

提供数据源

在ColdFusion应用程序可以使用任何被支持的数据源之前，必需先建立该数据源。用ColdFusion Administrator建立数据源后，与数据源的集成只需指定数据源的名字，即可执行对数据源的操作。通常，这要用到CFQUERY标签，如下例：

```
<CFQUERY NAME="EmployeeName" DATASOURCE="BigCompany">
```

```
SELECT FirstName + " " + LastName
```

```
FROM Employees AS FullName
```

```
</CFQUERY>
```

在这个例子中，用DATASOURCE属性指定数据源的名字，表示要执行的SELECT语句作用于这个数据源。它要求该数据源已经建立才行。

实现安全性

建立ColdFusion应用程序时，你需要考虑几个方面的安全性——要实现开发环境以及应用程序资源的安全。ColdFusion Server目前支持几个等级的高级安全性：

- 远程开发服务安全性（RDS——Remote Development Services Security）：开发者要先通过安全认证，才可以使用ColdFusion Studio访问服务器资源，从而实现了对服务器资源的保护。

- 客户安全性：由程序开发者在ColdFusion应用程序中实现，提供运行时的安全认证和授权。

- 服务器沙盒安全：由ColdFusion管理器控制，提供基于目录访问的运行时的安全性（只在企业版中实现）。

- 管理员安全：管理操作可以受到安全保护，防止未经授权的访问。

建立应用程序文件

ColdFusion应用程序由一组CFM文件组成，开发应用程序的过程就是产生这些文件的过程。CFM文件可以包含CFML标签，HTML标签，客户定制的CFML标签，CFSript,JavaScript,以及任何可以包含在HTML文件中的内容。

当浏览器（如Netscape,IE）请求一个标准HTML文件时，Web服务器提供该文件给浏览器，浏览器解释HTML文件的内容，将输出显示在浏览器窗口中。当浏览器请求一个CFM文件时，将由ColdFusion首先对该文件进行处理，产生一个标准HTML文件，由Web服务器提供给浏览器。对浏览器来说，CFM文件与HTML文件并无区别。在客户端不需要任何客户端软件，组件，或插件来支持ColdFusion应用程序，所有的处理都在服务器上完成。

一般地，一个ColdFusion应用程序由许多文件组成，每个文件都完成特定的功能，如提供客户输入界面，格式化输出，或操纵数据库等。建立应用程序文件有以下几个要点：

- 效率——对web应用来说，效率是很关键的，你需要以最快的速度将数据提供给客户。为此在编程时对一些服务器端的操作，如大型数据库查询等，要注意提高执行效率。如果你是在为公司的内网开发应用程序，那么你不需要考虑带宽的限制，如果开发Internet应用程序，则要考虑网络传输速度的限制。

- 安全性——利用ColdFusion的认证和授权特性，你可以控制客户对应用程

序的访问。在ColdFusion管理器中设置安全框架后，你可以对客户进行基于NT域或LDAP目录的认证。

·客户输入界面——为了程序的交互性，用表单（form）接收客户的输入是必要的。在CFM文件中，你可以建立HTML表单，也可以使用基于Java的ColdFusion表单控件（如树，数据表格，滑动控件等）而不需要任何Java的知识。

用表单接收客户输入很容易，把表单数据传递给ColdFusion应用程序同样容易。当一个表单被提交时，表单数据以“表单变量”的形式被传递给该表单的ACTION属性所指定的文件。该文件用来处理客户的输入，在该文件中可以引用表单变量以显示数据，执行查询，或执行其它各种操作。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

ColdFusion 初级教程

第一章 概述

ColdFusion 4.0的特点	ColdFusion 的组成部分
建立ColdFusion应用程序	计划ColdFusion应用
建立应用程序文件	提供数据源
实现安全性	调试应用程序
手册的内容	

第二章 变量

变量的创建	用CFPARAM创建缺省变量
变量命名规范	变量的种类
利用URL和Form传递变量	

第三章 流程控制

条件选择	重定向 (CFLOCATION)
停止处理文件 (CFABORT)	包含文件
建立循环 (CFLOOP)	

第四章 应用程序框架

理解应用程序框架	实现应用程序框架
应用程序级设置	客户状态管理
错误处理	Web服务器安全性集成
缺省变量和常量	用CFLOCK进行排斥性锁定
Application变量, Session变量, 和Client变量的使用	

第五章 数据源

数据源基础	添加ODBC数据源
ODBC命名规范	使用本地数据库驱动程序
使用OLE DB连接	结构化查询语言 (SQL) 概述

第六章 选择和显示数据

用CFQUERY标签选择数据	使用动态查询参数
执行存储过程	日期格式函数
显示查询结果	在表格中显示查询结果
建立HTML查询表单	返回MIME内容

第七章 插入,更新,和删除数据

插入数据	用CFINSERT建立一个插入执行文件
用CFQUERY建立插入执行文件	更新数据
建立数据更新表单	用CFUPDATE建立数据更新执行文件
用CFQUERY建立数据更新执行文件	删除数据
数据输入检查	动态HTML表单
动态SQL	事务处理 (CFTRANSACTION)

第八章 建立动态Java表单

用CFFORM标签建立表单	CFFORM控件的输入检查
---------------	---------------

用CFTREE建立树控件	在CFTREE中嵌入URL
用CFGRID建立数据网格	建立可更新的网格
网格数据选择选项	建立滑动条控件
建立文本输入框	建立下拉列表框
建立FORM控件	嵌入Java程序片
第九章 服务器端的文件操作	
使用CFFILE	文件上载
设置文件和目录的属性	评价文件上载的结果
移动、重合名、复制和删除服务器端文件	读、写和追加一个文本文件
执行目录操作	
第十章 用CFFTP执行文件操作	
建立连接	文件和目录操作
连接缓存	CFFTP变量
第十一章 通过HTTP协议访问远程服务器	
使用CFHTTP与Web相交互	CFHTTP的语法
使用CFHTTP的Get方法	根据一个文本文件建立查询
使用CFHTTP的Post方法	
第十二章 发送和收取Email	
在ColdFusion中使用邮件服务器	发送email信息 (SMTP)
CFMAIL示例	为多个收信者定制Email
高级选项	收取邮件 (CFPOP)
处理POP邮件	

调试应用程序

ColdFusion Studio提供了交互式的调试工具，程序开发者可以用它设置断点，监视变量，计算表达式和变量的值，分步执行代码，查看堆栈。

你也可以在浏览器中打开程序文件来测试它，不需要进行任何编译和连接。多数开发者在本地计算机上运行ColdFusion和Web服务器，随时用浏览器查看应用程序的运行时状态。ColdFusion提供了几个调试选项，帮助开发者排除程序中的错误。对每个ColdFusion对话——即每次浏览器请求一个CFM文件时，如果设置了相应的debug选项，则调试信息可以显示在浏览器窗口中，显示在网页内容的底部。

提供数据源

在ColdFusion应用程序可以使用任何被支持的数据源之前，必需先建立该数据源。用ColdFusion Administrator建立数据源后，与数据源的集成只需指定数据源的名字，即可执行对数据源的操作。通常，这要用到CFQUERY标签，如下例：

```
<CFQUERY NAME="EmployeeName" DATASOURCE="BigCompany">
```

```
SELECT FirstName + " " + LastName
```

```
FROM Employees AS FullName
```

```
</CFQUERY>
```

在这个例子中，用DATASOURCE属性指定数据源的名字，表示要执行的SELECT语句作用于这个数据源。它要求该数据源已经建立才行。

计划ColdFusion应用

在开始编写代码之前，应该先有一个总体的规划，包括：

- 研究实际问题，制定解决方案，明确应用程序的功能。
- 确定应用程序需要什么数据，需要输出什么数据。
- 确定应用程序是否需要客户安全性或资源安全性。
- 设计应用程序的整体结构。

许多程序开发者都发现，在编写代码之前先画出程序流程图是有必要的，流程图可以使编程时思路更加清晰。当然，在编写代码的过程中，随时可以调整计划，使之更加完善。

ColdFusion 4.0的特点：

ColdFusion4.0是ColdFusion发展史上的一个里程碑，与以前的版本相比，ColdFusion4.0有许多新的特性或增强特性，简述如下：

快速性开发

简单而快速地开发web应用程序一直是ColdFusion的重要特征，在ColdFusion4.0中，ColdFusion Studio与ColdFusion Server结合得更加紧密，可视化工具的功能进一步增强，CFML语言的功能也得到了扩展，从而使程序的开发更加快速。

可升级性配置

ColdFusion已可以用来处理非常大的站点和服务于上万个客户的web应用程序。在4.0版中提供了强大的新特性，显著地增强了可升级性。如下载平衡，多服务器聚集，高级线程池，以及与Windows NT性能监视器相集成等。

开放式集成

ColdFusion比其它IRAD系统提供了与邮件服务器，web服务器，和目录系

统的更好的集成。在4.0版中，这种集成已经可以支持XML（ Extensible Markup Language ）。

安全性

ColdFusion目前在开发和管理两方面提供了安全环境。4.0版的安全特性允许对开发和管理进行更复杂的安全控制，如开放的认证系统，远程开发安全性等。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

ColdFusion 的组成部分：

作为一个集成开发系统，ColdFusion的主要组成部分简述如下：

ColdFusion服务器（ColdFusion Server）

ColdFusion Server是ColdFusion的核心部分，它在NT中运行为一项服务，在WIN95中作为应用程序运行，其作用是等待Web服务器的请求。当客户（浏览器）向Web服务器请求文件时，发生如下过程：Web服务器根据文件名判断文件的类型，若是ColdFusion文件（后缀名为CFM），Web服务器转而向ColdFusion Server请求该文件，ColdFusion Server对该文件进行处理后，产生一个标准的HTML文件，交付给Web服务器，Web服务器再将HTML文返回到客户端。

ColdFusion管理器（ColdFusion administrator）

系统管理员用ColdFusion管理器来配置ColdFusion服务器的各种选项，包括：

- ColdFusion 数据源

- 调试信息

·服务器设置

·应用程序安全性

·服务器集

·执行预定网页

·目录映射

ColdFusion工作室 (ColdFusion Studio)

ColdFusion Studio是ColdFusion的开发环境。它提供了可视化的开发工具，包括动态预览，交互式的调试，查询生成器，表达式生成器，工程管理，以及一些控件工具和其他工具，用这些工具可以快速方便地产生ColdFusion应用程序文件，也可以产生标准的HTML文件。ColdFusion Studio的功能主要包括：

·查看数据源

·快速生成SQL语句

- 从标签选择器中使用HTML和CFML的所有标签
- 在特定标签的编辑器中或标签检查器中编辑代码
- 在内部浏览器或外部浏览器中显示网页，在设计窗口中可视地编辑网页。
- 建立工程来管理应用程序文件，使文件易于维护和上载。
- 使用扩展的查询和替换，快速更新一组文件。
- 存储代码片段，以便重复使用。
- 用表达式生成器生成包含ColdFusion函数，常量，变量，和运算符的表达式。
- 调试程序代码。
- 查看站点的整体结构。
- 检查HTML代码和CFML代码的合法性。

- 检验单个文件或整个工程的链接。

- 启动文件的版本控件，以利于团队开发。

ColdFusion应用程序

ColdFusion应用程序由一组后缀名为CFM的文本文件构成，这些文件实现了该应用程序的功能，其中一个重要功能是接收客户输入和格式化输出的客户界面。CFM文件中可以包含ColdFusion标签，HTML标签，CFScript,JavaScript,以及任何可以嵌入HTML文件中的内容。

ODBC数据源

ColdFusion应用程序可以操纵任何支持ODBC标准的数据库。

其它数据源

ColdFusion并不只限于操纵ODBC数据源。你也可以通过OLEDB，本地数据库驱动程序，支持LDAP的目录服务器，支持POP的邮件服务器等途径来取得数据。

ColdFusion扩展

ColdFusion提供了一个基于XML的开放式框架，用来以新的服务器组件扩展ColdFusion的功能，并可以使用COM，CORBA，C/C++，VBScript，JavaScript,ActiveX,或者CFML等与企业系统建立连接。

ColdFusion 文档

ColdFusion文档包括两个版本，打印版和联机版，两个版本都很容易定位到你所需要的信息。文档的内容包括：

- Getting Started with ColdFusion——包括系统的安装和基本配置，ColdFusion开发系统的组成，以及CFML语言的简介。

- Administering ColdFusion Server——讲述如何配置系统以提高性能，如何管理数据源，设置安全等级，以及其他有关开发环境和站点管理的选项设置。

- Developing Web Applications with ColdFusion——ColdFusion应用程序开发基础。包括数据源，客户界面，和Web技术。详细介绍了ColdFusion Studio的开发工具。

- Advanced ColdFusion Development——介绍ColdFusion的函数，表达式，数组，脚本，以及XML数据交换。还讨论了客户定制标签，CF API标签，集成对象技术，以及站点管理。

- CFML Language Reference——提供了所有CFML元素的完整语法和例子程

序。

·Quick Reference Card——CFML的联机参考 (Acrobat) 。

你可以通过如下途径获得ColdFusion的文档信息：

·PDF格式的文档存储在web根目录的cfdocs子目录下，可以用Acrobat阅读这些文档。

·在任务栏的ColdFusion Sever程序组中，运行ColdFusion Documentation,可以阅读ColdFusion联机文档。

·在ColdFusion Studio中使用联机帮助。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

建立ColdFusion应用程序

建立完整的ColdFusion应用程序涉及如下几个方面的内容：

·计划ColdFusion应用。

·建立应用程序文件。

·提供数据源。

·实现安全性。

·调试应用程序。

下面对这几项内容作一简要介绍。

手册的内容

根据以上的概述可以看出，ColdFusion4.0 包括很多方面的内容，掌握这些内容可以开发出非常强大的Web应用程序。作为 入门级手册，本手册将不会涵盖ColdFusion的全部。本手册主要侧重于CFML语言的使用，通过本手册的学习，读者将学会如何用CFML语言编写交互式的Web应用程序。而关于ColdFusion Administrator 和ColdFusion Studio等方面的详细内容，请读者参考其他书籍或ColdFusion的文档。

学习本手册的内容不需要读者有开发Web应用程序的经验，但应具备HTML的基本知识以及关于WWW的基本概念。

变量的创建

在ColdFusion中，一个变量就是分配了一个值的参数，变量的使用存在于ColdFusion应用程序的方方面面，不使用变量而开发应用程序是不可想象的。例如，在处理表单输入时，要用变量将表单中的数据传递给ACTION属性所指定的网页。这种变量也被称为动态参数。

变量的创建

用CFSET创建变量

创建变量的最简单的方法是用CFSET标签，你可以在应用程序的任何地方使用CFSET标签，所创建的变量可以在同一文件中该标签后面的任何位置引用。只需指定变量名，并分配一个静态值或一个参数，或表达式。其语法为：

```
<CFSET VariableName = Value, Parameter, or Expression>
```

例1：静态值

```
<CFSET FirstName = "Jack">
```

此例创建了一个变量 `FirstName`，其值为静态值 `Jack`，注意变量的值为字符串时要使用双引号。再如，创建一个变量 `UserNumber`，其值为数 `26`：

```
<CFSET UserNumber=26>
```

例2：动态参数

```
<CFSET CurrentUserID=GetUserID.User_ID>
```

例中，`GetUserID`是一个数据库查询操作的查询名，`User_ID`是所查询的一个字段名。此例定义了变量 `CurrentUserID`，其值为一个数据库查询结果的 `User_ID`字段的值。再如：

```
<CFSET UserDescription = "#UserName# is a wonderful person.">
```

其中 `UserName`是另一个变量。

例3：表达式

你可以用 `CFSET` 创建一个变量，将一个表达式的值分配给它，如：

```
<CFSET TotalValue = 2*(4+5)>
```

```
<CFSET Pay="John's take home pay is"&(TotalValue-1000)>
```

注意：在ColdFusion中，变量没有类型之分，也就是说，你不需要指定变量值的数据类型。

检测变量的存在

在使用一个变量之前，你可以用函数IsDefined检测它是否存在。例如，下面的例子检测名为Order_ID的变量是否存在：

```
<CFIF Not IsDefined("FORM.Order_ID")>
```

```
<CFLOCATION URL="previous_page.cfm">
```

```
</CFIF>
```

如果你试图使用一个没有定义的变量，ColdFusion将无法处理该文件。

用CFPARAM创建缺省变量

创建变量的另一个方法是使用CFPARAM标签，该标签先检测变量是否存在，如果该变量不存在，则提供一个缺省值。CFPARAM标签的语法为：

```
<CFPARAM NAME="VariableName" DEFAULT="DefaultValue">
```

根据不同的需要，使用CFPARAM标签的方式有两种：

·只提供NAME属性，检测所必需的变量是否存在，如果不存在，ColdFusion Server停止处理该文件。例如，下面的语句指出，该文件需要两个变量StartRow和RowsToFetch,处理才能继续进行：

```
<CFPARAM NAME="Form.StartRow">
```

```
<CFPARAM NAME="Form.RowsToFetch">
```

如果包含这两个标签的文件被调用，但没有提供两个变量中的任一个，则会发生错误，文件被停止处理。

·同时提供NAME属性和DEFAULT属性，检测一个可选的变量是否存在，如果存在，则处理继续进行，该变量的值不作改变；如果变量不存在，则创建该变量，赋值为DEFAULT属性所提供的值。例如：

```
<CFPARAM NAME="cookie.SearchString" DEFAULT="temple">
```

你可以用CFPARAM标签代替分支语句，为URL及Form变量设置缺省值。

在网页中显示变量及井号的使用

要在网页中显示一个变量，需把变量包含在CFOUTPUT标签中，例如：

```
<CFOUTPUT>
```

```
Your favorite color is #Clint.FavoriteColor#.
```

```
</CFOUTPUT>
```

注意，在CFOUTPUT标签内，要始终把变量名括在井号（#）中。这表示该变量名要被当作动态参数来计算其值，ColdFusion将输出变量的值而不是变量名本身。

当要区分一个字符串是变量还是文本时，需要使用井号，以表示该位置要用变量的值来代替。例如：

```
<CFQUERY NAME="search" DATASOURCE="Company">
```

```
Select * From Employees Where LastName='#Form.LastName#'
```

```
</CFQUERY>
```

在CFSET标签中，不要滥用井号，例如，不要用`<CFSET x=#Cos(0)#+>`；而要用

```
<CFSET X=Cos(0)+1>
```

由于井号在ColdFusion中有特殊用途，要表示井号本身时，需用两个连续的井号来表示。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

变量命名规范

ColdFusion变量及form字段的命名应遵循如下规范：

- 变量名必须以字母开头，后面可以跟任意多个字母，数字，或下划线。

- 变量名中不能包含空格和特殊字符。例

如，UserName_1,UserName_2,UserName是合法的命名，1stUset,WhatAName!,和User-Name都是非法的变量名。

- 对字段名和变量名，应使用描述性的名字，而不是简写。这使其他人更容易读懂你的程序，也帮助你自己记住该变量的用途。

- 注意，在同一个CFM文件中，变量名和查询名不能相同。

- ColdFusion变量不区分大小写。

变量的引用范围

你已经注意到，在上面的某些例子中，变量名带有前缀，下面就将讨论这方面的内容。

ColdFusion变量没有数据类型之分，但变量有不同的来源（如来自表单，查询等）和不同的引用范围（如本文件范围内，整个应用程序范围内等）。ColdFusion用前缀来区分不同来源或引用范围的同名变量。例如，Form.State表示来自Form提交的变量State，而URL.State表示来自URL的变量State，这是两个不同的变量。在创建变量时，除非两个不同范围的变量同名，你不必要使用前缀，但使用前缀使程序更加清晰，并且可提高处理速度。

下表列出了ColdFusion中变量前缀的类型，关于这些类型的详细内容将在后文讨论：

变量前缀	
类型	使用方法
查询变量	QueryName.variablename
局部变量	Variables.variablename
URL参数变量	URL.variablename
Form字段变量	Form.variablename
Client变量	Client.variablename
Server变量	Server.variablename

Session变量	Session.variablename
应用程序变量	Application.variablename
HTTP Cookies变量	Cookie.variablename
CGI环境变量	CGI.variablename

当你没有指定变量前缀时，ColdFusion将按如下的顺序搜索该变量：

1. 用CFSET和CFQUERY创建的局部变量
2. CGI变量
3. File变量
4. URL变量
5. Form变量
6. Cookie变量
7. Client变量

注意：Coldfusion不会自动搜索Application变量和Session变量，定义和使用这两种变量必须用前缀。

变量的种类

以下对ColdFusion的各种变量进行逐一介绍。

局部变量

局部变量是用CFSET标签和CFPARAM标签所创建的变量的缺省类型。这种变量只能在创建它的文件中或包含该文件的文件中引用。

查询变量

一旦执行了一个查询，就可以通过查询变量来访问查询的结果。查询变量的前缀是查询名，变量名即是表中的字段名。例如，你执行了一个名为LookupUser的查询，所查询的字段之一是名为ID的字段，那么你可以用变量LookupUser.ID来访问查询结果中的ID字段的值。

URL变量

URL变量以variablename=value的形式追加在URL后面，传递给URL所指的网页，在该网页中用URL.variablename引用该变量。

Form变量

网页之间传递参数的最常用方法就是用Form变量。当客户在Form字段中输入数据并提交，其输入的数据以Form变量的形式传递给ACTION属性所指的网页，在该网页中用Form.variablename引用该变量，variablename即是Form字段的名称。

Cookie变量

Cookie是服务器端的应用程序（如ColdFusion）在浏览器中存储信息的一种机制。存储在浏览器中的Cookie可以返回给服务器端应用程序。利用Cookie，应用程序可以创建与某个特定浏览器相关的变量。例如，你可以为背景颜色建立一个Cookie，从而使你的站点在不同客户的浏览器中具有不同的颜色。

Cookie是与特定站点相关的，当浏览器访问一个特定的站点时，基于该站点的Cookie被传至服务器端。

可以用SSL(Secure Sockets layer)对Cookie进行加密。Cookie将一直存在，直到它们到期或被删除。目前的主流浏览器都支持Cookie。

创建Cookie

标签CFCOOKIE用来建立Cookie，例如：

```
<CFCOOKIE NAME="User_ID" VALUE="2344"
```

```
EXPIRES="100">
```

这个例子创建了变量Cookie.User_ID，其值为2344，将在100天后到期。

Cookie建立之后，同一站点上的所有ColdFusion网页均可以引用该Cookie。这意味着你可以用Cookie在网页间传递参数。注意，Cookie不适用于用来存储安全信息，如密码，信用卡号码等。

注意：如果在创建Cookie的文件中，在CFCOOKIE标签后执行了CFLOCATION标签，则CFCOOKIE标签创建的Cookie将会丢失。

在网页中使用Cookie

一旦在客户浏览器中存储了Cookie，每当浏览器请求你站点的网页时，Cookie变量会被自动传给Web服务器。与其它类型的变量一样，你可以使用前缀来引用Cookie变量的值，例如：

```
<CFOUTPUT> #Cookie.User_ID# </CFOUTPUT>
```

删除Cookie

要是删除一个Cookie，你可以用CFCOOKIE标签，将EXPIRES属性设为“now”，例如：

```
<CFCOOKIE NAME="User_ID" VALUE="#User_ID#"
EXPIRES="now">
```

当浏览器关闭时，这个Cookie会被删除。

Client变量

Client变量是与特定客户相关的变量。当客户在一个应用程序的不同文件间移动（此处“移动”指客户从一个文件中链接到另一个文件）时，Client变量可以用来维持客户的状态，识别一个特定的客户。

对一个启动了客户状态管理的应用程序，ColdFusion为每一个请求该应用程序文件的浏览器建立一个客户记录，客户记录由一个唯一的令牌来标识，令牌存储在客户浏览器的一个Cookie中。有了客户记录后，就可以在客户记录中定义Client变量，这些变量可在整个应用程序的任何文件中引用。当客户请求一个文件时，ColdFusion根据浏览器提供的令牌来选择相应的客户记录，于是所有的Client变量均可以使用。

创建client变量的步骤如下：

1. 在Application.cfm文件的CFAPPLICATION标签中，设置CLIENTMANAGEMENT="yes",启动客户状态管理。（关于Application.cfm文件后文有介绍）
2. 在CFAPPLICATION标签中，通过设置CLIENTSTORAGE属性，选择client变量的存储位置。其缺省位置在ColdFusion Administrator中设置，可以是注册表，已存在的数据源，或Cookies。
3. 用CFSET或CFPARAM创建该变量。例如：

```
<CFSET Client.FavoriteColor="Conflower Blue">
```

标准Client变量

ColdFusion提供了几个标准Client变量，这些变量有助于记录客户访问你站点的次数和上次访问的时间等信息。例如，下面的例子显示了客户上次访问你站点的时间：

```
<CFOUTPUT>
```

```
Welcome back to the Web SuperShop, your last visit
```

```
was on #DateFormat(Client.LastVisit)#.
```

```
</CFOUTPUT>
```

标准Client变量具有只读属性，你可以访问它们，但不能在程序中设定它们

的值。这些变量包括：

·CFID

·CFToken

·URLToken

·HitCount

·TimeCreated

·LastVisit

不利用Cookie而实现客户状态管理

在Coldfusion中，客户状态管理被设计为使用Cookie，这是识别一个客户的标准方法。你也可以不利用Cookie而实现客户状态管理，要做到这一点，你必须通过隐藏的Form字段或URL变量，在网页之间传递客户ID（CFID）和客户安全令牌（CFTOKEN），这需要教高的编程技巧。

Client变量的存储

Client变量的缺省存储位置是系统注册表，你可以用ColdFusion Administrator改变其缺省位置，将Client变量存在SQL数据库或Cookie中。在Application.cfm文件中用CFAPPLICATION标签的CLIENTSTORAGE属性可以设置Client变量的存储位置，该设置将覆盖ColdFusion Administrator的缺省设置。例如：

```
<CFAPPLICATION NAME="myapplication"
```

```
CLIENTMANAGEMENT="Yes"
```

```
CLIENTSTORAGE="mydatasource">
```

如果不设置CLIENTSTORAGE属性，ColdFusion将使用缺省设置。注意，Client变量的存储机制具有排斥性，当使用某一种存储类型时，其他存储类型的Client变量不可访问。

用Cookie存储Client变量时，应考虑如下的限制因素：

- Netscape Navigator只允许为一个特定主机设置20个Cookie。ColdFusion使用了两个Cookie用于CFID和CFTOKEN，并用一个名为CFGLOBALS的Cookie存储客户的全局数据如HitCount,TimeCreated,和LastVisit。

- Netscape Navigator设定了每个Cookie不能大于4K字节的限制。也就是说你不能在Cookie中存储过多的数据。

·如果客户浏览器禁止使用Cookie，Client变量将不能工作。

得到Client变量列表

使用函数GetClientVariableslist()可以获得Client变量的列表，该函数返回的是用逗号隔开的，由CFAPPLICATION所确定的应用程序范围内的Client变量的名字。标准Client变量不在返回列表中。

删除Client变量

用函数DeleteClientVariable()删除Client变量，例如：

```
<CFSET IsDeleteSuccessful=DeleteClientVariable("MyClientVariable")>
```

该函数只能删除由CFAPPLICATION确定的应用程序范围内的Client变量。

同样，通过ColdFusion Administrator中的设置，可以在一个特定的天数之后删除Client变量，对存储在注册表中的变量，其缺省值是90天，对存储在数据库中的变量，缺省值是10天。注意，你不能删除系统提供的标准Client变量。

Application变量和Session变量

Application变量和Session变量是整个应用程序范围内的变量，在应用程序的任何文件中都可以访问。你要在变量名前面加前缀才能访问这些变量，例如，“Session.MyVariable”，“Application.MyVariable”。由于它们可在整个应用程序范围内访问，因此可以利用它们在网页间传递数据。

允许Application变量和Session变量

与Client变量一样，可以用CFAPPLICATION标签允许Application变量和Session变量。（具体语法请参阅ColdFusion文档CFML language reference部分）

与Client变量不同的是，Application变量和Session变量始终保存在服务器的内存中，这使得它们的存取速度更快。另外，你可以用ColdFusion Administrator或在CFAPPLICATION中，设置这两种变量的生存时间。你也可以完全禁止Application变量和Session变量。

Session变量

与Client变量一样，Session变量与某个特定的客户相关，它也需要一个客户ID，并始终与客户ID连系在一起。当客户第一次请求一个ColdFusion应用程序文件时，即建立一个Session，Session有一定的生存时间，如果在限定的时间内客户没有向服务器发出请求，则该Session结束。

如果两个客户连接到服务器上的同一个应用程序，这是两个独立的Session，每个客户都有自己的Session变量，它们可能有不同的值，客户不能访问别人的Session变量。

Session变量适于存储与特定客户有关的全局性信息。当客户第一次请求网页时，有关信息被初始化在Session变量中，客户访问其它网页时，这些信息仍可以直接使用。

要允许Session变量，可在Application.cfm文件的CFAPPLICATION标签中设置SESSIONMANAGEMENT

Session变量的生存期

Session变量有一个特定的生存期，正是这个生存期定义了一个“Session”。当你在一个Session变量的生存期内访问它时，该变量可以访问，如果超过了生存期，该变量将不再存在。

Session变量的缺省过期时间是20分钟，即如果浏览器在20分钟内没有与服务器联系，该Session将被结束。你可以用ColdFusion Administrator改变这一缺省设置。

你也可以用CFAPPLICATION标签的SESSIONTIMEOUT属性来设置Session变量的过期时间，这将覆盖Administrator的缺省设置。

Application变量

Application变量也是可以在整个应用程序范围内访问的变量。与Session变量不同的是，Application变量不针对特定的客户，它不需要与特定的客户ID相关联，一个应用程序的所有客户可以访问同一个Application变量。

这种变量适于存储应用程序级的不与特定客户相关的，由所有客户共同使用的数据。

这里有必要解释一下什么是“应用程序”。在ColdFusion中，通过在Application.cfm文件的CFAPPLICATION标签中设置NAME属性，为应用程序指定一个应用程序名，所有具有相同应用程序名的文件属于同一个应用程序，而同一个应用程序的所有文件共享所有的Application变量，这些变量对所有的客户都相同。（关于Application.cfm文件及“应用程序”的定义，后文有更多叙述）

