收到rtsp的回复,检查是集成接口,使用VruRealViVruRealVideo使用RtspClientKit::startRtsp(m\_strUrl
- 3、以client为例,RtspClientKit继承IRtspEvent,而RTSPClientspClientKit在创建RTSPClient的时候,把this指针传给RTRTSPClient执行onConnect调用onRtspConnect,多态执行到
- 4、VruRealVideo继承RtspBase,而RtspClientKit关联RtspBas

而VruRealVideo通过m\_pRtspClient->setCallback(this, 多态执行到VruRealVideo

- 5、rtsp连接成功,执行到VruRealVideo::onRtspConnect,检查是
- 6、服务端也能收到onRtspConnect, VruSlave继承RtspBase,但是
- 7、VruSlave继承RtspBase,s\_pRtspServer->setRtspCallbac 客户端发送describe和setup,服务端从RtspServerKit回调」
- 8、对于请求setup,打开rtp有关的udp端口,保存setup请求。 打开udp端口(只是打开端口,并没有连接),会从RtspServerkORTSPServer::OnRtpConnect-->RtspServerKit::onRtpC 找到setup的请求,向客户端发送setup回复
- 9、VruRealVideo收到setup的回复,执行onMessage,处理setupfetspClientKit::startRtp(m\_remoteRtpIp,m\_remoteF--> RTSPClient::setUp(rIp, rPort, m\_localIp, m\_lc--> mediaNodeI::open((t\_int8 \*)rIp.c\_str(), rPort 作为udp的客户端,需要指定远程的ip和port,以及作为udp客户发送穿网包,发送穿网包之后回调上来。然后执行onRtpConnect,再去发送play
- 10、注意:对于rtsp的流程都是在RtspBase的子类中处理的,重写onrtsp的打包解析协议是rtsp\_protocol,class CRtspSetUpFpe协议是载体,暴露出结构体的字段,打包解析协议继承pe协议,dhtp是内部默认的打包解析协议

# 理解我们的rtsp流程【客户端】

- 1、以vru为例, vru录像会从vtdu拉取码流, 看看这个过程。
- 2、vru收到录像请求,master线程向vtdu请求rtsp地址(通过与vtd
- 3、从vtdu获取到rtsp地址,找到原始录像请求,将rtsp的有关信息( 设置到录像请求协议中,转发给salve线程。注意:rtsp的端口是
- 4、slave线程收到录像请求,判断是前端录像还是中心录像。

前段录像录在DVR或者NVR上面

中心录像录在平台上面,比如vru从vtdu拉取码流,保存在平台上DVR:Digital Video Recorder 数字硬盘录像机,也就是在[NVR:Network Video Recorder 网络硬盘录像机,通过网络指

- 5、如果是中心录像,登录vtdu的rtsp地址【VTDU\_A\_RtspConnect/设置回调rtsp的回调【VTDU\_A\_SetRtspMsgCallBack】
- 6、收到rtsp连接成功的回调,构造setup请求,调用rtsp的异步集成。 同时设置码流回调【VTDU\_A\_SetRealDataCallBack】

- 7、现在分析setup的异步集成接口
  - a、在调用VTDU\_A\_RtspConnect4Server连接vtdu【9816】的
  - b、先发送setup请求,里面带着本地的rtp端口
  - c、然后再onMessage处理setup的回复,先设置码流解析器,并
  - d、端口绑定成功,回调onRtpConnect,检查是异步集成接口,发
- 8、停止录像的时候, master线程向vtdu【端口9810】发送peCloseF同时转发给slave线程, slave向vtdu【端口9816】发送teardc

# 理解我们的rtsp流程【服务端】

- 1、rtsp走控制信令, setup沟通端口, 用于码流的发送, play发送,
- 2、先看rtsp服务端,以vru为例
- 3、Rtsp\_Server在9836监听,当客户端连接成功【onAccept】,创加 nSeq为自增Id,用于标识new MediaSession MediaSession用于标识一个rtsp连接(也就是客户端连接9836
- 4、在rtsp的服务端收到setup请求,先保存请求,然后在本地打开端口打开rtp的端口,在MediaSession::onConnect 然后判断出rt注意:服务端的端口打开,客户端就会发送穿网包,内容是tunnel
- 5、在打开本地端口的时候,创建mediaNodeI,用于码流的发送,mediable说,一个rtsp连接可以回放多路视频。MediaSession和m

# 理解码流有关的概念

- 1、H264是视频的编码方法,是一种标准的格式。
- 2、在H264的码流数据上,可以加上不同的头部信息,可以加上大华头部
- 3、大华码流(大华头+H264)必须识别这种格式才能播放。大华码流不
- 4、PS流是标准的打包格式,大家都认识。在PS流上再封装一层PG头,i

# 理解采样率

- 1、声音是由于振动产生的。想象出一个振膜,随着时间上下振动。
- 2、因此,音频信息可以使用振膜的位置来表示,相对于时间,这个位置但是计算机没法表示连续的值,只能表示离散值。这些离散值就是当离散值足够多的时候,就能够基本上还原出音频。离散值越多,
- 3、每秒离散值的个数就是采样率,采样率是个折中,当人的耳朵分辨不
- 4、对于视频, 也是同样的道理。要表示一组连续的图片, 也需要采集这

