Apache 服务器的配置

PHP 支持 Apache、Nginx、IIS 等多种 Web 服务器，本章主要以 Linux 操作系统上安装的 Apache 服务器为例，为 PHP 的应用介绍 Apache 配置和使用的基本过程。包括 Apache 服务器的目录结构、配置文件的结构说明、文件和目录的访问限制，以及虚拟主机的设置等。本章虽然是围绕 Linux 操作系统下以源代码包安装的 Apache 为例，但同样也适用于在 Windows 操作系统上运行的 Apache 服务器，只是两个系统的目录结构及相关位置有所差异。

1 Apache 简介

网站只有发布以后，用户才能通过 Web 浏览器访问到该网站中的信息资源。Web 服务器软件就是用于发布网站的服务器，而 Apache 是世界使用排名第一的 Web 服务器软件，它可以运行在几乎所有广泛使用的计算机平台上。

如图 1 所示为 Web 浏览的 B/S 工作模型。Apache 服务器提供一些重要数据，这些数据需要在客户端的浏览器中支持显示。服务器启动 Web 服务，拥有可供浏览的资源，客户端使用可解析服务器信息的浏览器查看资源，同时向服务器端请求需要的资源，服务器和客户端即通过这一方式实现信息交互。

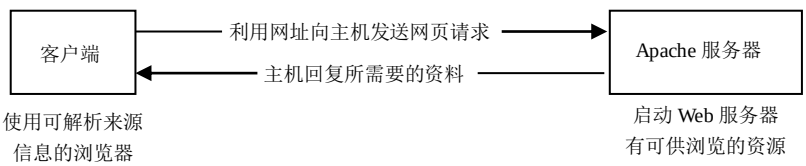

图 1　Web 工作模型

Apache 服务器所提供的资源其实就是一些文件，因此，管理员必须在服务器端先将数据文件写好，并放置在某个特殊的目录下，这个目录就是整个网站的首页位置。另外，客户端必须在浏览器的“地址栏”输入所需要的网址才行。

1.1　Apache 的诞生

Apache 源于 NCSAhttpd 服务器，经过多次修改，成为世界上最流行的 Web 服务器软件之一。Apache 取自“a patchy server”的读音，意思是充满补丁的服务器。因为它是自由软件，所以不断有人来为它开发新的功能、新的特性、修改原来的缺陷。Apache 的特点是简单、速度快、性能稳定，并且可以作为代理服务器使用。最初 Apache 只用于小型或试验 Internet 网络，后来逐步扩充到各种 UNIX 系统中，尤其对 Linux 的支持相当完美。Apache 有多种产品，可以支持 SSL 技术，支持多个虚拟主机。Apache 是以进程为基础的结构，进程要比线程消耗更多的系统开支，不太适合于多处理器环。因此，在一个

Apache Web 站点扩容时，通常是增加服务器或扩充群集节点而不是增加处理器。到目前为止，Apache 仍然是世界上用得最多的 Web 服务器，市场占有率达 60%左右。世界上很多著名的网站如 Amazon.com、Yahoo!、W3 Consortium、Financial Times 等都是 Apache 的产物。它的成功之处主要在于它的源代码开放、有一支开放的开发队伍、支持跨平台的应用（可以运行在几乎所有的 UNIX、Windows、Linux 系统平台上），以及它的可移植性等方面。

Apache 的诞生极富有戏剧性。当 NCSA WWW 服务器项目停顿后，那些使用 NCSA WWW 服务器的人们开始交换他们用于该服务器的补丁程序，他们也很快认识到成立管理这些补丁程序的论坛是必要的。就这样，诞生了 Apache Group，后来这个团体在 NCSA 的基础上创建了 Apache。

1.2 Apache 的特性

因为 Apache 是开源软件，这就意味着用户可以免费从 Apache 的官方网站下载。另外，对于开源软件任何人都可以参加组成部分的开发。如果你准备选择 Web 服务器，毫无疑问 Apache 是你的最佳选择。最新版本的 Apache Web 服务器软件拥有以下特性：

- 支持最新的 HTTP/1.1 通信协议。
- 拥有简单而强有力的基于文件的配置过程。
- 支持通用网关接口。
- 支持基于 IP 和基于域名的虚拟主机。
- 支持多种方式的 HTTP 认证。
- 集成 Perl 处理模块。
- 集成代理服务器模块。
- 支持实时监视服务器状态和定制服务器日志。
- 支持服务器端包含指令（SSI）。
- 支持安全 Socket 层（SSL）。
- 提供用户会话过程的跟踪。

2 Apache 服务器的目录结构

在 Linux 系统下使用源代码包安装 Apache 时，我们将 Apache 服务器的家目录设置在 /usr/local/apache2 目录下面。在学习 Apache 服务器配置之前首先了解一下 Apache 的目录结构。进入到 Apache 的安装目录中，使用 ls 命令显示下面的目录结构。如下所示：

```
[root@localhost root]# cd /usr/local/apache2/          //进入 apache 的安装目录
[root@localhost apache2]# ls                           //显示 apache 安装目录下所有目录
bin     cgi-bin  htdocs  include  logs  manual
Build   error    icons   lib      man   modules
```

在 Apache 安装目录 usr/local/apache2 下面存在以上所列的子目录，常见子目录的存放内容及说明如表 C-1 所示。

表 1　Apache 家目录下的子目录说明

目录名称	描　　述

Bin	是 Apache 执行文件所在的目录。例如，Apache 的主要执行文件 apachectl，使用这个命令可以用来控制 Apache 服务器开启或停止。还有 Apache 的密码保护文件 htpasswd，当用户在某些网页需要输入账号与密码时，Apache 本身提供一个最基本的密码保护方式，该密码的产生就是透过这个指令来完成的
cgi-bin	预设给一些可执行的 CGI 网页程序存放的目录，当用户输入 http://localhost/cgi-bin/时所显示的数据所存储的目录
error	默认错误应答文件目录。如果因为服务器设置或者浏览器要求的数据错误时，在浏览器上出现的错误信息存放的位置
htdocs	默认 Web 文档根目录，就是用户存放网页和存放默认首页的位置。用户通过浏览器中输入地址访问这个下面的文件，并将文件内容传回给用户浏览器，显示成美妙的页面供给用户浏览
icons	这个目录提供 Apache 预设给予的一些小图示
Logs	Apache 日志文件存放的位置。其中包括访问日志 access_log 和错误日志 error_log 文件。访问日志 access_log 记录了所有对 Web 服务器的访问活动，例如，记录了客户端的 IP 地址和在什么时间访问了哪个页面等信息。错误日志 error_log 记录了所有对 Web 服务器的错误访问活动
Man	Apache 服务器配置文件的帮助手册所在目录
modules	Apache 服务器所支持的动态加载模块所在的位置

除了要了解 Apache 家目录下的子目录的作用，还有两个比较重要的目录需要掌握，就是存入 Apache 配置文件和子配置文件的两个目录，如下所示。

- **/etc/httpd/目录：**在通过源代码包安装 Apache 时，为 Apache 指定的主配置文件 httpd.conf 存放的目录。
- **/etc/httpd/extra/目录：**该目录为 Apache 扩展及模块配置文件目录。最新版本 Apache 服务器不仅有一个 httpd.conf 作为主配置文件，还存在一些辅助配置文件被存放在这个目录下。

3 Apache 服务器配置文件结构说明

Apache 服务器的各种功能特性主要是由配置文件 httpd.conf 决定。如果读者希望通过文件方式配置 Apache 服务器，则只需要修改 Apache 配置文件的指令参数即可。

3.1 Apache 服务器配置文件位置

Apache 的配置文件是包含若干“指令”的纯文本文件，其位置是编译时确定的。这里介绍的是以第 2 章源码包方式安装的 Apache 服务器为例，配置文件的位置在安装时我们将其指定到了/etc/httpd/目录下。另外，在主配置文件 httpd.conf 中只包含少量与服务自身相关的指令，可以用 Include 指令和通配符附加许多其他配置文件，默认不附加任何其他配置文件。这是 Apache2.2 以后的最新版本服务器，为了使 Apache 配置文件 httpd.conf 简洁，默认的配置布局已经被简化并模块化了。将 httpd.conf 文件按配置功能拆分为多个常用特性的配置片段和 Apache 捆绑在一起，可以被轻易地添加到主配置文件中。每个配置片段都有一个对应的配置文件，存放在主配置文件 httpd.conf 所在目录的 extra 子目录下。httpd.conf 为主配置文件，扩展及模块配置文件目录 extra 下的为辅配置文件，要在主配置文件中通过 Include 指令符加上这些辅配置文件一起使用。一些常用的辅配置文件名称和作用如下：

```
[root@localhost root]# ls /etc/httpd/extra/        //显示/etc/httpd/extra/下面的所有附加配置文件
httpd-autoindex.conf                               //配置目录列表的辅配置文件
httpd-dav.conf                                     //配置 DAV 的辅配置文件
httpd-default.conf                                 //配置与 Apache 服务自身相关的辅配置文件
```

httpd-info.conf	//配置用于服务器信息和状态显示的辅配置文件
httpd-languages.conf	//配置语言支持的辅配置文件
httpd-manual.conf	//配置提供 Apache 文档访问的辅配置文件
httpd-mpm.conf	//配置多路处理模块(MPM) 的辅配置文件
httpd-multilang-errordoc.conf	//配置多语言错误应答的辅配置文件
httpd-ssl.conf	//配置 SSL 模块的辅配置文件
httpd-userdir.conf	//配置用户主目录的辅配置文件
httpd-vhosts.conf	//配置虚拟主机的辅配置文件