Application变量的生存期

应用程序也有一定的生存期，当第一个访问者请求一个应用程序中的某个文件时，ColdFusion初始化该应用程序，如果在限定的时间内没有客户再请求该应用程序的文件，则该应用程序被结束，Application变量不再存在。

Application变量的缺省过期时间是2天。你可以用ColdFusion Administrator改变这个缺省值。你也可以用CFAPPLICATION标签的APPLICATIONTIMEOUT属性设置Application变量的过期时间，从而覆盖Administrator的缺省设置。

注意：使用Application变量和Session变量时，必须使用前缀。

Server变量

Server变量可以在当前Web服务器所服务的所有应用程序中，由所有的客户引用，直到ColdFusion Server关闭。

Server变量适于存储不经常改变的，可以被多个客户和多个应用程序共享的数据。你可以用Server变量存储运行在你站点上的所有应用程序都需要的信息，例如关于某个数据源的信息。

ColdFusion带有下列内建的Server变量：

·Server.ColdFusion.ProductName----返回ColdFusion产品名。

·Server.ColdFusion.ProductVersion---返回ColdFusion的版本信息。

·Server.ColdFusion.ProductLevel---返回ColdFusion的产品等级信息。

·Server.ColdFusion.SerialNumber---返回ColdFusion的序列号。

·Server.OS.Name---返回服务器操作系统的名字。

·Server.OS.AdditionalInformation---返回关于操作系统的附加信息。

·Server.OS.Version---返回服务器操作系统的版本信息。

·Server.OS.BuildNumber---返回服务器操作系统的建立号。

注意：Server变量具有读 - 写属性，小心不要覆盖了内建的Server变量。

CGI环境变量

每当浏览器向Web服务器请求服务时，就会创建一系列环境变量，有些由浏览器创建，有些由服务器创建。这些变量称为CGI变量，用前缀“CGI”来引用（即使服务器使用API而不是CGI与ColdFusion相通信）。

环境变量中是浏览器和服务器之间对话的一些相关数据，如IP地址，浏览器类型，通过认证的客户名等。你可以在浏览器所请求的文件中的任何位置引用CGI环境变量。所有的CGI变量都是只读的。

注意，在你的应用程序中可以引用的CGI变量与具体的浏览器和Web服务器软件有关。

检测CGI变量

因为并非每种浏览器都支持所有的CGI变量，当测试一个CGI变量的存在时，ColdFusion总是返回TRUE，即使浏览器并不支持该变量。避免这种情况的方法是检测一个空字符串，而不是检测逻辑值，例如：

```
<CFIF CGI.varname IS NOT "">
```

CGI variable exists

```
<CFELSE>
```

CGI variable does not exist

```
</CFIF>
```

服务器CGI变量

下表列出了由服务器创建的最常用的CGI变量（并非每种浏览器都支持这些变量）：

服务器CGI变量	
变量	描述

SERVER_SOFTWARE	响应请求的服务器软件的名字和版本。格式：名字/版本。
SERVER_NAME	服务器的主机名，DNS别名，或IP地址。
GATEWAY_INTERFACE	服务器所遵循的CGI规范的版本。格式：CGI/版本。
SERVER_PROTOCOL	这次请求所使用的协议的名字和版本。格式：protocol/版本
SERVER_PORT	请求传至的端口号。
REQUEST_METHOD	请求服务时使用的方法，对HTTP是Get,Head,Post等。
PATH_INFO	由浏览器提供的特殊路径信息。访问脚本可以使用虚拟路径，后面跟以特殊路径信息，该信息存在PATH_INFO中。
PATH_TRANSLATED	服务器转换过的PATH_INFO，虚拟路径映射为物理路径。
SCRIPT_NAME	指向要执行的脚本的虚拟路径，用于自引用的URLs.
QUERY_STRING	跟在URL中“？”后面的查询信息。
REMOTE_HOST	请求服务的主机名。如果服务器不知道这项信息，它将创建REMOTE_ADDR,而不是REMOTE_HOST。
REMOTE_ADDR	请求服务的远程主机的IP地址。
AUTH_TYPE	如果服务器支持客户认证，并且脚本受保护，此变量包含基于特定协议的认证方法。
REMOTE_USER AUTH_USER	如果服务器支持客户认证，并且脚本受保护，此变量包含已认证的客户名。

REMOTE_IDENT	如果HTTP服务器支持RFC931标识，这个变量被设置为远程的客户名。
CONTENT_TYPE	对带有附加信息的查询，如HTTP POST和PUT，此变量指出附加信息的类型。
CONTENT_LENGTH	客户提供的信息的长度。

客户CGI变量

下表列出了由浏览器创建后传递给服务器的，最常用的CGI环境变量：

客户CGI变量	
变量	描述
HTTP_REFERER	引用的文档。这是链接所来自的文档或提交Form数据的文档。
HTTP_USER_AGENT	客户当前使用的浏览器。格式：软件/版本库/版本。

利用URL和Form传递变量

上文中已讲到，利用Client变量，Session变量，Application变量等，可以在不同的网页之间传递数据，下面介绍利用URL和FORM来传递变量的方法。这两种方法可以在网页之间传递局部变量。

利用URL传递参数

你可以把参数附加在URL中，传递给该URL所指向的目标文件。用一个问号将要传递的参数与URL的文件地址隔开，在问号后面追加变量名及其值，于是在目标文件内就可以引用该变量，这种变量即前文所述的URL变量。例如：

```
<A HREF="example.cfm?user_id=5&color=#mycolor#">
```

在这个例子中，变量user_id,值为5，和变量color，值为表达式 # mycolor# 的值，被传递给超链接所指的文件example.cfm。

在目标文件example.cfm中，你可以用URL.user_id和URL.color的形式来引用这两个变量，如：

```
<CFOUTPUT>
```

Your user ID is #URL.user_id# and

your favorite color is #URL.mycolor#.

</CFOUTPUT>

利用URL传参时，应注意以下几点：

- 用一个问号将URL地址与要传递的参数隔开。
- 用符号“&”将不同的参数隔开。
- 不要使用空格，有些浏览器在检测到空格时，会将URL截断。
- 不要使用特殊字符，如&?,.,和#。

如果你要传递的值有可能包含空格和特殊字符，应使用函数
URLEncodeFormat（），例如：

<CFSET FullName="Bob Smith">

```
<CFOUTPUT>
```

```
<A HREF="printname.cfm?FullName=#URLEncodedFormat(FullName)#">
```

```
Click here</A>
```

```
</CFOUTPUT>
```

关于此函数的详细信息请参阅CFML language reference。

用Form传递参数

通常，使用Form接收客户的输入，然后在Action属性所指的文件中利用Form变量得到客户输入的数据并作处理。你也可以利用隐藏字段从一个网页向另一个网页传递参数。例如，下面的Form中包含一个名为Customer_ID的隐藏字段，被传递到文件example.cfm中。

```
<FORM ACTION="example.cfm" METHOD="Post">
```

```
<INPUT TYPE="Hidden"
```

```
NAME="Customer_ID" VALUE="24">
```

```
<INPUT TYPE="Submit" VALUE="Enter">
```

```
</FORM>
```

在文件example.cfm中，你可以用Form.Customer_ID的形式引用该变量。

也可以用隐藏字段传递动态参数，如查询结果等，例如：

```
<FORM ACTION="example.cfm" METHOD="Post">
```

```
<CFOUTPUT QUERY="GetCustomer">
```

```
<INPUT TYPE="Hidden" NAME="Customer_ID"
```

```
VALUE="#Customer_ID#">
```

```
</CFOUTPUT>
```

```
<INPUT TYPE="Submit" VALUE="Enter">
```

</FORM>

小节：

本章介绍了ColdFusion的变量，其要点是各种变量的引用范围和生存期，另外要注意井号的使用。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

条件选择

每当客户请求一个应用程序文件时，ColdFusion根据该文件的CFML标签和HTML标签，动态地生成一个HTML网页。程序流程控制使你可以控制ColdFusion对文件的处理，根据不同的条件生成不同内容的网页。

条件选择

ColdFusion提供了两种处理条件选择的方法：CFSWITCH和CFIF。

使用CFSWITCH，CFCASE和CFDEFAULTCASE

CFSWITCH标签计算一个表达式的值，然后将控制传给与表达式的值相匹配的CFCASE标签。你也可以提供一个CFDEFAULTCASE标签，当所有的CFCASE标签都不能与表达式的值相匹配时，控制转到CFDEFAULTCASE标签。

使用CFSWITCH标签比使用一串CFIF/CFELSEIF标签的效率要高，而且代码易于阅读。但是，CFSWITCH标签的VALUE属性必须是已知的常量。

下面的简单例子显示了CFSWITCH标签的语法：

<CFSWITCH EXPRESSION=#Switch#>

<CFCASE VALUE="4"> Case four </CFCASE>

<CFCASE VALUE="1"> Case one </CFCASE>

<CFCASE VALUE="2"> Case two </CFCASE>

<CFCASE VALUE="3"> Case three </CFCASE>

<CFCASE VALUE="2.5"> Case two and a half </CFCASE>

<CFCASE VALUE="5"> Case five </CFCASE>

<CFCASE VALUE="6"> Case six </CFCASE>

<CFDEFAULTCASE> Default case </CFDEFAULTCASE>

</CFSWITCH>

每个CFCASE标签的VALUE属性都是一个常量，CFSWITCH标签将表达式的值与各个CFCASE标签的VALUE值作比较，并执行相匹配的CFCASE标签内的代码。

使用CFIF，CFELSEIF和CFELSE

你也可以用CFIF，CFELSEIF和CFELSE来选择执行某一段代码。典型的用法如下，其中CFELSE不是必须要的。：

```
<CFIF value operator value>
```

```
... HTML and CFML tags
```

```
<CFELSE>
```

```
... HTML and CFML tags
```

```
</CFIF>
```

注意，在CFIF中要使用运算符，如IS，IS NOT等，而不是等于号。

例1：根据条件返回查询结果

要检测一个查询是否返回了记录，你可以检测该查询的记录数是否为0：

```
<CFIF #CustomerSearch.RecordCount# IS 0>
```

```
<!-- Inform user that we had no hits -->
```

```
<P>Sorry, no customers matching your
```

```
criteria were found.</P>
```

```
<CFELSE>
```

```
<!-- Show the list of customers retrieved -->
```

```
<CFOUTPUT Query="Customers">
```

```
#FirstName# #LastName# <BR>
```

```
</CFOUTPUT>
```

```
</CFIF>
```

在下面的这个例子中，只有客户明确地请求这段代码的内容时（记录在变量showcustomers中），才显示这部分内容：

```
<CFIF #Form.ShowCustomers# IS "Yes">
```

```
Customer List: <P>
```

```
<CFOUTPUT Query="Customers">
```

```
#FirstName# #LastName# <BR>
```

```
</CFOUTPUT>
```

```
</CFIF>
```

例2：根据条件显示记录内容

条件选择标签最强大的功能之一是对查询结果的每条记录作不同的处理。要实现这一点，需把条件选择标签放在CFOUTPUT代码段中。这时，将对查询结果中的每条记录循环执行条件选择语句。这使你可以根据一个记录中是否包含某个字段而作不同的显示。例如，在下面的例中，并非所有的

记录都有电话号码，因此，将CFIF放在CFOUTPUT中，确保只有Phone字段不为空时，才显示电话号码的内容。

```
<CFOUTPUT QUERY="Contacts">
```

```
<HR>
```

```
Name: #Name# <BR>
```

```
Title: #Title# <BR>
```

```
<CFIF #Phone# IS NOT "">
```

```
Phone: #Phone# <BR>
```

```
</CFIF>
```

```
</CFOUTPUT>
```

综合条件选择语句

综合条件选择就是用逻辑运算符将多个条件结合起来。其典型语法为：

```
<CFIF (value operator value) Boolean Operator
```

```
(value operator value) Boolean Operator
```

```
(value operator value)>
```

```
... HTML and CFML tags
```

```
</CFIF>
```

最常用的逻辑运算符是AND,OR,和NOT。

例如，下面的例子假设一个名为“GetEmployee”的查询返回了一个雇员的有关信息。如果查询结果表明该雇员是销售部的，并且获得了超过5000元的奖金，则显示一条信息。

```
<CFIF (#GetEmployee.Department# IS "Sales") AND
```

```
(#GetEmployee.Bonus# GE 5000)>
```

<H4>Congratulations on your sales bonus!</H4>

</CFIF>

使用CFELSEIF

CFELSEIF使你能够进行复杂的条件处理，将多个条件结合起来，而不需要使用嵌套的IF语句。其语法为：

<CFIF condition1>

Display this text only if condition1 is true.

<CFELSEIF condition2>

Display this text only if condition1 is false and condition2 is true.

<CFELSEIF condition3>

Display this text only if condition1

and condition2 are false and condition3 is true.

<CFELSE>

Display this if condition1, condition2,

and condition3 are false.

</CFIF>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

重定向（CFLOCATION）

使用CFLOCATION标签，你可以将对某个文件的请求重定向到另一个文件或另一个URL。这有助于定义一个网页，在其中执行一个或多个查询，然后转到另一个网页，或者你希望客户所指向的URL依赖于一个动态参数。

例如，你可以用CFIF检测一个客户是否已登录（客户密码是否经过了确认？），如果没有，则将该客户重定向到另一个网页，在该网页中告诉客户其密码没有经过确认：

```
<CFIF #NewPassword# IS NOT '#PasswordConfirmation#'
```

```
<CFLOCATION URL="invalidpassword.cfm">
```

```
</CFIF>
```

也可以在CFLOCATION标签的URL属性中使用动态参数。例如，要根据一个名为“Page”的变量动态地决定将客户重定向到哪个网页，可以用如下方法：

```
<CFLOCATION URL="#Page#">
```

后页
前页
目录
北极星书库

停止处理文件 (CFABORT)

ColdFusion处理CFM文件时若遇到CFABORT标签，则停止文件处理，将CFABORT标签之前的处理结果返回给浏览器。CFABORT通常用来根据特定的条件而停止文件的处理。如下例所示，CFABORT标签没有属性：

```
<P>This HTML content is returned to in the final page.</P>
```

```
<CFABORT>
```

```
<P>This HTML content is NOT returned in the final page.</P>
```

包含文件

当你的ColdFusion应用程序越来越复杂时，应该考虑代码的重用。CFINCLUDE标签可以实现这个目的。该标签将另一个文件插入当前的文件，从而很方便地重复使用通用的代码。

注意：使用CFINCLUDE时不要将CFIF，CFLOOP，或CFOUTPUT等代码块分隔到不同的文件中。

CFINCLUDE标签有一个TEMPLATE属性，该属性指定了一个已经存在的文件，当ColdFusion遇到CFINCLUDE标签时，会将这个文件视为当前文件的一部分来处理并返回到客户端。例如，如果你在一个文件中创建了一个变量，并使用了一个CFINCLUDE标签，则在被包含的文件中你可以引用这个创建于主文件中的变量。CFINCLUDE标签的语法为：

```
<CFINCLUDE TEMPLATE=" FileName">
```

此处，FileName代表被包含文件的相对路径。例如要包含当前文件的上一级目录中的文件index.cfm：

```
<CFINCLUDE TEMPLATE="../index.cfm">
```

在下面这几种情况下，要使用CFINCLUDE：

·你需要在不同的网页中显示同一个表格，可以建立一个文件用于显示该表格，然后将这个文件包含在需要它的地方，而不必从一个网页拷贝和粘贴相同的代码到另一个网页。

·你可以建立两个文件”header.cfm”和”footer.cfm”，分别将这两个文件包含在每个网页的开始和结束处，从而只要修改这两个文件就可以改变整个站点的外观。

·可以将复杂的CFIF/CFELSE嵌套作为包含文件，从而使程序更加清晰。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

建立循环 (CFLOOP)

循环是非常强大的编程技术，它使你能够根据特定的条件而重复执行某些代码。用CFLOOP可以执行四种类型的循环：

- 索引循环（也叫For循环）
- 条件循环（也叫While循环）
- 对一个查询进行循环
- 对一个列表进行循环
- 对一个COM集合进行循环

你通过CFLOOP标签的不同属性来选择使用不同的循环类型。

索引循环

索引循环根据一个数字范围而重复执行循环体，当你知道循环重复执行的次数时，可以使用索引循环。每次执行循环体时，变量INDEX就会加1。下面的例子循环5次，每次循环显示当前的索引值：

```
<CFLOOP INDEX="LoopCount" FROM="1" TO="5">
```

```
The loop index is <CFOUTPUT>#LoopCount#</CFOUTPUT>.<BR>
```

```
</CFLOOP>
```

这个循环在浏览器中显示的结果类似于：

The loop index is 1.

The loop index is 2.

The loop index is 3.

The loop index is 4.

The loop index is 5.

在缺省情况下，每次循环INDEX增加1，你可以设置STEP属性来改变这一缺省值。如下例：

```
<CFLOOP INDEX="LoopCount" FROM="5" TO="1" STEP="-1">
```

```
The loop index is <CFOUTPUT>#LoopCount#</CFOUTPUT>.<BR>
```

```
</CFLOOP>
```

在浏览器中显示的结果类似于：

The loop index is 5.

The loop index is 4.

The loop index is 3.

The loop index is 2.

The loop index is 1.

条件循环

条件循环当一个给定的条件为真时重复执行循环体。使用这种循环时，每次执行循环体都要改变循环条件，直到该条件的值成为“FALSE”。

下面的例子将参数“CountVar”从1增加到5。

```
<!-- Set the variable CountVar to 1 -->
```

```
<CFSET CountVar= 0>
```

```
<!-- Loop until CountVar is 5 -->
```

```
<CFLOOP CONDITION="CountVar LT 5">
```

```
<CFSET CountVar=#CountVar# + 1>
```

```
The loop index is <CFOUTPUT>#CountVar#</CFOUTPUT>.<BR>
```

```
</CFLOOP>
```

在浏览器中显示的结果类似于：

The loop index is 1.

The loop index is 2.

The loop index is 3.

The loop index is 4.

The loop index is 5.

查询循环

查询循环对查询结果记录集中的每条记录重复执行循环体。CFLOOP的结果与CFOUTPUT相似。每次执行循环体时，当前记录的各个字段可以被输出。但是，CFLOOP的执行速度比CFOUTPUT慢很多。

例1：用CFLOOP显示一个记录集

下面的例子中，CFLOOP实现了与CFOUTPUT相同的功能：

```
<CFQUERY NAME="MessageRecords" DATASOURCE="Customer">
```

```
SELECT * FROM Messages
```

```
</CFQUERY>
```

```
<CFLOOP QUERY="MessageRecords">
```

```
<CFOUTPUT> #Message_ID# </CFOUTPUT><BR>
```

```
</CFLOOP>
```

例2：一个查询结果的“下n个记录”

CFLOOP也可以对给定了起始点和结束点的一组记录进行循环。这样，你可以从第10个记录开始循环，到第20个记录结束循环。这种机制提供了一个简单的方法，用来得到一个记录集中的下n条记录。下面的例子对查询“MyQuery”的结果中第10到第20条记录进行循环：

```
<CFSET Start=10>
```

```
<CFSET End=20>
```

```
<CFLOOP QUERY="MyQuery" STARTROW="#Start#" ENDROW="#End#">
```

```
<CFOUTPUT>#MyQuery.MyColName#</CFOUTPUT><BR>
```

```
</CFLOOP>
```

当没有记录或当前记录的序号大于ENDROW的值时，循环结束。

例3：对全部查询结果进行循环

下面的例子用CFINCLUDE将一个查询所查到的所有文件包含到同一个文档中：

```
<CFQUERY NAME="GetFile" DATASOURCE="Library" MAXROWS=5>
```

```
SELECT FileName FROM Templates
```

```
</CFQUERY>
```

```
<CFLOOP QUERY="GetFile">
```

```
<CFINCLUDE TEMPLATE="#FileName#">
```

```
</CFLOOP>
```

如果你要对记录集中的所有记录进行循环，也可以使用CFOUTPUT，如下例所示：

```
<CFOUTPUT QUERY="MyQuery">
```

```
Text and #variablename#
```

```
</CFOUTPUT>
```

注意，在CFOUTPUT标签中指定QUERY属性，即表示对该查询结果的每条记录分别执行CFOUTPUT代码块。

列表循环

列表循环对一个列表中的所有元素分别执行循环体。在列表循环中，

用LIST属性指定一个列表或一个包含列表的变量，用INDEX属性指定一个变量名，该变量名用来引用列表中的下一个元素。下面的例子显示列表中的每个人名：

```
<CFLOOP INDEX="ListElement" LIST="John,Paul,George,Ringo">
```

```
<CFOUTPUT>#ListElement#</CFOUTPUT><BR>
```

```
</CFLOOP>
```

缺省情况下，CFLOOP认为列表中的元素以逗号为分隔符，但你可以在DELIMITER属性中指定你所喜欢的分隔符。下面的例子与上一个有相同的效果，不同的是逗号，冒号，和斜杠都会被视为元素分隔符。

```
<CFLOOP INDEX="ListElement"
```

```
LIST="John/Paul,George::Ringo" DELIMITERS=",:/ ">
```

```
<CFOUTPUT>#ListElement#</CFOUTPUT><BR>
```

```
</CFLOOP>
```

指定分隔符时不需要特定的顺序。注意连续的分隔符被视为单个分隔符；

因此上例中“George”和“Ringo”之间的两个冒号被视为一个。

对一个COM集合进行循环

CFLOOP的COLLECTION属性使你能对一个COM/DCOM集合对象进行循环。一个COM/DCOM集合对象是一组相似对象（称为“项”）的整体。例如，一个应用程序中一组打开的文档就是一种集合。你用CFLOOP的ITEM属性指定一个变量名，用来在循环体中引用集合中的每一项。COLLECTION属性用来指定要对之进行循环的COM/DCOM集合。

在下面例子中，ITEM属性指定了一个变量file2，于是，在每次执行循环体时，该变量代表集合中的当前项。在CFOUTPUT中，当前项的name属性被显示。

```
<CFOBJECT CLASS="FileFunctions.files"
```

```
NAME="FFunc"
```

```
ACTION="Create">
```

```
<CFSET FFunc.Path="c:\">
```

```
<CFSET FFunc.Mask="*.*)">
```

```
<CFSET FFunc.attributes=16>
```

```
<CFSET x=FFunc.GetFileList()>
```

```
<CFLOOP COLLECTION=#FFUNC# ITEM="file2">
```

```
<CFOUTPUT> #file2.name# <BR>
```

```
</CFOUTPUT>
```

```
</CFLOOP>
```

循环的嵌套

一个CFLOOP代码块中可以包含其它的CFLOOP标签。在这种情况下，每次执行外层循环时，内层循环都会被完整地执行一次。循环可以嵌套任意多层。

在下面的例子中，内层循环体总共会被执行15次：

```
<CFLOOP INDEX="OuterLoopCount" FROM="1" TO="3">
```

```
<CFOUTPUT>Outer loop #OuterLoopCount#</CFOUTPUT><BR>
```

```
<CFLOOP INDEX="InnerLoopCount" FROM="1" TO="5">
```

```
<CFOUTPUT>Inner loop #InnerLoopCount#</CFOUTPUT><BR>
```

```
</CFLOOP>
```

```
</CFLOOP>
```

当ColdFusion在循环体内遇到CFBREAK标签时，就终止循环体的执行，直接处理循环体后面的语句。对于复杂的标签结构以及深层的嵌套，这可以简化程序结构。

下面的例子并不太好，但它显示了CFBREAK标签的功能：

```
<CFLOOP INDEX="LoopCount" FROM=1 TO=100>
```

```
The value is <CFOUTPUT>#LoopCount#</CFOUTPUT>.<BR>
```

```
<CFIF LoopCount IS 7>
```

<CFBREAK>

</CFIF>

</CFLOOP>

在浏览器中显示的结果将是：

The value is 1.

The value is 2.

The value is 3.

The value is 4.

The value is 5.

The value is 6.

The value is 7.

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

理解应用程序框架

前文中已多次出现关于Application.cfm文件和“应用程序”的内容，本章将对此进行较为详细的介绍。

理解应用程序框架

一个ColdFusion应用程序是一组协同工作的文件的集合。从最简单的客户目录，到复杂的Internet商务系统，都可以是一个应用程序。而且你可以把单个应用程序结合起来，组成复杂而强大的Web系统。

ColdFusion利用一个特殊的文件Application.cfm来定义一个应用程序。当浏览器请求一个ColdFusion文件时，ColdFusion向上搜索该文件的目录树，查找Application.cfm文件。找到该文件时，将其包含到当前被申请文件的开头处。如果被申请文件的目录树中有多个Application.cfm文件，ColdFusion使用所找到的第一个文件。

注意：ColdFusion向上搜索目录树，一直搜索到磁盘的根目录。

按照上述的搜索规则，所有包含了同一个Application.cfm的文件属于同一个“应用程序”，此应用程序的名字即为Application.cfm文件中CFAPPLICATION标签的NAME属性所提供的名字。

由于应用程序的每个文件都在开头处包含了同一个Application.cfm文件，

因此该文件中的内容将对每个文件都执行一次，从而该文件的内容对整个应用程序都有效。这正是Application.cfm的特殊之处，除此之外，该文件只是个标准的CFM文件。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

实现应用程序框架

利用Application.cfm而实现的ColdFusion应用程序框架有四个基本组成部分：

·应用程序级设置

·客户状态管理

·错误处理

·Web服务器安全性集成

这些功能使你更容易组织应用程序文件，建立复杂的Web应用。

定义应用程序

开发ColdFusion应用程序的一个重要步骤是设计其目录结构。建立应用程序之前，首先要为该应用程序建立一个根目录。有些程序文件也许需要存在子目录中。

建立应用程序框架的步骤如下：

1. 在应用程序的根目录建立Application.cfm文件，用于整个应用程序的设置。
2. 设定应用程序级设置和错误处理
3. 用CFAPPLICATION标签允许客户状态管理。
4. 允许你所需要的其它特性，如安全（用CFAUTHENTICATE），锁定（用CFLOCK），或错误处理（用CFERROR）。

你可以只用一个Application.cfm，也可以对应用程序的不同部分使用不同的Application.cfm文件。下图所示的目录树显示了两种不同的应用程序框架。

·在左面的例子中，一个名为Web Wonders,Inc的公司，用应用程序根目录下的一个Application.cfm文件控制整个应用程序。

·在右面的例子中，Bandwidth Associates公司对不同的部门使用不同的Application.cfm文件。只有Products目录下的文件使用根目录下的Application.cfm，Marketing和Sales目录有自己的Application.cfm。



每个ColdFusion文件只能包含一个Application.cfm。这是一种隐式的CFINCLUDE，如果目录树中存在Application文件，你没有办法不包含它。（正因如此，该文件是设置应用程序级变量的理想位置。）

当被申请的文件中有一个CFINCLUDE标签，显式地包含了另一个文件时，ColdFusion不会为被包含的文件搜索Application.cfm。

应用程序级设置

在Application.cfm文件中可以进行应用程序级的设置，如：

- 应用程序名：设置CFAPPLICATION标签的NAME属性。
- 客户状态管理选项：设置CLIENTMANAGEMENT,CLIENTSTORAGE属性。
- Application变量和Session变量：设置SESSIONMANAGEMENT,SESSIONTIMEOUT,APPLICATIONTIMEOUT等属性。
- 缺省变量
- 执行错误处理的网页
- 数据源

·排斥性锁定

·应用程序级常量

这些设置在整个应用程序范围内有效。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

客户状态管理

前面已经讲过，对Web服务器来说，浏览器的每一次请求都是独立的，两次请求之间没有任何联系。而对一个应用程序来说，在不同的文件中识别同一个客户是至关重要的。这正是客户状态管理要实现的功能。允许了客户状态管理（CLIENTMANAGEMENT="Yes"）后，即可以用Client变量在不同的文件中识别特定的客户。

注意，要在需要客户状态管理的每个文件中都设置CLIENTMANAGEMENT = "Yes"，因此Application.cfm是进行这项设置的最好的地方。

错误处理

使用CFERROR标签，你可以在发生错误时显示定制的网页。该标签通常与CFTRY，CFCATCH，和CFTHROW一同使用。

ColdFusion有两种类型的错误：

1. 请求（REQUEST）——当被请求的文件中的代码发生错误时，产生请求错误。
2. 合法性（VALIDATION）——当客户提交Form时违反了Form字段的合法性检查时，产生合法性错误。

缺省情况下，ColdFusion返回一个显示错误信息的标准网页。但是你可以定制错误信息网页，使它们与你站点的风格相一致。定制的错误信息网页还使你能控制客户所见的信息。

CFERROR标签用来设置错误信息网页。你可以为每个网页分别设置，但由于定制的错误信息网页通常应用于整个站点，将CFERROR标签放在Application.cfm中效率更高。

错误信息网页是一个包含HTML标签和有关的错误参数的文件，注意，这个文件中不能包含CFML标签。有关的错误参数依赖于错误的类型。所有的错误参数都使用前缀“Error”（例如，Error.Diagnostics）。

下面的例子显示了一个请求错误（request error）的定制错误信息网页：

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Products - Error</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<CFOUTPUT>
```

```
<H2>Sorry</H2><P>An error occurred when you requested this page.
```

```
Please email the Webmaster to report this error. We
```

```
will work to correct the problem and apologize
```

```
for the inconvenience.</P>
```

<TABLE BORDER=1>

<TR><TD>Error Information

#Error.DateTime#

#Error.Template#

#Error.RemoteAddress#

#Error.HTTPRefer#

</TD></TR></TABLE>

</CFOUTPUT>

</BODY>

</HTML>

下面的例子显示了一个合法性错误（validation error）的定制错误信息网页：

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Products - Error</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<H2>Oops</H2>
```

```
<P>You failed to complete all the fields
```

```
in the form. The following problems occurred:</P>
```

```
#Error.InvalidFields#
```