采样率就是1秒钟采集多少个样本。

- 5、网络上发包,需要表示两个音频数据包或者视频数据包的前后,使用 这个时间戳是逻辑概念,不表示真实的时间,与unix时间戳不同。 因此可以使用采样的次数表示。
- 6、考虑音频,采样率是8000,20毫秒发送一个音频数据包。两个音频 也就是1秒钟有1000/20=50个音频包,每个音频包的采样次数 (
- 7、考虑视频,采样率是90000,视频播放的时候,帧率是25,一秒钟报两个数据包间隔的采样次数来表示,也就是一个画面对应多少采样。90000/25=3600采样次数

### 码流有关的端口

```
1、码流的传输使用udp,rtp有个服务端和客户端,二者都有个端口范[
```

2、以vtdu为例,客户端在vtdu\_sdk设置端口范围,如下:

```
t_int32 funcClub::VTDU_Func_Init()
{
    logI log;
    log.regMod(VTDU_SDK);//注册日志模块

    VtduRealVideo::getPortManager()->setPortRange
    int ret = VtduSdkManager::instance()->VtduSdk
    streamTransInit();
    return ret;
```

3、服务端使用配置文件,如下:

<send\_stream\_first\_port>20000</send\_stream\_first\_ <send\_stream\_last\_port>30000</send\_stream\_last\_port 并且设置

pPortManager->setPortRange(portBegin, portEnd, 2)

- 4、特别注意:在同一台设备获取码流,rtp的客户端与服务端,udp的显示者,端口范围一样,打开端口失败,无法获取码流。
- 5、rtp客户端发送setup请求,填写client的ip和端口rtp服务端回复setup,填写server的ip和端口

# 网页中嵌入播放器控件

```
1、比如嵌入VLC控件,如下:
   <!DOCTYPE html>
   <html>
   <head>
   <meta http-equiv="Content-Type" content="text/htm</pre>
   <title>RTSP</title>
   </head>
   <body>
       <object id="vlc" classid="clsid:9BE31822-FDA[</pre>
           <param name="AutoLoop" value="0" />
           <param name="AutoPlay" value="-1"/>
           <param name="Toolbar" value="0"/>
           <param name="ExtentWidth" value="21167"/>
           <param name="ExtentHeight" value="11906",</pre>
           <param name="MRL" value="rtsp://10.36.65.</pre>
           <param name="Visible" value="-1" />
           <param name="Volume" value="50"/>
           <param name="StartTime" value="0"/>
           <param name="BackColor" value="0"/>
       </object>
   </body>
   </html>
2、这里起作用的关键一点是classid="clsid:9BE31822-FDAD-461
   clsid就是classID类的标识码,是控件的身份证号码。
   注:这是在windows中注册的,只能使用IE打开这个控件,使用g
3、怎么找到VLC对应的clsid?
   a、运行regedit
   b、选择HKEY CLASSES ROOT,快速输入CLS,定位到目录CLSI
   c、查找,选择项,查找关键字VLCPlugin,注:F3是下一个的快
   d、可以找到对应的项,有一个子目录就是CLSID,打开就能看到数
```

# 重变

PV操作
理解内存
线程栈空间的大小
理解浮点数精度
理解字符编码
理解unicode和utf8
编程中的契约精神

## 变量说明

#### PV操作

1、首先应弄清PV操作的含义:PV操作由P操作原语和V操作原语组成(》 对信号量进行操作,具体定义如下:

#### P(S):

- ①将信号量S的值减1,即S=S-1;
- ②如果S>=0,则该进程继续执行;否则该进程置为等待状态V(S):
  - ①将信号量S的值加1,即S=S+1;
  - ②该进程继续执行(不阻塞),并且,如果该信号的等待队列
- 2、PV操作的意义:我们用信号量及PV操作来实现进程的同步和互斥。P
- 3、什么是信号量?

信号量(semaphore)的数据结构为一个值和一个指针,指针指向信号量的值与相应资源的使用情况有关。

当它的值大于0时,表示当前可用资源的数量。

当它的值小于0时,其绝对值表示等待使用该资源的进程个数。

注意,信号量的值仅能由PV操作来改变。

4、P操作,S的值减1。

如果S>0,说明还有资源,线程不阻塞,继续运行下去。

如果S=0,说明刚好用完资源,当前线程也获取到了资源,线程不下如果S<0,说明没有可用资源,当前线程移到等待队列中,等待被下

5、V操作,S的值加1。

如果S>0,说明还有资源,没有线程在等待,不需要做什么。

如果S=0,说明S加1之前取值是-1,有一个线程在等待,需要做的如果S<0,说明之前有多个线程在等待,假如当前S取值为-4,说见这时候需要做的事情是:唤醒队列中第一个线程,还有

注意:V操作本身不会被阻塞,只是判断是否需要唤醒阻塞的线程。

6、特别注意:PV操作的P操作会导致线程阻塞,而V操作不会导致线程图

参见

#### 理解unicode和utf8

- 1、首先思考编码的本质,计算机只能认识和存储0和1,要表示一个符号呈现符号的时候,把这个序号解析出来,找到对应的符号,系统宣存储符号的时候,把符号编码成一个序号,也就是数字,写到内存
- 2、最开始只有英文字符,一个字节可以表示200多个数字,足够给这些
- 3、但是问题来了,每个国家的符号不一样,个数可能达到几万,因此各也就是对自己国家的符号,进行标序号,使用两个字节表示一个符合 ASCII是单字节编码,ANSI是双字节编码。

在中国ANSI就是gb2312,2个字节可以表示6万多个符号,足够了

4、这又导致问题,各个国家都有自己的一套编码规则,同一个序号在不 那么,在一个页面中,不能同时表现中文和日文。

(注意:乱码的本质是用一种编码规则去解析另一种编码规则)

5、怎么解决这个问题?