如果以二进制包（rpm 包）安装 Apache 时，主配置文件中包含了大部分配置指令，默认通过 include 指令将/etc/httpd/conf.d 目录中的以.conf 为扩展名的文件自动附加到主配置文件。本节以源代码包安装形式为例，在这里二进制包安装的环境将不再介绍。

任何配置文件都可以使用任何指令。只有在启动或重新启动 Apache 后，配置文件的更改才会生效。重新启动可以使用 Apache 安装目录下，bin 子目录下的 apachectl 命令完成。具体命令如下：

[root@localhost root]# **/usr/local/apache2/bin/apachectl stop**	//停止 Apache 服务
[root@localhost root]# **/usr/local/apache2/bin/apachectl start**	//启动 Apache 服务
或	
[root@localhost root]# **/usr/local/apache2/bin/apachectl restart**	//重启 Apache 服务

3.2 配置文件的语法

Apache 服务器配置文件都是使用 Linux 的脚本风格，例如以“#”作为注释符等。本节内容将主要介绍配置文件中常用的各种配置命令。在配置文件中，很多是使用变量赋值语法风格，文件的每一行包含一个指令，它告诉 Apache 服务器以某种特定的方式完成某一项特定的任务。在指令的行尾使用反斜杠"\"可以表示续行，但是反斜杠与下一行之间不能有任何其他字符（包括空白字符）。具体指令格式说明如下：

#指令名	参数	//指令和参数中间有一个空格
Listen	*80*	//Listen 是指令名，80 是该指令的参数
ServerRoot	*"/usr/local/apache2"*	//ServerRoot 是指令名，后面是该指令的参数

配置文件中的指令是不区分大小写的，但是指令的参数（argument）通常是大小写敏感的。以“#”开头的行被视为注解并被忽略，注解不能出现在指令的后边。空白行和指令前的空白字符将被忽略，因此可以采用缩进以保持配置层次的清晰。当在非 UNIX/Linux 平台上输入文件路径的时候，要特别注意即使平台本身是使用反斜杠（\）来分隔路径的，比如 Windows 系统平台，在这里也只能使用正斜杠（/）。通常在配置文件里只用正斜杠（/）来分隔路径总是不会错的。可以用 apachectl configtest 或者命令行选项-t 检查配置文件中的错误，而无须启动 Apache 服务器。

[root@localhost root]# **/usr/local/apache2/bin/apachectl configtest**	//检查配置文件中的错误
Syntax OK	//表示没有错误
[root@localhost root]# **/usr/local/apache2/bin/apachectl –t**	//检查配置文件中的错误
Syntax OK	//表示没有错误

Apache 是模块化的服务器，这意味着核心中只包含实现最基本功能的模块，扩展功能可以作为模块动态加载。默认情况下，只有 base 组的模块被编译进了服务器，如果服务器在编译时包含了 DSO（Dynamic Shared Object）模块，那么各模块可以独立编译，并可随时用 LoadModule 指令加载。比如

PHP 5 模块就是这样动态加载到 Apache 服务器中的。否则，要增加或删除模块必须重新编译整个 Apache。用命令行参数–1 可以查看已经编译到服务器中的模块，命令格式如下：

```
[root@localhost root]# /usr/local/apache2/bin/apachectl –l          //查看已经编译到 Apache 服务器中的模块
```

主配置文件中的指令对整个服务器都有效。用于特定模块的指令可以用<IfModule>指令包含起来，使之有条件地生效。如果你只想改变某一部分的配置，你可以把指令嵌入到<Directory>、<DirectoryMatch>、<Files>、<FilesMatch>、<Location>、<LocationMatch>配置段中，这样就可以限制指令的作用域为文件系统中的某些位置或特定的 URL。这些配置段还可以进行嵌套，以进行更精细的配置。Apache 还具备同时支持多个站点的能力，称为虚拟主机。<VirtualHost>配置段中的指令仅对该段中的特定站点（虚拟主机）有效。

Apache 可以使用分布在整个网站文件目录树结构中的特殊文件来进行分散配置，这些特殊的文件通常叫.htaccess，但是也可以用 AccessFileName 指令来改变它的名字。.htaccess 文件中指令的作用域是存放它的那个目录及其所有子目录。.htaccess 文件的语法与主配置文件相同。由于对每次请求都会读取.htaccess 文件，所以对这些文件的修改会立即生效。不需要重启 Apache 服务器。服务器管理员可以在主配置文件中使用 AllowOverride 指令来决定哪些指令可以在.htaccess 文件中生效。

4 Apache 服务器全局参数设置

在 Apache 2.2 以上以源码包安装的版本中，主配置文件 httpd.conf 中只包含少量的必须使用的一些与服务自身相关的指令，大部分指令都被分散保存在辅助配置文件中。可以用 Include 指令和通配符附加这些辅助配置文件到主配置文件 httpd.conf 中，但默认不附加任何其他辅助配置文件。如果使用其他辅助配置文件中的设置，就需要手动在主配置文件中加载。主配置文件中已经写好了附加其他的配置文件的指令，但默认使用“#”注释掉了，需要加载哪个辅助配置文件，去掉前面的注释即可。在主配置文件 httpd.conf 中附加辅助配置文件的指令和说明如下：

```
[root@localhost root]# cat /etc/httpd/httpd.conf
… …
#Include /etc/httpd/extra/httpd-mpm.conf                    #此指令附加多路处理模块(MPM)配置文件
#Include /etc/httpd/extra/httpd-multilang-errordoc.conf     #此指令附加多语言错误应答的辅配置文件
#Include /etc/httpd/extra/httpd-autoindex.conf              #此指令附加目录列表的辅配置文件
#Include /etc/httpd/extra/httpd-languages.conf              #此指令附加语言支持的辅配置文件
#Include /etc/httpd/extra/httpd-userdir.conf                #此指令附加用户主目录的辅配置文件
#Include /etc/httpd/extra/httpd-info.conf                   #用于服务器信息和状态显示的辅配置文件
#Include /etc/httpd/extra/httpd-vhosts.conf                 #此指令附加虚拟主机的辅配置文件
#Include /etc/httpd/extra/httpd-manual.conf                 #此指令附加提供 Apache 文档访问的文件
#Include /etc/httpd/extra/httpd-dav.conf                    #此指令附加 DAV 的辅配置文件
#Include /etc/httpd/extra/httpd-default.conf                #此指令附加与 Apache 服务自身相关的文件
#Include /etc/httpd/extra/httpd-ssl.conf                    #此指令附加 SSL 模块的辅配置文件
… …
```

以下是 Apache 服务器配置文件常用的全局配置参数。全局环境将影响全局操作，用于设置 Apache 的一些基本信息，如服务器根目录、Apache 能够支持的并行请求数目、文档根目录、超时时间等。这些参数主要在主配置文件 httpd.conf 以及部分附加配置文件中，将分别列举每个文件中的全局指令，并详细说明。

4.1 在主配置文件 httpd.conf 中的全局参数

```
ServerRoot "/usr/local/apache2"                    #设置服务器目录的绝对路径
#Listen 12.34.56.78:80                             #允许 Apache 绑定指定的 IP 或者端口，实现对其监听
Listen 80                                          #Apache 绑定指定的端口 80
#以下几行加载特定的 DSO（Dynamic Shared Object）模块
LoadModule deflate_module modules/mod_deflate.so
LoadModule expires_module modules/mod_expires.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule php5_module     modules/libphp5.so      #动态加载 PHP5 模块
#设置子进程的用户和组，<IfModule test>...</IfModule>配置段用于封装根据指定的模块是否启用而决定是否生效的指令。在<IfModule>配置段中的指令仅当 test 为真的时候才进行处理。如果 test 为假，所有其间的指令都将被忽略
<IfModule !mpm_netware_module>
    <IfModule !mpm_winnt_module>
        User daemon                                #设置实际提供服务的子进程的用户
        Group daemon                               #设置提供服务的 Apache 子进程运行时的用户组
    </IfModule>
</IfModule>
ServerAdmin you@example.com                        #设置在返回给客户端的错误信息中包含的管理员邮件地址
#ServerName www.example.com:80                     #设置服务器用于辨识自己的主机名和端口号
DocumentRoot "/usr/local/apache2/htdocs"           #设置 Web 文档根目录，默认存放网页位置
<IfModule dir_module>
    DirectoryIndex index.html                      #指令用于指定目录中默认的索引文件名称
</IfModule>
```