</BODY>

</HTML>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

Web服务器安全性集成

你可以将Web服务器所提供的客户认证和安全性与你的应用程序相集成。如果你的应用程序文件放在Web服务器的文档目录中，你就可以使用Web服务器的本地认证和加密服务来实现应用程序的安全性。

每种Web服务器都用不同的方法配置安全选项，建立用户，建立用户组，以及设置特权。关于这些信息可以参阅你的Web服务器的文档。

你也可以使用Web服务器的认证系统。当Web服务器认证一个客户时，会唯一地返回一个变量，在你的应用程序中可以通过CGI变量“Auth_User”引用它。你可以用这个变量访问存于数据库中的用户附加信息。

另外，ColdFusion Server提供了一种安全框架，控制对应用程序，网页，数据源等的访问。你可以在Application.cfm文件中用CFAUTHENTICATE标签为应用程序建立一个安全域。有关的详细信息请参阅ColdFusion文档的Advanced ColdFusion Development部分。

ColdFusion提供了两个用于加密字符串的函数：Encrypt()和Decrypt()。关于这两个函数请参阅ColdFusion文档的CFML Language Reference部分。因为ColdFusion将网页返回给Web服务器，你也可以用Web服务器的加密技术对网页进行加密。对于要求高度安全的商务程序，这会特别有用。

北极星书库

缺省变量和常量

在Application.cfm文件中设置缺省变量和常量是有用的。例如，你可以指定：

- 你所使用的数据源
- 域名
- 风格设置，如字体，颜色。
- 其它重要的应用程序级变量

下面显示了一个完整的Application.cfm文件：

```
<!-- Set application name and enable client
```

```
variables option, with client variables stored in
```

a data source called mycompany --->

```
<CFAPPLICATION NAME="Products"
```

```
CLIENTMANAGEMENT="Yes"
```

```
CLIENTSTORAGE="mycompany">
```

```
<!-- Install custom error pages --->
```

```
<CFERROR TYPE="REQUEST"
```

```
TEMPLATE="RequestErr.cfm"
```

```
MAILTO="admin@company.com">
```

```
<CFERROR TYPE="VALIDATION"
```

```
TEMPLATE="ValidationErr.cfm">
```

```
<!-- Set application constants --->
```

<CFSET HomePage="http://www.mycompany.com">

<CFSET PrimaryDataSource="CompanyDB">

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

用CFLOCK进行排斥性锁定

CFLOCK标签用来对一段代码进行锁定。括在此标签内的代码在同一时间只能被一个请求来执行。执行CFLOCK代码块的请求对该段代码进行排斥性锁定，因此当一个请求正在执行CFLOCK代码块时，不允许其它的请求执行该段代码。ColdFusion按照“先来先服务”的原则进行排斥性锁定。

将修改共享数据的代码括在CFLOCK标签中，可以确保不会有两个用户同时修改数据。通常，更新应用程序范围的变量，如Server变量，Application变量,Session变量时，应该使用CFLOCK标签。如下例所示：

```
<HTML>
```

```
<HEAD>CFLOCK Example</HEAD>
```

```
<BODY>
```

```
<H3>CFLOCK Example</H1>
```

```
<CFLOCK NAME="ApplicationData" TIMEOUT=30>
```

```
<CFIF NOT IsDefined("Application.IsApplicationDataInitialized")>
```

```
<CFSET Application.IsApplicationDataInitialized=TRUE>
```

```
<CFSET Application.ImportantValue = 5>
```

```
</CFIF>
```

```
</CFLOCK>
```

```
<CFOUTPUT>
```

```
Important value is #Application.ImportantValue#
```

```
</CFOUTPUT>
```

```
</BODY>
```

```
</HTML>
```

北极星书库

Application变量，Session变量，和Client变量的使用

关于这三种变量在第二章已经有较为详细的讨论，在此仅作一些补充说明。

1. Application.cfm是个标准的CFM文件，可以在Application.cfm中使用动态参数，根据查询结果或客户信息等动态地决定应用程序级的设置。
2. 这三种变量是在整个应用程序范围内存在的变量，此处“应用程序范围”指的是由同一个Application.cfm文件所确定的范围，应用程序根目录下的所有文件不一定都处在同一个“应用程序范围”。
3. 应注意这三种变量与缺省变量的区别。缺省变量也可以在整個应用程序范围中访问，但它是定义在Application.cfm中，靠每个文件都包含了Application.cfm而实现应用程序范围的共享，实际上在每个文件中，缺省变量都是局部变量，在一个文件中改变缺省变量的值不会影响到其它文件（除非使用了动态参数）。而Application变量，Session变量，和Client变量既使不定义在Application.cfm中，也可以在整個应用程序范围共享，在一个文件中改变这三种变量的值将会影响到整个应用程序。
4. 当使用CFLOCATION重定向一个文件时，Client.URLToken会自动被追加在URL后面。你可以在CFLOCATION标签中设置ADDTOKEN = "NO"来禁止这种追加。
5. 如果Client变量存储在系统注册表中，而你需要把它转移到另一台计算机中，你可以输出（export）注册表中的响应键值，然后输入（import）到目标计算机的系统注册表中。

Application变量，Session变量，和Client变量的比较

下表显示了Application变量，Session变量，和Client变量之间的关系：

变量种类					
变量类型	应用程序名	客户IDs	Client Mgmt	Session Mgmt	过期时间
Client	可选	必需	必需	不需要	可选
Session	可选	必需	必需	必需	可选
Application	必需	不需要	不需要	不需要	可选

注意Session变量需要客户状态管理。

Application变量和Session变量的使用技巧

通常，Application变量和Session变量用来存储不常需要写但经常需要读的信息。在多数情况下，这些变量的值只在应用程序启动时（Application变量）或Session建立时（Session变量）被赋值。然后变量的值在应用程序或Session的生存期内被多次引用。

使用Application变量时，要记住这些变量被一个应用程序的多个实例共享。因此，Application变量的值有可能会被同时运行的其它实例所覆盖。如果用这些变量存储“一次写入，多次读出”的信息，则不会出什么问题，否则应该考虑同一应用程序的不同实例之间的相互影响。（注：当多个客户同时链接到一个应用程序时，即产生该应用程序的多个“实例”。）

下面的例子演示了Application变量的使用。注意这段代码位于Application.cfm文件中，因为要执行对每个应用程序文件都需要的操作时，Application.cfm是最好的地方。在这个例子中，一个变量“Application.Initialized”被用来作为标志，指明应用程序变量是否已经初始化。

```
<!-- Declare a name for this application. This
```

```
automatically turns on application scope --->
```

```
<CFAPPLICATION NAME="AccountCheck">
```

```
<!-- Test to see if application variables have already been defined --->
```

```
<CFIF NOT IsDefined("application.initialized")>
```

```
<CFSET application.query1= ??? >
```

```
<CFSET application.query2= ??? >
```

<CFSET application.initialized=1>

</CFIF>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

数据源基础

在ColdFusion中，“数据源”包括可以从本地或通过网络访问的许多种数据形式。你可以查询一个站点，LDAP服务器，POP邮件服务器，以及各种形式的文档。然而在大多数情况下，应用程序需要访问的是数据库，在本章中数据源被定义为“操作数据库的人口点”。

开放数据库连接（Open Database Connectivity——ODBC）

ODBC是应用程序连接到数据源的标准接口。系统中必须安装了ODBC驱动程序，并为每个数据源配置该驱动程序。通过Windows控制面板中的ODBC数据源管理器，可以查看系统中已安装的ODBC驱动程序。

从<http://www.microsoft.com/data/odbc/kill/download.htm>可以了解关于ODBC的更多内容，并可以下载ODBC3.0编程参考。

安装ODBC驱动程序

你可以从Windows开始菜单中的ColdFusion程序组中安装Microsoft Data Access Components(MDAC2.0)。由此可安装的ColdFusion ODBC驱动程序包括：

·Microsoft SQL Server

·Microsoft Access和FoxPro 数据库

·Borland dBase 兼容数据库

·Microsoft Excel Worksheet

·分隔的文本文件

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

添加ODBC数据源

要在ColdFusion应用程序中对一个已经存在的数据库进行操作，首先要为该数据库建立ColdFusion数据源。从ColdFusion Administrator中打开Data Source ODBC网页，在此处可建立数据源，步骤如下：

1. 为新数据源输入名字。
2. 从列表中选择相应的ODBC驱动程序。
3. 单击Add，打开Create ODBC Data Source 网页。
4. 输入新数据源的相关信息，单击CF Settings，显示可选设置。
5. 单击Create，新数据源即被添加到系统中并显示在数据源列表中。单击列表中的数据源名字可以编辑数据源的相关信息。
6. 要测试一个数据源，打开Verify Data Source 网页，从列表中选择该数据源即可。

建立数据源后，就可以在应用程序中使用它。

ODBC命名规范

不同的数据源其表和字段的名称有许多不同之处。作为数据源的标准，ODBC定义了一个命名规范。遵循该命名规范，可使你的代码更易于移植。

该规范要求名称以字母开头，只能包含字母，数字和下划线。ColdFusion使用与此相同的标准来标识查询字段。因此，你在ColdFusion和ODBC中使用的表和查询都要符合这一规范。

虽然ODBC允许表名中带有空格，ColdFusion却不允许。你必须用别名来访问包含空格的表和字段名。另外，有些数据源要求你指定表的所有者和限定词才能访问该数据源。

通过SQL的“AS”关键字可以用一个标准的名称作为非标准名称的别名，从而绕过ODBC/ColdFusion的命名要求。但是，这种做法很烦，并导致代码难于读懂，难以移植。

表的所有者和限定词

通常，在SQL语句中用ODBC数据源的名字即可以访问该数据源。而SQL Server是个例外，它要求扩展的信息。

下面的语句用于返回SQL Server的表Orders中所有记录。要实现这一点，必须在表名前面加上数据库名(orderdb)和数据库所有者（dbo）。

```
SELECT * FROM orderdb.dbo.order
```

其他的客户/服务器数据库如Oracle和Sybase也有自己特殊的要求。详细内容可参阅产品文档。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

使用本地数据库驱动程序

ColdFusion企业版可以安装用于从ColdFusion中直接访问数据库的驱动程序。由于它们不需要通过ODBC驱动程序管理器来执行SQL查询，因此操作大量的数据时效率更高。

ColdFusion目前提供的本地驱动程序包括：

- Oracle 7.3----需要Oracle 7.3.3 客户程序。
- Oracle 8.0----需要Oracle 8.0.4.0.0
- Sybase System 11----需要Sybase 11.11客户库。

允许本地驱动程序的属性

你可以使用下面的属性覆盖Administrator中的设置：

- DBSERVER——输入Sybase或Oracle服务器名。

·DBNAME (仅Sybase使用)——输入数据库名。

·BLOCKFACTOR(仅ODBC和Oracle使用)——设置每个查询所返回的最多记录数。缺省值是1，最大值是100。某些ODBC驱动程序会在运行时动态地减少这个参数的值。

CFQUERY, CFINSERT, CFUPDATE, CFGRIDUPDATE, 和 CFSTOREDPROC标签支持这些属性。

注意：对Sybase SQL Server, 用CFQUERY调用存储过程的语法对ODBC和本地Sybase驱动程序有所不同。ODBC需要括号和“Call”，而本地驱动程序不需要。

ODBC语法：

```
<CFQUERY DATASOURCE="" NAME="">
```

```
{call dbo.stored_procedure_name}
```

```
</CFQUERY>
```

本地驱动程序语法：

<CFQUERY DATASOURCE="" NAME="">

dbo.stored_procedure_name

</CFQUERY>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

使用OLE DB连接

OLE DB驱动程序包含在Win32平台的ColdFusion企业版中。通过安装第三方的OLE DB提供者，可以在ColdFusion中访问许多新的数据存储类型，包括：

- 基于MAPI的数据，如Microsoft Exchange和Lotus Mail

- 非关系型数据库，如Lotus Notes

- LDAP 2.0数据

- 来自OLE应用程序的数据

- 主框架数据

- HTML和文本数据

安装和配置了第三方软件后，你可以打开Administrator的Data Sources OLE DB页，添加该数据提供者所支持的数据源。与其他数据源一样，你可以输入数据源名字和描述，当需要时选择高级设置。你还需要输入：

Provider----ProgID

ProvederDSN----数据源名字

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

结构化查询语言 (SQL) 概述

SQL是操纵数据库的标准语言。其语法非常简单，但功能非常强大，可以进行复杂的数据操作。在ColdFusion中，将SQL语句包装在CFQUERY标签中来执行数据操作。

一个SQL语句是一个由从句组成的命令，从句指定了要执行的操作，数据源以及任何完成操作所需的结构。每个从句必须起始于一个关键字。下面是包装在CFQUERY标签中的SQL语句的一个简单例子：

```
<CFQUERY NAME="zip_02140"
```

```
DATASOURCE="Customer">
```

```
SELECT first_name, last_name, phone
```

```
FROM Customer
```

```
WHERE zip_code = 02140
```

```
ORDER BY last_name
```

</CFQUERY>

这个语句用关键字SELECT搜索数据库。第一行指定了要选择的数据库字段。第二行用一个从句指定了要对其进行操作的数据源，本例中即表的名字。第三个从句设定了返回记录的限定条件，只返回满足条件的记录。第四个从句按指定的字段对返回结果进行排序。

注意：有些DBMS使用非标准的SQL语法（称为方言）。ColdFusion不对CFQUERY中的SQL语句进行检查，因此你可以使用任何你的数据源所支持的SQL语法。关于具体的非标准SQL语法请参考你的DBMS文档。

最常用的SQL命令包括：

·SELECT——取出特定的记录

·INSERT——添加一条新记录

·UPDATE——改变特定记录的值

·DELETE——删除特定的记录

最常用的从句包括：

·FROM——指定要对其进行操作的数据源

·WHERE——对操作设定一个或多个条件

·ORDERBY——对查询结果进行排序

·GROUP BY——对查询结果进行分组

SQL扩展

ODBC对SQL进行了一些扩展，这些扩展为不同的数据库系统提供了统一的语法，从而使程序更加容易移植。最常用的ODBC SQL扩展包括日期/时间格式，转换函数，和存储过程。

日期/时间格式

ODBC定义了日期和时间的统一格式。如果你需要在SQL语句中使用日期/时间值，则必须使用ODBC的格式。日期，时间，和日期/时间值的格式如下表所示：

值	格式	例子
日期	{d'yyyy-mm-dd'}	{d'1995-06-20}

时间	{t'hh:mm:ss'}	{t'15:34:08'}
日期/时间	{ts'yyyy-mm-dd hh:mm:ss'}	{ts'1995-06-20 15:34:08'}

当你使用ColdFusion的Insert和Update操作时，日期字段会自动被检查并转换为ODBC日期格式，因此你不需要格式化用户输入。但是，当你在CFQUERY标签中使用SQL语句时，你应该保证日期和时间有正确的格式。实现的方法之一是对相应的字段进行合法性检查，这可以确保用户输入的日期值的合法，并自动转换为ODBC格式。

转换函数

许多数据库驱动程序允许你在SQL语句中调用函数。这些函数通常完成字符串操作，日期/时间操作，以及数值计算。由于不同的数据源中这些函数通常具有不同的名字和语法，ODBC定义了一套与数据源类型无关的转换函数。

调用ODBC函数的语法为：

```
{ fn scalar-function }
```

其中scalar-function代表函数名及其参数。下面的例子展示了两个常用的转换函数在SQL语句中的使用。第一个例子调用CURDATE获得当前的日期：

```
SELECT *
```

```
FROM Orders
```

```
WHERE ShipDate>={ fn CURDATE() }
```

第二个例子调用LEFT获得用户输入的邮编的最左边5个字符：

```
SELECT *
```

```
FROM Customers
```

```
WHERE PostalCode={ fn LEFT( '# Form.PostalCode#', 5) }
```

小结：

本章对ColdFusion数据源进行了简单介绍。对一般的开发而言，最常使用的是ODBC数据源。其要点是：在操作数据库之前，先要为数据库建立数据源，该数据源作为应用程序与数据库之间的接口。然后在应用程序中指定要对之进行操作的数据源，使用标准的SQL语句，即可完成数据库的查询，添加，修改等操作。

另外要注意ODBC的标准规范和ODBC的扩展。

其它类型的数据源属于较高级的内容，在此仅作一般了解即可。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

用CFQUERY标签选择数据

要从数据库中选择和输出数据，你需要在CFM文件中包含选择数据和输出结果的段落。第一步是定义一个数据库查询。ColdFusion使用CFQUERY标签定义查询。CFQUERY标签的完整语法为：

```
<CFQUERY NAME=" query_name"
```

```
DATASOURCE=" ds_name"
```

```
USERNAME=" username"
```

```
PASSWORD=" password"
```

```
MAXROWS=" number"
```

```
TIMEOUT=" milliseconds"
```

```
BLOCKFACTOR=" 1" (default)
```

DBPOOL=" database connection pool name"

DEBUG=" yes/no">

SQL statements

</CFQUERY>

这里有一个简单的例子，从一个与ODBC数据源CompanyDB相连的数据库Company中选择所有记录：

<CFQUERY NAME="EmployeeList" DATASOURCE="CompanyDB">

SELECT *

FROM Employees

</CFQUERY>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

使用动态查询参数

通过使用动态参数（变量）动态地确定SQL语句的内容，你可以充分利用CFQUERY标签的强大功能。动态参数包括Form自段，URL参数，和CGI环境变量。

在SQL语句中包含动态参数时要将其扩在井号（#）中，如# State #。当ColdFusion遇到扩在井号中的文本时，就搜索所有的Form变量,URL变量,cookies变量,Client变量和CGI变量，寻找与井号内相匹配的变量名。找到之后，即用变量的值代替变量名。

例子：动态SQL

如果你建立了一个表单，用户用它来根据名字搜索员工信息，你可以使用下面的动态SQL语句：

```
SELECT *
```

```
FROM Employees
```

```
WHERE LastName = 'Form.LastName#'
```

如果用户在LastName字段中输入”Rucker”，则对数据库执行的SQL语句为：

```
SELECT *
```

```
FROM Employees
```

```
WHERE LastName = 'Rucker'
```

使用单引号

在上面的例子中你已经看到，参数Form.LastName扩在单引号中。对字符串来说，这是必须的，对数字则不需要。那么当一个参数中本身包含一个或多个单引号时会不会产生错误呢？这时需要特殊的处理。ODBC允许在扩在单引号中的字符串里用两个连续的单引号代表一个单引号。但在ColdFusion中，你不需要作这种处理。在把参数包含在SQL语句之前，ColdFusion会自动将单引号替换为两个连续的单引号。

注意：在特殊情况下，你也许不需要将单引号自动替换，这时可以使用PreserveSingleQuotes函数。

执行存储过程

许多数据库系统允许为常用的查询建立存储过程。关于存储过程的建立请参考你的DBMS文档。

ColdFusion提供了两种执行存储过程的方法。

从CFQUERY中执行存储过程

你可以在CFQUERY中调用存储过程，如下例所示。这个查询调用了一个SQL Server存储过程，返回特定装卸日期的所有定单信息：

```
<CFQUERY NAME="GetOrdersForDate"  
  
DATASOURCE="Orders Database">  
  
{ call OrderDB.dbo.sp_getorders( #OrderDate# ) }  
  
</CFQUERY>
```

这种方法是可行的，但推荐使用的方法是用CFSTOREDPROC标签，以实

现更大的灵活性。

从CFSTOREDPROC标签中执行存储过程

CFSTOREDPROC标签通过标签属性设置SQL调用参数，简化了取出数据的过程，并提供了CFQUERY标签所不具备的功能。它支持ODBC数据源和本地数据源。

CFSTOREDPROC用来指定存储过程及其数据源及设置选项。该标签中嵌有另外两个标签：

- CFPROCPARAM——设置数据和存储过程类型，变量及其值，以及其他选项。

- CFPROCRESULT——指定输出结果，如果返回多个记录还可以限制输出结果。

在决定使用以上哪一种方法之前，要考虑以下问题：

- 这个查询需要指定输入/输出参数吗？

- 这个查询需要返回结果代码吗？

·需要从返回的多个记录中再作选择吗？

如果这三个问题的回答都是“[Yes](#)”，那么要使用CFSTOREDPROC标签。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

日期格式函数

CodFusion返回的日期格式可能不是你需要的显示格式。为此ColdFusion提供了一组函数，用于格式化日期值。下表中的函数可以对日期，时间，和数字进行格式转换：

日期，时间，和数字格式函数	
函数	描述
DateFormat(Date[,mask])	区分大小写。建立日期值的用户定制格式。
TimeFormat(Date[,mask])	区分大小写。建立时间值的用户定制格式。
NumberFormat(Number[,mask])	建立数字的用户定制格式

下表所列的函数可用于特殊的格式转换：

格式化函数	

函数	描述
HTMLCodeFormat	用于显示来自TEXTAREA的HTML代码。删除回车符，保留特殊字符（<,>,”,&等）。
HTMLEditFormat	与HTMLCodeFormat类似，不同之处在于它不在输出文本中加入<PRE>标签。
ParagraphFormat	用于显示输入TEXTAREA的数据。将CR/CF序列转换为空格，将成对CR/CF序列转换为HTML段落符（<P>）。
PreserveSingleQuotes	<p>用在SQL语句中，禁止ColdFusion自动处理动态参数中的单引号。例如，在SQL语句中使用动态参数，不对单引号进行转换：</p> <pre>SELECT * FROM Customers WHERE CustomerName IN (#PreserveSingleQuotes(CustNames)#)</pre>
StripCR	用于显示输入TEXTAREA字段的预先格式化（PRE）的数据。删除所有的回车符。
URLEncodeFormat	将字符串参数进行URL编码（用加号“+”代替空格，用相应的16进制

	<p>数代替非字母的字符)。这个函数使你可以通过URL传递任意的字符串（包括含有空格的）。ColdFusion会自动对URL参数进行解码。</p>
ValueList&QuotedValueList	<p>这两个输出函数有助于使用查询结果来进行后续的查询。</p> <p>函数ValueList以一个查询字段为参数，如Customers.CustomerID，返回查询结果中每条记录的该字段的值，用逗号隔开。</p> <p>例如，如果你执行了一个查询，返回四个记录，则函数ValueList的输出结果类似于：22，43，50，96。函数QuotedValueList的结果则为：'21','43','50','96'。</p>
YesNoFormat	<p>将逻辑型数据显示为Yes或No的形式。非零值显示为Yes，零值显示为No。</p>

显示查询结果

一旦你在应用程序文件中建立了一个查询，你就可以在其他CFM文件中引用该查询的结果。查询结果可以用来动态地产生HTML网页。

当你学习使用CFML标签时，要记住你也可以在程序中使用HTML标签和文本。在你使用HTML标签和文本的地方，ColdFusion简单地将标签和文本直接传递给客户浏览器。输出查询结果的最灵活的方式是使用CFOUTPUT标签。CFOUTPUT标签可以对某个特定的查询进行输出，也可以输出多个查询的内容。一个CFOUTPUT标签中可以包含：

·纯文本

·HTML标签

·引用查询字段

·引用动态参数，如Form字段

·函数

基本的输出代码有如下的语法：

```
<CFOUTPUT QUERY=" queryname" MAXROWS= n >
```

Literal text, HTML tags, and

dynamic field references (e.g., #FullName#)

```
</CFOUTPUT>
```

CFOUTPUT中的数据库字段名

有些数据库，如Microsoft Access，允许字段名中带有空格，例如”Region Name。”。ColdFusion在CFOUTPUT中不支持对这种字段的引用。要和ColdFusion兼容，应避免在表名和字段名中使用空格，这使应用程序对不同的数据库系统具有最大的兼容性。你可以用下划线作分隔符，如”Region_Name”。

字段名必须以字母开头，且只能包含字母和数字。你 可以用SQL关键字AS为字段名起一个别名。例如，语句：

```
SELECT 401K AS FK FROM Employee_Withholding
```

在CFQUERY中可以得到正确处理。

例子：CFOUTPUT

在下面的例子中，我们用CFOUTPUT标签显示查询EmployeeList的查询结果。选择显示的内容是每个员工的姓，名，和电话号码。在变量名中指定查询名是个很好的编程习惯。

下面给出程序的完整代码：

```
<!-- Query to select customers --->
```

```
<CFQUERY NAME="EmployeeList" DATASOURCE="CompanyDB">
```

```
SELECT *
```

```
FROM Employees
```

```
</CFQUERY>
```

```
<HTML>
```

```
<HEAD>
```

<TITLE>Employee List</TITLE>

</HEAD>

<BODY>

<H2>Employee List</H2>

<!-- Output section -->

<CFOUTPUT QUERY="EmployeeList">

<HR>

#EmployeeList.FirstName# #EmployeeList.LastName#

(Phone: #EmployeeList.PhoneNumber#)

</CFOUTPUT>

</BODY>

</HTML>

你可以通过URL来申请这个网页，如：

<http://myserver/cfdocs/employeeelist.cfm>

或者通过超链接引用：

Employee List

这个例子的输出结果的HTML代码类似于：

<HR>

Deborah Jones (Phone: 612-227-1019)

<HR>

John Smith (Phone: 507-452-7224)

<HR>

Frank Wilson (Phone: 612-831-9555)

嵌套CFOUTPUT和分组

你可以对CFOUTPUT进行嵌套，以实现查询结果的分组显示。分组的实现是通过指定CFOUTPUT标签的GROUP属性，然后将另一个CFOUTPUT嵌入前一个CFOUTPUT。内层CFOUTPUT上面和下面的内容用于给每一组显示一个标题和脚注信息。

例子：分组

下面的代码使用CFOUTPUT的GROUP属性，根据”CourseLevel”的值将查询结果分组显示。ORDER BY关键字根据CourseLevel字段的值对查询结果进行排序。CFOUTPUT嵌套没有层数的限制，如果你想使用多级分组，你需要在SQL查询中设置多级排序（例如，”ORDER BY Region,State”）。

```
<CFQUERY NAME="Courses" DATASOURCE="CourseDB">
```

```
SELECT *
```

```
FROM CourseList
```

WHERE Department_ID = ?Form.Department#

ORDER BY CourseLevel

</CFQUERY>

<CFOUTPUT QUERY="Courses" GROUP="CourseLevel">

<H4>#CourseLevel#</H4>

<CFOUTPUT>

 #CourseNumber# - #CourseName#

</CFOUTPUT>

</CFOUTPUT>

这个例子在浏览器中显示的结果为：

Basic

100 - Physiology

Intermediate

510 - Neurobiology

500 - Plant Biology

Advanced

820 - Neurobiology

800 – Microbiology

CFQUERY 属性

执行每个CFQUERY时，有三个属性可以提供记录数信息。

记录数	
记录数	描述
RecordCount	查询所返回的记录总数。
CurrentRow	CFOUTPUT正在处理的当前记录。
ColumnList	返回用逗号隔开的查询字段列表。

例子：查询字段

这个例子返回查询CustomerList的记录数和查询字段的列表：

<CFOUTPUT>

The search returned information

```
on #CustomerList.RecordCount# customers.<BR>
```

```
Columns queried were #CustomerList.ColumnList#.
```

```
</CFOUTPUT>
```

要显示每个记录的记录序号，可以用如下语法：

```
<CFOUTPUT QUERY="CustomerList">
```

```
#CurrentRow# - #FirstName# #LastName# <BR>
```

```
</CFOUTPUT>
```

返回记录集的一部分

对于较大的记录集，你也许希望只显示该记录集的一部分。用CFOUTPUT标签的STARTROW和MAXROWS属性可以做到这一点。STARTROW指定记录集中第一条要返回的记录。MAXROWS指定了要返回的记录总数。

例子：记录集的一部分

这个例子显示查询EmployeeList的结果记录集中第10到20条记录：

```
<CFOUTPUT QUERY="EmployeeList"
```

```
STARTROW="10" MAXROWS="20">
```

```
#FirstName# #LastName# #Phone# <BR>
```

```
</CFOUTPUT>
```

在CFOUTPUT中使用参数

CFOUTPUT不仅仅用于显示查询到的数据。你也可以用CFOUTPUT显示Form变量，URL变量，Cookie，Client变量，Server变量，Session变量，Application变量，和CGI环境变量。引用这些变量时要注意使用正确的前缀，这样ColdFusion才能将这些变量与查询结果的字段名区分开。与查询字段一样，变量必须扩在井号中。对查询结果的每条记录，参数值都被显示一次。注意，若要显示井号本身，应用两个连续的井号表示一个井号。

在表格中显示查询结果

当查询返回的记录数较少时，用简单的HTML文本显示就够了。但当记录很多时，你需要更有条理的显示。CFOUTPUT标签可以包含HTML代码，因此你可以使用标准的HTML表格标签，动态地生成表格。

CFML 表格

CFML提供了两个表格标签，CFTABLE和CFCOL，用来以表格的形式显示查询结果。这种格式使用HTML的<PRE>标签精确地控制显示内容的宽度和对齐方式。如下例所示：

```
<CFTABLE QUERY="Messages" MAXROWS=10>
```

```
<CFCOL HEADER="Subject" WIDTH=25
```

```
TEXT="<I>#Subject#</I>">
```

```
<CFCOL HEADER="User Name" WIDTH=15
```

```
TEXT="#UserName#">
```

```
<CFCOL HEADER="Date" WIDTH=15
```

```
ALIGN=RIGHT
```

```
TEXT="#DateFormat(DateSubmitted)#">
```

```
</CFTABLE>
```

这段程序建立了一个三列的表格，通过CFCOL标签的HEADER属性为三个列分别指定了标题“Subject”，“User Name”，和“Date”。该表格的数据来自名为Messages的查询，最多显示10条记录。显示的内容由TEXT属性所给的HTML标签和动态参数决定。

CFCOL中的双引号和井号

ColdFusion在CFCOL的TEXT属性中用双引号和井号作为分隔符。因此当用到这两个符号本身时，要用两个连续的双引号来代替双引号，用两个连续的井号来代替井号。