需要一种编码规则,对地球上已经存在的每个符号,都指定一个唯特别注意:unicode只是一个标准,为每个符号分配一个唯一的序

6、unicode的实现有utf8, utf16, utf32

注:这里的utf是universal transformation format 8、16、32分别表示8bit、16bit、32bit为编码单位。 其中utf16和utf32是定长编码,而utf8是可变长编码

7、现场考虑中文字符 "汉",分别用utf16和utf8表示。

在unicode的约定中,"汉"的序号是0x6c49,也就是第0x6c49个使用utf8表示有些复杂,utf8是可变长编码,前缀有几个1,就表Unicode序号 UTF-8 字节流(二进制)

000000 - 00007F || 0xxxxxx

000080 - 0007FF | 110xxxxx 10xxxxxx

0x6c49的二进制是 0110 1100 0100 1001,在范围000800 - 按顺序放入1110xxxx 10xxxxxx 10xxxxxx中,0110放入第一

00 1001放入第三个字节的后面6bit,也就是 1110[0110] 10

8、相对于unicode的其它编码实现,utf8使用更广泛,原因是:

- a、完美兼容ascii码,使用一个字节表示。
- b、节省存储空间
- 9、这里存在一个问题,对于unicode,怎么知道是哪一种编码实现?以 (注:字节顺序就是大端和小端,比如有个表示45,大端取值是45 因此需要在字节流的开头,标识是哪一种的unicode的编码实现和

EF BB BF UTF-8

FE FF UTF-16/UCS-2, little endian

FF FE UTF-16/UCS-2, big endian

FF FE 00 00 UTF-32/UCS-4, little endian

00 00 FE FF UTF-32/UCS-4, big-endian

对于utf16和utf32,有大端和小端的表示方式,但是对于utf8只因此,对于unicode的编码实现,utf8并不需要字节流的开头标识utf8的编码设计很巧妙,首先没有大端和小端,前缀有几个1,就

10、也就是说utf8不需要包含BOM,包含BOM在linux上面反而会带来-因此,utf8最好不要包含BOM

#### 理解内存

- 1、32位机器,4G内存,指针使用4个字节表示,可以表示的地址个数是也就是说4G内存共有40多亿个房间,可以存放40多亿个整数,可以但是,每个房间大小是2^32次方,可以表示的整数范围[0,2^32)注意:这里有两个概念,多少个房间,每个房间的大小
- 2、32位机器,1G内存,指针还是使用4个字节表示,可以表示的地址个可以存放10亿个整数,但是房间大小还是2^32次方,可以表示的图
- 3、32位机器,8G内存,指针还是使用4个字节表示,可以表示的地址个但是每个房间的大小还是2^32次方,可以表示的整数范围是40多位也就是说,房间能够表示的指针大小是0到40多亿,编号为50亿的是浪费的。
- 4、怎么解决上面的问题?

上面问题产生的原因是:房间个数足够多,但是房间大小太小,解定房间能够表示的指针大小是0到2^64次方,编号0多亿到80多亿的度是2^64次方,这个数字非常大。

5、BTY,顺便说一下,IPV4是4个字节表示网络Ip地址,而IPV6不是6 每一组是2个字节,使用4个16进制的数字。一个16进制范围[0,2 调试的时候,指针使用8个16机制表示,也就是4个字节。

#### 理解字符编码

- 1、可以这样理解,字符编码就是对于字符集合,每个字符设置一个id,
- 2、举例来说,有两套编码,编码A只有三个字符 甲a1,编码B只有六个甲[00] a[01] 1[10]; a[000] b[001] c[010] d[011] 1
- 3、对于不同字符编码,同一个id对应不同的字符,如甲[00] 和 a[06
- 4、这里要区分, 当成某种编码解释和转化为某种编码。
- 5、当成某种编码解释,也就是只看Id,存在两种情况:
  - 一是编码A有意义的字符,当成编码B来看,是一堆没有意义的字符 二是,当成编码B来看,编码B中可能没有对应的Id,错误。
- 6、转化为某种编码,先看编码A的Id,找到对应的字符,再看这个字符: 比如:编码A [0110]-->对应a1-->从编码B中找到a1对应的Id-这里存在一个问题,假如编码A中的一个字符,比如甲,在编码B中

#### 理解浮点数精度

- 1、为什么叫浮点数?
  - 相对于浮点数,就是固点数,小数点固定在最右边,也就是整数。浮
- 2、考虑二进制整数,假设只有2个bit,可以表示00,01,10,11,共四
- 3、考虑二进制小数,假设只有2个bit,可以表示多少个小数? 答案也是四个。假设小数点在最左边,分别为00,01,10,11,表示特别注意:不同于整数,整数可以表示范围内的每一个整数,如[0 我们知道从0.0到0.75有无数个小数,两位二进制只能表示其中的
- 4、十进制0.0到0.9,有几个可以使用二进制表示? 只有两个,0.0和0.5。考虑十进制,小数第一位是1/10,小数第二 小数第一位是1/2,小数第二位是1/4,小数第三位是1/8,那么0.1 a1\*1/2 + a2\*1/4 + a3\*1/8 + .... 存在这样的a1,a2,a3吗? 答案是不存在。
- 6、那么怎么解释 0.2+03 会等于0.5呢? 举个例子,计算 1.6+1.8,现在假设不能准确表示1.6和1.8,只 这与1.6+1.8=3.4,再转化为最接近的3,相差为1。