以下是在主配置文件 httpd.conf 中全局参数的详细说明。

1．ServerRoot 指令

```
语法：ServerRoot directory-path
例如：ServerRoot "/usr/local/apache2"
```

ServerRoot 用于指定 Apache 服务器的配置文件及日志文件存放的根目录，服务器的基础目录，一般来说它将包含 conf/和 logs/子目录。但配置文件在安装时指定到了/etc/httpd 目录下，其他配置文件的相对路径即基于此目录。默认为安装目录/usr/local/apache2，不需更改。

2．Listen 指令

```
语法：Listen [IP-address:]portnumber [protocol]
例如：Listen 80
```

Listen 指令用于设置 apache 服务器监听指定 IP 和（或）端口上的连接请求。如果只指定一个端口，服务器将在所有地址上监听该端口。如果指定了地址和端口的组合，服务器将在指定地址的指定端口上监听。可选的 protocol 参数在大多数情况下并不需要，若未指定该参数，则将为 443 端口使用默认的 HTTPS 协议，为其他端口使用 HTTP 协议。使用多个 Listen 指令可以指定多个不同的监听端口或地址端口组合。默认为“Listen 80”，如果让服务器接受 80 和 8080 端口上请求，可以这样设置：

```
Listen 80                                          #服务器接受 80 端口上的请求
Listen 8080                                        #服务器接受 8080 端口上的请求
```

如果让服务器在两个确定的地址端口组合上接受请求，可以这样设置：

```
Listen 192.168.2.1:80                              #服务器接受 192.168.2.1 地址的 80 端口上的请求
Listen 192.168.2.2:8080                            #服务器接受 192.168.2.2 地址的 8080 端口上的请求
```

如果使用 IPV6 地址，必须用方括号把 IPV6 地址括起来：

```
Listen [2001:db8::a00:20ff:fea7:ccea]:80        #监听 2001:db8::a00:20ff:fea7:ccea 地址的 80 端口上的请求
```

仅在使用非标准端口时才需要指定 protocol 参数。比如在 8443 端口运行 HTTPS 协议：

```
Listen 192.170.2.1:8443 https                   #服务器接受 192.168.2.1 地址的 8443 端口上的以 https 协议请求
```

3．LoadModule 指令

```
语法：LoadModule   模块名称        模块文件路径全名
例如：LoadModule   php5_module    modules/libphp5.so
```

Apache 默认将已编译的 DSO（Dynamic Shared Object）模块存放于 Apache 安装目录的子目录 modules/中，如果模块文件路径全名使用相对路径，则路径是相对于 ServerRoot 所指示的相对路径。Apache 服务器采用动态共享对象 DSO 的机制，在启动 Apache 服务器时可根据实际需要载入适当的模块，使其具有相应的功能。LoadModule 指令用于动态载入模块，即将模块外挂在 Apache 服务器上。Apache 服务器可以根据需要去加载相应的模块，这里设置动态加载的 DSO 模块有如下几个：

```
#加载此模块是服务器在将输出内容发送到客户端以前进行压缩以节约带宽
LoadModule deflate_module modules/mod_deflate.so
#加载此模块是服务器允许通过配置文件控制 HTTP 的"Expires:"和"Cache-Control:"头内容
LoadModule expires_module modules/mod_expires.so
#加载此模块是服务器需要基于一定规则实时重写 URL 请求
LoadModule rewrite_module modules/mod_rewrite.so
#加载此模块是服务器将用户请求的 PHP 内容使用 PHP 模块处理
LoadModule php5_module    modules/libphp5.so
```

4．User 指令

```
语法：User   unix-userid
例如：User   daemon
```

User 指令用于设置实际提供服务的子进程的用户。为了使用这个指令，服务器必须以 root 身份启动和初始化。如果你以非 root 身份启动服务器，子进程将不能够切换至非特权用户，并继续以启动服务器的原始用户身份运行。如果确实以 root 用户启动了服务器，那么父进程将仍然以 root 身份运行。Unix-userid 是下列值之一：

```
User daemon                 #通过用户名引用用户 daemon
User #-1                    #通过用户编号引用用户，"#"号后面跟一个用户编号 1
```

用于运行子进程的用户必须是一个没有特权的用户,这样才能保证子进程无权访问那些不想为外界所知的文件，同样的，该用户还需要没有执行那些不应当被外界执行的程序的权限。强烈推荐你专门为 Apache 子进程建立一个单独的用户和组。一些管理员使用 nobody 用户，但是这并不能总是符合要求，因为可能有其他程序也在使用这个用户。

5．Group 指令

```
语法：Group   unix-group
例如：Group   daemon
```

Group 指令指定了用于对客户端请求提供服务的 Apache 子进程运行时的用户组。为了使用这个指令，Apache 必须以 root 初始化启动，否则在切换用户组时会失败，并继续以初始化启动时的用户组运行。Unix-group 可以是下列之一：

```
Group daemon                    #通过名称引用组 deamon
Group #-1                       #通过编号引用组,"#"号后跟一个组编号(GID)
```

建议你专门为 Apache 服务器新建一个用户组。一些管理员使用 nobody 用户，但是这并非总是可用或是合适的。为了安全，不要将 Group（或 User）设置成 root，除非你明确知道自己在做什么，并且明白其风险所在。

6．ServerAdmin 指令

```
语法：ServerAdmin   email-address|URL
例如：ServerAdmin   you@example.com
```

应该被设置为管理 Web 服务器管理员的邮件地址。当服务器运行出错时，这一地址将被返回给访问者，访问者可以向此邮件地址发信和 Web 服务器管理员联系。

7．ServerName 指令

```
语法：ServerName    fully-qualified-domain-name[:port]
例如：ServerName    www.example.com:80
```

ServerName 指令设置了服务器用于辨识自己的主机名和端口号，这主要用于创建重定向 URL。比如，一个放置 Web 服务器的主机名为 simple.example.com，但同时有一个 DNS 别名 www.example.com。而你希望 Web 服务器更显著一点，你可以使用“ServerName www.example.com:80”。当没有指定 ServerName 时，服务器会尝试对 IP 地址进行反向查询来推断主机名。如果在 ServerName 中没有指定端口号，服务器会使用接受请求的那个端口。为了加强可靠性和可预测性，你应该使用 ServerName 显式的指定一个主机名和端口号。如果使用的是基于域名的虚拟主机，在<VirtualHost>段中的 ServerName 将是为了匹配这个虚拟主机，在"Host:"请求头中必须出现的主机名。

8．DocumentRoot 指令

```
语法：DocumentRoot   directory-path
例如：DocumentRoot   "/usr/local/apache2/htdocs"
```

DocumentRoot 指令设置 Web 文档根目录。在没有使用类似 Alias 这样的指令的情况下，服务器会将请求中的 URL 附加到 DocumentRoot 后面以构成指向文档的路径。比如说“DocumentRoot/usr/web”，于是对 http://www.my.host.com/index.html 的访问就会指向/usr/web/index.html。如果 directory-path 不是绝对路径，则被假定为是相对于 ServerRoot 的路径。指定 DocumentRoot 时不应包括最后的“/”。

9．DirectoryIndex 指令

```
语法：DirectoryIndex local-url [local-url ] ...
例如：DirectoryIndex index.html index .htm index.php index.phtml
```

DirectoryIndex 指令设置了当客户端在请求的目录名的末尾刻意添加一个“/”以表示请求该目录的索引时，服务器需要寻找的资源列表。Local-url（%已解码的）是一个相对于被请求目录的文档的 URL（通常是那个目录中的一个文件）。可以指定多个 URL，服务器将返回最先找到的那一个。若一个也没有找到，并且那个目录设置了 Indexes 选项，服务器将会自动产生一个那个目录中的资源列表。示例如下：

```
DirectoryIndex index.html                    #当用户直接请求目录时，服务器将返回这个目录下的 index.html
```

上例配置指示对 http://myserver/docs/的请求返回 http://myserver/docs/index.html（若存在），或返回

该目录下所有资源的列表。注意，指定的文档不一定必须位于被请求的目录下，也可以指定一个绝对URL 来指向其他位置：

```
DirectoryIndex  index.html  index.txt  /php-bin/index.php
```

这样的设置将导致在 index.html 或 index.txt 都不存在的情况下执行 PHP 脚本/php-bin/index.php。

4.2 附加配置文件 httpd-default.conf 中的全局参数

```
Timeout 300                          #服务器在断定请求失败前等待的秒数
KeepAlive On                         #启用 HTTP 持久链接
MaxKeepAliveRequests 100             #一个持久链接中允许的最大请求数量
KeepAliveTimeout 5                   #持久链接中服务器在两次请求之间等待的秒数
HostnameLookups Off                  #启用对客户端 IP 的 DNS 查找
```

httpd-default.conf 文件是 Apache 服务器默认设置文件，以上列出的也是 Apache 服务器常用的全局配置参数。默认没有在主配置文件中加载这个文件，如果想使用这个文件的参数，在主配置文件中找到下面这条指令，去掉前面的“#”即可。

```
#Include /etc/httpd/extra/httpd-default.conf      #在 Apache 的主配置文件 httpd.conf 中加载附加配置文件
```