例如，要显示一个文档列表，为每个文档提供一个超链接，你必须在TEXT属性中使用超链接标签（该标签的HREF属性要用双引号作为分界符）。因此必须使用如下语法：

```
TEXT="<A HREF=""/documents/#DocFile#">#DocName#</A>"
```

HTML表格

虽然CFTABLE标签产生的预定义格式表格是有效的，但HTML表格具有更大的灵活性。为了使这种转换更容易，CFTABLE提供了另一个属性HTMLTABLE。这个属性将CFTABLE提供为一个HTML表格，而不是预定义格式的表格。使用HTMLTABLE属性会使CFTABLE和CFCOL的某些属性具有另外的含义：

- CFTABLE的HEADLINES属性和COLSPACING属性将被忽略。

- CFCOL的WIDTH属性被解释为占可用屏幕宽度的百分比，而不是字符数。

注意：使用HTML表格不是一定要通过CFTABLE的HTMLTABLE属性，你可以用CFOUTPUT直接建立HTML表格。

记录细节的动态显示

显示查询结果的最常用的方法是用表格，如CFTABLE，或其他更简洁的列表。用这种列表可以快速浏览记录集，但显示的信息量有限。解决这一问题的方法是以简洁的方式显示记录，然后为每个记录创建一个超链接，链接到另一个提供详细信息的网页。

建立到记录细节的链接

建立到记录细节的链接包括如下步骤，假设已经有一个文件显示了记录摘要列表。

1. 从记录摘要中选择一个关键词作为可点击的链接，如公司名。
2. 将所选的关键词扩在超链接标签(<A>)中，链接到一个显示细节信息的文件。在URL中应传递一个唯一的记录标识，这样被链接的文件才知道要显示的是哪一条记录。
3. 建立一个用于显示记录细节的CFM文件。

例子：记录细节信息

假设你要显示一个公司列表，让用户点击一个公司名来获得关于该公司的详细信息。再假设在数据库中每个公司用一个唯一的Company_ID来标识，而文件compinfo.cfm可以显示给定Company_ID的公司的详细信息。在这种情况下，使用如下的语法：

```
<CFOUTPUT Query="CompanySearch">
```

```
<A HREF="compinfo.cfm?Company_ID=#Company_ID#">
```

```
#CompName#</A><BR>
```

```
#ContactPerson# (#ContactPhone#) <BR>
```

</CFOUTPUT>

公司名显示为一个超链接，它将Company_ID作为一个URL参数，激活文件compinfo.cfm。由该文件显示相应公司的详细信息。使用这种技术，实现记录细节的动态显示的通用形式如下（注意HREF属性应处在单独的一行中）：

```
<A HREF="TemplateFile?Record_ID=#Record_ID_Data#"># RecordName#  
</A>
```

这里TemplateFile是实现记录细节显示的文件。该文件用Record_ID_Data（通过URL参数Record_ID传递）确定要显示哪一条记录。用户点击RecordName来激活该文件。

在CFTABLE中嵌入细节链接

你也可以在CFCOL标签的TEXT属性中嵌入超链接。由于TEXT属性用双引号作为分界符，超链接的HREF属性也用双引号作为分界符，要注意引号的正确使用。下面是这种技巧的通用形式：

```
<A HREF="" TemplateFile? Record_ID=# Record_ID_Data#">#RecordName#  
</A>
```

注意在HREF属性中用两个连续的双引号表示一个双引号。如果上一个例子改为使用TEXT属性在表格中嵌入超链接，其语法应为：

```
TEXT="<A                                HREF=""compinfo.cfm?
Company_ID=#Company_ID#">#CompName#</A>"
```

再次强调，整个TEXT属性要写在同一行中。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

建立HTML查询表单

创建动态参数的最常用的方法是使用作为用户界面的表单（Form）。你可以在表单中包含文本框，选择框，单选按钮，多选按钮等，供用户用来输入信息。表单很容易建立，并可提供许多灵活的数据输入选项。表单提交时，Form变量会自动被创建，你可以利用这些变量执行对数据源的查询。

除了HTML表单，你还可以用ColdFusion的CFFORM标签建立表单。关于这方面的详细内容后文有述。

例子：查询表单

下面的例子是一个简单的查询表单：

```
<FORM ACTION="employeeSearch.cfm" METHOD="Post">
```

```
<PRE>
```

```
Last Name: <INPUT TYPE="text"
```

```
NAME="LastName">
```

Department:

```
<SELECT NAME="Department">
```

```
<OPTION>Accounting
```

```
<OPTION>Administration
```

```
<OPTION>Engineering
```

```
<OPTION>Sales
```

```
</SELECT>
```

```
<INPUT TYPE="Submit" VALUE="Search">
```

```
</PRE>
```

```
</FORM>
```

这个表单有两项输入：LastName和Department.用户可以在文本框中输入名字，从选择列表中选择一个部门。每个OPTION标签提供给用户一个选项。当提交按钮被点击时，ACTION属性所指定的文件会被打开，Form变量作为动态参数被传递给该文件。

假设用户输入了名字“Peterson”并选择了“Sales”。当她点击提交按钮时，下面的Form变量会被传递给文件employeeearch.cfm：

```
LastName=Peterson
```

```
Department=Sales
```

设置表单的ACTION属性和METHOD属性

你必须设置表单的ACTION属性，告诉浏览器当用户点击提交按钮时调用哪一个文件。你还必须将表单的METHOD属性设置为Post。在下面的例子中，当用户点击提交按钮时，调用employeeearch.cfm文件。

```
<FORM ACTION="employeeearch.cfm" METHOD="Post">
```

实现数据查询字段

为表单建立查询字段非常简单。你只需要为你想搜索的数据库字段建立表单查询字段。为了使文件有更好的可读性，可以让表单字段与数据库字段

有相同的名字。

例如，如果你有一个表Employees，表中有三个字段
FirstName，LastName，和Department，你的表单字段可以是：

Name: <INPUT TYPE="text" NAME="FirstName">

LastName: <INPUT TYPE="text" NAME="LastName">

Department: <INPUT TYPE="text" NAME="Department">

URL参数REQUESTTIMEOUT

当请求一个ColdFusion文件时，你可以在URL中传递REQUESTTIMEOUT参数，指定数据源连接超时的秒数，以防止一个数据源连接花费太多的时间。这个参数将覆盖ColdFusion Administrator中的缺省值。其使用的语法为：

<http://myserver.com/cfpages?REQUESTTIMEOUT=100>

例子：动态SQL

下面的文件searchform.cfm建立了一个查询表单：

<HTML>

<HEAD>

<TITLE>Employee Search</TITLE>

</HEAD>

<BODY>

<FORM ACTION="employeeesearch.cfm" METHOD="Post">

<PRE>

Last Name: <INPUT TYPE="text" NAME="LastName">

Department: <SELECT NAME="Department">

<OPTION>Accounting

```
<OPTION>Administration
```

```
<OPTION>Engineering
```

```
<OPTION>Sales
```

```
</SELECT>
```

```
<INPUT TYPE="Submit" VALUE="Search">
```

```
</PRE>
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```

当用户点击提交按钮时，下面的文件EmployeeSearch.cfm用来选择和显示查询结果：

<!-- CFML application page to implement employee search --->

<!-- Query the database --->

<CFQUERY NAME="EmployeeList"

DATASOURCE="CompanyDB">

SELECT *

FROM Employees

WHERE LastName = ?LastName#

AND Department = ?Department#

</CFQUERY>

<!-- Page header --->

<HTML>

<HEAD>

<TITLE>Employee Search Results</TITLE>

</HEAD>

<BODY>

<H2>Organization Search Results</H2>

<!-- Summarize search criteria for user --->

<CFOUTPUT>

<P>The search for #Form.LastName# in

the #Form.Department#

returned these results:</P>

</CFOUTPUT>

<--- Display results --->

<CFOUTPUT QUERY="EmployeeList">

<HR>

#FirstName# #LastName# (Phone: #PhoneNumber#)

</CFOUTPUT>

<!--- Page footer --->

<P>

Thank you for searching the employee database!</P>

<HR>

</BODY>

</HTML>

模式匹配查询

当你允许用户输入文本作为查询条件的一部分时，你通常不想查询与输入完全匹配的值，你通常需要查询与输入文本类似的值。在这种情况下，可以使用模式匹配查询。要实现模式匹配查询，可以将一个或多个通配符（在ODBC中用“%”表示）与LIKE运算符结合使用。

例如，让用户填充一个名为“LastName”的Form字段，查询一员工信息，可以用如下语法：

```
SELECT *
```

```
FROM Employees
```

```
WHERE LastName LIKE '#LastName#%'
```

这个查询返回每个其名字以用户输入值开头的员工的信息。例如，若输入“Jon”，返回的记录可能有“Jones”和“Jonson”。

北极星书库

返回MIME内容

MIME (Multipurpose Internet Mail Extensions) 的目标之一是在数据文件和用于显示和编辑该文件类型的应用程序之间建立联系。MIME通过为每种数据文件提供一个内容类型来实现这一点。

在Web上，数据的交换要通过超文本传输协议 (HTTP)。Web服务器和Web客户必须对文件的内容类型达成一致，并且客户端软件必须能够理解该种文件，或者能找到另外的能使用该文件的应用程序。

缺省情况下，ColdFusion使用的内容类型是text/html。尽管text/html是Web上最常使用的内容类型，但在某些情况下，你需要用ColdFusion动态地产生其他内容类型的文档。

例子：返回VRML

根据一个URL参数动态地返回一个VRML(Virtual Reality Modeling Language)原型，使用如下语法：

```
<CFCONTENT TYPE="x-world/x-vrml">
```

```
<CFQUERY NAME="GetCyberCafeRoom"
```

```
DATASOURCE="CyberCafe">
```

```
SELECT VRML_Script
```

```
FROM CyberCafeRooms
```

```
WHERE FloorNumber=#URL.FloorNumber#
```

```
</CFQUERY>
```

```
<CFOUTPUT QUERY="GetCyberCafeRoom">
```

```
#VRML_Script#
```

```
</CFOUTPUT>
```

向客户端传送非HTML的内容时，一个重要的考虑是浏览器是否能理解该内容类型。所有的主流浏览器都预先定义了一系列内容类型，这些类型或者被浏览器本身所支持，或者被一个辅助的应用程序（通常称为查看器）所支持。因此，在使用浏览器有可能不支持的内容类型之前，先对用户作出提示是很重要的。

例子：生成Excel表格

下面的例子用ColdFusion动态生成一个Excel表格，该表格可以在Microsoft Excel中直接打开。要无缝地实现这一点，你必须将本地的Microsoft Excel与text/tab delimited内容类型相关联。

```
<CFCONTENT TYPE="text/tab delimited">
```

```
<CFQUERY NAME="GetFinancialData"
```

```
DATASOURCE="MutualFunds">
```

```
SELECT *
```

```
FROM Funds
```

```
WHERE Fund_ID=#Fund_ID#
```

```
</CFQUERY>
```

```
<P>Month Value Percentage Gain Estimate</P>
```

```
<CFOUTPUT QUERY="GetFinancialData">
```

```
#Month# #Value# #PercentageGain# #Estimate#
```

```
</CFOUTPUT>
```

字段之间和标题之间的空格实际上是制表符（Tab）。Excel需要制表符才能正确地处理表中的字段。

注意，ColdFusion会把程序文件中出现在CFCONTENT标签之前的输出文本删掉。

例子：返回一个文件

下面的例子演示了如何用CFCONTENT标签从服务器返回一个文件。这种方法的好处在于你可以控制对文件的访问。FILE属性允许你返回不存在在Web服务器文档根目录下的文件。

```
<CFCONTENT TYPE="text/tabdelimited"
```

```
FILE="c:\land\topo.txt">
```

注意，在这种用法中，如果ColdFusion在CFCONTENT标签之后发现其他

输出内容，将会产生错误。

小结：

本章主要介绍了如何进行数据查询和结果显示，实现数据库查询和显示的最常用的流程如下：

1. 建立数据源。
2. 通过表单字段接收用户输入。
3. 在ACTION所指的文件中，使用CFQUERY标签，用DATASOURCE属性指定数据源，根据Form变量的内容（即用户的输入信息），用标准SQL语句对数据源进行查询。然后使用CFOUTPUT，用QUERY属性指定查询名，对查询结果中的每条记录进行显示。

在CFOUTPUT中可以使用各种HTML标签。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

插入数据

向数据库中插入数据通常由两个文件完成：

- 一个数据插入表单
- 一个插入执行文件

你可以用CFFORM标签或标准HTML Form标签建立数据插入表单。当表单被提交时，Form变量被转递给一个文件，该文件执行对数据源的插入操作（以及其他需要的操作）。该文件中应该包含显示给用户的响应信息（如“记录已插入”等）。

建立HTML数据插入表单

与所有的表单一样，你必须指定ACTION属性和METHOD属性。METHOD属性通常为“POST”，ACTION属性指定表单提交时调用的CFM文件。

你需要为数据库中每个要插入数据的字段建立Form输入字段。Form字段的名称必须与数据库字段的名称相同。例如，如果有一个名为Employees的表，表中有三个字段FirstName，LastName，和Phone，则你建立的Form字段应类似于：

First Name: <INPUT TYPE="text" NAME="FirstName">

Last Name: <INPUT TYPE="text" NAME="LastName">

Phone: <INPUT TYPE="text" NAME="Phone">

隐含Form字段

隐含字段是一种特殊的Form字段，它是表单的一部分，但不显示给用户。当用户提交表单时，隐含字段的值（建立字段时指定）与用户输入的其他字段值一同返回。例如，如果你希望用户提交的表单中始终包含提交的位置（哪个站点）信息，你可以使用下面的隐含字段：

```
<INPUT TYPE="Hidden" NAME="SiteName" VALUE="CompanyName">
```

使用这个隐含字段，每次用户提交表单时，将返回一个名为“SiteName”的变量，值为“CompanyName”。

例子：HTML数据插入表单

下面的例子是个简单的HTML表单：

```
<FORM ACTION="insdata.cfm" METHOD="Post">
```

```
<!-- Data entry fields -->
```

```
<PRE>
```

```
First Name: <INPUT TYPE="text" NAME="FirstName">
```

```
Last Name: <INPUT TYPE="text" NAME="LastName">
```

```
Phone: <INPUT TYPE="text" NAME="Phone">
```

```
<INPUT TYPE="Submit" VALUE="Enter Information">
```

```
</PRE>
```

```
</FORM>
```

这个表单有三个输入：FirstName,LastName和Phone。用户可以在文本框中输入数据，然后点击提交按钮。Form提交后，ACTION属性激活的文件可以访问所有Form字段的值（包括隐含字段），该文件可能会显示这些变量，或将它们插入到数据库中。假设用户在三个字段中分别输入了"William","Gibson","(212)323-9734"，则表单提交时，下面的变量被传

递：

FirstName=William

LastName=Gibson

Phone=(212)323-9734

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

用CFINSERT建立一个插入执行文件

执行简单插入操作的最容易的方法是用CFINSERT标签。在多数情况下，不需要使用可选的属性。TABLEOWNER属性和TABLEQUALIFIER属性很少用到，提供这两个属性是为了与某些需要指定表所有者和（或）表限定词的ODBC驱动程序相兼容。需要指定OWNER和QUALIFIER的ODBC驱动程序包括SQL Server和Oracle驱动程序，以及Intersolv Q&E驱动程序。

CFINSERT数据源

假设ODBC数据源名为“Employees DB”，你要插入数据的表名为“Employees”。那么CFINSERT标签的使用方法为：

```
<CFINSERT DATASOURCE="Employee DB" TABLENAME="Employees">
```

例子：插入数据

下面的文件是一个用于插入数据的表单：

```
<!-- HTML form to input data -->
```

<HTML>

<HEAD>

<TITLE>Input Form</TITLE>

</HEAD>

<BODY>

<FORM ACTION="EmployeeInsert.cfm"

METHOD="Post">

<PRE>

First Name: <INPUT TYPE="text"

NAME="FirstName">

Last Name: <INPUT TYPE="text"

```
NAME="LastName">
```

```
Phone: <INPUT TYPE="text"
```

```
NAME="Phone">
```

```
<INPUT TYPE="Submit"
```

```
VALUE="Insert Information">
```

```
</PRE>
```

```
</FORM>
```

```
</BODY>
```

```
</HTML>
```

其ACTION属性所指向的Employeeinsert.cfm文件如下，该文件中用CFINSERT完成简单的数据插入：

<!-- Inserts the data from the the HTML Form -->

<CFINSERT DATASOURCE="Employee DB"

TABLENAME="Employees">

<HTML>

<HEAD>

<TITLE>Input Form</TITLE>

</HEAD>

<BODY>

<CENTER><H2>Thank You!</H2></CENTER>

<HR>

<P>Thank you for entering your data into

our database - please visit our site often!</P>

<HR>

</BODY>

</HTML>

注意：CFINSERT执行的是简单的插入操作，只能将Form变量的值（即用户输入到表单字段中的数据）插入到指定数据源的指定表中。它要求Form字段名与表的字段名相同，因为它是根据名字将Form变量与表中的字段相对应起来。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

用CFQUERY建立插入执行文件

要执行复杂的插入操作，可以在CFQUERY标签中使用SQL INSERT语句。SQL INSERT语句更加灵活，你可以有选择地插入数据，还可以在INSERT语句中使用函数。

SQL INSERT语句的基本语法为：

```
INSERT INTO tablename (columnnames)
```

```
VALUES ( values)
```

VALUES关键字指定新记录的各字段的值。你必须按照与columnnames相同的顺序键入这些值。在前一个例子中，如果使用CFQUERY标签插入表单数据，应使用如下语法：

```
<CFQUERY NAME="AddEmployee"
```

```
DATASOURCE="Employee DB">
```

```
INSERT INTO Employees (FirstName, LastName, Phone)
```

VALUES ('#Form.FirstName#'#Form.LastName#'

'#Form.Phone#')

</CFQUERY>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

更新数据

更新数据库中的数据通常也需要两个文件：

- 一个数据更新表单
- 一个更新执行文件

数据更新表单接收用户的输入，表单被提交时激活更新执行文件，该文件中包含CFUPDATE标签或CFQUERY标签，用于执行对数据库的更新操作。该文件中还应该包含响应给用户的信息（如“数据已更新”等）。

建立数据更新表单

数据更新表单与数据插入表单类似，主要的不同有两点。一个数据更新表单包含对要更新的记录的主关键字的引用。主关键字是一个数据库表格中的一个或多个字段，主关键字可以唯一地确定表中的每一条记录。例如，在一个员工姓名和地址表中，只有Employee_ID对每个记录具有唯一值。另外，由于数据更新表单用于更新已经存在的数据，因此表单的内容通常取自数据库中。你必须首先用CFQUERY标签从数据库中选择记录，然后在CFOUTPUT中建立表单，用选择出的记录的值填充表单字段。

设计主关键字

在数据更新表单中指定主关键字的最容易的方法是使用一个隐含字段，其值为要更新的记录的主关键字的值。如下例所示：

```
<!-- Query to select record --->
```

```
<CFQUERY NAME="EmployeeRecord"
```

```
DATASOURCE="Employee DB">
```

```
SELECT *
```

FROM Employees

WHERE Employee_ID = #URL.EmployeeID#

</CFQUERY>

<HTML>

<HEAD>

<TITLE>Input Form</TITLE>

</HEAD>

<BODY>

<CFOUTPUT QUERY="EmployeeRecord">

<!-- Input form -->

<FORM ACTION="EmployeeUpdate.cfm" METHOD="Post">

<!-- Primary Key value indicating record to update --->

<INPUT TYPE="Hidden" NAME="Employee_ID"

VALUE="#Employee_ID#">

<PRE>

FirstName: <INPUT TYPE="Text" NAME="FirstName"

VALUE="#FirstName#">

LastName: <INPUT TYPE="Text" NAME="LastName"

VALUE="#LastName#">

Phone: <INPUT TYPE="Text" NAME="Phone"

VALUE="#Phone#">


```
<INPUT TYPE="Submit" VALUE="Update Information">
```

```
</PRE>
```

```
</FORM>
```

```
</CFOUTPUT>
```

```
</BODY>
```

```
</HTML>
```

在这个例子中，Employee_ID是表Employees的主关键字，于是在HTML表单中使用了一个隐含字段Employee_ID（注意Form字段与数据库字段名字相同）。该隐含字段指示ColdFusion要更新的是哪一条记录。本例中主关键字来自URL参数。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

用CFUPDATE建立数据更新执行文件

根据前台的数据更新表单进行简单数据更新操作的最容易的方法是使用CFUPDATE标签。CFUPDATE标签的使用方法与CFINSERT完全一样。

要使用CFUPDATE标签，你必须在表单中包含组成主关键字的所有字段。CFUPDATE会自动检测待更新的表中的主关键字字段并将它们从被提交的Form变量中搜索出来。ColdFusion根据主关键字选择要更新的记录，然后用其余的Form字段更新该记录。

在下面的例子中，ODBC数据源名为"Employee DB"，要更新的表为"Employees"。利用前台的表单，CFUPDATE标签的使用如下：

```
<!-- This is the page EmployeeUpdate.cfm -->
```

```
<CFUPDATE DATASOURCE="Employee DB"
```

```
TABLENAME="Employees">
```

```
<HTML>
```

<HEAD>

<TITLE>Reply</TITLE>

</HEAD>

<BODY>

<H2>Thank You!</H2>

<HR>

<P>Thank you for updating your data in

our database - please visit our site often!</P>

<HR>

</BODY>

</HTML>

后页
前页
目录
北极星书库

用CFQUERY建立数据更新执行文件

要执行复杂的数据更新，你可以在CFQUERY标签中使用SQL UPDATE语句。SQL UPDATE语句比CFUPDATE标签具有更大的灵活性。其基本语法为：

UPDATE tablename

SET columnname = value

WHERE condition

在SET从句中必须指定一个或多个字段，然后指定一个常量或表达式作为该字段的值。

对于上一个例子，如果使用CFQUERY代替CFUPDATE，其语法应为：

```
<CFQUERY NAME="UpdateEmployee"
```

```
DATASOURCE="Employee DB">
```

```
UPDATE Employees
```

```
SET Firstname='#Form.Firstname#'
```

```
LastName='#Form.LastName#'
```

```
Phone='#Form.Phone#'
```

```
WHERE Employee_ID=#Employee_ID#
```

```
</CFQUERY>
```

注意，WHERE从句是可选的。但如果你在UPDATE语句中不使用WHERE从句，则数据库表中的所有记录都会被更新。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

删除数据

删除数据库中的数据可以由一个单独的文件完成，该文件包含一个执行SQL DELETE语句的CFQUERY标签。SQL DELETE语句的语法为：

```
DELETE FROM tablename
```

```
WHERE condition
```

WHERE从句控制删除语句删除一条记录，几条记录，或所有记录。

下面的例子从表Employees中删除一条记录：

```
DELETE FROM Employees
```

```
WHERE Employee_ID = 1
```

下面的例子从表Employees中删除几条记录。这个例子假设在销售部（Sales department）有不止一个员工：

```
DELETE FROM Employees
```

```
WHERE Department = 'Sales'
```

下面的例子从表Employees中删除所有记录：

```
DELETE FROM Employees
```

注意，对数据库的删除操作是不可恢复的，因此要小心使用delete语句。

例子：执行删除操作的完整文件

```
<!-- Page to delete single employee record -->
```

```
<CFQUERY NAME="DeleteEmployee"
```

```
DATASOURCE="Employee DB">
```

```
DELETE FROM Employees
```

```
WHERE Employee_ID = #URL.EmployeeID#
```

</CFQUERY>

<HTML>

<HEAD>

<TITLE>Delete Employee Record</TITLE>

</HEAD>

<BODY>

<H3>The employee record has been deleted.</H3>

</BODY>

</HTML>

数据输入检查

当你使用表单接收用户输入以执行数据库查询时，你通常希望在执行查询之前能对用户的输入进行检查。尤其是SQL语句需要特定的数据类型时，这种检查更加必要。

你可以对提交给ColdFusion Server的任何表单数据进行检查。ColdFusion提供了几种不同的数据输入检查类型。

数据输入检查类型	
检查类型	描述
客户端	在CFFORM中，你可以通过ONVALIDATE属性指定一个JavaScript函数，用该函数完成输入检查。CFINPUT，CFGRID，CFSLIDER，CFTEXTINPUT，和CFTREE等标签都具有ONVALIDATE属性。
服务器端	在CFFORM中，你可以在支持输入检查的标签（如CFINPUT，CFTEXTINPUT）中使用VALIDATE属性，允许输入检查。 你也可以在HTML表单中使用隐含字段，这可以对几种通用数据类型进行输入检查。

必须输入的Form字段

HTML表单的缺点之一是它不能将字段定义为必须输入的字段。而对数据库应用程序来说，这是个很重要的要求，因此ColdFusion提供了一种服务器端的机制，强制用户必须在字段中输入数据。

要将一个字段定义为必须输入的字段，可以用一个隐含字段，其名字由后缀“_required”和要求必须输入数据的字段名组合而成。例如，要强制用户在字段“FirstName”中输入数据，可以使用如下语法；

```
<INPUT TYPE="hidden" NAME="FirstName_required">
```

如果用户没有在FirstName字段中输入数据，ColdFusion将不接受表单提交，而是返回一个错误信息，通知用户该字段必须输入数据。你可以用隐含字段的VALUE属性提供定制的错误信息。例如你想显示错误信息“You must enter your first name.”，可以使用如下语句：

```
<INPUT TYPE="hidden"
```

```
NAME="FirstName_required"
```

```
VALUE="You must enter your first name.">
```

隐含Form字段

HTML表单的另一个缺点是你不能对用户输入的数据进行检查。而ColdFusion可以利用表单中的隐含字段，对数据输入进行几种类型的检查。你可以用来进行输入检查的隐含字段后缀有如下几种：

使用隐含字段进行表单输入检查		
字段名后缀	VALUE属性	描述
_integer	定制的错误信息	检查用户是否输入了数字，如果用户输入的是浮点数，则转换为整数。
_float	定制的错误信息	检查用户输入的是否是数字，不进行浮点数到整数的转换。
_range	MIN=最小值 MAX=最大值	检查输入的数字是否在指定的范围内，你可以指定上界和下界或其中之一。
_date	定制的错误信息	检查用户输入的是否是日期值，并将日期值转换为ODBC日期格式。它可以接受最

		常用的日期格式，如9/1/98,Sept.9,1998。
_time	定制的错误信息	检查时间值的正确输入，并将时间值转换为ODBC时间格式。
_eurodate	定制的错误信息	检查用户输入的是否是标准欧洲格式的日期值，并将其转换为ODBC格式。

例子：隐含字段

下面的例子演示了如何用隐含字段进行数据输入检查。在这个例子中，被检查的表单包含字段“Rooms”，“Guests”，和“ArrivalDate”。为了保证Rooms字段输入一个整数，Guests字段的值在1到12之间，ArrivalDate字段输入一个合法的日期值，可以在表单中加入下列的隐含字段：

```
<INPUT TYPE="hidden"
```

```
NAME="Rooms_integer"
```

```
VALUE="You must enter a number for the Rooms field.">
```

```
<INPUT TYPE="hidden"
```

```
NAME="Guests_range"
```

```
VALUE="MIN=1 MAX=12">
```

```
<INPUT TYPE="hidden"
```

```
NAME="ArrivalDate_date"
```

```
VALUE="This is not a valid arrival date.">
```

VALUE属性是可选内容，如果你不指定该属性，则显示缺省的错误信息。

当表单被提交时，ColdFusion扫描所有的字段，寻找你所指定的检查规则，然后对用户的输入进行分析。如果输入的数据违反了任何规则，ColdFusion会显示给用户一个错误信息，用户必须返回前一个屏幕，修改输入的数据，然后重新提交表单。只有通过数据输入检查的表单提交才能被ColdFusion所接受。

数值型和日期型字段的自动检查

如果你使用CFINSERT或CFUPDATE，并且要操纵的数据库字段是数值型，日期型，或时间型，那么向这些数据库字段插入数据的Form字段会自动被检查。

两点补充：

· 对一个字段设置检查规则并不能使该字段成为必须输入的字段。如果要求用户必须输入该字段，仍需要使用带有”_required”的隐含字段。

· 数值数据通常包含逗号字符和美元符，对于使用了_integer,_float,或_range检查规则的字段，在检查输入数据和存入数据库之前，这些字符会被自动删除。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

动态HTML表单

动态HTML表单在Web应用程序中操纵数据库的最重要的工具之一。动态表单是利用数据库查询结果和CFML标签动态生成的表单。你可以通过动态表单保证数据的完整性，加快编码速度，以及在数据库表格之间建立联系。

下面的例子利用对表parks的查询结果，动态地建立一个选择列表：

```
<CFQUERY NAME="ParkNames"
```

```
DATASOURCE="ParkDB?"
```

```
SELECT ParkName_ID, ParkName
```

```
FROM Parks
```

```
</CFQUERY>
```

```
<FORM ACTION="example.cfm" METHOD="Post">
```

```
<SELECT NAME="ParkName_ID">
```

```
<CFOUTPUT QUERY="ParkNames">
```

```
<OPTION VALUE="#ParkName_ID#">#ParkName#
```

```
</CFOUTPUT>
```

```
</SELECT>
```

```
<INPUT TYPE="submit" VALUE="Submit">
```

```
</FORM>
```

当这个表单被提交时，返回的是ParkName_ID，在example.cfm文件中可以使用这个参数。其他表单元素可以用同样的方法动态地建立。

在HTML表单中使用多选按钮和多选列表

当一个HTML表单中包含一系列同名的多选按钮或一个多选列表框时，用户的输入被传递为用逗号隔开的一系列被选值。

注意，如果用户没有作任何选择，则没有变量被创建。这有可能导致SQL INSERT语句不能正确执行，解决问题的办法是强制用户必须作出选择（用隐含字段），或者使用动态SQL。

多选按钮

当你在表单中为一列多选按钮指定同一个名字时，相应Form变量的值是用逗号隔开的一列值。这些值可以是数字，也可以是字符串。这两种类型的值在处理时有点差别。

假设你想为用户提供一系列组织名，让用户通过多选按钮选择一个或多个组织。然后通过查询返回被选组织的详细信息。可以使用如下的语句：

Select one or more companies to get information on:

```
<INPUT TYPE="checkbox"
```

```
NAME="SelectedOrgs"
```

```
VALUE="5">
```

```
Mobil Corporation<BR>
```

<INPUT TYPE="checkbox"

NAME="SelectedOrgs"

VALUE="19">

Shapeware, Inc.

<INPUT TYPE="checkbox"

NAME="SelectedOrgs"

VALUE="13">

BankBoston

<INPUT TYPE="hidden"

NAME="SelectedOrgs_required"

```
VALUE="You must select at least one organization.">
```

注意虽然显示给用户的文本是组织的名字，但每个多选按钮的VALUE属性的值对应于数据库中每个组织的记录主关键字。

如果用户选择了Shapeware和BankBoston两项，则Form字段SelectOrgs的值为“19，13。”。如果在下面的SQL语句中使用这个参数：

```
SELECT *
```

```
FROM Organizations
```

```
WHERE Organization_ID IN ( #SelectedOrgs# )
```

对数据库实际执行的SQL语句为：

```
SELECT *
```

```
FROM Organizations
```

```
WHERE Organization_ID IN ( 19,13 )
```

要查询字符串类型的字段，多选按钮和CFQUERY标签的使用都与数值型字段稍有不同。在上面的例子中，我们根据一个数值型主关键字Organization_ID对数据库进行查询。如果主关键字是字符串型字段”OrganizationName”，则要对代码作如下修改：

多选按钮的VALUE属性中必须加上单引号：

```
<INPUT TYPE="checkbox"
```

```
NAME="SelectedOrgs"
```

```
VALUE=""Mobil Corporation"">
```

```
Mobil Corporation<BR>
```

```
<INPUT TYPE="checkbox"
```

```
NAME="SelectedOrgs"
```

```
VALUE=""ShapeWare, Inc."">
```

```
Shapeware, Inc.<BR>
```

```
<INPUT TYPE="checkbox"
```

```
NAME="SelectedOrgs"
```

```
VALUE=""'Bank of Boston'">
```

```
BankBoston<BR>
```

```
<INPUT TYPE="checkbox"
```

```
NAME="SelectedOrgs_required"
```

```
VALUE="You must select at least one organization.">
```

如果用户选择了Shapeware和BankBoston两项，则Form字段SelectOrgs的值为'ShapeWare,Inc.', 'BankBoston'。

注意：你必须在SQL语句中使用ColdFusion的PreserveSingleQuotes函数，以防止ColdFusion对Form字段值中的单引号进行转换：

```
SELECT *
```

```
FROM Organizations
```

```
WHERE OrganizationName IN (#PreserveSingleQuotes(SelectedOrgs)#)
```

对数据库执行的实际查询为：

```
SELECT *
```

```
FROM Organizations
```

```
WHERE OrganizationName IN ('ShapeWare, Inc.', 'BankBoston')
```

多选列表框

ColdFusion将多选列表框（HTML `<select multiple>` SELECT输入控件，指定属性MULTIPLE）与多选按钮作同样处理，即返回的Form变量值是逗号隔开的一列用户所选值。例如，一个多选列表框包括三项：red,green,blue。用户选择了red和green，则Form变量的值为'red','green'。与多选按钮类似，利用多选列表框进行数据库查询也有两种情况：查询数值型数据库字段和查询字符串型数据库字段。

假设你想让用户从一个多选列表中选择组织名，然后通过数据库查询返回被选中组织的详细信息，可以使用如下语句：

Select one or more companies to get more information on:

```
<SELECT Name="SelectOrgs" MULTIPLE>
```

```
<OPTION VALUE="5">Mobil Corporation
```

```
<OPTION VALUE="19">ShapeWare, Inc.
```

```
<OPTION VALUE="13">BankBoston
```

```
</SELECT>
```

```
<INPUT TYPE="hidden"
```

```
NAME="SelectedOrgs_required"
```

```
VALUE="You must select at least one organization.">
```

如果用户选择了Shapeware和BankBoston两项，则Form变量SelectOrgs的值为19,13。如果这个变量用在下面的SQL语句中：

```
SELECT *
```

```
FROM Organizations
```

```
WHERE Organization_ID IN (#SelectedOrgs#)
```

对数据库实际执行的语句为：

```
SELECT *
```

```
FROM Organizations
```

```
WHERE Organization_ID IN (19,13)
```

假如你想让用户通过多选列表框选择组织名，然后对数据库中的字符串型字段进行查询，可以使用下面的语句：

Select one or more companies to get

more information on:

```
<SELECT Name="SelectOrgs" MULTIPLE>
```

```
<OPTION VALUE=""Mobil Corporation"">Mobil Corporation
```

```
<OPTION VALUE=""ShapeWare, Inc."">ShapeWare, Inc.
```

```
<OPTION VALUE=""BankBoston"">BankBoston
```

```
</SELECT>
```

```
<INPUT TYPE="hidden"
```

```
NAME="SelectedOrgs_required"
```

```
VALUE="You must select at least one organization.">
```

如果用户选择了Shapeware和BankBoston两项，则Form变量SelectOrgs的值为'ShapeWare,Inc.','BankBoston'。

与多选按钮一样，对数据库中的字符串型字段执行查询时，要使
用PreserveSingleQuotes函数：

```
SELECT *
```

```
FROM Organizations
```

```
WHERE OrganizationName IN (#PreserveSingleQuotes(SelectedOrgs)#)
```

使用上面的语句，对数据库实际执行的查询为：

```
SELECT *
```

```
FROM Organizations
```

```
WHERE OrganizationName IN ('ShapeWare, Inc.', 'BankBoston')
```

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

动态SQL

嵌入使用动态参数的SQL查询是将用户输入与数据库字段相对应起来的强大机制。但是，在更复杂的应用程序中，通常不仅需要根据用户输入决定查询内容，而且需要根据用户输入决定查询的结构。

动态SQL使你能够动态地决定（根据运行时参数）对数据库执行SQL语句的哪一部分。例如，如果用户没有在一个查询字段中输入数据，你可以忽略WHERE从句中与该字段相关的部分。如果用户没有指定按哪个字段排序，则整个ORDER BY从句可以被忽略。

在ColdFusion中动态SQL的实现是利用CFIF/CFELSEIF/CFELSE标签控制SQL语句的建立。在CFQUERY中一个动态SQL语句有如下形式：

```
<CFQUERY NAME=" queryname"
```

```
DATASOURCE=" datasourcename" >
```

```
...Base SQL statement
```

```
<CFIF value operator value >
```

...additional SQL

</CFIF>

</CFQUERY>

在下面的例子中，用一系列CFIF标签来确定将哪一个SQL语句追加到基本SQL SELECT语句后面：

```
<CFQUERY NAME="GetParkList"
```

```
DATASOURCE="CF 4.0 Examples">
```

```
SELECT *
```

```
FROM Parks
```

```
WHERE 0=0
```

```
<CFIF #ParkName# is not "">
```

AND ParkName LIKE ‘%#ParkName#%’

</CFIF>

<CFIF #ParkType# is not "AllTypes">

AND ParkType = ‘#ParkType#’

</CFIF>

<CFIF #Region# is not "AllRegions">

AND Region = ‘#Region#’

</CFIF>

<CFIF #State# is not "">

AND State = ‘#State#’

</CFIF>

```
</CFQUERY>
```

提示：WHERE 0=0从句对数据库查询不产生任何影响。但如果所有的CFIF条件都不成立，该从句可以保证WHERE从句不会产生SQL语法错误。

下面的例子演示了如何用动态SQL来确保一个多选列表框的输入：

```
<CFQUERY NAME="GetParkList"
```

```
DATASOURCE="CF 4.0 Examples">
```

```
SELECT *
```

```
FROM Parks
```

```
WHERE 0=0
```

```
<CFIF IsDefined("ParkName_ID")>
```

```
AND ParkName_ID IN (#Form.ParkName_ID#)
```

</CFIF>

</CFQUERY>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

事务处理 (CFTRANSACTION)

你可以用CFTRANSACTION标签实现事务处理。一个CFTRANSACTION标签内的所有查询被视为一个事务处理单元。这意味着只有事务处理代码块中的所有查询都正确执行后，才能完成对数据库的修改。如果其中的一个查询发生了任何错误，则事务处理代码块中此查询之前的所有操作都被还原。

下面的例子演示了CFTRANSACTION标签的使用。如果第二个查询产生了错误，CFTRANSACTION保证了前一个查询所作的改动被还原，从而维持数据库中数据的一至性。

```
<CFTRANSACTION>
```

```
<CFQUERY NAME="WithdrawCash"
```

```
DATASOURCE="BankDB">
```

```
UPDATE Accounts
```

```
SET Balance=Balance - #Amount#
```

```
WHERE Account_ID=#AccountFrom#
```

```
</CFQUERY>
```

```
<CFQUERY NAME="DepositCash"
```

```
DATASOURCE="BankDB">
```

```
UPDATE Accounts
```

```
SET Balance=Balance + #Amount#
```

```
WHERE Account_ID=#AccountTo#
```

```
</CFQUERY>
```

```
</CFTRANSACTION>
```

注意，只有对同一个数据源的操作才支持事务处理。

你可以指定CFTRANSACTION标签的可选属性ISOLATION。该属性提供了

更精细的控制，控制数据库引擎在执行事务处理时如何锁定数据。

ISOLATION属性的合法值有：

????????????????·READ_UNCOMMITTED

????????????????·READ_COMMITTED

????????????????·REPEATABLE_READ

????????????????·SERIALIZABLE

注意：VERSIONING不再是ISOLATION属性的合法值，ColdFusion不再支持这个值。

事务处理的ODBC驱动程序支持

并非所有的驱动程序都支持事务处理，也并非所有支持事务处理的驱动程序都支持所有的ISOLATION级别。当你试图对某个驱动程序使用事务处理/锁定级别的组合时，ColdFusion会查询该驱动程序的事务处理能力。如果驱动程序不具备所需的事务处理能力，则ColdFusion返回一个错误。

关于驱动程序所支持的ISOLATION级别及其对每种级别如何实现等内容，需参考具体驱动程序的有关文档。

小结：

本章内容是本手册的重点之一，建议仔细阅读，并读懂所有的例子程序。需要强调的内容有：

- Form中隐含字段的使用（传参，数据输入检查）。
- CFINSERT和CFUPDATE标签要求Form字段名与要操作的数据库字段名相同。
- 根据数据库查询结果建立动态Form。
- 根据Form提交结果建立动态SQL语句。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

用CFFORM标签建立表单

这一章将重点介绍如何用CFFORM标签建立基于Java程序片的动态表单，通过本章的学习，你将可以使用一组基于Java的图形控件，而不需要直接编写Java代码。

用CFFORM标签建立表单

CFFORM标签使你能够建立动态表单，并提供了一组基于Java的表单控件，如树控件，滑动条控件和网格控件。在CFFORM中也可以使用标准的HTML控件，如多选按钮，单选按钮、文本框、编辑框等，CFFORM的优点在于你不需了解Java语言就可以象使用标准HTML控件一样使用这些基于Java程序片的控件，从而使编程更加简单易学。

用CFFORM建立的表单与标准HTML表单有相同的结构，在<CFFORM>和</CFFORM>这两个标签之间，你可以放置各种表单控件，如多选按钮和单选按钮（用CFINPUT），数据网格（用CFGRID），树控件（CFTREE），以及下拉列表（用CFSELECT）等。

你可以将HTML FORM标签与CFFORM标签结合使用，CFFORM标签由ColdFusion处理，ColdFusion根据CFFORM标签动态地生成HTML表单，然后将HTML代码传递给浏览器，如果你将现有的FORM标签都替换为CFFORM，该表单仍可以正常工作。

使用CFFORM建立表单比使用HTML FORM标签有如下优点：

- 为了使用Cold Fusion扩展，你不再需要在HTML FORM中频繁使用CFOUTPUT，在CFFORM中可以直接使用ColdFusion的扩展内容。
- 在CFFORM中可以使用许多基于Java的控件，而不需要学习Java语言，这些动态控件可以为表单提供强大的功能。
- 多数CFFORM控件提供了输入检查属性，你可以用这些属性来检查用户输入的合法性。
- 使用CFAPPLET，可以更简单地在CFFORM 中嵌入你自己的Java程序片。

对几乎所有的CFFORM控件，你都可以指定字体特征，布局，大小，滚动属性，以及许多其它选项。

由于每个基于Java的控件都需要在客户端运行一个Java程序片，这样，不支持Java或禁止了Java的浏览器将无法支持包含了这些控件的表单。使用NOTSUPPORTED属性，可以避免在浏览器中显示空白的程序片运行空间，而是提供一个错误信息。每个基于Java 的控件以及CFAPPLET标签都包含这一属性，你用NOTSUPPORTED属性指定一个错误信息，当浏览器不支持Java时显示该信息。

北极星书库

CFFORM控件的输入检查

CFINPUT和CFTEXTINPUT标签带有VALIDATE属性，利用该属性你可以为控件指定一个合法的数据类型。你可以检查的数据类型如下表：

检查输入的合法性	
数据类型	描述
Date	确保输入的日期是美国格式mm/dd/yyyy。
Eurodate	确保输入的是欧洲日期格式dd/mm/yyyy。
Time	确保输入的时间格式为hh:mm:ss。
Float	确保输入的是一个浮点数。
Integer	确保输入的是一个整数。
Telephone	检查电话号码的输入。电话号码的格式必须为### - ##

	# - # # # #。连字符 (-) 也可以用空格代替，区号必须以1到9之间的数字开头。
Zipcode	可以是5位或9位数字，写成 # # # # - # # # # 的格式。连字符可以用空格代替。
Creditcard	空格和破折号会被剔除，并使用Mod10算法检查数字。
Social_security_number	数字必须写成 # # # - # # - # # # # 的格式，也可以用空格代替连字符。

当你通过VALIDATE属性指定了一个输入数据类型时，ColdFusion会检查用户的输入与指定的类型是否匹配，只有成功匹配时才提交表单数据。

除了上述的数据检查方法之外，下面的这些标签支持ONVALIDATE属性，该属性允许你指定一个JavaScript函数来处理CFFORM的数据输入检查：

- CFINPUT
- CFGRID
- CFSLIDER

- CFTEXTINPUT

- CFTREE

ColdFusion将下列JavaScript对象传递给你在ONVALIDATE属性中所指定的Javascript函数。

- form-object

- input-object

- object-value

ONERROR属性允许你指定一个在输入检查失败时执行的JavaScript函数，例如，如果你在ONVALIDATE中指定了一个函数来执行数据输入检查，你还可以在ONERROR中指定另一个函数来执行检查失败处理，下列的标签均支持ONERROR属性：

- CFGRID

- CFINPUT

- CFSELECT
- CFSLIDER
- CFTEXTINPUT
- CFTREE

ColdFusion将下列JavaScript对象传递给ONERROR属性所指定的函数：

- torm-object
- inpnt-object
- object-value
- 错误信息文本

例子：表单输入检查

下面的例子利用JavaScript对输入的email地址进行检查：

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>JavaScript Validation</TITLE>
```

```
<SCRIPT>
```

```
<!--
```

```
function testbox(form) {
```

```
    Ctrl = form.inputbox1;
```

```
    if (Ctrl.value == "" || Ctrl.value.indexOf (执? 0) == -1) {
```

```
        return (false);
```

```
} else
```

```
return (true);
```

```
}
```

```
//-->
```

```
</SCRIPT>
```

```
</HEAD>
```

```
<BODY>
```

```
<H2>JavaScript validation test</H2>
```

```
<P>Please enter your email address:</P>
```

```
<CFFORM NAME="UpdateForm"
```

```
ACTION="update.cfm" >
```

```
<CFINPUT TYPE="text"
```

```
NAME="inputbox1"
```

```
REQUIRED="YES"
```

```
ONVALIDATE="textbox"
```

```
MESSAGE="Sorry, invalid entry."
```

```
SIZE="10"
```

```
MAXLENGTH="10">
```

```
<INPUT TYPE="Submit" VALUE=" Update... ">
```

```
</CFFORM>
```

```
</BODY>
```

</HTML>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

用CFTREE建立树控件

树控件适于用来显示有层次的信息。利用CFTREE，你可以根据数据库查询结果建立树形显示结构。你用CFTREEITEM建立树形结构的各个分支，你还可以选择使用六种内建的图标来代表树控件中的各个分支。

根据查询结果建立树形结构。

下面的简单例子根据CFQUERY的查询结果建立树形显示结构，它使用了最少的CFTREE标签和CFTREEITEM标签，演示了如何用CFML语言方便地建立树形结构。

首先，利用查询从数据源选择数据：

```
<CFQUERY NAME="Engineering" DATASOURCE="cfsnippets">
```

```
SELECT FirstName +' '+ LastName AS FullName
```

```
FROM EMPLOYEES
```

```
</CFQUERY>
```

然后，利用查询结果构建树控件：

```
<CFFORM NAME="form1" ACTION="submit.cfm"
```

```
METHOD="Post">
```

```
<CFTREE NAME="tree1" REQUIRED="yes"
```

```
HSCROLL="no" vSCROLL="yes">
```

```
<CFTREEITEM VALUE=FullName
```

```
QUERY="Engineering"
```

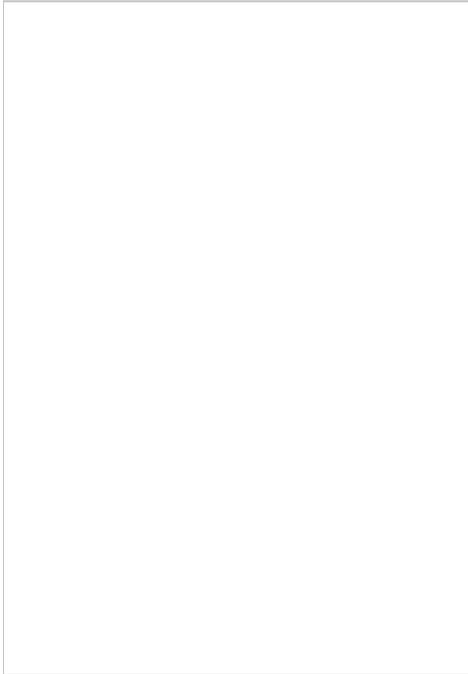
```
QUERYASROOT="yes"
```

```
IMG="folder,document">
```

```
</CFTREE>
```

```
</CFFORM>
```

此例产生的树控件如下图所示：



这个例子用QUERYASROOT属性指定将查询名（Engineering）作为树控件的根。QUERYASROOT属性的值可是Yes/No或者你想显示为查询结果的根的名字。

你可以根据几个不同的查询，利用来自不同数据源的数据建立一个树控件，使用QUERYASROOT，你可以为每个查询指定一个根（ROOT）。

分组显示查询结果

假设你想根据员工所在的部门来组织员工信息，并显示一个能反映部门与

员工关系的完整列表。在这种情况下，ColdFusion提供了一个非常简单的方法，在CFTREE中显示排序的CFQUERY查询结果。你只需在CFTREEITEM的VALUE属性中指定用逗号隔开的字段名，ColdFusion就会认为你希望树控件能反映SQL语句的排序结果。如下例：

```
<!-- CFQUERY with an ORDER BY clause -->
```

```
<CFQUERY NAME="myquery" DATASOURCE="cfsnippets">
```

```
SELECT DEPARTMENT, FirstName + ??+ LastName
```

```
AS FullName
```

```
FROM EMPLOYEES
```

```
ORDER BY DEPARTMENT
```

```
</CFQUERY>
```

```
<!--Build the tree control -->
```

```
<CFFORM NAME="form1" ACTION="submit.cfm"
```

METHOD="Post">

<CFTREE NAME="tree1"

HSCROLL="no"

VSCROLL="no"

BORDER="yes"

HEIGHT="350"

REQUIRED="yes">

<CFTREEITEM VALUE="Department, FullName"

QUERY="myquery"

QUERYASROOT="Department"

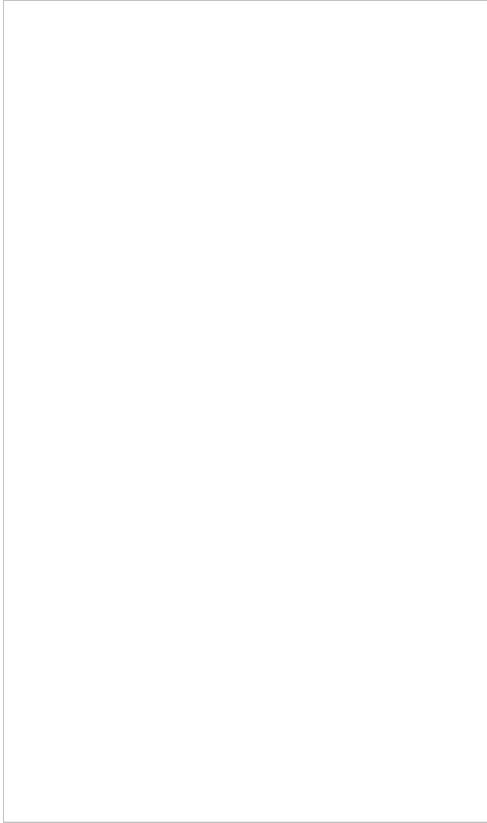
```
IMG="cd,folder">
```

```
</CFTREE>
```

```
<BR><INPUT TYPE="Submit" VALUE="Submit">
```

```
</CFFORM>
```

注意IMG属性与VALUE属性之间的对应。第一个字段， Department，使用ColdFusion的内建图标CD。第二个字段，FullName，使用另一个内建图标folder。如果IMG属性被省略，则ColdFusion为树控件的每一级都使用folder图标。此例的显示结果如下图所示：



如果用户选择了树控件中的“Peter Jacobsen”一项，则由Coldfusion返回下面的form变量：

```
form treel node=peter Jacobsen
```

```
form treel path=pepartment\sales\peter Jacobsen
```

注意：后面关于树控件的例子都使用上面的CFQUERY的查询结果，此查询的数据源是一个Microsoft Access数据库cfsnippets.mdb，要运行这些关于树控件的例子，只需引用“myquery”即可。

CFTREE表单变量

在CFTREE标签中将REQUIRED 属性设为YES，可以强制用户必须从树控件中选择一项，无论是否设置REQUIRED属性，ColdFusion都返回两个Form变量：

- `form.treename.rod`——返回用户所选的节点
- `form.treename.path`——返回用户选择的节点的完整路径，其格式为 `root\node1\node2\node_n\value`

只有当你将CFTREE的COMPLETEPATH属性设置为YES时，返回的路径才包括root部分，否则路径将从第一个节点开始。

CFTREE的输入检查

CFTREE不支持VALIDATE属性，但你可以用REQUIRED属性强制用户对树控件作出选择。另外，你可以在ONVALIDATE属性中指定一个JavaScript函数，由该函数执行输入检查。

构造树控件

用CFTREE可以建立非常复杂的树控件，而知道如何指定多个CFTREEITEM之间的关系将有助于处理甚至是最复杂的树控件结构。

例子：一级树控件

下面的例子用一个根和多个项建立了一个树控件：

```
<CFFORM NAME="form1" ACTION="submit.cfm">
```

```
<CFTREE NAME="tree1">
```

```
<CFTREEITEM VALUE="FullName"
```

```
QUERY="myquery"
```

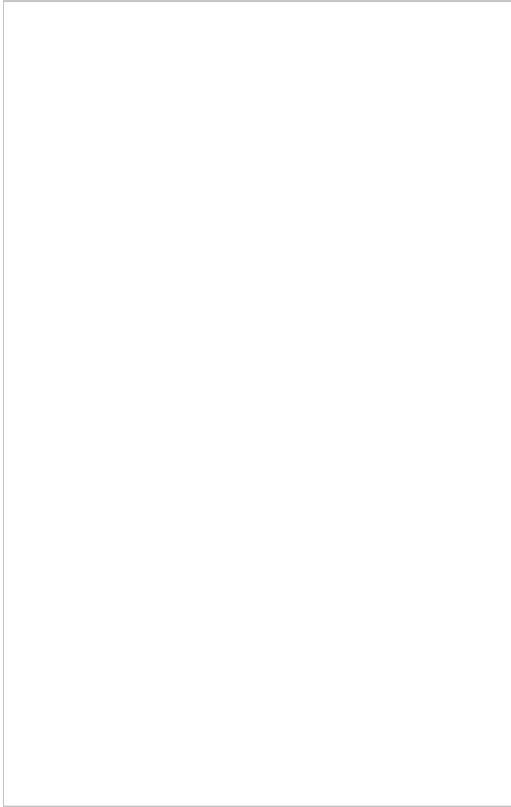
```
QUERYASROOT="Department">
```

```
</CFTREE>
```

```
<BR><INPUT TYPE="submit" VALUE="Submit">
```

```
</CFFORM>
```

这个例子的显示结果为:



例子：多级树控件

构造一个树形结构时，你通过为树的每一项指定一个“父”来定义树的结构。在下面的这个例子中，除最顶级以外，树的每一项都指定了一个“父”，PARENT属性使你可以定义树控件的各元素之间关系。

```
<CFFORM NAME="form1" ACTION="cform_submit.cfm"
```

```
METHOD="Post">
```

```
<CFTREE NAME="tree1" HSCROLL="no" VSCROLL="no"  
  
BORDER="no">
```

```
<CFTREEITEM VALUE="Divisions">
```

```
<CFTREEITEM VALUE="Development"
```

```
PARENT="Divisions" IMG="folder">
```

```
<CFTREEITEM VALUE="Product One"
```

```
PARENT="Development">
```

```
<CFTREEITEM VALUE="Product Two"
```

```
PARENT="Development">
```

```
<CFTREEITEM VALUE="GUI"
```

PARENT="Product Two" IMG="document">

<CFTREEITEM VALUE="Kernel"

PARENT="Product Two" IMG="document">

<CFTREEITEM VALUE="Product Three"

PARENT="Development">

<CFTREEITEM VALUE="QA"

PARENT="Divisions" IMG="folder">

<CFTREEITEM VALUE="Product One"

PARENT="QA">

<CFTREEITEM VALUE="Product Two" PARENT="QA">

<CFTREEITEM VALUE="Product Three"

PARENT="QA">

<CFTREEITEM VALUE="Support"

PARENT="Divisions" IMG="fixed">

<CFTREEITEM VALUE="Product Two"

PARENT="Support">

<CFTREEITEM VALUE="Sales"

PARENT="Divisions" IMG="cd">

<CFTREEITEM VALUE="Marketing"

PARENT="Divisions" IMG="document">

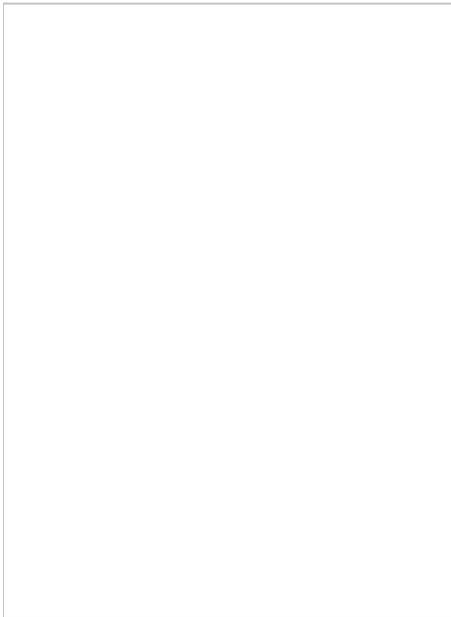
<CFTREEITEM VALUE="Finance"

```
PARENT="Divisions" IMG="element">
```

```
</CFTREE>
```

```
</CFFORM>
```

这个例子的显示结果为：



CFTREE中的图象名

当你使用TYPE = “Image”属性时，ColdFusion会试图显示一个与字段值相应的图象，该图象可以是ColdFusion的内建图象，也可以是你自选的图象，自选图象应位于cfide\classes目录或其子目录下，并应用相对的URL来

引用。

内建的图象包括：

- cd
- computer
- document
- element
- folder
- floppy
- fixed
- remote

在CFTREEITEM中使用逗号

逗号用来分隔CFTREEITEM, VALUE, DISPLAY, IMG和HREF等属性的值。在CFTREE中你用逗号来分隔字段名。如下例所示, VALUE属性指定了查询所返回的两个字段名:

```
<CFTREE NAME="tree1" VSCROLL="no"
```

```
HSCROLL="no" BORDER="no">
```

```
<CFTREEITEM VALUE="Department,LastName"
```

```
QUERY="myquery"
```

```
QUERYASROOT="Company B"
```

```
IMG="folder,folder,document"
```

```
PARENT="Company B">
```

```
</CFTREE>
```

在这个树控件中，Department和LastName是查询返回的两个字段。这个树根据不同的部门（Department）名字来分组显示LastName的值（假设查询中包含了相应的order by子句）。还应注意的是IMG属性的值。由于这个树控件共有三级，你可以通过用逗号分隔的图象名，为每一级指定一个内建的或定制的图象。

[后页](#)

[前页](#)

[目录](#)

[北极星书库](#)

在CFTREE中嵌入URL

CFTREEITEM标签的HREF属性允许你将树的一项指定为一个链接，要在CFTREE中使用这一特性，你只需在HREF属性中定义该链接的目标地点（即URL）

例子：添加Web链接

```
<CFFORM ACTION="submit.cfm">
```

```
<CFTREE NAME="oak"
```

```
HIGHLIGHTHREF="yes"
```

```
HEIGHT="100"
```

```
WIDTH="200"
```

```
HSPACE="100"
```

VSPACE="6"

HSCROLL="no"

VSCROLL="no"

BORDER="no"

DELIMITER="?">

<CFTREEITEM VALUE="Important Links">

<CFTREEITEM VALUE="Allaire Home"

PARENT="Important Links"