那么是不是,所有的计算结果都不准确呢? 不是这样,考虑1.2+1.8 =3,转化为处理是 1+2 =3,计算的结 也就是说,转化过程中精度缺失,如果两个加数都多了一点,其和 如果一个加数多了一点,一个加数少了一点,刚好相互抵消,其和 7、二进制与十进制的转化, 二进制整数转为十进制整数,二进制小数转为十进制小数都简单。 十进制整数转为二进制小数,除2取余,倒序排列。 十进制小数转为二进制小数,乘2取整,顺序排列。用这种方法,可 8、思考一下,十进制小数0.1不能用二进制准确表示,那么是不是所有 可以。二进制小数,转为十进制,就是a1/2+a2/4+a3/8...., a 那问题就转化为,1除2的n次方,是不是都能除尽。类推一下,0.1 9、从数学的角度分析,对于小数,2进制只能表示1/2, 3进制只能表: 你会说1.5/3 就是一半呀,这就是一个递归的问题,那1.5怎么用 那么10进制,只能表示1/10, ...9/10, 而恰巧5/10就是1/2 10、如果我想让0.1+0.2 等于0.3,怎么办? 从上面分析知道,二进制可以表示可表示范围内的任意一个整数。 我们把0.1和0.2根据小数点分成两部分,同时记住小数点的位置。 在字符串中添加小数点的位置即可。需要注意的是:小数点左边右 比如:12.46+5.5400,分别为12+5,4600+5400。 代码如下: string NzbUtils::GetRightDouble(string a, string { vector<string> aVec; StringSplit(a,".",aVec,true); vector<string> bVec; StringSplit(b, ".", bVec, true); int  $c1 = atoi(aVec[0].c_str()) + atoi(bVec[6])$ int maxLen = aVec[1].size() > bVec[1].size() if(maxLen > aVec[1].size()) { while(maxLen > aVec[1].size()) { aVec[1]+="0"; } if(maxLen > bVec[1].size())

#### 线程栈空间的大小

```
1、Linux下面线程栈默认是10M,单位是K,设置和查询如下:
        [root@localhost IBP]# ulimit -s
        10240
        [root@localhost IBP]# ulimit -s 5120
        [root@localhost IBP]# ulimit -s
        5120
        因此特别注意:Linux线程开得太多,会导致内存使用特别多,解测少线程的个数,同时线程栈的大小设置小一点。
        因为线程栈只是用于记录程序的调用过程和栈上自动变量的内存分2、Windows下线程栈默认只有1M,一般情况下也是够用的。
```

#### 编程中的契约精神

#### 1、问题来源:

软件开发中有一个本质性的难题,那就是错误处理。错误处理是程,探察错误的一方(接口提供者)不知道如何处理错误,而知道如何;使用防御式代码来解决,往往会破坏程序的正常结构,而带来更多

- 2、这个问题的解决办法是:异常机制。
- 3、但是,这就要搞清楚什么是真正的异常,只有在真正异常的情况下使以往,我们认为异常就是:接口没有完成它所声称可以完成的任务。 还要加上一句,在满足约束条件的情况下,接口没有完成它所声称。
- 4、这也就是软件开发中的契约精神,接口提供者和接口使用者是平等的 我没有义务"排除万难"地去完成任务。接口提供者声称我能完成什
- 5、如果没有契约精神,会导致责权不清的问题:我说你给的条件不满足

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

 首页
 模块

 设计模式

# 重变

策略模式

简单工厂\_\_工厂方法\_\_抽象工厂

设计模式六大原则

适配器模式

## 变量说明

#### 策略模式

项目应用场景:协议本身是一个载体,与不同的外部服务交互,格式不同因此需要不同的打包解包子类,根据交互的格式需要,指定打包解包的引

#### 参见

#### 简单工厂 工厂方法 抽象工厂

- 1、考虑用户(这个用户可能是自己也可能是其他的程序员)的需求:给
- 2、使用简单工厂。如下:
  - 创建抽象类ICar,具体类Sedan(轿车)和具体类SportCar继承创建简单工厂类,根据汽车类型(使用switch),创建汽车对象证
- 3、用户需求变化,要求给出SUV类型,获取SUV汽车对象。怎么办?
- 4、增加具体类SUV,继承ICar。在简单工厂类增加switch的分支。但 a、具体类SUV满足开闭原则,而简单工厂类不满足开闭原则,要修 b、同时简单工厂类是个上帝类,需要返回一切需要的汽车对象。 怎么办?
- 5、使用工厂方法。如下:
  - 创建抽象工厂IFactory,以及具体工厂类SedanFactory和Spor当用户需要SUV的时候,增加一个工厂类SUVFactory,满足开闭原
- 6、用户需求变化,期望获取一个品牌的轿车和跑车,比如给出奔驰,期如果用户还是使用工厂方法,用户可能会用SedanFactory获取奔这就导致了使用windows的锤子去敲linux的钉子,不匹配。要把
- 7、这就涉及到产品族的概念,解决办法是使用抽象工厂。如下: 创建抽象工厂IFactory,以及具体工厂类BenZFactory和BmwFa BenZFactory有两个方法,分别负责生产奔驰轿车和奔驰跑车,