1．TimeOut 指令

```
语法：TimeOut  seconds
例如：TimeOut  300
```

TimeOut 指令用于设置 Apache 等待以下三种事件的时间长度：

```
（1）接受一个 GET 请求耗费的总时间。
（2）POST 或 PUT 请求时，接受两个 TCP 包之间的时间。
（3）应答时 TCP 包传输中两个 ACK 包之间的时间。
```

计时器在 1.2 版本之前的默认值为 1200，而现在已经设置为 300 了，但对于绝大多数情况来说仍然是足够的。没有把它默认值设的更小的原因在于代码里还有点问题：有时发送一个包之后，计时器没有复位。

2．KeepAlive 指令

```
语法：KeepAlive  On|Off
例如：KeepAlive  On
```

Keep-Alive 扩展自 HTTP/1.0 和 HTTP/1.1 的持久链接特性。提供了长效的 HTTP 会话，用以在同一个 TCP 连接中进行多次请求。在某些情况下，这样的方式会对包含大量图片的 HTML 文档造成的延时起到 50%的加速作用。在 Apache1.2 版本以后，您可以设置 KeepAlive On 以启用持久链接。

对于 HTTP/1.0 的客户端来说，仅当客户端指定使用的时候才会使用持久链接连接。此外，仅当能够预先知道传输的内容长度时，才会与 HTTP/1.0 的客户端建立持久链接连接。这意味着那些长度不定的内容，诸如 CGI 输出、SSI 页面，以及服务器端生成的目录列表等内容一般来说将无法使用与 HTTP/1.0 客户端建立的持久链接连接。而对于 HTTP/1.1 的客户端来说，如果没有进行特殊指定，持久将是默认的连接方式。如果客户端进行了请求，将使用分块编码以解决在持久链接里发送未知长度内容的问题。

3．MaxKeepAliveRequests 指令

语法：MaxKeepAliveRequests *number*
例如：MaxKeepAliveRequests *100*

MaxKeepAliveRequests 指令限制了当启用 KeepAlive 时，每个连接允许的请求数量。如果将此值设为“0”，将不限制请求的数目。建议最好将此值设为一个比较大的值，以确保最优的服务器性能。

4．KeepAliveTimeout 指令

语法：KeepAliveTimeout *seconds*
例如：KeepAliveTimeout *5*

Apache 在关闭持久连接前等待下一个请求的秒数。一旦收到一个请求，超时值将会被设置为 Timeout 指令指定的秒数。对于高负荷服务器来说，KeepAliveTimeout 值较大会导致一些性能方面的问题：超时值越大，与空闲客户端保持连接的进程就越多。

5．HostnameLookups 指令

语法：HostnameLookups *On|Off|Double*
例如：HostnameLookups *Off*

此指令启用了对客户端 IP 的 DNS 查询，可选值为 On|Off|Double。使得主机名能被记入日志。DNS 查询会造成明显的时间消耗，建议设置为 Off。

4.3 附加配置文件 httpd-mpm.conf 中的全局参数

```
<IfModule !mpm_netware_module>
    PidFile "logs/httpd.pid"                    #用于指定记录 httpd 进程号（PID）的文件位置
</IfModule>
#设置 prefork 多路处理模块
<IfModule mpm_prefork_module>
    StartServers          5                     #设置服务器启动时建立的子进程数量
    MinSpareServers       5                     #设置空闲子进程的最小数量
    MaxSpareServers      10                     #设置空闲子进程的最大数量
    MaxClients           150                    #设置 Apache 的最大连接数
    MaxRequestsPerChild   0                     #设置每个子进程在其生存期内允许伺服的最大请求数量
</IfModule>
#设置 worker 多路处理模块
<IfModule mpm_worker_module>
    StartServers          2
    MaxClients          150
    MinSpareThreads      25
    MaxSpareThreads      75
    ThreadsPerChild      25
    MaxRequestsPerChild   0
</IfModule>
```

Apache 服务器被设计为一个强大的、灵活的、能够在多种平台，以及不同环境下工作的服务器。不同的平台和不同的环境经常产生不同的需求，或是为了达到同样的最佳效果而采用不同的方法。Apache 凭借它的模块化设计很好地适应了大量不同的环境。这一设计使得网站管理员能够在编译时和运行时凭借载入不同的模块来决定服务器的不同附加功能。

Apache2.0 将这种模块化的设计延伸到了 Web 服务器的基础功能上。这个版本带有多路处理模块（MPM）的选择以处理网络端口绑定、接受请求并指派子进程来处理这些请求。将模块化设计延伸到

这一层次主要有以下两大好处：

- Apache 可以更简洁、更有效地支持各种操作系统。
- 服务器可以为某些特定的站点进行定制。

从用户角度来看，MPM 更像其他的 Apache 模块。不论何时，必须有且仅有一个 MPM 被载入到服务器中。现有的 MPM 列表可以在模块索引中找到。MPM 必须在编译配置时进行选择，并静态编译到服务器中。如果编译器能够确定线程功能被启用，它将会负责优化大量功能。因为一些 MPM 在 UNIX 上使用了线程，而另外一些没有使用，所以如果在编译配置时选择 MPM 并静态编译进 Apache，Apache 将会有更好的表现。表 2 列出了不同操作系统上默认的 MPM。如果你在编译时没有进行选择，这将是默认选择的 MPM。

表 2　不同操作系统上使用的默认 MPM

操作系统	默认选择的 MPM
BeOS	Beos
Netware	mpm_netware
OS/2	mpmt_os2
Unix/Linux	prefork
Windows	mpm_winnt

我们主要阐述 prefork 和 worker 这两种和性能关系最大的产品级 MPM。

```
Apache MPM prefork                  #一个非线程型的、预派生的 MPM
Apache MPM worker                   #支持混合的多线程多进程的多路处理模块
```

Apache MPM prefork 多路处理模块（MPM）实现了一个非线程型的、预派生的 Web 服务器，它的工作方式类似于 Apache 1.3。它适合于没有线程安全库，需要避免线程兼容性问题的系统。它是要求将每个请求相互独立的情况下最好的 MPM，这样若一个请求出现问题就不会影响到其他请求。这个 MPM 具有很强的自我调节能力，只需要很少的配置指令调整。最重要的是将 MaxClients 设置为一个足够大的数值以处理潜在的请求高峰，同时又不能太大，以致需要使用的内存超出物理内存的大小。

Apache MPM worker 多路处理模块（MPM）使网络服务器支持混合的多线程多进程。由于使用线程来处理请求，所以可以处理海量请求，而系统资源的开销小于基于进程的 MPM。但是，它也使用了多进程，每个进程又有多个线程，以获得基于进程的 MPM 的稳定性。控制这个 MPM 的最重要的指令是，控制每个子进程允许建立的线程数的 ThreadsPerChild 指令，和控制允许建立的总线程数的 MaxClients 指令。常用指令如下。

1．StartServers 指令

StartServers 指令设置了服务器启动时建立的子进程数量。因为子进程数量动态的取决于负载的轻重，所有一般没有必要调整这个参数。

2．MinSpareServers 指令

MinSpareServers 指令设置空闲子进程的最小数量。所谓空闲子进程是指没有正在处理请求的子进程。如果当前空闲子进程数少于 MinSpareServers，那么 Apache 将以最大每秒一个的速度产生新的子进程。只有在非常繁忙机器上才需要调整这个参数。将此参数设的太大通常是一个坏主意。

3．MaxSpareServers 指令

MaxSpareServers 指令设置空闲子进程的最大数量。所谓空闲子进程是指没有正在处理请求的子进程。如果当前有超过 MaxSpareServers 数量的空闲子进程，那么父进程将杀死多余的子进程。只有在非常繁忙机器上才需要调整这个参数。将此参数设的太大通常是一个坏主意。如果你将该指令的值设置为比 MinSpareServers 小，Apache 将会自动将其修改成"MinSpareServers+1"。

4．MaxClients 指令

MaxClients 指令设置了同一时间服务器允许的最大接入请求数量。任何超过 MaxClients 限制的请求都将进入等候队列，直到达到 ListenBacklog 指令限制的最大值为止。一旦一个链接被释放，队列中的请求将得到服务。

对于非线程型的 MPM（也就是 prefork），MaxClients 表示可以用于伺服客户端请求的最大子进程数量，默认值是 256。要增大这个值，你必须同时增大 ServerLimit。对于线程型或者混合型的 MPM（也就是 beos 或 worker），MaxClients 表示可以用于伺服客户端请求的最大线程数量。线程型的 beos 的默认值是 50。对于混合型的 MPM 默认值是 16（ServerLimit）乘以 25（ThreadsPerChild）的结果。因此要将 MaxClients 增加到超过 16 个进程才能提供的时候，你必须同时增加 ServerLimit 的值。