IMG="document"

href="http://www.allaire.com">

<CFTREEITEM VALUE="Allaire Forums"

```
PARENT="Important Links"
```

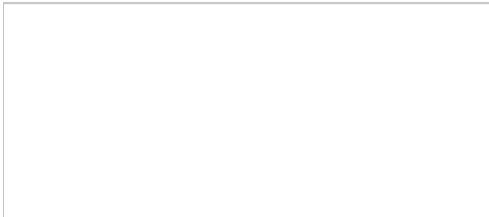
```
IMG="document"
```

```
HREF="http://forums.allaire.com">
```

```
</CFTREE>
```

```
</CFFORM>
```

这个例子的显示结果为：



CFTREEITEM的APPENDKEY属性

当用户选择了树控件中的一项并提交表单时，变量CFTREEITEMKEY会被追加到URL后面，传递给ACTION属性所指的文件，其格式为：

http://myserver.com? CFTREEITEMKEY=Selected_value

通过设置CFTREE的APPENDKEY属性为NO，你可以禁止该URL变量。

CFTREEITEM的TARGET属性

可以在CFTREEITEM标签中用TARGET属性为链接指定一个目标（TARGET）。当利用查询结果构造树控件时，你可以将一系列TARGET属性值用逗号隔开，使这些值分别对应于树控件中不同级的链接项，例如：

TARGET="FRAME_BODY,blank,_top"

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

用CFGRID建立数据网格

CFGRID是另一个基于Java程序片的CFFORM控件。一个网格控件类似于一个表，可以包含来自CFQUERY的查询结果或其它来源的数据。与别的CFFORM标签一样，CFGRID也提供了一系列数据格式选项以及用户输入检查选项，CFGRID的其它特性包括：

- 对网格中的数据进行排序
- 更新，插入和删除数据
- 网格中可以嵌入图象

当用户选择网格中的数据并提交表单时，ColdFusion将被选信息以form变量的形式传递给ACTION属性所指的文件。

注意：如果你在CFGRID中指定了QUERY属性，但没有指定相应的CFGRIDITEM属性，则缺省的网格将包含查询中的所有字段。

在CFGRID中使用CFGRIDCOLUMN标签就象在CFTREE中使用CFTREEITEM标签一样。CFGRIDCOLUMN标签用于定义网格中的一列。

利用查询结果建立网格控件

下面的例子利用一个CFQUERY的查询结果构造了一个简单的数据网格：

```
<CFQUERY NAME="getdata" DATASOURCE="cfsnippets">
```

```
SELECT * FROM Employees
```

```
</CFQUERY>
```

```
<CFFORM NAME="Form1" ACTION="submit.cfm" METHOD="Post">
```

```
<CFGRID NAME="employee_grid" QUERY="getdata"
```

```
SELECTMODE="single">
```

```
<CFGRIDCOLUMN NAME="Employee_ID">
```

```
<CFGRIDCOLUMN NAME="LastName">
```

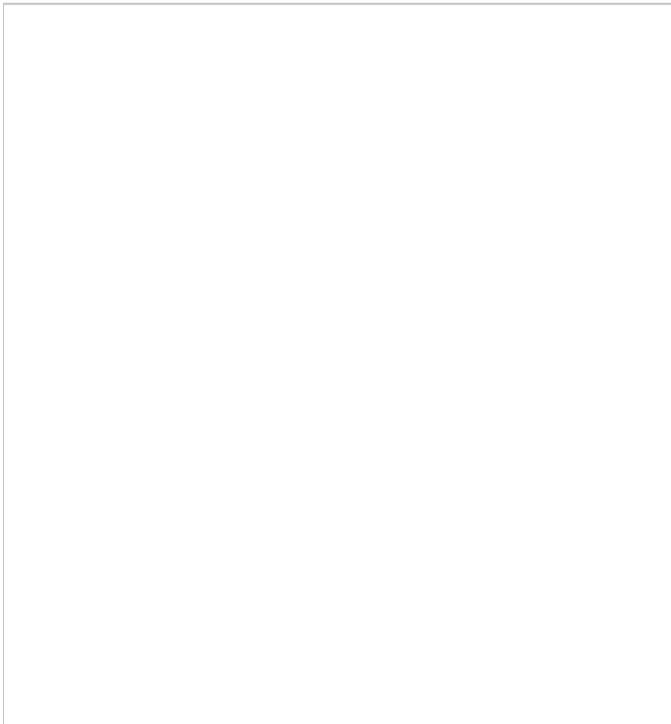
```
<CFGRIDCOLUMN NAME="Department">
```

```
</CFGRID>
```

```
<BR><INPUT TYPE="Submit" VALUE="Submit">
```

```
</CFFORM>
```

这个例子的显结果类似于：



隐藏网格字段

你可以用CFGRIDCOLUMN的DISPLAY属性将你想从数据源中选择但不想显示给用户的字段从网格中隐去，如用户ID号或其它的主关键字字段等。在下面的例子中，员工ID字段被查询，但不被显示：

```
<CFQUERY NAME="getdata" DATASOURCE="cfsnippets">
```

```
SELECT * FROM Employees
```

```
</CFQUERY>
```

```
<CFFORM NAME="Form1"
```

```
ACTION="submit.cfm"
```

```
METHOD="Post"
```

```
ENABLECAB="Yes">
```

```
<CFGRID NAME="grid1" QUERY="getdata"
```

```
SELECTMODE="single">
```

<CFGRIDCOLUMN DISPLAY="No"

NAME="Employee_ID">

<CFGRIDCOLUMN NAME="LastName">

<CFGRIDCOLUMN NAME="Department">

</CFGRID>

<INPUT TYPE="Submit" VALUE="Submit">

</CFFORM>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

建立可更新的网格

你可以建立允许用户进行数据编辑的网格，根据用户的编辑结果，你可以用CFQUERY标签对数据源进行修改，也可以使用CFGRIDUPDATE标签，将用户对网格数据的编辑直接写到数据库中。

单元格中的数据可以被编辑，一行数据可以被插入，删除或更新，但在此之前。你需要在CFGRID中指定SELECTMODE=“EDIT”来允许编辑数据，还许要允许INSERT属性和DELETE属性（将其值设为“YES”）。完成这些设置后，该网格即可用来对数据源进行操作。

使用可更新网格对ODBC数据源进行修改的方法主要有两种。你可以建立一个文件，将CFGRID表单变量传递给该文件，在该文件中用CFQUERY标签执行对数据源的更新，另一种方法是将网格编辑结果传递给一个包含CFGRIDUPDATE标签的文件，该标签直接将编辑结果写入数据源，虽然使用CFQUERY可以实现对数据源的完全控制，但对不需要复杂控制的操作来说，CFGRIDUPOATE使用起来更加简单。

编辑网格中的数据

设置SELECTMODE=“EDIT”后，用户即可编辑网格中的数据。用户可以点击一个单元格，然后用简单的编辑操作改单元格中的数据。设置INSERT=“YES”和DELETE=“YES”后，用户可以选择一行数据并删除之，也可以选择一行数据，在该位置插入一个新行。当一个包含CFGRID标签的CFFORM被提交时，数据的修改被记录在几个一维数组中，你可以象引用其它的ColdFusion数组一样引用这些数组。

下列的这些数组用来跟踪网格数据的修改：

用于存储单元格编辑信息的数组	
数组	描述
gridname.colname[row_index]	存储被编辑单元格的新值
gridname.Original.colname[row_index]	存储被编辑单元格的原始值
gridname.RowStatus.Action[row_index]	存储对单元格的编辑类型

例如，有一个网格“mygrid”，包含两个可显示字段CO11和CO12，一个隐藏字段CO13。当用户选择并修改了某一行中的数据时，则下列数组被创建，用于保存被更新，插入或删除的行的原始数据和新数据。

mygrid.col1[row_index]

mygrid.col2[row_index]

mygrid.col3[row_index]

```
mygrid.original.col1[ row_index ]
```

```
mygrid.original.col2[ row_index ]
```

```
mygrid.original.col3[ row_index ]
```

其中row_index是数组的索引值。

如果用户修改了CO12字段的某个单元格，你可以用如下的方式引用编辑操作类型、该单元格的原始值，以及编辑后的新值：

```
<CFSET edittype = mygrid.RowStatus.Action[1]><BR>
```

```
<CFSET new_value = mygrid.col2[1]><BR>
```

```
<CFSET old_value = mygrid.original.col2[1]>
```

为INSERT按钮和DELETE按钮指定替代文本

如果你希望可更新网格的插入和删除按钮使用不同于“Insert”和“Delete”的文本，你可以在INSERTBUTTON属性和DELETEBUTTON属性中指定替代

的文本。

对网格数据排序

CFGRID标签的SORT属性用于允许在网格控件中包含排序按钮，允许排序后，排序按钮会被自动添加到网格中。点击该按钮时，网格数据会按选中的字段进行排序。ColdFusion将字段值视为文本或数字而进行排序。

下列的CFGRID属性用于定义与排序有关的各个选项

- PICTUREBAR——当设置为YES时，排序按钮是一个图形按钮，而非文本按钮。
- SORTASCENDINGBUTTON——用于指定升序排序按钮的显示文本，缺省值为“A→Z”
- SORTDESCENDINGBUTTON——用于指定降序排序按钮的显示文本，缺省值为“Z→A”

注意：用户在点击排序按钮之前必须首先选择一个字段。

例子：可编辑的网格

下面的例子演示了可更新网格的使用，在这个例子中，SELECTMODE属性被设置为“Edit”，INSERT属性和DELETE属性被设置为“yes”。当这个表单被提交时，handle_grid.cfm文件被调用，该文件显示编辑操作的类型，并直接对数据源执行相应的更新、删除或插入等操作。

Grid.cfm

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Simple Update Grid Example</TITLE>
```

```
</HEAD>
```

```
<CFQUERY NAME="CourseList"
```

```
DATASOURCE="cfsnippets">
```

```
SELECT * FROM Courses
```

```
</CFQUERY>
```

<BODY BGCOLOR="#FFFFFF">

<CFFORM NAME="GridForm"

ACTION="handle_grid.cfm">

<CFGRID NAME="course_grid"

HEIGHT=170

WIDTH=400

HSPACE=10

VSPACE=6

ALIGN="RIGHT"

SELECTCOLOR="white"

SELECTMODE="edit"

ROWHEADERS="YES"

ROWHEADERWIDTH=25

ROWHEADERALIGN="right"

COLHEADERS="YES"

QUERY="CourseList"

GRIDDATAALIGN="left"

BGCOLOR="green"

INSERT="YES"

DELETE="YES"

SORT="YES"

MAXROWS=60>

<CFGRIDCOLUMN NAME="course_id"

HEADER="Course ID"

WIDTH=80

ITALIC="NO"

HEADERALIGN="center"

HEADERITALIC="NO"

HEADERBOLD="YES"

DISPLAY="NO">

<CFGRIDCOLUMN NAME="number"

HEADER="Course ##"

WIDTH=80

ITALIC="NO"

HEADERALIGN="center"

HEADERITALIC="NO"

HEADERBOLD="YES"

DISPLAY="YES"

SELECT="YES">

<CFGRIDCOLUMN NAME="description"

HEADER="Description"

WIDTH=240

ITALIC="No"

HEADERALIGN="center"

HEADERITALIC="No"

HEADERBOLD="Yes"

BOLD="Yes"

ITALIC="Yes"

DISPLAY="Yes">

</CFGRID>

<!--

<H3>Editable Grid</H3>

This is a grid that is populated

from a query. The select mode is

update. The description column

is presented in Bold and Italic.

 --->

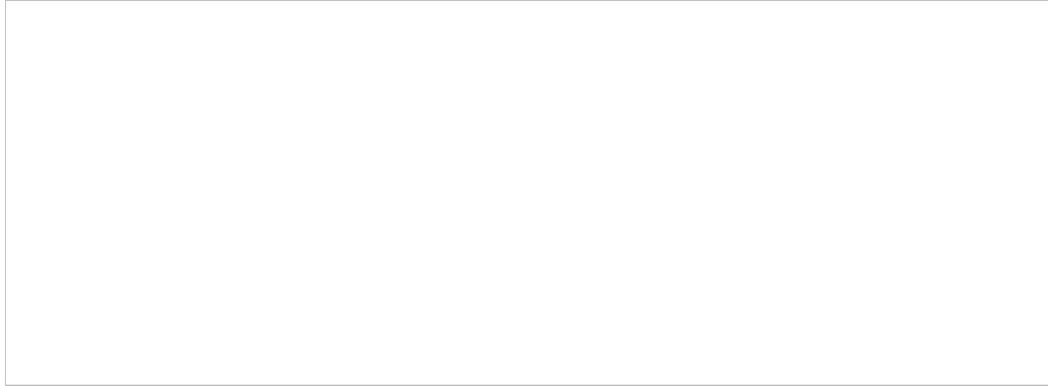
<INPUT TYPE="Submit" VALUE=" Push me... ">

</CFFORM>

</BODY>

</HTML>

这个例子的显示结果类似于下图：



你对网格数据的修改将反映在数据源cfsnippets的表Courses中，你可以在Studio中查开这个表。

Handle_grid.cfm

<HTML>

<HEAD>

<TITLE>Catch submitted grid values</TITLE>

</HEAD>

<BODY>

<H3>Grid values for FORM.Course_grid row updates</H3>

<CFIF IsDefined("form.course_grid.rowstatus.action")>

<CFLOOP INDEX = "Counter" FROM = "1" TO =

#ArrayLen(form.course_grid.rowstatus.action)#>

<CFOUTPUT>

The row action for #Counter# is:

#form.course_grid.rowstatus.action[Counter]#

</CFOUTPUT>

<CFIF form.course_grid.rowstatus.action[Counter] IS "D">

<CFQUERY NAME="InsertNewCourse"

```
DATASOURCE="cfsnippets">
```

```
DELETE from courses
```

```
WHERE course_id=#form.course_grid.original.course_id[Counter]#
```

```
</CFQUERY>
```

```
<CFELSEIF form.course_grid.rowstatus.action[Counter] IS "U">
```

```
<CFQUERY NAME="UpdateExistingCourse"
```

```
DATASOURCE="cfsnippets">
```

```
UPDATE courses
```

```
SET description='#form.course_grid.description[Counter]#',
```

```
"Number"='#form.course_grid.number[Counter]#'
```

WHERE

course_id=#form.course_grid.original.course_id[Counter]#

</CFQUERY>

<CFELSEIF form.course_grid.rowstatus.action[Counter] IS "I">

<CFQUERY NAME="InsertNewCourse"

DATASOURCE="cfsnippets">

INSERT into courses

("Number", description)

VALUES ('#form.course_grid.number[Counter]#',

'#form.course_grid.description[Counter]#')

</CFQUERY>

</CFIF>

</CFLOOP>

</CFIF>

</BODY>

</HTML>

使用CFGRIDUPDATE

CFGRIDUPDATE标签可以根据CFGRID的内容直接对数据源进行更新，你不需要在CFQUERY中使用SQL语句来完成数据的更新，CFGRIDUPDATE标签可以直接取得CFGRID中的数据编辑信息，自动完成整个处理过程。

在前面的例子中，来自可编辑网格的数据被传给一个文件，该文件在CFIF结构中使用CFQUERY标签完成对数据源的更新，而在许多情况下，用CFGRIDUPDATE标签使程序更加简单。

例如，下面的代码从一个名为“Courses”的网格接收数据，对数据源进行更新。

```
<CFGRIDUPDATE GRID="Courses"
```

```
DATASOURCE="CF 4.0 Examples"
```

```
TABLENAME="Courses"
```

```
KEYONLY="NO">
```

当表单被提交时，存储有关数据修改信息的数组被传递给CFGRIDUPDATE标签，此标签根据这些数组的内容构造必要的SQL语句来完成数据的更新，从而你不必要自己构造复杂的SQL语句，非常方便。

KEYONLY属性

CFGRIDUPDATE标签包含一个KEYONLY属性，该属性可以强制ColdFusion对被更新字段的原始值与表中的字段值进行比较。如果两者相同，也就是说，在网格数据被编辑的过程的没有其它进程曾改变过数据库中的数据，则更新成功，如果两者不相同，则产生错误。当你要确保没有其它进程曾更新过该数据时，应设置KEYONLY="NO"。如果不可能有其它进程更改该数据，可设置KEYONLY="YES"。

在网格中嵌入图象

CFGRIDROW标签使你能够在网格单元格中放入图象，首先你要在CFGRIDCOLUMN标签中设置TYPE="IMAGE"，以告诉ColdFusion将该列数据理解为一个图象。你可以使用ColdFusion的内建图象（与CFTREEITEM相同），也可以指定自选的图象。下面的这段代码演示了如何在网格中嵌入ColdFusion的内建图象。

...

```
<CFGRIDCOLUMN NAME="dept_name"
```

```
  HEADER="Dept"
```

```
  SELECT="NO"
```

```
  DATAALIGN="Center"
```

```
  WIDTH=40
```

```
  TYPE=IMAGE>
```

```
<CFGRIDCOLUMN NAME="emp_lname"
```

```
  HEADER="Name">
```

```
<CFGRIDROW DATA="folder,Jones">
```

```
<CFGRIDROW DATA="document,Smith">
```

...

注意，在CFGRIDROW中，用逗号将显示在第一列的图象名与显示在第二列的数据分隔开。

在网格中使用自己的图象

当你想用自己的图象代替ColdFusion的内建图象时，你需要指定该图象所在目录的相对路径，以及图象文件名。指定图象文件的相对路径时，要注意该路径是相对于支持CFGRID控件的Java类的路径，它通常是：

```
Webroot\cfide\classes\images
```

因此你可以指定一个此路径的相对路径，或者将要使用的图象文件放在cfide\classes\images目录下，然后即可直接使用文件名而无须指定路径，就象使用内建图象一样。

在下面的代码段中，CFGRIDROW标签使用了webroot\images目录下的图象文件：

...

<CFGRIDCOLUMN NAME="dept_name"

HEADER="Dept"

SELECT="NO"

DATAALIGN="Center"

WIDTH=40

TYPE=IMAGE>

<CFGRIDCOLUMN NAME="emp_lname"

HEADER="Name">

<CFGRIDROW DATA="..\..\..\images\icon1.gif,Jones">

<CFGRIDROW DATA="..\..\..\images\icon2.gif,Smith">

...

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

网格数据选择选项

你可以控制用户如何与你的网格控件相交互，你可以限制一个用户只能浏览网格中的数据。你可以在SELECTMODE属性中指定几个不同的选择选项。

- Single——限制用户只能选择网格控件的一个单元格。
- Column——当用户选择一个单元格时，包含此单元格的一列数据均被选取。
- Row——当用户选择一个单元格时，包含此单元格的一行数据均被选取。
- Browse——用户不能选择单元格，只能浏览数据。
- Edit——用户可以编辑单元格数据。

选择模式与form变量

根据SELECTMODE属性的不同取值，网格数据被提交为不同形式的form变量：

- 当SELECTMODE=“single”时，返回的变量为grid-name.selectedname及被选中的值。
- 当SELECTMODE=“Column”时，返回的是被选列的用逗号隔开的所有字段值。
- 当SELECTMODE=“Row”时，返回的变量为gridname.Column1_name和grid_name.Column2_name等，以及相应的被选字段值。
- 当SELECTMODE=“Browse”时，没有返回数据。
- 当SELECTMODE=“Edit”时，如果单元格数据被修改，则创建并返回三个一维数组。

使用URL属性

当在单元格中使用URL时，SELECTMODE属性的值决定了该链接是限制于一个单元格，还是扩展到该单元格所在的行或列。当用户点击一个链接时，一个CFGRIDKEY变量被追加到URL后面，其格式为：

<http://myserver.com?CFGRIDKEY=selection>

Selection的值由SELECTMODE属性的值决定：

- 当SELECTMODE="single"时，Selection即用户选中的单元格的值。
- 当SELECTMODE="ROW"时，Selection是被选单元格所在行的各单元格的值，用逗号将各单元格的值隔开，起始于该行的第一个单元格。
- 当SELECTMODE="Column"时，Selection是被选单元格所在列的各单元格的值，起始于该列的第一个单元格，各单元格的值用逗号隔开。

HREF属性

你可以用HREF属性将一个超链接与被选中的行或单元格相关联。ColdFusion将HREF属性的值理解为一个存储链接文本的查询字段或链接文本自身。HREF的链接目标仅限于当前的应用程序文件。

在下面的这段代码中，两个CFGRIDCOLUMN使用了HREF属性，由于HREF属性引用了另一个CFGRIDCOLUMN，从而每一行都链接一个不同的URL，此URL由被选中行的dept_url字段的值决定。注意最后一个CFGRIDCOLUMN是一个隐藏字段，用于隐藏dept_url字段的值。

```
<CFFORM NAME="GridForm"
```

ACTION="catch_grid10.cfm"

TARGET="Lower">

<CFGRID NAME="grid_ten"

HEIGHT=170

WIDTH=400

HSPACE=10

VSPACE=6

ALIGN="Right"

SELECTMODE="Row"

ROWHEADERS="Yes"

COLHEADERS="Yes"

QUERY="DeptList"

GRIDDATAALIGN="Left"

HIGHLIGHTHREF="No"

APPENDKEY="No"

SORT="Yes">

<CFGRIDCOLUMN NAME="dept_id"

HEADER="Department"

WIDTH=80

ITALIC="No"

HEADERALIGN="Center"

HEADERITALIC="No"

HEADERBOLD="Yes"

HREF="dept_url"

TYPE="Numeric">

<CFGRIDCOLUMN NAME="dept_name"

HEADER="Name"

ITALIC="No"

HEADERALIGN="Center"

HEADERITALIC="No"

HEADERBOLD="Yes"

HREF="dept_url">

```
<CFGRIDCOLUMN NAME="dept_url"
```

```
DISPLAY="No">
```

```
</CFGRID>
```

```
<INPUT TYPE="Submit" VALUE="Submit"> <BR>
```

```
</CFFORM>
```

CFGRIDKEY的APPENDKEY属性

当用户选择了网格数据并提交表单时，CFGRIDKEY变量即被追加到URL后面，传递给ACTION属性所指的文件。你可以通过设置APPENDKEY="NO"来禁止这个URL变量。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

建立滑动条控件

CFSLIDER是ColdFusion提供的另一个基于Java的表单控件。使用CFSLIDER你可以建立滑动条，并可以为滑动条标识文本设置一系列格式选项，可以设置滑动条的比例，范围，位置等。

与CFTREE和CFGRID一样，通过ONVALIDATE 属性指定的JavaScript函数，可以进行输入检查。

例子：CFSLIDER控件

下面的例子建立了一个简单的CFSLIDER控件：

```
<CFFORM NAME="Form1" ACTION="submit.cfm"
```

```
METHOD="Post">
```

```
<CFSLIDER NAME="myslider"
```

```
GROOVECOLOR="black"
```

BGCOLOR="white"

TEXTCOLOR="black"

FONT="Trebuchet MS"

BOLD="yes"

RANGE="0,1000"

SCALE="10"

VALUE="640"

FONTSIZE="24"

LABEL="Slider %value%"

WIDTH="400">

</CFFORM>

显示结果如下图：



CFSLIDER表单变量

来自CFSLIDER控件的表单变量的值决定于滑动条的位置。该表单变量被传递为：

```
slider_name=Slider_value
```

在上面的例子中，表单变量为：

```
myslider=slider_value
```

CFSLIDER的格式选项

与其它的CFFORM控件一样，CFSLIDER提供了许多关于格式，位置和布

局的选项。你可以指定滑槽的颜色，以及标识文本的字体、大小、加粗、倾斜和颜色等选项。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

建立文本输入框

CFTEXTINPUT标签与HTML INPUT=text标签功能相似，但利用CFTEXTINPUT你可以指定字体和布局选项，并可以通过JavaScript或VALIDATE属性进行输入检查。

例子：CFTEXTINPUT控件

下面的例子是一个简单的CFTEXTINPUT控件，这个例子对日期输入进行检查，即用户必须按照mm/dd/yy的格式输入一个合法的日期值。

Please enter a date:

```
<CFFORM NAME="Form1" ACTION="cform_submit.cfm"
METHOD="Post">
```

```
<CFTEXTINPUT NAME="entertext"
```

```
VALUE="mm/dd/yy"
```

```
MAXLENGTH="10"
```

```
VALIDATE="date"
```

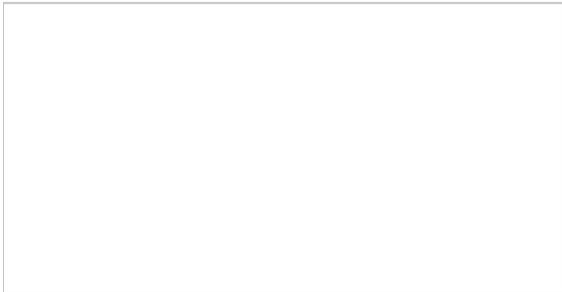
```
FONT="Trebuchet MS">
```

```
<BR><INPUT TYPE="Submit"
```

```
VALUE="Submit">
```

```
</CFFORM>
```

此例的显示结果为：



CFTEXTINPUT表单变量

从CFTEXTINPUT控件返回的form变量的值即用户的输入值，其格式为：

textinput_name=textinput_value

在上一个例子中，返回的form变量为：

entertext=textinput_value

在ACTION属性所指的文件用，应将该变量引用为#entertext#。

CFTEXTINPUT的输入检查

你可以利用VALIDATE属性对以下格式的数据进行输入检查：

检查输入的合法性	
数据类型	描述
Date	确保输入的日期是美国格式mm/dd/yyyy。
Eurodate	确保输入的是欧洲日期格式dd/mm/yyyy。
Time	确保输入的时间格式为hh:mm:ss。

Float	确保输入的是一个浮点数。
Integer	确保输入的是一个整数。
Telephone	检查电话号码的输入。电话号码的格式必须为### - ## # - #####。连字符(-) 也可以用空格代替，区号必须 以1到9之间的数字开头。
Zipcode	可以是5位或9位数字，写成# #### - #####的格式。 连字符可以用空格代替。
Creditcard	空格和破折号会被剔除，并使 用Mod10算法检查数字。
Social_security_number	数字必须写成### - ## - #####的格式，也可以用空 格代替连字符。

建立下拉列表框

你可以用CFSELECT建立与HTML `SELECT`标签相似的下拉列表框，但CFSELECT提供了对用户输入，错误处理等的更全面的控制，并允许你根据数据库查询结果构造下拉列表。

用查询结果构造CFSELECT

利用查询结果构造下拉列表时，你只需指定为CFSELECT提供数据的查询名和要显示在下拉列表中的查询字段名。如下例：

```
<CFQUERY NAME="myquery"
```

```
DATASOURCE="cfsnippets">
```

```
SELECT * FROM Employees
```

```
</CFQUERY>
```

```
<CFFORM NAME="Form1" ACTION="submit.cfm"
```

METHOD="Post">

<CFSELECT NAME="myselectbox"

QUERY="myquery"

VALUE="Employee_ID"

DISPLAY="FirstName"

REQUIRED="yes"

MULTIPLE="yes"

SIZE="8">

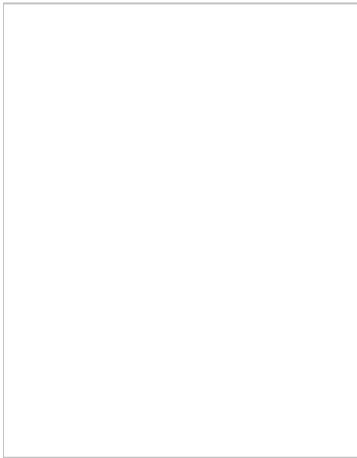
</CFSELECT>

<INPUT TYPE="Submit"

```
VALUE="Submit">
```

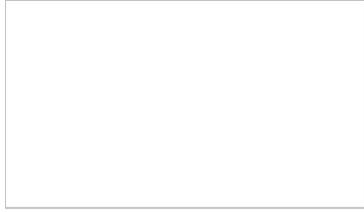
```
</CFFORM>
```

此例的显示结果类似于：



注意例子中使用了MULTIPLE属性，用户可以选择多项内容。当MULTIPLE属性被忽略或设置为SINGLE，并且SIZE属性被设置为零时，下拉列表框的显示结果则为：

另外应该注意的是，在此例中用VALUE属性指定了表Employee的主关键字，因此传递给ACTION属性所指文件的form变量的值是该主关键字的值，而不是显示在列表框中的文本。



[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

建立FORM控件

与HTML INPUT标签相似，CFINPUT标签支持下列表单输入控件：

- 单选按钮
- 多选按钮
- 文本输入框
- 密码输入框

CFINPUT也支持用VALIDATE属性或JavaScript函数进行数据输入检查。

例子：CFINPUT控件

下面的代码是一个包含了多种CFINPUT控件的表单：