BmwFactory有两个方法,分别负责生产宝马轿车和宝马跑车。 这样,再傻的用户也不会出错,生产出来的轿车和跑车是匹配的,

#### 参见

#### 设计模式六大原则

- 1、开闭原则
- 2、里氏代换
- 3、面向抽象编程(依赖倒转)
- 4、接口隔离(最小接口)
- 5、合成聚合原则(使用合成聚合替代继承)
- 6、迪米特法则(不和陌生人说话)

#### 参见

#### 适配器模式

项目应用场景:我们的服务会接入不同的设备,比如大华设备,海康设备需要根据不同的设备类型进行适配。下层有适配器管理器和一组适配器,适配器管理器创建出一个适配器,设备适配器与具体的设备交互。这个omysql的odbc驱动是适配器,负责与mysql服务交互,oracle的odbc。

#### 参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

首页	模块	
读书		模块   变量

# 模块

冷幽默

名人名言

文化修养

# 变量

仗义每多屠狗辈,负心多是读书人

看上去荒谬实际上科学的理论

看上去脑残实际上精妙的设计

路径依赖

### 详细描述

## 变量说明

#### 仗义每多屠狗辈,负心多是读书人

先说这副联的出处,作者是明代的曹学佺。

一种说法是当时他在桂林当官,明宗室家奴放狗咬人,一个屠户杀狗救门结果判案时秀才慑于贵胄势力又拿了钱,想改口供陷害屠户,被曹学佺认事后曹学佺就写下了这副联。

#### 参见

#### 看上去脑残实际上精妙的设计

- 1、飞机上厕所的冲水按钮,在背后。
- 2、自助取票机器,放身份证的位置是斜坡。
- 3、动车车票没有E,航空机票是3+3座,分别为A、B、C、D、E、F,AF如果有E,那么考虑客户拿到E,对于3+3座,就是中间,对于3+2层同理对于2+2只有AC和DF
- 4、有些地方的掉头车道在最右边,这是为了大型车准备的,方便掉头。
- 5、有些道路人为设置一些弯道。是为了逼迫司机把速度降下来,特别是
- 6、有些超时的易拉罐饮料倒着放,是为了防止落上灰尘。
- 7、可乐塑料瓶,开口螺旋有缺口,是为了开的时候,一部分一部分溢出
- 8、淞沪会战,战壕挖两米。自己要拿着板凳来防守。当撤退的时候,日
- 9、洗衣机的开始按钮,要长按一会才能启动。防止人无意间碰到,触动
- 10、有些药瓶,要下压转动才能打开,是防止小孩打开误食。

注意:解决问题的办法由100种,最好的解决办法是问题不产生。

#### 参见

#### 看上去荒谬实际上科学的理论

- 1、返回的轰炸机,弹孔多的地方不需要加强防护,弹孔少的地方反而需
- 2、汽车的油耗越低,总的石油消耗就越大。

#### 参见

#### 路径依赖

- 1、路径依赖是指一旦选择了一条路径(不管是好是坏),就会对这条路惯性的力量会使这一选择不断自我强化和锁定,轻易走不出去。也就是说,人们过去做出的选择,决定了其现在可能的选择。
- 2、一个经典的例子是:美国航天火箭助推器的宽度,竟然是由2千多年前美国航天火箭助推器的宽度为什么是4英尺左右呢?因为火箭助推器需要从生产地运到发射基地,因此轨道宽度决定了那么火车轨道宽度是怎么定下来的?轨道的宽度是由早期造电车的人设计的,电车的宽度标准是怎么来造电车的人以前是造马车的,电车的宽度标准沿用马车的轮距。马达是因为英国马路上的辙迹决定的,因为其它宽度的马车很容易撞突国的马路是由罗马人为他的军队建设的,辙迹是由罗马战车形成

参见

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

 首页
 模块

 冷幽默

 读书

# 重变

## 亲子有关

## 详细描述

## 变量说明

#### 亲子有关

- 1、爸爸带女儿买衣服,见到女儿的男朋友,说,我背着老婆来看你。
- 2、老妈喂小弟吃饺子,掉到地上,捡起来还去喂。怎么还给他吃,老妈
- 3、十一回家,看到丰盛的晚餐,想到爸妈平时省吃俭用,自己回来就做
- 4、冬天冷,没人愿意起来,家里座机电话一直响。我刚起床,老爸就说
- 5、过节,给了100元,你出去玩吧,今天情人节,和你老妈单独过节。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.



# 变量

自律四个原则

借钱不换

尼采有关

## 详细描述

## 变量说明

#### 借钱不换

1、钱钟书说,对有些人来说,最重要的理财方式是借钱不换。

#### 尼采有关

- 1、我们飞得越高,在那些不能飞的人眼里就越渺小。
- 2、我们的眼睛就是我们的监狱,而目光所及之处就是监狱的围墙。

#### 自律四个原则

1、斯科特.派特,说的自律包括四个原则:推迟满足感,承担责任,忠于

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

 首页
 模块

 文化修养
 读书

## 变量

博学、审问、慎思、明辨、笃行

三省吾身

振衣千仞岗,濯足万里流

鹪鹩巢于深林,不过一枝,偃鼠饮河,满腹而已

不迁怒,不二过

八风吹不动,端坐紫金莲

一个说前门楼子,一个说胳膊肘子

两个厨子烧一锅汤,不是太咸就是盐没放

龙生一子定乾坤,猪生一窝拱墙根

## 详细描述

### 变量说明

- 一个说前门楼子,一个说胳膊肘子
- 1、一个说前门楼子,一个说胳膊肘子

#### 三省吾身

1、三省吾身

孔子弟子曾参深得老师喜爱,有人问他为何进步这么快,曾子曰:

#### 不迁怒,不二过

1、不迁怒,不二过

不把怒气发泄到别人身上,不重复犯同样的错误。这是一种极高的,

两个厨子烧一锅汤,不是太咸就是盐没放

1、两个厨子烧一锅汤,不是太咸就是盐没放

#### 八风吹不动,端坐紫金莲

1、八风吹不动,端坐紫金莲

话说苏东坡在地方任职,和一江之隔金山寺的住持佛印经常谈经论籍首天中天,毫光照大千,八风吹不动,端坐紫金莲。 洋洋自得,派书童送给佛印,佛印看后,写上放屁两个字。 苏东坡收到回信,顿时发火,渡江去找佛印理论。 佛印说,哎,你不是八风吹不动吗?怎么一屁打过江了? 苏东坡顿时惭愧不已。 博学、审问、慎思、明辨、笃行

1、博学、审问、慎思、明辨、笃行

振衣千仞岗,濯足万里流

1、振衣千仞岗,濯足万里流 晋代左思的诗,在很高的山上抖落衣服的灰尘,在长河中洗去脚上

鹪鹩巢于深林,不过一枝,偃鼠饮河,满腹而已

1、鹪鹩巢于深林,不过一枝,偃鼠饮河,满腹而已

龙生一子定乾坤,猪生一窝拱墙根

1、龙生一子定乾坤,猪生一窝拱墙根

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.

# 重变

机制与策略
软件的本质就是封装
软件的功能就是增删改查和数据可视化
软件的运行
概要设计
通用性和针对性
缓存
流程驱动和事件驱动

### 详细描述

## 变量说明

#### 机制与策略

- 1、透过现象看本质,机制与策略到底是什么?为什么要将机制与策略分
- 2、反映在系统设计上,机制是目标功能,策略是实现方法,也可以认为
- 3、反映在管理上,领导负责机制,员工负责策略,也就是说,领导决定
- 4、为什么要将机制与策略分离?

假如不分离,会出现什么问题?考虑下面的场景,要完成排序的功那么问题来了,我想换一种排序算法,比如快速排序,接口和实现也就是机制与策略分离,提供一个排序接口,继承接口,提供不同Context关联一个抽象的策略(也就是接口),用不同的具体策略

#### 参见

#### 概要设计

概要设计过程:确定边界,划模块,定接口,理流程。分别如下:

- 1、确定边界:确定边界后,才能明确与外部哪些实体交互,交互需要确消息的大小,消息的格式,消息是同步还是异步。
- 2、划模块:模块之间有竖直关系和水平关系,比如MVVM模式可以认为是水平关系比如:接受消息,处理消息,转发消息。
- 3、定接口:模块可以认为是一个IPO(Input-Process-Output)控
- 4、理流程:使用时序图,说明对于一个功能,若干模块是如何协作完成

#### 参见

#### 流程驱动和事件驱动

- 1、每个模块都是消息处理单元,外部线程往这个模块里面Push消息,证 启动线程while死循环,主动轮巡检查消息队列,然后处理消息。
- 2、有没有更好的处理办法呢?

也就是,外部线程Push进去一个消息,回调上来一次,然后处理消给人感觉就是,这里触发一下,从另外一个地方回调上来处理消息。

3、怎么实现事件驱动呢?

第一种方法:把while死循环,下沉到底层(也就是底层启动一个检测到一个消息,从底层回调上来。也就是说,事件驱动是对流程

第二种方法:使用PV操作,当前线程底层while死循环,每次进行其他线程Push消息的时候,内部调用notify一下底层(也就是V指也就是,任何模块向当前模块发送一个消息,当前模块都回调上来。

#### 参见

#### 缓存

- 1、《精通正则表达式》,孟岩作序:Bill Joy 软件世界的爱迪生,设现在已离职,2003年离职的时候,SUN股票应声下跌3.2%
- 2、Bill Joy曾经调侃说,在计算机领域中,缓存是唯一称得上伟大的。
- 3、典型的场景应用:

Mysql的缓存

线程池

对象池

客户端频繁查询某些信息,这些信息保存到内存中,避免重复读取注意:保证内存和数据库的信息同步,信息变化需要从数据库插入数据库,缓存起来批量写入。

#### 参见

软件的功能就是增删改查和数据可视化

所有的软件,无非是增删改查和数据的可视化。Google也就是做了一个互联网上的用户进行增删改操作。淘宝也就是对商品的增删改查。

#### 参见

#### 软件的本质就是封装

软件的本质就是封装。根据已知的实现,对这些实现封装,对外暴露接口高级编程语言对操作系统进行封装,对外暴露接口。应用软件对高级语言

#### 参见

#### 软件的运行

可以把软件的运行看成做一件事,比如做菜。做菜需要菜谱,菜谱列出了人按照操作步骤,加工数据。菜谱就是程序,原料就是数据,操作步骤就人做菜就是CPU执行程序,就是进程。