5．MaxRequestsPerChild 指令

MaxRequestsPerChild 指令设置每个子进程在其生存期内允许的最大请求数量。到达 MaxRequests PerChild 的限制后，子进程将会结束。如果 MaxRequestsPerChild 为“0”，子进程将永远不会结束。

不同的默认值在 mpm_netware 和 mpm_winnt 上的默认值是“0”。将 MaxRequestsPerChild 设置成非零值有两个好处：

- 可以防止（偶然的）内存泄漏无限进行，从而耗尽内存。
- 给进程一个有限寿命，从而有助于当服务器负载减轻的时候减少活动进程的数量。

这里要注意，对于 KeepAlive 链接，只有第一个请求会被计数。事实上，它改变了每个子进程限制最大链接数量的行为。

6．ThreadsPerChild 指令

这个指令设置了每个子进程建立的线程数。子进程在启动时建立这些线程后就不再建立新的线程了。如果使用一个类似于 mpm_winnt 只有一个子进程的 MPM，这个数值要足够大，以便可以处理可能的请求高峰。如果使用一个类似于 worker 有多个子进程的 MPM，每个子进程所拥有的所有线程的总数要足够大，以便可以处理可能的请求高峰。对于 mpm_winnt，ThreadsPerChild 的默认值是 64；对于其他 MPM 是 25。

7．ThreadLimit 指令

这个指令设置了每个子进程可配置的线程数 ThreadsPerChild 上限。任何在重启期间对这个指令的改变都将被忽略，但对 ThreadsPerChild 的修改却会生效。

使用这个指令时要特别当心。如果将 ThreadLimit 设置成一个高出 ThreadsPerChild 实际需要很多的值，将会有过多的共享内存被分配。如果将 ThreadLimit 和 ThreadsPerChild 设置成超过系统的处理能力，Apache 可能无法启动，或者系统将变得不稳定。该指令的值应当和 ThreadsPerChild 可能达到的最大值保持一致。对于 mpm_winnt，ThreadLimit 的默认值是 1920；对于其他 MPM 这个值是 64。

这里要注意，Apache 在编译时内部有一个硬性的限制"ThreadLimit 20000"（对于 mpm_winnt 是"ThreadLimit 15000"），你不能超越这个限制。

8．ServerLimit 指令

对于 preforkMPM，这个指令设置了 MaxClients 最大允许配置的数值。对于 workerMPM，这个指令和 ThreadLimit 结合使用设置了 MaxClients 最大允许配置的数值。任何在重启期间对这个指令的改变都将被忽略，但对 MaxClients 的修改却会生效。

使用这个指令时要特别当心。如果将 ServerLimit 设置成一个高出实际需要许多的值，将会有过多的共享内存被分配。如果将 ServerLimit 和 MaxClients 设置成超过系统的处理能力，Apache 可能无法启动，或者系统将变得不稳定。

对于 preforkMPM，只有在你需要将 MaxClients 设置成高于默认值 256 的时候才需要使用这个指令。要将此指令的值保持和 MaxClients 一样。

对于 workerMPM，只有在你需要将 MaxClients 和 ThreadsPerChild 设置成需要超过默认值 16 个子进程的时候才需要使用这个指令。不要将该指令的值设置的比 MaxClients 和 ThreadsPerChild 需要的子进程数量高。

这里要注意，Apache 在编译时内部有一个硬限制"ServerLimit 20000"（对于 preforkMPM 为"ServerLimit 200000"）。你不能超越这个限制。

5 Apache 服务器中使用配置段（容器）

配置文件中指令的作用范围可能是整个服务器，也可能是特定的目录、文件、主机、URL。这一节阐述如何使用配置段（容器），以及.htaccess 文件来改变配置指令的作用范围。配置段（容器）通常包括在< >括号内，较容易识别。常用的容器指令有：

```
<Directory> </Directory>                    #封装一组指令，使之仅对文件空间中的某个目录及其子目录生效
<Files> </Files>                            #包含作用于匹配指定文件名的指令
<Location> </Location>                      #将封装的指令作用于匹配的 URL
<VirtualHost> </VirtualHost>                #包含仅作用于指定主机名或 IP 地址的指令
```

容器有两种基本类型。大多数容器是针对各个请求的，包含于其中的指令仅对与该容器匹配的请求起作用，而容器<IfDefine>、<IfModule>、<IfVersion>仅在启动和重新启动中起作。如果在启动时指定的条件成立，则其中的指令对所有的请求都有效，否则将被忽略。

<IfDefine>容器中的指令只有在 httpd 命令行中设定了特定的参数后才有效。下例中，只有在服务器用 httpd-DClosedForNow 方式启动时，所有的请求才会被重定向到另一个站点：

```
<IfDefine ClosedForNow>                     #判断服务是否用 httpd-DClosedForNow 方式启动
        Redirect    /http://otherserver.example.com/    #如果条件成功请求才会被重定向到另一个站点
</IfDefine>                                 #条件的结束标记
```

<IfModule>容器与<IfDefine>容器很相似，但是其中的指令只有当服务器启用特定的模块时才有效（或是被静态地编译进了服务器，或是被动态装载进了服务器）。注意，配置文件中该模块的装载指令 LoadModule 行必须出现在此容器之前。这个容器应该仅用于你希望无论特定模块是否安装，配置文件

都能正常运转的场合；而不应该用于容器中的指令在任何情况下都必须生效的场合，因为它会抑制类似模块没找到之类的有用出错信息。下例中，MimeMagicFiles 指令仅当 mod_mime_magic 模块启用时才有效。

```
<IfModule mod_mime_magic.c>          #仅当 mod_mime_magic 模块启用时条件成功
    MimeMagicFile conf/magic         #条件成立才使用 MimeMagicFile conf/magic 指令
</IfModule>                          #条件的结束标记
```

<IfVersion>指令与<IfDefine>和<IfModule>很相似，但是其中的指令只有当正在执行的服务器版本与指定的版本要求相符时才有效。这个模块被设计用于测试套件，以及在一个存在多个不同 httpd 版本的大型网络中需要分别针对不同版本使用不同配置的情况。

```
<IfVersion >= 2.1>
    # 仅在版本高于 2.1.0 的时候才生效
</IfVersion>
```

<IfDefine>、<IfModule>、<IfVersion>都可以在条件前加一个“!”以实现条件的否定，而且都可以嵌套以实现更复杂的配置。

最常用的配置段是针对文件系统和网络空间特定位置的配置段。首先必须理解文件系统和网络空间这两个概念的区别，文件系统是指操作系统所看见的磁盘视图，比如，在 UNIX 文件系统中，Apache 会被默认安装到/usr/local/apache2，在 Windows 文件系统中，Apache 会被默认安装到“C:/Program Files/Apache Group/Apache2”（注意：Apache 始终用正斜杠而不是反斜杠作为路径的分隔符，即使是在 Windows 中）。相反，网络空间是网站被 Web 服务器发送，以及被客户在浏览器中所看到的视图。所以网络空间中的路径/dir/在 Apache 采用默认安装路径的情况下对应于 UNIX 文件系统中的路径/usr/local/apache2/htdocs/dir/。由于网页可以从数据库或其他地方动态生成，因此，网络空间无须直接映射到文件系统。

5.1 文件系统容器

<Directory>和<Files>指令与其相应的正则表达式版本（<DirectoryMatch>和<FilesMatch>）一起作用于文件系统的特定部分。<Directory>配置段中的指令作用于指定的文件系统目录及其所有子目录，.htaccess 文件可以达到同样的效果。下例中，/var/web/dir1 及其所有子目录被允许进行目录索引。

```
<Directory /var/web/dir1>            #为目录/var/web/dir1 设置属性
    Options +Indexes                 #被设置的目录允许被索引
</Directory>                         #结束一个目录的设置
```

<Files>配置段中的指令作用于特定的文件名，而无论这个文件实际存在于哪个目录。下例中的配置指令如果出现在配置文件的主服务器段，则会拒绝对位于任何目录下的 private.html 的访问。

```
<Files private.html>                 #为任意目录下的 private.html 文件设置属性
    Order allow,deny                 #顺序是先允许后被拒绝
    Deny from all                    #拒绝对位于任何目录下的 private.html 的访问
</Files>                             #结束一个文件的设置
```

<Files>和<Directory>段的组合可以作用于文件系统中的特定文件。在下面的示例中的配置会拒绝对/var/web/dir1/private.html 文件、/var/web/dir1/subdir2/private.html 文件、/var/web/dir1/subdir3/private.html

文件等任何/var/web/dir1/目录下 private.html 文件的访问。

```
<Directory /var/web/dir1>                #为目录/var/web/dir1 设置属性
        <Files private.html>              #为/var/web/dir1 目录及子目录下的 private.html 文件设置属性
            Order allow,deny             #顺序是先允许后被拒绝
            Deny from all                #拒绝对文件 private.html 的访问
        </Files>                         #结束一个文件的设置
</Directory>                             #结束一个目录的设置
```