```
<CFFORM NAME="Form1" ACTION="submit.cfm" METHOD="Post">
```

```
<TABLE CELLPADDING=5 border=0>
```

```
<TR><TD>Please enter your user login:<BR>
```

```
<CFINPUT TYPE="text" NAME="loginID" VALUE="name"></TD></TR>
```

```
<TR><TD>Please also enter your password:<BR>
```

```
<CFINPUT TYPE="password" NAME="pwd"
```

```
VALUE="password"></TD></TR>
```

```
<TR><TD>Please select one:<BR>
```

```
<CFINPUT TYPE="radio" NAME="radio1"
```

```
VALUE="select1">Embodied<BR>
```

```
<CFINPUT TYPE="radio" NAME="radio1" CHECKED="yes"
```

VALUE="select2">

Disembodied

<CFINPUT TYPE="radio" NAME="radio1"

VALUE="select3">Don'tKnow</TD></TR>

<TR><TD>Make your selections here:

<CFINPUT TYPE="checkbox" NAME="checkbox1"

VALUE="one">Derrida

<CFINPUT TYPE="checkbox" NAME="checkbox1" CHECKED="yes"

VALUE="two">Foucault

<CFINPUT TYPE="checkbox" NAME="checkbox1" VALUE="three">

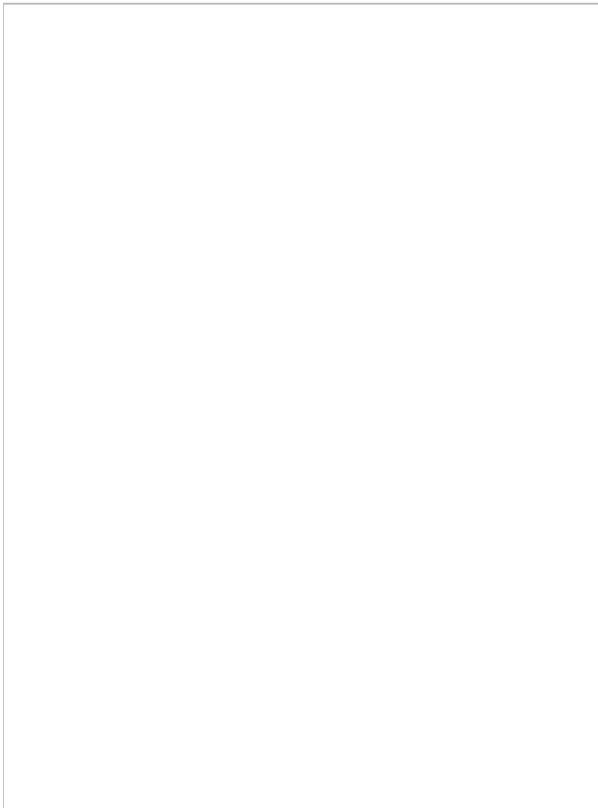
Kristeva</TD></TR>

```
<TR><TD><INPUT TYPE="Submit" VALUE="Submit"></TD></TR>
```

```
</TABLE>
```

```
</CFFORM>
```

这段代码可生成如下图所示的表单：



目录
北极星书库

嵌入Java程序片

CFAPPLET标签使你可以在CFFORM中嵌入Java程序片。要使用这个标签，首先要用ColdFusion Administrator对Java程序片进行注册。在Administrator中，你定义Java程序片的接口，对其进行封装，从而使CFAPPLET标签的使用非常简单。与HTML APPLET标签相比，CFAPPLET有如下几个优点：

- 返回值——由于CFAPPLET需要指定一个form字段名，因此你不必编写额外的JavaScript代码来获取程序片的返回值，你可以象对待其它form变量一样直接引用返回值：form.VariableName。
- 使用简单——由于程序片的接口在Administrator中定义，应用程序文件中的每个CFAPPLET标签只需引用程序片的名字并指定一个form变量名即可。
- 参数选项——通过在CFAPPLET中指定参数值，你可以覆盖在Administrator中设定的参数值。否则ColdFusion将使用你在Administrator中设定的参数值。

当一个程序片被注册之后，指定程序片名和form变量名即可使用该程序片：

```
<CFAPPLET APPLETSOURCE="Calculator"
```

NAME="calc_value">

与此相反，若使用HTML APPLET标签，你必须在每次使用Java程序片时设定所有的参数

注册Java程序片

在ColdFusion文件中使用Java程序片之前，必须先对程序片进行注册。注册的步骤为：

- 1.从ColdFusion程序组中点击Administrator图标，输入管理员密码（如果需要的话），打开ColdFusion Administrator。
- 2.点击Applets按钮，打开Registered Applets网页。
- 3.为要注册的程序片输入一个名字，然后点击Register New Applet。
- 4.输入程序片所需的信息，选择你想要的高度、宽度、垂直和水平方向的空间，以及布局设置。

程序片注册所需的信息如下表所列：



程序片注册域	
域	描述
Codebase	<p>程序片的URL，即程序片所在的目录。程序片类文件必须存放在浏览器根目录下，如：</p> <p>http://servername/classes</p>
Code	<p>包含程序片子类的文件名。此文件名与URL相对应。不需要指定后缀名.class。</p>
Method	<p>程序片中返回一个字符串的方法名。你在CFAPPLET标签的NAME属性中使用这个方法名，将该方法的返回值赋给一个FORM变量。如果此程序片没有方法，可另此域保持为空。</p>
Height	<p>该程序片在垂直方向的高度（象素个数）</p>
Width	<p>该程序片在水平方向的宽度（象素个数）</p>
Vspace	<p>该程序片上方和下方的空隙（象素个数）</p>

Hspace	该程序片左右两边的空隙（象素个数）
Align	布局方式
Java Not Supported Message	不支持Java的浏览器将显示此信息。如果你想覆盖此信息，可以在CFAPPLET NOTSUPPORTED属性中指定另外的信息。
Parameter Name	程序片所需参数的名字。
Value	对每个参数指定一个缺省值。程序片的文档中会说明各个参数的合法值。

5. 点击Create，完成注册过程

用CFAPPLET嵌入一个程序片

对一个Java程序片进行注册之后，你就可以用CFAPPLET标签在ColdFusion文件中嵌入该程序片。CFAPPLET有两个必须的属性，APPLETSOURCE和NAME。由于程序片已经注册，每个参数都已指定了缺省值，因此你可以用非常简单的形式嵌入该程序片。

```
<CFAPPLET APPLETSOURCE=" appletname"
```

```
NAME="form_variable">
```

覆盖布局设置和位置设置

要覆盖Administrator 中定义的各项选项设置，你可以使用可选的CFAPPLET参数，指定其它的选项值。例如，下面的CFAPPLET指定了定制的位置设置和布局设置。

```
<CFAPPLET APPLETSOURCE="myapplet"
```

```
NAME="applet1_var"
```

```
HEIGHT=400
```

```
WIDTH=200
```

```
VSPACE=125
```

```
HSPACE=125
```

```
ALIGN="left">
```

覆盖参数值

你也可以为Java程序片提供新的参数值，以覆盖Administrator中定义的缺省值。要注意的是，要覆盖一个参数，你必须已经在Administrator中为程序片定义了该参数并提供了一个缺省值：

```
<CFAPPLET APPLETSOURCE="myapplet"
```

```
NAME="applet1_var"
```

```
Param1="registered parameter"
```

```
Param2="registered parameter">
```

处理来自程序片的form变量

CFAPPLET标签要求你必须为程序片指定一个form变量名。这个变量可以象其它form变量一样以form.variable_name的形式被引用，其值是Java程序片的返回值。

并非所有的Java程序片都有返回值，有些程序片，如图形动画，并不返回数据。对这种程序片，Administrator中的方法（method）域保持为空。而其它的程序片都有一个会返回数据的方法。注册每个程序片时，你只能使

用一个方法。如果一个程序片具有多个方法，你可以用同一个名字为你想使用的每个方法再注册一次该程序片。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

使用CFFILE

CFFILE标签使你能以几种不同的方式进行服务器端的文件操作：

- 利用HTML表单，从客户端将文件上传到服务器。
- 移动,重命名，复制或删除服务器上的文件。
- 对服务器上的文本文件进行读、写或追加操作。

要设置的属性依赖于ACTION属性的值，例如，如果设定ACTION="Write"，则需要设置与写文本文件有关的属性。

注意：在允许用户访问服务器上的文件之前，应考虑安全性以及目录结构等问题。

文件上传

实现文件上传需要建立两个文件：

- 一个HTML表单，用于输入文件上传信息。
- 一个包含文件上传代码的cfm文件。

建立用于文件上传的HTML表单

大多数浏览器都支持文件上传。将HTML INPUT标签的类型设为“file”，则浏览器会准备从客户端读取并传输一个文件给服务器。将ENCTYPE属性设为“multipart/form-data”，则服务器会知道表单的提交包含了一个上传的文件。

例子：一个用于文件上传的HTML表单

```
<FORM ACTION="FileUpload.cfm"
```

```
ENCTYPE="multipart/form-data"
```

```
METHOD="Post">
```

```
<PRE>
```

```
File Name: <INPUT NAME="FileName" TYPE="text">
```

```
File: <INPUT NAME="FileContents" TYPE="file">
```

```
<INPUT TYPE="submit" VALUE="Upload File">
```

```
</PRE>
```

```
</FORM>
```

用户可以输入一个文件的路径，或浏览本地的文件系统并选择一个要上载的文件。注意必须将ENCTYPE属性设置为“multipart/form-data”。

建立一个文件上载的程序文件

提交一个文件后，该文件还没有保存到服务器上。它只是被译码然后与其它的表单数据一起被传递给ACTION属性所指的程序文件，在该文件中由CFFILE标签对上载的文件进行解码并保存到服务器上。

例子：上载一个文件

下面例子中的CFFILE标签可以放置在上一例子中所指的“FileUpload.cfm”文件中：

```
<CFFILE ACTION="UPLOAD"
```

```
FILEFIELD="FileContents"
```

```
DESTINATION="C:\Web\Uploads\">
```

如果请求此文件的文件上载表单传递了一个名为keyMemo.doc文件，则该文件在服务器上被保存为：

```
C:\web\Uploads\keyMemo.doc
```

要以另外的文件名保存上载的文件，可以在DESTINATION属性中指定新文件名。在下面的例子中，文件“keyMemo.doc”被保存为“UploadedFile.doc”：

```
<CFFILE ACTION="UPLOAD"
```

```
FILEFIELD="FileContents"
```

```
DESTINATION="C:\Web\Uploads\UploadedFile.doc">
```

你也可以将这些属性设置为动态参数。例如，你可以根据数据库查询的结果来设定文件名。

注意：FILEFIELD属性应设定为一个Form字段的名称，而不是Form字段的值，因此不能将字段名括在#号中。

处理文件名冲突

将一个文件保存到服务器上时，有可能已经有另一个同名的文件存在，在这种情况下，利用NAMECONFLICT属性，你可以采取多种处理方法。

例子：解决文件名冲突

在下面的例子中，如果文件上传时发生名字冲突，程序会生成一个唯一的文件名，文件后缀名保持不变：

```
<CFFILE ACTION="Upload"
```

```
FILEFIELD="FileContents"
```

```
DESTINATION="C:\Web\Uploads\"
```

```
NAMECONFLICT="MAKEUNIQUE">
```

控制上传文件的类型

对某些应用程序，你也许需要限制上传文件的类型。例如，在一个文档库中，你也许不想接受图形文件。

ACCEPT属性用于限制允许上传的文件类型。指定了ACCEPT属性后，被上传文件的MIME内容类型必须与指定的条件相匹配，否则就会产生错误。ACCEPT属性的值是一列由逗号隔开的MIME数据类型名，其中可以使用通配符。

一个文件的MIME类型决定于浏览器，普通的类型如“image/gif”和“text/plain”等已在浏览器中注册。

例子：限制文件类型

这个CFFILE 只保存GIF格式的图象文件：

```
<CFFILE ACTION="Upload"
```

```
FILEFIELD="UploadFile"
```

```
DESTINATION="c:\uploads\MyImage.GIF"
```

```
NAMECONFLICT="OVERWRITE"
```

```
ACCEPT="image/gif">
```

这个CFFILE 只保存GIF格式或JPEG格式的图象文件：

```
<CFFILE ACTION="Upload"
```

```
FILEFIELD="UploadFile"
```

```
DESTINATION="c:\uploads\MyImage.GIF"
```

```
NAMECONFLICT="OVERWRITE"
```

```
ACCEPT="image/gif, image/jpeg">
```

这个CFFILE只保存图象文件，但对其格式没有限制：

```
<CFFILE ACTION="Upload"
```

```
FILEFIELD="UploadFile"
```

```
DESTINATION="c:\uploads\MyImage.GIF"
```

```
NAMECONFLICT="OVERWRITE"
```

```
ACCEPT="image/*">
```

注意：如果ACCEPT属性被忽略或设为空或设为“*/*” ，则所有类型的文件都可以被上载。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

设置文件和目录的属性

在Windows中，用CFFILE的ATTRIBUTES属性定义文件属性；在UNIX中，用CFFILE和CFDIRECTORYMODE属性定义文件和目录权限。

UNIX

在UNIX中，你可以对文件和目录设置权限。MODE属性的值相对应于UNIX命令chmod的十进制数值。

- 4=读
- 2=读/写
- 1=读/写/执行

你应按照“所有者，组，其它用户”的顺序，为每种类型的用户设置权限值。例如，为所有用户设置读权限：

MODE=444

Windows

在Windows中，你可以设置下面的文件属性：

?????????• ReadOnly

?????????• Temporary

?????????• Archive

?????????• ?Hidden

?????????• ?System

?????????• ?Normal

如果不使用ATTRIBUTES属性，则文件现有的属性被保留。如果同时指定了Normal属性和任何其它属性，则Normal属性被其它属性所覆盖。

例子：设置文件属性

这个例子将上载的文件设为Archive属性：

```
<CFFILE ACTION="copy"
```

```
SOURCE="c:\files\upload\keymemo.doc"
```

```
DESTINATION="c:\files\backup\"
```

```
ATTRIBUTES="Archive">
```

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

评价文件上传的结果

完成文件上传之后，你可以利用文件上传变量获取状态信息。状态信息包括与上传文件有关的各种数据，如文件名和保存的路径等。

文件上传状态变量的推荐用法是使用CFFILE前缀，例如CFFILE.ClientDirectory。为了向后兼容，File前缀仍可以使用。在任何可以使用ColdFusion变量的地方都可以使用文件状态变量。

完成文件上传之后，下表中的各个状态变量即可以被引用：

文件上传变量	
参数	描述
AttemptedServerFile	ColdFusion最初用来存储被上传文件的文件名，如myfile.txt。
ClientDirectory	被上传文件在客户系统中的目录位置。

ClientFile	被上传文件在客户系统中的文件名，如myfile.txt。
ClientFileExt	被上传文件在客户系统中的扩展名，不包括句号,如txt（而不是.txt）。
ClientFileName	被上传文件在客户系统中的文件名，不包括后缀名。
ContentSubType	被上传文件的MIME内容子类型，例如对image/gif即为gif。
ContentType	被上传文件的MIME内容类型，例如对image/gif即为image。
DateLastAccessed	被上传文件最后一次被访问的时间。
FileExisted	指明是否（yes或no）相同路径的文件已经存在。
FileSize	被上传文件的大小。
FileWasAppended	指明是否（yes/no）要ColdFusion将被上传文件追加到一个已经存在的文件中。

FileWasOverwritten	指明是否 (yes/no) 要 ColdFusion 覆盖一个文件。
FileWasRenamed	指明被上载文件是否 (yes/no) 因文件名冲突而被重命名。
FileWasSaved	指明 ColdFusion 是否保存了一个文件。
OldFileSize	在文件上载操作中被覆盖的文件的大小。
ServerDirectory	服务器上实际保存了被上载文件的目录。
ServerFile	实际保存到服务器上的文件名。
ServerFileExt	被上载文件在服务器上的文件后缀名，不包括句号，如txt。
ServerFileName	被上载文件在服务器上的文件名，不包括后缀名。
TimeCreated	被上载文件的创建时间。
TimeLastModified	被上载文件最后一次被修改的日期和时间。

引用这些变量时可以使用File前缀或CFFILE前缀，如#File.FileExisted#。

注意：文件状态变量是只读的。这些变量的值是最近一次CFFILE操作的执行结果，如果执行了两个CFFILE标签，则第一个标签的执行结果被第二个所覆盖。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

移动、重合名、复制和删除服务器端文件

利用CFFILE，你可以对服务器上的文件进行移动、重命名、复制和删除等文件操作，在下面的这些例子中，许多属性值均被设置为静态值，但是，CFFILE标签的所有属性的值均可包含动态参数，这使得CFFILE的功能非常强大。

移动一个文件（ACTION=“MOVE”）

用CFFILE可以将服务器上的文件从一个位置移动到另一个位置。如下例，文件“KeyMemo.doc”被从C:\files\upload\目录移动到C:\files\memo\目录：

```
<CFFILE ACTION="MOVE"
```

```
SOURCE="c:\files\upload\KeyMemo.doc"
```

```
DESTINATION="c:\files\memo\">
```

重命名一个文件（ACTION=“PENAME”）

用CFFILE可以将服务器上的文件重命名。如下例，文件KeyMemo.doc被重

命名为oldmemo.doc :

```
<CFFILE ACTION="Rename"
```

```
SOURCE="c:\files\memo\KeyMemo.doc"
```

```
DESTINATION="c:\files\memo\OldMemo.doc">
```

复制一个文件 (ACTION="COPY")

用CFFILE可以将服务器上的文件从一个目录复制到另一个目录。下面的例子将文件KeyMemo.doc的一个备份保存到C:\files\backup\目录：

```
<CFFILE ACTION="Copy"
```

```
SOURCE="c:\files\upload\KeyMemo.doc"
```

```
DESTINATION="c:\files\backup\">
```

删除一个文件 (ACTION="DELETE")

用CFFILE可以删除服务器上的文件，下面的例子将指定的文件永久性删除：

```
<CFFILE ACTION="Delete"
```

```
FILE="c:\files\upload\uploadfile.txt">
```

[后页](#)

[前页](#)

[目录](#)

[北极星书库](#)

读、写和追加一个文本文件

用CFFILE不仅可以操作服务器上的文件，还可以对文本文件进行读取，创建和修改。这使你能够：

- 建立日志文件
- 生成静态HTML文档
- 用文本文件存储可以添加到网页中的信息

读文本文件（ACTION=“READ”）

你可以用CFFILE来读取一个已经存在的文本文件。文件被读取到一个动态参数中，你可以在程序文件的任何位置使用该动态参数。例如，你可以读取一个文本文件，然后将其内容插入到数据库中，你还可以读取文件然后使用查找和替换函数修改其内容。

下面的例子创建一个变量“Message”，用该变量保存文件“message.txt”的内容：

```
<CFFILE ACTION="Read"
```

```
FILE="C:\Web\message.txt"
```

```
VARIABLE="Message">
```

于是在应用程序文件中即可以使用变量“Message”。例如，你可以在网页中显示文件message.txt的内容：

```
<CFOUTPUT>#Message#</CFOUTPUT>
```

写文本文件（ACTION=“WRITE”）

你可以用CFFILE根据动态内容写一个文本文件。例如，你可以建立一个HTML文件或日志文件。

下面的例子用用户输入一个HTML插入表单中的信息建立一个文件：

```
<CFFILE ACTION="Write"
```

```
FILE="C:\files\updates\#Form.UpdateTitle#.txt"
```

OUTPUT="Created By: #Form.FullName#

Date: #Form.Date#

#Form.Content# ">

如果用户输入如下的信息并提交表单：

UpdateTitle="FieldWork"

FullName="John Lunch"

Date="10/1/98"

Content="We had a wonderful time in Cambridgeport."

则ColdFusion会在C:\files\updates\目录下建立一个文件Fieldwork.txt，该文件中包含如下文本：

Created By: John Lunch

Date: 10/1/98

We had a wonderful time in Cambridgeport.

追加一个文本文件（ACTION="APPEND"）

用CFFILE可以向一个已经存在的文本文件中追加文本，如日志文件。下面的例子向文件Fieldwork.txt中追加一个字符串“**But Davis Square was more fun.**”：

```
<CFFILE ACTION="Append"
```

```
DESTINATION="C:\files\updates\FieldWork.txt"
```

```
OUTPUT="<B>But Davis Square was more fun.</B>">
```

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

执行目录操作

CFDIRECTORY标签用于返回指定目录的文件信息，并可以对目录进行创建，删除和重命名等操作。

与CFFILE一样，在ColdFusion Administrator中可以禁止CFDIRECTORY标签。CFDIRECTORY的各属性如下表所列：

CFDIRECTORY 属性	
属性	描述
ACTION	可选。定义要对DIRECTORY属性中指定的目录所执行的操作。合法的输入包括： <ul style="list-style-type: none">• List• Create• Delete• Rename 缺省值为List.
DIRECTORY	要对之执行目录操作的目录名。

NAME	<p>目录列表输出的查询名。 ACTION="List"时要指定此属性，其他操作忽略此属性。</p>
FILTER	<p>用于过滤返回的文件名，如"*.*.cfm"。 ACTION="List"时可指定此属性，其他操作忽略此属性。</p>
MODE	<p>可选。只在Solaris中，在ACTION="Create"时使用，用于设置一个目录的许可权限。在Windows中将被忽略。</p> <p>其合法取值对应于UNIX的chmod命令的十进制值。许可权限按照“所有者，组，其它”的顺序分别设置。例如：</p> <p>MODE=644</p> <p>赋予所有者读/写权限，组和其它用户被赋予读权限。</p>
SORT	<p>ACTION="List"时可选，对其它操作被忽略。可用此属性指定一系列字段名，相互间用逗号隔开，根据这些字段对输出的目录列表进行排序。对每个字段可指定ASC或DESC，按升序或降序排序。例如：</p>

	SORT="colname ASC,filename2 DESC,size,datelastmodified"
NEWDIRECTORY	DIRECTORY属性中指定的目录的新目录名。 ACTION="Rename"时要指定此属性，其它操作忽略该属性。

返回文件信息（ACTION="LIST"）

当ACTION="LIST"时，CFDIRECTORY返回五个变量，你可以在CFOUTPUT中引用这些变量：

- Name——目录名或文件名。
- Size——目录大小或文件大小。
- Type——文件类型：F代表文件，D代表目录。
- DateLastModified——最后一次被修改的时间。

- Attributes——文件属性。
- Mode——（仅Solaris支持）代表指定目录的权限设置的十进制数。

例子

你可以在CFML扩展中使用查询结果字段，只需以查询名作为字段名的前缀。

```
<CFDIRECTORY
```

```
DIRECTORY="c:\winnt\system32"
```

```
NAME="mydirectory"
```

```
SORT="size ASC, name DESC, datelastmodified">
```

```
<CFOUTPUT QUERY="mydirectory">
```

```
Name: #mydirectory.name# <BR>
```

Size: #mydirectory.size#
Type: #mydirectory.type#

Date last modified: #mydirectory.datelastmodified#

Attributes: #mydirectory.attributes#

Mode: #mydirectory.mode#

</CFOUTPUT>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

建立连接

CFFTP标签使你能够通过文件传输协议 (ftp) 与远程服务器相交互。这是服务器与服务器之间的文件操作。要实现服务器与浏览器之间的文件操作，应使用CFFILE，CFCONTENT和CFDIRECTORY标签。

建立连接

CFFTP的使用涉及两种类型的操作，建立连接和传输文件。

注意：CFFTP是一个COM对象，Microsoft Windows NT 3.51不支持此对象。

例子：FTP登录

下面的例子演示了一个打开的FTP连接，并提供了用户名和密码。

```
<CFFTP CONNECTION=FTP
```

```
    USERNAME="betauser"
```

PASSWORD="monroe"

SERVER="beta.company.com"

ACTION="Open"

STOPONERROR="No">

CFFTP标签要求指定USERNAME和PASSWORD属性，为此当建立匿名连接时，要以“anonymous”作为用户名，以一个email地址作为密码。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

文件和目录操作

建立连接之后，你就可以与所连接的FTP服务器进行交互，执行文件和目录操作。实现文件和目录操作的CFFTP标签的各属性如下表所列：

CFFPT文件和目录操作属性	
属性	描述
ACTION	<p>如果连接没有被缓冲，则需要此属性，否则不需要。</p> <p>指明要执行的FTP操作，这些操作可以是下列之一：</p> <ul style="list-style-type: none">• ChangeDir• CreateDir• ListDir• GetFile• PutFile• Rename• Remove

	<ul style="list-style-type: none"> • GetCurrentDir • GetCurrentURL • ExistsDir • ExistsFile • Exists <p>注意：文件名和目录名是区分大小写的。例如，对test.log执行ListDir操作不会找到test.LOG。</p>
USERNAME	用于建立FTP连接的用户名。如果连接已经被缓冲，则不需要此属性。
PASSWORD	用于建立连接的密码。如果连接已经被缓冲，则不需要此属性。
NAME	当ACTION="ListDir"时需要此属性，用于指定用来保存目录列表的查询名。
SERVER	要与之建立连接的服务器名。如果连接已经被缓冲，则不需要此属性。
TIMEOUT	可选。用于指定所有操作的超时秒数。缺省值是30秒。
PORT	可选。用于指定远程服务器的端口号。

CONNECTION	可选。指定FTP连接的名字，用于缓冲当前的FTP连接或重用已缓冲的FTP连接。所有使用相同连接名的CFFTP都使用相同的连接信息。
ASCIIEXTENSIONLIST	可选。一系列用分号隔开的文件扩展名，用于在TRANSFERMODE="Autodetect"时强制进行ASCII传输模式。缺省的扩展名列表是： txt;htm;html;cfm;cfml;shtm;shtml;css;asp;asa
TRANSFERMODE	可选。指定你要用的FTP传输模式。合法的输入有ASCII,Binary,或Autodetect。缺省值是Autodect.
AGENTNAME	可选。管理传输的应用程序。
FAILIFEXISTS	可选。其值为Yes或NO，缺省为Yes。用于指明当同名的文件已经存在时，GetFile操作是否失败。
DIRECTORY	当ACTION=ChangeDir,CreateDir,ListDir,ExistsDir时需要此属性。用于指定要对之进行操作的目录。
LOCALFILE	当ACTION=GetFile,PutFile时需要此属性。用于指定用户本地文件系统中的文件名。
REMOTEFILE	当ACTION=GetFile,PutFile,ExistsFile时需要此属性。用于指定FTP服务器上文件系统中的文件名。

ATTRIBUTES	<p>可选。一系列用逗号隔开的文件属性。缺省值为"Normal"。用于在GetFile操作中指定本地文件的文件属性。其值可以是下列值的任意组合：</p> <ul style="list-style-type: none"> • ReadOnly • Hidden • System • Archive • Directory • Compressed • Temporary • Normal
ITEM	<p>当ACTION=Exists,Remove时需要此属性。用于指定这些操作的对象，文件或目录。</p>
EXISTING	<p>当ACTION=Rename时需要此属性。指定远程服务器上文件或目录的当前名字。</p>
NEW	<p>当ACTION=Rename时需要此属性。指定远程服务器上文件或目录的新名字。</p>
RETRYCOUNT	<p>可选。报错之前的重试次数。缺省值是1。</p>
STOPONERROR	<p>可选。其值为Yes或No，缺省值为NO。当为</p>

Yes时，停止所有的处理并显示合适的错误信息；当为No时，产生三个变量：

- CFFTP.Succeeded---Yes 或NO.
- CFFTP.ErrorCode---错误号。
- CFFTP.ErrorText---解释错误情况的错误信息。

连接缓存

一旦用CFFTP建立了一个连接，你可以再次使用该连接，执行附加的FTP操作。要实现这一点，在第一次建立连接时要用CONNECTION属性定义并命名一个FTP连接对象。此对象用于保存与连接有关的信息，任何附加的FTP操作只要使用相同的CONNECTION名字，即可自动使用存储在连接对象中的信息。这项特性节省了连接和登录到FTP服务器的时间，从而提高了文件传输操作的效率。

当你使用一个FTP连接时，你不需要重新指定下列属性：

- USERNAME
- PASSWORD
- SERVER

在这种情况下，如果使用了框架（frame），则你应该确保当多个请求同时到达时，只有一个框架将使用连接对象。

跨多个文件的连接缓存

CFFTP缓存只能维持在当前的网页中，除非你显式地将一个CFFTP连接分配给一个应用程序范围或session范围的变量。将一个CFFTP连接分配给一个Application变量对象有可能会出现问题，因为有可能会有多个用户同时访问同一个连接对象。为CFFTP连接创建一个Session变量是比较合适的。

将FTP连接名分配给一个session变量即可实现session范围内的连接缓存。如下例所示：

```
<CFFTP ACTION=connect
```

```
    USERNAME="anonymous"
```

```
    PASSWORD="me@home.com"
```

```
    SERVER="ftp.eclipse.com"
```

```
    CONNECTION="Session.myconnection">
```

在这个例子中，连接缓存在当前Session的所有程序文件中有效。当然，你首先应确保你的应用程序允许了Session变量。

注意：要修改一个缓存的连接，例如改变RETRYCOUNT或TIMEOUT的值，则必须重新建立该连接。

连接缓存的操作和属性

下表显示了使用连接缓存进行各种操作所需的CFFTP属性，如果没有使用连接缓存，则还需指定USERNAME，PASSWORD和SERVER属性。

CFFTP各操作所需的属性			
操作	属性	操作	属性
Open	无	Rename	EXISTING NEW
Close	无	Remove	SERVER ITEM
ChangeDir	DIRECTORY	GetCurrentDir	无
CreateDir	DIRECTORY	GetCurrentURL	无
ListDir	NAME DIRECTORY	ExistsDir	DIRECTORY
GetFile	LOCALFILE	ExistsFile	REMOTEFILE

	REMOTEFILE		
PutFile	LOCALFILE REMOTEFILE	Exists	ITEM

例子：一个FTP Session

下面的例子打开一个FTP连接，获取文件列表，显示文件或目录的名字，路径，URL，长度，以及修改日期。例中使用了连接缓存，并允许了自动检测错误。