#### 参见

#### 通用性和针对性

- 1、所有工具都是为了人类解决问题的,这也是人类发明工具的动力。
- 2、通用型的好处在于适用较大的范围,缺点是对于特定的场景会退化。 因为通用性要考虑不同的情况,这些情况要均衡地处理。
- 3、针对性在特定场景下很好地满足用户需求,如果用户需求经常变化,
- 4、通用性和针对性解决不同的问题,也就是有不同的应用场景。举例来但是它的效果肯定比不过专业的炖汤锅(注:同等价位)。如果家如果只是拿来炖汤,就要买专业的炖汤锅。
- 5、这就能解释,已经存在很多成熟的开源库,为什么有些公司还要自己 这里存在两个问题:

开源的网络库不能很好地满足自己特定的需求。

功能丰富的开源库,代码量很大,不吃透就使用,有很大的风险。

#### 参见

世程\_线程\_同步\_异步

# 变量

1 :		-	+>-	JE.
Lin	ux	N	个么	亚

Windows线程CloseHandle

线程同步信号量semaphore

设置回调的问题

同步与异步

共享资源竞争的解决办法

分离式线程

## 变量说明

#### Linux内核同步

- 1、原子操作,是其它同步方法的基础。
- 2、自旋锁,线程试图获取一个已经被别人持有的自旋锁,当前线程处于
- 3、读写自旋锁,根据通用性和针对性的特点,普通自旋锁在特定场景下 因此,提供了读写自旋锁,读锁可以加读锁,不能加写锁,写锁不
- 4、需要注意的几项:

普通自旋锁是不能递归的。读锁可以递归,写锁也不能递归。表面上锁的是代码,实际上锁的是共享数据。

使用读写锁的时候,需要注意,读锁可以加读锁,多个线程都占用。 这往往会导致写锁长时间处于饥饿状态。

- 5、自旋锁存在的问题,线程试图获取一个已经被别人持有的自旋锁,当使用信号量,信号量是一种睡眠锁。一个任务试图获取被别人占有一当请求的信号量被释放,处于等待队列的任务被唤醒,并获得信号。
- 6、需要注意的是,信号量是一种睡眠锁,但它本身也会带有开销,上下也即是说让其睡眠并唤醒它,花费一定的开销。如果每个线程锁的如果锁的时间长,使用信号量。
- 7、相比自旋锁,信号量还有更广泛的用处,使用PV操作不仅能保护共享对于锁,是谁加锁谁释放,而信号量可以再不同线程之间PV操作。
- 8、考虑信号量的一种特殊使用场景,可以睡眠的互斥锁。创建的信号量 这就是互斥体,互斥体加锁可以认为是P操作,再V操作。

#### Windows线程CloseHandle

这里为什么创建线程之后,马上CloseHandle?如下:

HANDLE hThread = CreateThread(NULL, 0, CommWithClient, &
CloseHandle(hThread);

原因是:创建线程后返回线程句柄,新创建的线程内核对象引用计数是2一个是创建线程的线程,也就是说当前线程对新创建的线程保持一个引用当前线程调用CloseHandle,使引用计数减1,新线程运行结束后,引序这时候为0,系统删除新创建的内核对象,这是正常的流程。

如果当前线程没有调用CloseHandle,新线程运行结束后,引用计数减系统不会删除新创建的内核对象。当然,整个程序运行结束后,系统还是也就是说,不调用CloseHandle,会导致程序运行期间的内存泄露。

当然,这还要考虑实际的需求,如果新创建的线程运行结束,后续还要修

#### 共享资源竞争的解决办法

- 1、多个线程访问共享资源,存在竞争关系。
- 2、解决共享资源的竞争关系,只有三个思路。分别如下:
- 3、多版本并发控制(Multi-Version Concurrency Control), 这个时候也就没有共享资源了。
- 4、对于共享资源,多个线程大家排队,一个一个来访问。
- 5、使用PV操作,控制线程的访问顺序。这个线程进行了V操作,那个线