5.2 网络空间容器

<Location>指令与其相应的正则表达式版本（<LocationMatch>）一起作用于网络空间的特定部分。下例中的配置会拒绝对任何以“/private”开头的 URL 路径的访问，比如：http://yoursite.example.com/private、http://yoursite.example.com/private123、http://yoursite.example.com/private/dir/file.html 等所有以“/private”开头的 URL 路径。

```
<Location /private>                      #为以/private 开头的 URL 路径设置属性
        Order Allow,Deny                 #顺序是先允许后被拒绝
        Deny from all                    #拒绝对/private 开头的 URL 路径的访问
</Location>                              #结束一个网络位置的设置
```

5.3 通配符和正则表达式

<Directory>、<Files>、<Location>指令可以使用与 C 标准库中的 fnmatch 类似的 shell 风格的通配符。“*”匹配任何字符串，“?”匹配任何单个的字符，“[seq]”匹配 seq 序列中的任何字符，符号“/”不被任何通配符所匹配，所以必须显式地使用。

如果需要更复杂的匹配，这些容器都有一个对应的正则版本：<DirectoryMatch>、<FilesMatch>、<LocationMatch>，可以使用与 Perl 兼容的正则表达式，以提供更复杂的匹配。但是还必须注意下面配置段的合并部分关于使用正则表达式会如何作用于配置指令的内容。下例使用非正则表达式的通配符来改变所有用户目录的配置：

```
<Directory /home/*/public_html>          #使用非正则表达式的通配符来改变所有用户目录的配置
        Options Indexes                  #允许使用目录索引
</Directory>                             #结束一个目录的设置
```

下例使用正则表达式一次性拒绝对多种图形文件的访问：

```
<FilesMatch \.(?i:gif|jpe?g|png)$>       #使用正则表达式一次性拒绝对多种图形文件的访问
        Order allow,deny                 #顺序是先允许后被拒绝
        Deny from all                    #拒绝对匹配上的图片格式的访问
</FilesMatch>                            #结束一个文件正则的设置
```

5.4 如何选择使用容器

选择使用文件系统容器还是使用网络空间容器其实很简单。当指令应该作用于文件系统时，总是用<Directory>或<Files>；而当指令作用于不存在于文件系统的对象时，就用<Location>，比如一个由数据

库生成的网页。绝对不要试图用<Location>去限制对文件系统中的对象的访问，因为许多不同的网络空间路径可能会映射到同一个文件系统目录，从而导致你的访问限制被突破。比如：

```
<Location /dir/>                    #为以/dir/开头的 URL 路径设置属性
    Order allow,deny                #顺序是先允许后被拒绝
     Deny from all                  #拒绝对/dir/开头的 URL 路径的访问
</Location>                         #结束一个网络位置的设置
```

上述配置对 http://yoursite.example.com/dir/请求的确起作用。但是设想一下，在一个不区分大小写的文件系统中，这个访问限制会被 http://yoursite.example.com/DIR/请求轻易突破。而<Directory>指令才会真正作用于对这个位置的任何形式的请求。但是有一个例外，就是 UNIX 文件系统中的符号连接（软连接），符号连接可以使同一个目录出现在文件系统中的多个位置。<Directory>指令将不重设路径名而直接追踪符号连接，因此，对于安全要求最高的，应该用 Options 指令禁止对符号连接的追踪。

不要认为使用大小写敏感的文件系统就无所谓了，因为有很多方法可以将不同的网络空间路径映射到同一个文件系统路径，所以，应当尽可能使用文件系统容器。但是也有一个例外，就是把访问限制放在<Location/>配置段中可以很安全地作用于除了某些特定 URL 以外的所有 URL。

<VirtualHost>容器作用于特定的虚拟主机，为同一个机器上具有不同配置的多个主机提供支持。在后面的章节中将详细介绍。

6 .htaccess 文件和访问限制

.htaccess 文件（或者“分布式配置文件”）提供了针对每个目录改变配置的方法，即在一个特定的目录中放置一个包含指令的文件，其中的指令作用于此目录及其所有子目录。任何出现在配置文件中的指令都可能出现在.htaccess 文件中。该文件在 httpd.conf 文件的 AccessFileName 指令中指定，用于进行针对单一目录的配置。在服务器配置文件中按以下方法配置：

```
AccessFileName   .htaccess                #在主配置文件中设置访问.htaccess 文件
```

一般情况下，不应该使用.htaccess 文件，除非你对主配置文件没有访问权限。有一种很常见的误解，认为用户认证只能通过.htaccess 文件实现，其实并不是这样，把用户认证写在主配置文件中是完全可行的，而且是一种很好的方法。

.htaccess 文件应该被用在内容提供者需要针对特定目录改变服务器的配置而又没有 root 权限的情况下。如果服务器管理员不愿意频繁修改配置，则可以允许用户通过.htaccess 文件自己修改配置，尤其是 ISP 在同一个机器上运行了多个用户站点，而又希望用户可以自己改变配置的情况下。

虽然如此，一般都应该尽可能地避免使用.htaccess 文件。任何希望放在.htaccess 文件中的配置，都可以放在主配置文件的<Directory>段中，而且更高效。避免使用.htaccess 文件有两个主要原因：首先是性能。如果 AllowOverride 启用了.htaccess 文件，则 Apache 需要在每个目录中查找.htaccess 文件，因此，无论是否真正用到，启用.htaccess 都会导致性能的下降。另外，对每一个请求，都需要读取一次.htaccess 文件。Apache 必须在所有上级的目录中查找.htaccess 文件，以使所有有效的指令都起作用。其次是安全。这样会允许用户自己修改服务器的配置，这可能会导致某些意想不到的修改，所以请认真考虑是否应当给予用户这样的特权。将 AllowOverride 设置为 none 可以完全禁止使用.htaccess 文件：

```
AllowOverride None                  #完全禁止使用.htaccess 文件
```

6.1 设置路径别名

在使用.htaccess 文件之前，我们使用 Alias 指令设置一个路径别名。将/var/www/html 目录的别名设置为 dir，这样我们可以在浏览器里通过别名 dir 去访问/var/www/html 目录下面的网页文件了，并且可以使用<Directory>容器指令对/var/www/html 目录做访问控制。之后在把同样的访问控制改为使用.htaccess 文件方式完成。设置路径别名和目录权限控制，可以在主配置文件 httpd.conf 中加入以下指令。具体设置如下：

```
Alias /dir/    "/var/www/html/"                    #使用 Alias 设置别名
<Directory "/var/www/html">                        #使用<Directory>容器指令设置/var/www/html 目录访问权限控制
        Options Indexes FollowSymLinks             #当访问时可以允许出现目录列表和符号链接
        AllowOverride None                         #禁止使用.htaccess 文件
        Order allow,deny                           #顺序是先允许后被拒绝
        allow from all                             #允许所有对这个目录的访问
</Directory>                                       #结束一个目录的设置
```

配置文件按上面的设置以后，重新启动 Apache 服务器。如果服务器所在的主机为 yoursite.example.com，则在浏览器中通过 http://yoursite.example.com/dir/地址访问服务器中/var/www/html/目录下面的文件。在/var/www/html/目录下新建一个 test.html 和一个 demo.html 文件，如果直接访问该目录就可以看到目录索引列表，如图 2 所示。

图 2　设置别名路径的访问结果

在上面的配置中，<Directory>容器指令中使用的指令具体说明如下：

1． Options 指令

Options 指令控制了在特定目录中将使用哪些服务器特性。Options 可以为 None、All 或者任何 Indexes、Includes、FollowSymlinks、ExecCGI 或者 MultiViews 的组合。MultiViews 不包含在 All 中，必须显式指定。这些选项解释如表 3 所示。

表 3　Options 指令选项值的解释

选 项 名	描　　述
None	在这种情况下，将不启用任何额外特性
All	除 MultiViews 之外的所有特性。这是默认设置
Indexes	如果一个映射到目录的 URL 被请求，而此目录中又没有 DirectoryIndex（例如：index.html），那么服务器会返回由 mod_autoindex 生成的一个格式化后的目录列表
Includes	允许使用 mod_include 提供的服务器端包含
FollowSymLinks	服务器允许在此目录中使用符号链接

符号链接的使用如下：

```
[root@localhost html]# ln –s /var/share/doc   /var/www/html/doc          #在 Linux 命令行创建软链接
```

通过上面的链接，我们在浏览器中访问 http://yoursite.example.com/dir/doc/，就可以将访问位置链接到/var/share/doc 目录下面。

注意：即使服务器会使用符号连接，但它不会改变用于匹配<Directory>段的路径名。

如果此配置位于<Location>配置段中，则此设置会被忽略。

2． AllowOverrides 指令

确定允许存在于.htaccess 文件中的指令类型。当服务器发现一个.htaccess 文件（由 AccessFileName 指定）时，它需要知道在这个文件中声明的哪些指令能覆盖在此之前指定的配置指令。AllowOverride 仅在不包含正则表达式的<Directory>配置段中才是有效的。在<Location>，<DirectoryMatch>，<Files>配置段中都是无效的。如果此指令被设置为 None，那么.htaccess 文件将被完全忽略。事实上，服务器根本不会读取.htaccess 文件。当此指令设置为 All 时，所有具有“.htaccess”作用域的指令都允许出现在.htaccess 文件中。