```
<--- open FTP connection --->
```

```
<CFFTP CONNECTION=FTP
```

```
USERNAME="betauser"
```

```
PASSWORD="monroe"
```

```
SERVER="beta.company.com"
```

```
ACTION="Open"
```

STOPONERROR="Yes">

<--- get current directory name --->

<CFFTP CONNECTION=FTP

ACTION="GetCurrentDir"

STOPONERROR="Yes">

<--- output directory name --->

<CFOUTPUT>

FTP directory listing of #cftp.returnvalue#.<p>

</CFOUTPUT>

<--- get directory info --->

<CFFTP CONNECTION=FTP

ACTION="listdir"

DIRECTORY="/*."

NAME="q"

STOPONERROR="Yes">

<--- output dir - list results --->

<HR>

<P>FTP Directory Listing:</P>

<CFTABLE QUERY="q" HTMLTABLE>

<CFCOL HEADER="Name" TEXT="#name#">

<CFCOL HEADER="Path" TEXT="#path#">

<CFCOL HEADER="URL" TEXT="#url#">

<CFCOL HEADER="Length" TEXT="#length#">

<CFCOL HEADER="LastModified"

TEXT="Date(Format#lastmodified#)">

<CFCOL HEADER="IsDirectory"

TEXT="#isdirectory#">

</CFTABLE>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

CFFTP变量

CFFTP操作返回下列变量：

- CFFTPResult.ReturnValue
- 当ACTION属性为“ListDir”时，返回CFFTP查询对象。
- 当STOPONERROR属性值为“NO”时，创建三个变量。

CFFTPResult.ReturnValue

变量CFFTPResult.ReturnValue的值决定于CFFTP的ACTION属性。

CFFTPResult.ReturnValue	
CFFTP ACTION	CFFTPResult.ReturnValue变量的值
GetCurrentDir	当前目录的字符串值
GetCurrentURL	当前URL的字符串值
ExistsDir	Yes或No
ExistsFile	Yes或No
Exists	Yes或No

STOPONERROR变量

当STOPONERROR属性被设置为“NO”时，将创建如下的变量：

- CFFTP.Succeeded——Yes或NO
- CFFTP.ErrorCode——错误码（参见下面的CFFTP.Errorcode值）
- CFFTP.ErrorText——解释错误情况的文本信息。

下表列出了CFFTP.ErrorCode变量的可能值：

CFFTP.ErrorCode变量的值	
错误码	描述
0	操作成功
1	系统错误
2	无法建立Internet会话
3	无法打开FTP会话
4	无法识别文件传输模式
5	无法建立搜索连接

6	被调用的操作只有在搜索时才合法
7	非法的超时值
8	非法的端口号
9	没有足够的内存
10	本地文件不可读取
11	本地文件不可写入
12	无法打开远程文件
13	无法读取远程文件
14	无法打开本地文件
15	不能写入远程文件
16	未知错误
17	保留
18	文件已经存在

19	保留
20	保留
21	非法的重试次数

CFFTP查询对象的属性

当你使用CFFTP的ListDir操作时，你必须指定NAME属性的值。NAME属性的值用于将ListDir操作的查询结果保存在一个查询对象中。这个查询对象的各个字段可以用如下的格式来引用：

queryname Columnname[row]

其中queryname即是在NAME属性中指定的查询名，Columnname是返回到查询对象中的各字段之一，Row是由ListDir操作所返回的每个文件/目录项的行号，每个文件或子目录都占用单独的一行。

下表列出了查询对象中的各个字段。

CFFTP查询对象字段	
字段	描述
Name	当前元素的文件名
Path	当前元素的文件路径（不包括盘符）

URL	当前元素（文件或目录）的完整URL
Length	当前元素的文件大小
LastModified	当前元素的未格式化的日期/时间
Attributes	当前元素的属性
Isdirectory	布尔值，指明当前元素是文件还是目录

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

使用CFHTTP与Web相交互

CFHTTP标签是功能最强大的CFML标签之一，它可以用来：

- 从网络上获取网页和查询结果。
- 向远程服务器，CFML模板或CGI应用程序传送数据并处理返回数据。

超文本传输协议（HTTP）处理Web服务器之间的各种通信。执行一个事务处理所采取的方法决定了服务器处理什么以及什么内容被传送给浏览器。除了HTML文档之外，HTTP还支持二进制文件，图象、音频和视频等内容类型。HTTP协议标准规定，用户的请求信息和服务器的响应信息，以及事务处理的状态信息均以请求标题和响应标题的形式被传递。HTTP协议有几种通信方法，用于在客户和服务器之间交换信息。CFHTTP标签实现了最常用的HTTP通信方法：GET和POST。CFHTTP的语法非常简单，但它提供了许多选项，用于指定输出，处理链接以及建立查询等任务。

CFHTTP的语法

CFHTTP 标签的基本语法如下：

<CFHTTP URL=" hostname"

USERNAME=" username"

PASSWORD=" password"

NAME=" queryname"

COLUMNS=" query_columns"

PATH=" path"

FILE=" filename"

METHOD=" get_or_post"

DELIMITER=" character"

TEXTQUALIFIER=" character"

RESOLVEURL=" Yes/No"

PROXYSERVER=" hostname">

</CFHTTP>

处理所获网页中的链接

要保持通过CFHTTP从Web服务器获取的网页中的相对链接，应将RESOLVEURL属性设为“Yes”。当RESOLVEURL=YES时，下列可能包含链接的标签会被处理：

- IMG SRC
- A HREF

- RORM ACTION
- APPLET CODE
- SCRIPT SRC
- EMBED SRC
- EMBED PLUGINSPLACE
- BODY BACKGROUND
- FRAME SRC
- BGSOUND SRC
- OBJECT DATA
- OBJECT CLASSID

- OBJECT CODEBASE

- OBJECT USEMAP

注意：如果获取的网页中某个URL指向一个二进制文件，则该网页不被显示。你可以输入一个路径作为PATH属性的值，从而把该二进制文件保存到本地计算机中。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

使用CFHTTP的Get方法

Get方法于从指定的服务器获取文本文件或二进制文件。下面的几个例子演示了几种常用的Get操作。Get方法是一种单向的事务处理方法，CFHTTP用这种方法获得一个对象，而Post方法是双向的处理方法，CFHTTP用Post方法向ColdFusion文件或CGI程序传递变量，并接收和处理返回的数据。

例子：获取到变量中

下面的例子用Get方法请求Yahoo站点的缺省主页，并将文件内容保存到一个变量中，然后用CFOUTPUT将该变量的内容显示在浏览器中。

```
<CFHTTP METHOD="Get"
```

```
URL="http://www.yahoo.com/index.htm"
```

```
RESOLVEURL="Yes">
```

```
<CFOUTPUT>
```

```
#CFHTTP.FileContent# <BR>
```

</CFOUTPUT>

当FILE属性和PATH属性被忽略时，ColdFusion将文件index.htm的内容保存到变量CFHTTP.FileContent中。要注意的是，若不把RESOLVEURL属性设为“yes”，则下载的网页中相对链接会被打断。若将该属性设为“yes”，则相对链接会被转换为绝对链接。

例子：获取到一个文件中

下面的例子也用Get方法执行一个简单的文件请求，但PATH属性和FILE属性的使用导致下载的文件被保存到本地的文件中。

```
<CFHTTP METHOD = "get"
```

```
URL="http://www.yahoo.com/index.htm"
```

```
PATH="c:\temp"
```

```
FILE="yahooindex.htm">
```

注意，当使用了PATH属性和FILE属性时，RESOLVEURL属性会被忽略（即使指定了该属性，也将被忽略）。

例子：获取一个二进制文件

与上一个例子相似，这个例子也从服务器下载一个文件并保存到PATH属性和FILE属性所指定的位置。唯一的不同是被下载文件的MIME类型不一样。

```
<CFHTTP METHOD="Get"
```

```
URL="http://maximus/downloads/quakestuff/q2_test.zip"
```

```
PATH="c:\quake2\install"
```

```
FILE="quake2beta.zip">
```

```
<CFOUTPUT>
```

```
#CFHTTP.MimeType#
```

```
</CFOUTPUT>
```

CFOUTPUT代码块显示了被下载文件的MIME类型，在此例中是

application/zip。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

根据一个文本文件建立查询

使用CFHTTP的Get方法，你可以根据一个有分隔符的文本文件创建查询对象。这是处理生成的文本文件的一个有力方法。一旦查询对象创建之后，对查询字段的引用和处理是非常简单的。

文本文件的处理方式如下：

- 用DELIMITER属性指定一个分隔符。如果某个字段的数据包含分隔符本身，则必须将其括在引号中或括在某个其它字符中，你可以用TEXTQUALIFIER属性指定该字符。
- 缺省情况下，文本文件的第一行被解释为字段标题文本。你可以在COLUMNS属性中指定替代的标题文本。如果这样，你应确保为文本文件的每一列都指定了替代的标题文本。
- 当遇到重复的字段标题时，ColdFusion为重复的字段名添加一个下划线字符。例如，如果发现两个字段标题都是CustomerID，则第二个字段标题被更名为“CustomerID_”。

例子：根据一个文本文件创建查询

这个例子根据一个以逗号为分隔符的文本文件创建一个查询对象。该文本文件有六列数据，互相以逗号分隔，文件的第一行为：

```
OrderID,OrderNum,OrderDate,ShipDate,ShipName,ShipAddress
```

这个例子接受文本文件的第一行作为字段名：

```
<CFHTTP METHOD="Get"
```

```
URL="http://127.0.0.1/orders/june/orders.txt"
```

```
NAME="juneorders"
```

```
DELIMITER=","
```

```
TEXTQUALIFIER=""""">
```

```
<CFOUTPUT QUERY="juneorders">
```

```
OrderID: #OrderID#<BR>
```

Order Number: #OrderNum#

Order Date: #OrderDate#

</CFOUTPUT>

你可用COLUMNS属性指定不同与上例的字段名：

<CFHTTP METHOD="Get"

URL="http://127.0.0.1/orders/june/orders.txt"

NAME="juneorders"

COLUMNS="ID, Number,Date"

DELIMITER=","

TEXTQUALIFIER="''''''>

<CFOUTPUT QUERY="juneorders">

Order ID: #ID#

Order Number: #Number#

Order Date: #Date#

</CFOUTPUT>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

使用CFHTTP的Post方法

Post方法用于向指定的ColdFusion文件或CGI程序发送Cookie变量，form变量，CGI变量，URL变量以及文件变量。使用Post方法时，必须对每个要发送的变量使用CFHTTPPARAM标签。与Get方法不同，Post方法将数据传送给ColdFusion文件或其它可执行文件，该文件对传送的变量进行处理并有数据返回。

例如，当你建立一个使用Post方法的HTML表单时，你要指定将form数据传递给哪个程序文件。在CFHTTP中使用Post方法与此相同。

例子：将变量传递给ColdFusion文件

下面的例子向URL属性所指定的文件传递五种类型的变量。该文件返回各变量的值并显示在浏览器中。这个例子在接受Post数据的文件中用CFFILE标签将文件变量的内容上载到C:\temp\junk目录下。

注意：文件Posttest.cfm中的CFOUTPUT引用了变量CFHTTP.Filecontent，用于显示Server.cfm文件的输出。如果不使用CFHTTP.Filecontent变量，浏览器的输出显示将只限于Posttest.cfm的内容。

例子文件: posttest.cfm

<CFHTTP METHOD="Post"

URL="http://housebeat/cfdocs/server.cfm"

USERNAME="user1"

PASSWORD="user1pwd">

<CFHTTPPARAM TYPE="Cookie"

VALUE="cookiemonster"

NAME="mycookie6">

<CFHTTPPARAM TYPE="CGI"

VALUE="cgivar "

NAME="mycgi">

<CFHTTPPARAM TYPE="URL"

VALUE="theurl"

NAME="myurl">

<CFHTTPPARAM TYPE="Formfield"

VALUE="wbfreuh@allaire.com"

NAME="emailaddress">

<CFHTTPPARAM TYPE="File"

NAME="myfile"

FILE="c:\temp\cyberlogo.gif">

</CFHTTP>

<CFOUTPUT>

#CFHTTP.filecontent#

#CFHTTP.mimetype#

</CFOUTPUT>

例子文件： server.cfm

You have POSTed to me.

<CFFILE DESTINATION="c:\temp\junk"

NAMECONFLICT="Overwrite"

FILEFIELD="myfile"

ACTION="Upload"

ATTRIBUTES="Normal">

<CFOUTPUT>

The URL variable is: #url.myurl#

The Cookie variable is: #cookie.mycookie6#

The CGI variable is: #cgi.mycgi#.

The Formfield variable is: #form.myformfield#.

</CFOUTPUT>

例子：CGI程序的返回结果

下面的例子运行一个CGI程序Search.exe，该程序搜索一个站点，返回VALUE属性所指定的值的命中次数。

<CFHTTP METHOD="Post"

URL="http://www.that site.com/search.exe"

RESOLVEURL="Yes">

```
<CFHTTPPARAM TYPE="Formfield"
```

```
NAME="search"
```

```
VALUE="hello">
```

```
</CFHTTP>
```

```
<CFOUTPUT>
```

```
#CFHTTP.MimeType#<BR>
```

```
Length: #len(cfhttp.filecontent)# <BR>
```

```
Content: #htmlcodeformat(cfhttp.filecontent)#<BR>
```

```
</CFOUTPUT>
```

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

在ColdFusion中使用邮件服务器

利用CFMAIL标签和CFPOP标签，你可以为你的应用程序增添交互式的email特性，提供与邮件服务器的双向接口。

在ColdFusion中使用邮件服务器

在应用程序中加入email功能使你能对用户的请求自动作出响应。在应用程序中你可以通过许多方式使用email。例如；

- 根据用户的请求触发email信息
- 允许用户通过email请求或接收附加信息
- 根据订购情况确认客户信息
- 根据数据库查询结果发送支票或余额信息

ColdFusion提供了几种方式来将email集成到应用程序中。要发送email，通常应使用“简单邮件传输协议”（SMTP），要接收email，通常应使用“邮局

协议” (POP)。要在应用程序中使用email，你必须可以访问一个SMTP服务器，还要有一个合法的POP帐号和密码。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

发送email信息 (SMTP)

发送email信息之前，你必须可以访问一个SMTP邮件服务器。另外，在运行需要使用SMTP服务器的应用程序文件之前，你必须配置ColdFusion Administrator，指定使用哪个SMTP服务器。配置ColdFusion的email设置步骤如下：

- 1、在ColdFusion Administrator中打开Mail页。
- 2、在Mail Server框中，输入你要使用的SMTP服务器的地址
- 3、服务器端口设置和连接超时设置保持其缺省值
- 4、点击Apply，保存以上的设置
- 5、点击Verify按钮，确证ColdFusion可以访问你所设定的邮件服务器。

用CFMAIL发送SMTP邮件

CFMAIL标签支持从ColdFusion应用程序内发送SMTP（简单邮件传输协议）邮件。CFMAIL标签与CFOUTPUT标签相似，只不过CFMAIL将文本

输出为SMTP邮件，而不是输出到屏幕上。在CFOUTPUT中使用的所有属性都可以用于CFMAIL。

下面的例子是一个简单的CFMAIL标签，其中用form变量构造邮件信息的FROM，TO，Subject和信息体等部分。

```
<CFMAIL QUERY="GetList"
```

```
TO="#Email#"
```

```
FROM="Sales DepartmentList Manager"
```

```
SUBJECT="#Form.Subject#">
```

```
#FORM.Body#
```

```
</CFMAIL>
```

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

CFMAIL示例

包含CFMAIL标签的文件可以根据该标签的设置动态地生成email信息，你可以利用CFMAIL实现的功能有：

发送一个email，其接收者及其内容取决于用户输入到一个HTML表单中的数据。

根据查询结果将email发送给一批收信者。

根据查询结果发送定制的email信息，如帐单信息，其收信者动态地由数据库查询决定。

与邮件信息一起发送一个MIME文件附件。

CFMAIL与CFOUTPUT很相似，两者有一些相同的属性——QUERY，GROUP，STARTROW和MAXROWS等。而CFMAIL还有一些与邮件有关的特殊属性，用于处理邮件标题信息以及MIME文件和HTML文件等。

发送基于form的email

在下面的例子中，用户在表单中输入的咨询信息被发送到市场部：

```
<CFMAIL FROM="#Form.EmailAddress#"
```

```
TO="marketing@allaire.com"
```

```
SUBJECT="Customer Inquiry">
```

A customer inquiry was posted to our Web site:

```
Name: #Form.FirstName# #Form.LastName#
```

```
Subject: #Form.Subject#
```

```
#Form.InquiryText#
```

```
</CFMAIL>
```

发送基于查询的email

在下面的例子中，查询ProductRequest的任务是获取过去七天中对某个产品

进行咨询的客户列表。然后这个列表被加上合适的标题和脚注，发送给市场部：

```
<CFMAIL QUERY="ProductRequests"
```

```
FROM="webmaster@allaire.com"
```

```
TO="marketing@allaire.com"
```

```
SUBJECT="ColdFusion status report">
```

Here is a list of people who have inquired about

Allaire ColdFusion over the last seven days:

```
<CFOUTPUT>
```

```
#ProductRequests.FirstName# #ProductRequests.LastName#
```

```
(#ProductRequests.Company#) - #ProductRequests.EMailAddress#
```

</CFOUTPUT>

Regards,The WebMaster

webmaster@allaire.com

</CFMAIL>

注意在标准的CFMAIL信息中嵌入了一个CFOUTPUT标签，用于提供动态内容。CFOUTPUT内的语句将对查询“ProductRequest”的查询结果中每一条记录重复执行，而CFOUTPUT上面和下面的部分别作为邮件的标题和脚注信息。

向多个收信者发送email

在下面的例子中，查询CFBetaTesters用于获取一个正在使用ColdFusion β 测试版的用户列表。然后根据此查询的结果向每个用户发送一个通知。告知他们β 测试版已有一个新版本发布：

```
<CFMAIL QUERY="CFBetaTesters"
```

```
FROM="beta@allaire.com"
```

```
TO="#TesterEMail#"
```

SUBJECT="ColdFusion Beta Four Available">

To all ColdFusion beta testers:

ColdFusion Beta Four is now available for downloading

from the Allaire site. The URL for the download is:

<http://beta.allaire.com>

Regards,

ColdFusion Technical Support

beta@allaire.com

</CFMAIL>

在这个例子中，email信息不包含动态内容。但此信息的目标email地址是一个动态参数。注意：TO属性中“TesterEmail”字段来自查询结果。

后页
前页
目录
北极星书库

为多个收信者定制Email

在下面的例子中，查询“GetCustomers”用于获取一批用户的联系信息。然后查询结果被用来向每个用户发送一个email，要求他们证实其联系信息是否仍然有效：

```
<CFMAIL QUERY="GetCustomers"
```

```
FROM="service@allaire.com"
```

```
TO="#EMail#"
```

```
SUBJECT="Contact Info Verification">
```

```
Dear #FirstName# -
```

```
We'd like to verify that our customer
```

```
database has the most up-to-date contact
```

information for your firm. Our current

information is as follows:

Company Name: #Company#

Contact: #FirstName# #LastName#

Address:

#Address1#

#Address2#

#City#, #State# #Zip#

Phone: #Phone#

Fax: #Fax#

Home Page: #HomePageURL#

Please let us know if any of the above

information has changed, or if we need to

get in touch with someone else in your

organization regarding this request.

Thanks,

Allaire Customer Service

service@allaire.com

</CFMAIL>

注意在CFMAIL的TO属性中用到了查询字段#Email#，将邮件发送到查询所得的email地址。在CFMAIL的信息体中用到了其它查询变量（FirstName，LastName等），为不同的收信者定制不同的邮件内容。

附带一个MIME文件

在下的例子中，一个MIME文件与email信息一起被发送：

```
<CFMAIL FROM="info@allaire.com"
```

```
TO="jdoe@supercomputer.com"
```

```
SUBJECT="File you requested"
```

```
MIMEATTACH="c:\cfmanual.doc">
```

Dear Joe,

Here is a copy of the file you requested.

Regards,

The Allaire Team

```
</CFMAIL>
```

[后页](#)

前页
目录
北极星书库

高级选项

ColdFusion所实现的SMTP邮件使用缓冲结构。也就是说，当处理程序文件中的CFMAIL标签时，并不立即发送邮件，而是将邮件保存在硬盘上，在后台进行处理。这种结构有两个显著的优点：

- 1、用户不必等待SMTP发送过程结束即可继续下面的操作。当用户的操作导致大量的邮件发送时，这一优点特别有用。
- 2、用CFMAIL发送信息是可靠的，即使出现意外事故如断电，服务器死机等情况，邮件仍可发送到目的地。

在大多数情况下，缓冲在硬盘上的邮件信息会立即被ColdFusion处理。但是，如果ColdFusion特别忙或者已经有许多邮件等待处理，则邮件会被推迟发送。

将邮件发送为HTML

许多新的Internet邮件程序都可以读取并解释邮件信息中的HTML代码。CFMAIL标签允许你用TYPE属性将邮件信息类型指定为HTML。TYPE属性只接受HTML作为参数，用于通知客户端软件，邮件信息中嵌有需要处理的HTML标签。只有客户端软件可以解释HTML代码时，这一特性才有用处。

覆盖缺省的SMTP服务器设置

你可以用下表所列的可选CFMAIL属性覆盖ColdFusion Administrator中的缺省SMTP服务器设置：

CFMAIL 标签	
可选属性	描述
SERVER	发送邮件所使用的SMTP服务器的地址。
PORT	SMTP服务器监听请求的TCP/IP端口号，通常总是25。
TIMEOUT	与SMTP服务器连接超时的秒数。

错误日志和未发送信息

SMTP信息的处理过程中所发生的错误都被记录在文件errors.log中。错误日志中的每条记录包含发生错误的日期和时间以及关于错误原因的诊断信息。

由于发生错误而未能发送的邮件信息被写入\cfusion\mail\undelivr目录。与未发送信息相对应的错误日志记录中包含被写入undelivr目录的文件名。

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)

收取邮件（CFPOP）

CFPOP标签为ColdFusion应用程序增添了Internet邮件客户端的特性。当你在应用程序中收取邮件时，可以使用CFPOP。下面是可以使用CFPOP的两个实例：

- 如果你的站点有多人共用的邮箱（Sales@yourcompany.com），那么构造一个ColdFusion前台邮件界面，为每个用户实现取信功能，将是一种有效的方法。
- 在许多应用程序中，当邮件的格式服务于特定任务时，可以利用CFPOP对邮件进行自动处理。

CFPOP示例

这个例子演示了CFPOP标签的基本语法：

```
<CFPOP SERVER="my.mailserver.com"
```

```
    USERNAME=#username#
```

PASSWORD=#pwd#

ACTION="GetHeaderOnly"

NAME="getmsghdrs">

<BODY>

<CFOUTPUT>

You have #getmsghdrs.RecordCount# messages to read.

</CFOUTPUT>

<CFOUTPUT QUERY="getmsghdrs">

<P>Date: #getmsghdrs.date#</P>

<P>From: #getmsghdrs.from#</P>

</CFOUTPUT>

</BODY>

使用CFPOP

在ColdFusion应用程序中实现CFPOP标签的步骤如下：

- 1、选择要在ColdFusion应用程序中访问的邮箱。
- 2、确定你需要处理哪些邮件信息部件：信息标题，信息体，附件等。
- 3、决定是否需要将邮件信息保存到数据库中。
- 4、决定在收取邮件后是否从POP服务器上删除该邮件。
- 5、在应用程序中利用CFPOP标签建立访问指定邮箱的用户界面。
- 6、建立一个用于输出处理的程序文件。收取的信息中可以包含浏览器不能正确显示的ASCII字符。

在CFOUTPUT标签中可以用函数HTMLCodeFormat和HTMLEditFormat来控制浏览器的输出显示。在后面的例子中要注意这两个函数的使用。

CFPOP查询变量

对每个CFPOP查询，都将返回两个变量：

- RecordCount：查询结果的记录总数。
- CurrentRow：在CFOUTPUT循环中正在处理的当前记录。

你可以在CFOUTPUT中引用这些变量，只需以查询名作为变量名的前缀。查询名即CFPOP的NAME属性的值。例如：

```
<CFOUTPUT>
```

```
This operation returned #Sample.RecordCount# messages.
```

```
</CFOUTPUT>
```

处理POP邮件

CFPOP的ACTION属性用来指定要获取什么数据。你可以获取所有数据（GetAll），只获取标题数据（GetHeaderOnly），或者删除邮件（DELETE）。后面将给出几个例子。注意要使下面的例子能成功运行，你需要以一个合法的用户名和密码引用某个有效的POP服务器。

只返回信息标题

使用CFPOP标签时，若ACTION="GetHeaderOnly"，则ColdFusion只将邮件的标题信息返回到NAME属性所指定的查询中。标题信息有下面几种：

- DATE
- FROM
- MESSAGENUMBER
- REPLYTO

- SUBJECT

- CC

- TO

标题信息

CFPOP所返回的邮件标题有可能被括在HTML代码中。你可以用ColdFusion函数HTMLCodeFormat将HTML标签转换为替代字符，例如用“>”表示“>”，用“<”表示“<”。

另外，CFPOP返回的日期值可以用函数ParseDateTime进行处理，该函数将POP日期时间对象转换为GMT（格林威治时间）。

你可以在CFOUTPUT中引用这些标题，如下例所示：

```
<CFOUTPUT>
```

```
#ParseDateTime(queryname.date, "POP")#
```

```
#HTMLCodeFormat(queryname.from)#
```

```
#HTMLCodeFormat(queryname.messageNumber)#
```

```
</CFOUTPUT>
```

例子：获取邮件标题

用户可以从POP服务器获取或删除邮件信息，下面的例子文件holronly.cfm演示了如何获取邮件的标题信息：

例子文件: hdronly.cfm

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>POP Mail Message Header Example</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

<H2>This example retrieves message

header information:</H2>

<CFPOP SERVER="mail.company.com"

USERNAME=#username#

PASSWORD=#password#

ACTION="GetHeaderOnly"

NAME="Sample">

<CFOUTPUT QUERY="Sample">

MessageNumber: #HTMLFormat(Sample.MESSAGENUMBER)#

To: #HTMLFormat(Sample.TO)#

From: #HTMLFormat(Sample.FROM)#

Subject: #HTMLFormat(Sample.SUBJECT)#

Date: #HTMLFormat(Sample.DATE)#

Cc: #HTMLFormat(Sample.CC)#

ReplyTo: #HTMLFormat(Sample.REPLYTO)#

</CFOUTPUT>

</BODY>

</HTML>

返回整个邮件

使用CFPOP时指定ACTION="GetA11" , ColdFusion除了返回与
GetHeaderOnly

相同的字段外，还返回另外两个字段，BODY和HEADER。

在下面的例子文件hdrbody.cfm中，ACTION属性被设为“GetA11”，因此它将获取邮件的标题和信息体。

例子文件: hdrbody.cfm

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>POP Mail Message Body Example</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<H2>This example adds retrieval of
```

```
the message body:</H2>
```

<CFPOP SERVER="mail.company.com"

USERNAME=#username#

PASSWORD=#password#

ACTION="GetAll"

NAME="Sample">

<CFOUTPUT QUERY="Sample">

MessageNumber: #HTMLEditFormat(Sample.MESSAGENUMBER)#

To: #HTMLEditFormat(Sample.TO)#

From: #HTMLEditFormat(Sample.FROM)#

Subject: #HTMLEditFormat(Sample.SUBJECT)#

Date: #HTMLEditFormat(Sample.DATE)#

Cc: #HTMLFormat(Sample.CC)#

ReplyTo: #HTMLFormat(Sample.REPLYTO)#

Body: #HTMLCodeFormat(Sample.BODY)#

Header: #HTMLCodeFormat(Sample.HEADER)#

</CFOUTPUT>

</BODY>

</HTML>

返回附件

若ACTION属性设置为“GetA11”，并同时指定ATTACHMENTPATH属性，则ColdFusion返回两个额外的字段：

- ATTACHMENTS——包含一系列附件源的名字，以制表符为分隔符。

- ATTACHMENTFILES——包含实际写入服务器的一系列临时文件名：以制表符为分隔符。

你可以用CFFILE标签删除这些临时文件。

并非所有的邮件都带有附件。如果一个邮件没有附件，则ATTACHMENTS属性和ATTACHMENTFILES属性均为一空字符串。

处理附件文件名

为避免文件重名所带来的问题，ColdFusion为所有的附件建立临时文件。如果你收取的邮件带有附件，你可以用ATTACHMENTPATH属性选择附件的保存位置。应用程序开发者的任务是管理这些文件，使对它们的访问不会混淆。

程序开发者还应注意清除CFPOP标签执行时所产生的临时文件。

下面的例子文件attch.cfm返回包括附件在内的邮件信息的所有内容：

例子文件: attach.cfm

<HTML>

<HEAD>

<TITLE>POP Mail Message Attachment Example</TITLE>

</HEAD>

<BODY>

<H2>This example retrieves message header,

body, and all attachments:</H2>

<CFPOP SERVER="mail.company.com"

USERNAME=#username#

PASSWORD=#password#

ACTION="GetAll"

ATTACHMENTPATH="c:\attachdir"

NAME="Sample">

<CFOUTPUT QUERY="Sample">

MessageNumber: #HTMLFormat(Sample.MESSAGENUMBER)#

To: #HTMLFormat(Sample.TO)#

From: #HTMLFormat(Sample.FROM)#

Subject: #HTMLFormat(Sample.SUBJECT)#

Date: #HTMLFormat(Sample.DATE)#

Cc: #HTMLFormat(Sample.CC)#

ReplyTo: #HTMLFormat(Sample.REPLYTO)#

Attachments: #HTMLFormat(Sample.ATTACHMENTS)#

Attachment Files: #HTMLEditFormat(Sample.ATTACHMENTFILES)#

Body: #HTMLCodeFormat(Sample.BODY)#

Header: #HTMLCodeFormat(Sample.HEADER)#

</CFOUTPUT>

</BODY>

</HTML>

删除邮件

缺省情况下，收取邮件时并不将邮件从POP服务器上删除，如果你想删除已收取的邮件，则必须将ACTION属性设为“Delete”。

CFPOP返回的MESSAGENUMBER变量包含了你需要送回到POP服务器的邮件序号，该序号指明要删除哪个邮件。应注意的是：

- 一旦删除了一个邮件，就无法恢复它。

- 每次对POP服务器执行完一个Delete操作后，邮件序号就会被重新分配。例如，如果从POP服务器收取了四个邮件，返回的邮件序号为1，2，3，4。如果随后在一个CFPOP标签内删除了两个邮件，则邮件3和邮件4会分别被分配给序号1和2。

例子：删除邮件

例子文件: msgdel.cfm

<HTML>

<HEAD>

<TITLE>POP Mail Message Delete Example</TITLE>

</HEAD>

<BODY>

<H2>This example deletes messages:</H2>

<CFPOP SERVER="mail.company.com"

USERNAME=#username#

PASSWORD=#password#

ACTION="Delete"

MESSAGENUMBER="1,2,3">

</BODY>

</HTML>

[后页](#)
[前页](#)
[目录](#)
[北极星书库](#)