#### 分离式线程

```
1、技术都是为了解决实际问题的,考虑下面的场景:
    主线程创建一个子线程,子线程做一些任务,在主线程上,等待子
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

void* FuncA(void* arg)
{
    printf("FuncA Time[%d]\n", time(NULL));
    sleep(2);
}

int main(int argc,char* argv[])
{
```

```
pthread_t threadA;
       pthread_create(&threadA, NULL, FuncA, NULL);
       pthread_join(threadA, NULL);
       printf("main Time[%d]\n", time(NULL));
       getchar();
       return 0;
   }
    [niu_zibin@localhost thread]$ g++ -o main main.cr
    [niu zibin@localhost thread]$ ./main
   FuncA Time[1477297071]
   main Time[1477297073]
2、可以看到,主线程阻塞在pthread_join,那么问题来了,如何让主
3、上面产生的原因是:默认创建的线程A不是分离的,也就是被主线程:
   因此,解决办法是:创建线程A的时候,把它设置成分离的,不再补
   #include <stdio.h>
   #include <pthread.h>
   #include <unistd.h>
   void* FuncA(void* arg)
   {
       printf("First FuncA Time[%d]\n", time(NULL))
       sleep(2);
       printf("Second FuncA Time[%d]\n", time(NULL))
   }
   int main(int argc,char* argv[])
   {
       pthread_t threadA;
       pthread_attr_t pAttr;
       pthread_attr_init(&pAttr);
       pthread attr setdetachstate(&pAttr,PTHREAD CF
       pthread_create(&threadA, &pAttr, FuncA, NULL)
       int ret = pthread_join(threadA, NULL);
       printf("pthread_join ret[%d]\n", ret);
```

```
printf("main Time[%d]\n", time(NULL));
       getchar();
       return 0;
    }
    [niu_zibin@localhost thread]$ g++ -o main main.cr
    [niu_zibin@localhost thread]$ ./main
    pthread_join ret[22]
    main Time[1477298407]
    First FuncA Time[1477298407]
    Second FuncA Time[1477298409]
    不再阻塞。
4、注意:设置了分离状态【PTHREAD CREATE DETACHED】, pthre
    改成可结合状态【PTHREAD CREATE JOINABLE】, pthread j
    pthread_attr_setdetachstate(&pAttr,PTHREAD_CREATE
    [niu_zibin@localhost thread]$ g++ -o main main.cr
    [niu_zibin@localhost thread]$ ./main
    First FuncA Time[1477298637]
    Second FuncA Time[1477298639]
    pthread_join ret[0]
    main Time[1477298639]
5、还有一种办法,就是创建线程A之后,也就是在线程A运行的时候,进
    #include <stdio.h>
    #include <pthread.h>
    #include <unistd.h>
    void* FuncA(void* arg)
    {
       printf("First FuncA Time[%d]\n", time(NULL))
       sleep(2);
        printf("Second FuncA Time[%d]\n", time(NULL))
    }
    int main(int argc,char* argv[])
        pthread_t threadA;
```

```
pthread_create(&threadA, NULL, FuncA, NULL);
      pthread_detach(threadA);
      pthread join(threadA, NULL);
      printf("main Time[%d]\n", time(NULL));
      getchar();
      return 0;
   }
   [niu_zibin@localhost thread]$ g++ -o main main.cr
   [niu zibin@localhost thread]$ ./main
   main Time[1477298924]
   First FuncA Time[1477298924]
   Second FuncA Time[1477298926]
6、线程是可结合(joinable)或者分离的(detached)。
   对于可结合线程A,被主线程回收资源(比如A的线程栈)和杀死,
   对于分离式线程A,在它终止后,系统会自动释放线程A的资源。
7、对于分离式线程A,考虑一种极端的情况,分离式线程执行特别快,?
   这就意味着,pthread_create返回的数据是垃圾数据。
8、怎么解决上面的问题?
   在分离式线程A中执行pthread_cond_timewait函数,让当前线
   还有一种办法,使用PV操作,分离式线程内先执行P操作,卡在这!
   从而确保pthread_create之后,分离式线程刚开始执行。
```

#### 同步与异步

- 1、同步就是做完这个任务,再执行下一个任务。
- 2、异步就是在当前线程创建一个线程,把这个任务放到新的线程上去做
- 3、任务独立是异步的前提,任务耗时是异步的理由。怎么理解? 任务独立就是,其他任务不依赖这个任务的完成。如果必须在这个 因为在新创建的线程上进行这个任务,这个任务在进行中,其他任实 任务耗时就是指做这个任务耗费时间比较多,用户等不了,想进行
- 4、基于网络的开发,一般都是异步,先设置异步回调,发送消息,然后同步是对异步的封装,使用PV操作。也是要设置异步回调,但是在而发消息的时候,使用Timeout\_P操作,卡在这里,等待异步回调注意:Timeout\_P是在当前线程执行的,而异步回调是在新创建的说明,代码和线程的关系,代码段可以在运行任何线程上,也可以

#### 线程同步信号量semaphore

```
windows CreateSemaphore, ReleaseSemaphore, Waitfors
模拟场景,
1、一个生产者,一个消费者,共享缓冲区大小为1,
2、一个生产者,一个消费者,共享缓冲区大小为5(注:可以同时生产5
3、两个生产者,一个消费者,共享缓冲区大小为5(注:可以同时生产5
注意:可以同时生产5个,满了就不能生产了,这一点和tcp的滑动窗口
另外信号量PV操作,可以表达语义:共享区没有了资源,消费者要等待。
也就是说,只有P操作要等待。
如果要表达,共享区满了,生产者要等待。需要使用反向的信号量。
sem_for_consume(0,5); 容量为5,可以消耗的数量为0,与共享缓
produce();
sem for consume.V();
sem_for_consume.P();
consume();
加上一个反向的信号量。
sem_for_produce(5,5); 容量为5,可以生产的数量为5,与共享缓
sem_for_produce.P();
produce();
sem_for_consume.V();
sem_for_consume.P();
consume();
sem_for_produce.V();
```

### 设置回调的问题

1、设置回调,往往需要把用户数据设置进去,回调的时候,把用户数据

- 一般的使用场景是:把普通方法(或者是静态方法)作为回调方法回调上来强转为对象指针,调用对象的成员方法。
- 2、回调设置的用户数据往往是全局的,如果多个对象都要设置回调,把 这是一种情况,还有另外一种情况:即使只有一个对象,会有多次·
- 3、因此还需要一个标识。

对于第一种情况,往往不再需要用户数据。比如创建一个device对用户管理好登录句柄和device对象的对应关系。以后回调上来,也对于第二种情况,还需要设置用户数据。但是每次请求都会返回一以后回调上来,也都带着请求序号,就知道是哪一次请求了。

4、当然这个过程也可以在底层处理好,比如第一种情况。

每次设置回调的时候,参数需要登录句柄和device对象的指针(月回调上来的时候,回调上来带着登录句柄,然后根据登录句柄,把上层就可以直接使用用户数据,转化为device指针。

也就是说,这种情况下,设置回调的用户数据不是全局的。

Copyright (c) 2015~2016, Andy Niu @All rights reserved. By Andy Niu Edit.