3． Order 指令

Order 指令控制默认的访问状态与 Allow 和 Deny 指令生效的顺序。表明用户是先设置允许的访问地址还是先设置禁止访问的地址。

4． Allow 指令

Allow 指令控制哪些主机可以访问服务器的该区域。可以根据主机名、IP 地址、IP 地址范围或其他环境变量中捕获的客户端请求特性进行控制。如指定“Allow from all”，则允许所有主机访问。

5． Deny 指令

这条指令允许基于主机名、IP 地址或者环境变量限制对服务器的访问。Deny 指令的参数设置和 Allow 指令完全相同。如指定“Deny from all”，则禁止所有主机访问。

6.2 在.htaccess 文件中设置目录的访问限制

启用并控制使用.htaccess 文件，可以在 Apache 的主配置文件中将 AccessFileName 指令的参数设置为.htaccess，按如下内容修改即可启用.htaccess 文件功能。启用.htaccess 文件如下：

```
AccessFileName .htaccess                    #某个目录启用分布式配置文件功能
<FilesMatch "^\.ht">                        #设置文件名称，并且设置客户端无法修改.htaccess 文件
    Order allow,deny
    Deny from all
</FilesMatch>
```

任何出现在配置文件中的指令都可能出现在.htaccess 文件中。通常，.htaccess 文件使用的配置语法和主配置文件一样。AllowOverride 指令按类别决定了.htaccess 文件中哪些指令才是有效的。我们将上例别名的设置路径/var/www/html 的访问限制，重新改写成使用.htaccess 文件进行控制。首先在主配置文件 httpd.conf 中将上例<Directory>容器指令中的内容修改写成如下内容：

```
Alias /dir/    "/var/www/html/"             #使用 Alias 设置别名
<Directory "/var/www/html">                 #使用<Directory>容器指令设置/var/www/html 目录访问权限控制
        AllowOverride All                   #将上例参数 none 改写成 all 即可以使用.htaccess 文件
</Directory>
```

上面所示将<Directory>容器指令中的内容除了留下 AllowOverride 指令，删除了其余三个指令。并将指令 AllowOverride 的值由原来的 None 改成 All，即具有“.htaccess”作用域的指令都允许出现

在.htaccess 文件中。

在别名设置目录/var/www/html 中新建一个.htaccess 文件，将删除掉的三个指令写入到这个文件中，这样在 Apache 服务器寻找目录时就可以使用.htaccess 文件来对这个目录进行访问限制了。如下所示：

```
[root@localhost html]# vi /var/www/html/.htaccess          //编辑此文件写入下面内容
Options Indexes FollowSymLinks
Order allow,deny
Allow from all
```

不用重新启动 Apache 服务器即可以使用.htaccess 文件中的目录访问限制，打开浏览器同样输入 URL 为 http://yoursite.example.com/dir/，同样可以看到目录列表，表示使用.htaccess 文件设置些目录访问限制成功。

7 设置虚拟主机

虚拟主机是指在一个机器上运行多个网站（例如，www.company1.com 和 www.company2.com）。有两种方法在 Apache 服务器上设置虚拟主机，如果每个网站拥有不同的 IP 地址，则虚拟主机可以是“基于 IP”的；如果只有一个 IP 地址，也可以是“基于主机名”的。简单地说，使用基于 IP 的虚拟主机时，每一个虚拟主机都有一个不同的 IP 地址，而基于名称的虚拟主机都有相同的 IP 地址，但它们的名称不同。两种类型各有优点，但它们的实现方法并没有大的差别。

7.1 基于 IP 地址的虚拟主机

基于 IP 的虚拟主机使用连接的 IP 地址来决定相应的虚拟主机。就像它的名字“基于 IP”所暗示的那样，这样的服务器中每个基于 IP 的虚拟主机必须拥有不同的 IP 地址。如图 3 所示，同一个 Web 服务器设置四个 IP 地址，配置成一个主服务器和三个虚拟主机。第一个虚拟主机，通过域名 www.company1.com 访问 IP 地址为 192.168.1.11 对应的站点、第二个虚拟主机，通过域名 www.company2.com 访问 IP 地址为 192.168.1.12 对应的站点、第三个虚拟主机，通过域名 www.company3.com 访问 IP 地址为 192.168.1.13 对应的站点。主 Web 服务器则和没有配置虚拟主机之前一样访问。

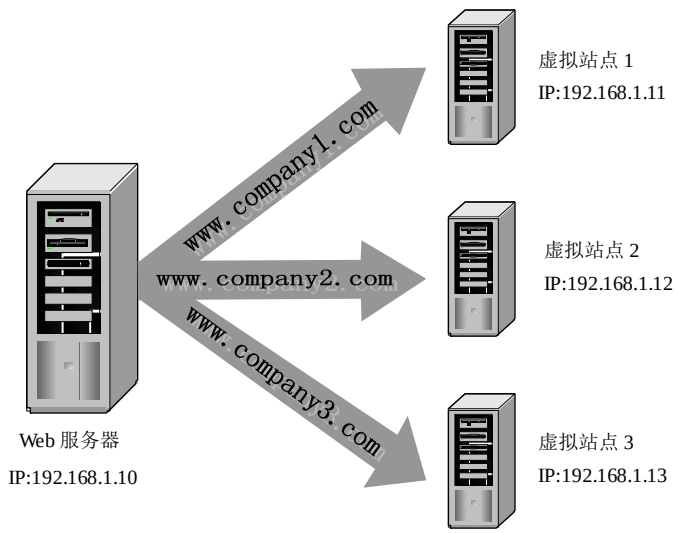

图 3　基于 IP 的虚拟主机

其于 IP 的虚拟主机具体的配置步骤如下：

1． 设置服务器的 IP 地址

可以通过配备多个真实的物理网络接口来达到这一要求，也可以使用几乎所有流行的操作系统都支持的虚拟界面来达到这一要求。假设现在服务器有一个 IP 地址为 192.168.1.10，这里是测试环境，可以使用 Linux 下的 ifconfig 命令在同一个网络接口 eth0 上再绑定 3 个 IP 地址。命令如下所示；

```
[root@localhost root]# ifconfig  eth0:1  192.168.1.11      //添加第一个 IP 地址
[root@localhost root]# ifconfig  eth0:2  192.168.1.12      //添加第二个 IP 地址
[root@localhost root]# ifconfig  eth0:3  192.168.1.13      //添加第三个 IP 地址
```

2． 设置域名对应各自的 IP 地址

可以在 DNS 中添加三个域名来分别指向上面设置的三个 IP 地址。也可以在您的 hosts 文件中添加三行条目来进行测试，但这种方法仅适用于那些有 hosts 文件的机器来使用。这里我们使用 hosts 文件将域名分别映射到各自的 IP。在 Windows 系统中 hosts 文件在 C:\WINDOWS\system32\drivers\etc\文件夹下，在 Linux 系统中 hosts 文件在/etc/目录下。使用 hosts 文件做域名解释比较简单，只要在该文件中加入以下内容即可，具体操作如下：

```
[root@localhost root]# vi /etc/hosts              //使用 vi 编辑 host 文件加入下面三行内容
192.168.1.11          www.company1.com    //不同 IP
192.168.1.12          www.company2.com    //不同 IP
192.168.1.13          www.company3.com    //不同 IP
```

3． 设置各个虚拟主机存放网页的根目录

建立每个虚拟主机需要存放网页的目录，可以在任意目录下建立，这里在/www/目录建三个目录，具体操作如下所示：

```
[root@localhost root]# mkdir /www                 //在根目录下建立 www 目录
[root@localhost root]# mkdir /www/company1        //建立第一个虚拟主机存入网页目录
[root@localhost root]# mkdir /www/company2        //建立第二个虚拟主机存入网页目录
[root@localhost root]# mkdir /www/company3        //建立第三个虚拟主机存入网页目录
```

还需要在每个目录下建立各自的网页文件以供测试，在 company1/目录下建一个 one.html 文件、在 company2/目录下建一个 two.html 文件、在 company3/目录下建一个 three.html 文件。在每个文件中可以任意写入一些网页内容。

4． 使用配置文件配置虚拟主机

在主配置文件 httpd.conf 中将附加配置文件 httpd-vhosts.conf 包含进来，在 httpd.conf 文件中找到下面的 Include 指令，将前面的“#”注释去掉即可。如下所示：

```
[root@localhost root]# vi /etc/httpd/httpd.conf      //使用 vi 编辑 httpd.conf 文件
# Virtual hosts
Include /etc/httpd/extra/httpd-vhosts.conf
```

打开 Apache 虚拟主机的附加配置文件 httpd-vhosts.conf，写入下面的内容配置三个虚拟主机，如下所示：

```
[root@localhost root]# vi /etc/httpd/extra/httpd-vhosts.conf
<VirtualHost 192.168.1.11:80>                #配置 IP 为 192.168.1.11 的虚拟主机，使用 www.company1.com 访问
    ServerName www.company1.com              #设置了该虚拟机用于辨识自己的主机名
    DocumentRoot /www/company1/              #设置了该虚拟机存放网页文件的根目录
  # 你可以在这里添加其他指令，例如访问日志、错误日志，以及管理员邮箱等
    <Directory "/www/company1">              #设置/www/company1/目录的访问权限
        Options Indexes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
<VirtualHost 192.168.1.12:80>                #配置 IP 为 192.168.1.12 的虚拟主机，使用 www.company2.com 访问
    ServerName www.company2.com              #设置了该虚拟机用于辨识自己的主机名
    DocumentRoot /www/company2/              #设置了该虚拟机存放网页文件的根目录
    # 你可以在这里添加其他指令，例如访问日志、错误日志，以及管理员邮箱等
    <Directory "/www/company2">              #设置/www/company2/目录的访问权限
        Options Indexes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
<VirtualHost 192.168.1.13:80>                #配置 IP 为 192.168.1.13 的虚拟主机，使用 www.company3.com 访问
    ServerName www.company3.com              #设置了该虚拟机用于辨识自己的主机名
    DocumentRoot /www/company3/              #设置了该虚拟机存放网页文件的根目录
    # 你可以在这里添加其他指令，例如访问日志、错误日志，以及管理员邮箱等
    <Directory "/www/company3">              #设置/www/company3/目录的访问权限
        Options Indexes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

5． 测试每个虚拟主机

打开浏览器分别输入每个虚拟主机对应的主机名，访问其 IP 所对应的网页根目录，如果出现自己主机文档根目录下面的文件列表，即基于 IP 的虚拟主机配置成功。具体操作如图 4 至图 6 所示。

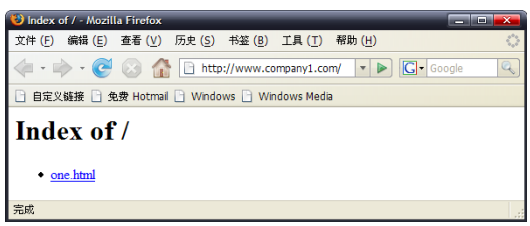

图 4　请求 www.company1.com 的结果

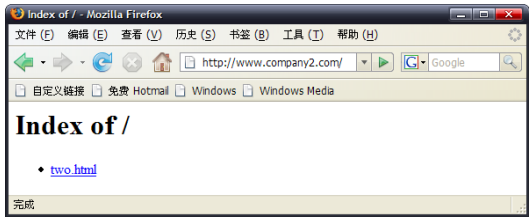

图 5　请求 www.company2.com 的结果

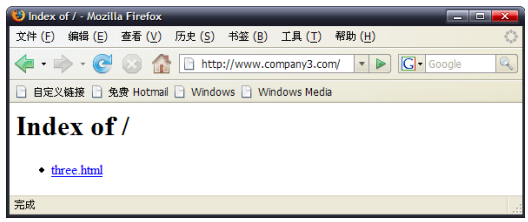

图 6　请求 www.company3.com 的结果

7.2　基于主机名的虚拟主机

基于域名的虚拟主机是根据客户端提交的 HTTP 头中标识主机名的部分决定的。使用这种技术，很多虚拟主机可以共享同一个 IP 地址。基于域名的虚拟主机相对比较简单，因为只需要配置你的 DNS 服务器将每个主机名映射到同一个 IP 地址，然后配置 Apache HTTP 服务器，令其辨识不同的主机名就可以了。

基于域名的服务器也可以缓解 IP 地址不足的问题。所以，如果没有特殊原因使你必须使用基于 IP 的虚拟主机，您最好还是使用基于域名的虚拟主机。如图 7 所示，同一个 Web 服务器只有一个 IP 地址，需要配置成一个主服务器和三个虚拟主机。第一个虚拟主机，通过域名 www.company1.com 访问第一个站点、第二个虚拟主机，通过域名 www.company2.com 访问第二站点、第三个虚拟主机，通过域名 www.company3.com 访问第三站点。主 Web 服务器则和没有配置虚拟主机之前一样访问。

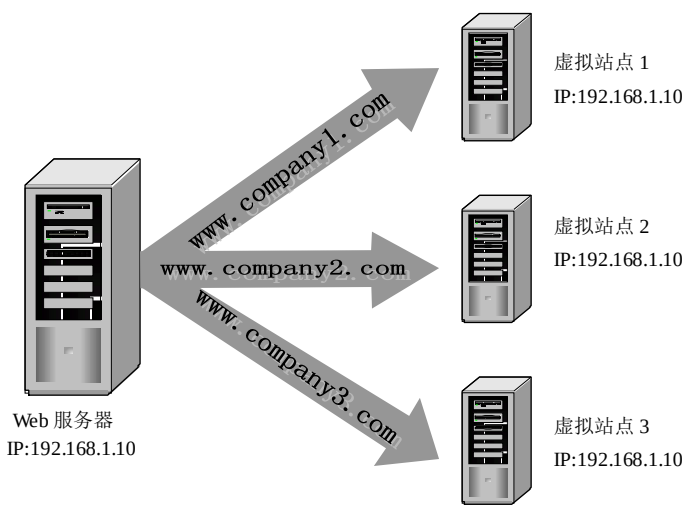

图 7　基于主机名的虚拟主机

基于域名的虚拟主机具体的配置步骤如下。

1． 设置域名映射同一个 IP 地址

可以在 DNS 中添加三个域名来指向同一个 IP 地址，也可以在您的 hosts 文件中添加三行条目来进行测试。这里我们使用 hosts 文件将三个域名映射到同一个 IP 地址。具体操作如下：

```
[root@localhost root]# vi /etc/hosts                    //使用 vi 编辑 host 文件加入下面三行内容
192.168.1.10            www.company1.com                //相同 IP
192.168.1.10            www.company2.com                //相同 IP
192.168.1.10            www.company3.com                //相同 IP
```

2． 设置各个虚拟主机存放网页的根目录

建立每个虚拟主机需要存放网页的目录，可以和上例一样在/www/目录建三个目录，具体操作如下所示：

```
[root@localhost root]# mkdir /www                       //在根目录下建立 www 目录
[root@localhost root]# mkdir /www/company1              //建立第一个虚拟主机存入网页目录
[root@localhost root]# mkdir /www/company2              //建立第二个虚拟主机存入网页目录
[root@localhost root]# mkdir /www/company3              //建立第三个虚拟主机存入网页目录
```

还需要在每个目录下建立各自的网页文件以供测试，在 company1/目录下建一个 one.html 文件、在 company2/目录下建一个 two.html 文件、在 company3/目录下建一个 three.html 文件。在每个文件中可以任意写入一些网页内容。

3． 使用配置文件配置虚拟主机

同样在主配置文件 httpd.conf 中将附加配置文件 httpd-vhosts.conf 包含进来，在 httpd.conf 文件中找到下面的 Include 指令，将前面的“#”注释去掉即可。如下所示：

```
[root@localhost root]# vi /etc/httpd/httpd.conf        //使用 vi 编辑 httpd.conf 文件
# Virtual hosts
Include /etc/httpd/extra/httpd-vhosts.conf
```

为了使用基于域名的虚拟主机，你必须指定服务器 IP 地址（和可能的端口）来使主机接受请求，这个可以用 NameVirtualHost 指令来进行配置。如果服务器上所有的 IP 地址都会用到，你可以用“*”作为 NameVirtualHost 的参数。如果你打算使用多端口（如运行 SSL）你必须在参数中指定一个端口号，比如“*:80”。请注意，在 NameVirtualHost 指令中指定 IP 地址并不会使服务器自动侦听那个 IP 地址。另外，这里设定的 IP 地址必须对应服务器上的一个网络接口。

下一步就是为每个虚拟主机建立<VirtualHost>段。<VirtualHost>的参数与 NameVirtualHost 的参数必须是一样的（比如说，一个 IP 地址或“*”代表的所有地址）。在每个<VirtualHost>段中，至少要有一个 ServerName 指令来指定伺服哪个主机和一个 DocumentRoot 指令来说明这个主机的内容位于文件系统的什么地方。

如果你想在现有的 Web 服务器上增加虚拟主机，你必须也为现存的主机建造一个<VirtualHost>定义块。这个虚拟主机中 ServerName 和 DocumentRoot 所包含的内容应该与全局的 ServerName 和 DocumentRoot 保持一致。还要把这个虚拟主机放在配置文件的最前面，来让它扮演默认主机的角色。

打开 Apache 虚拟主机的附加配置文件 httpd-vhosts.conf，写入下面的内容配置三个虚拟主机。如下

所示：

```
[root@localhost root]# vi /etc/httpd/extra/httpd-vhosts.conf
#必须指定服务器 IP 地址（和可能的端口）来使主机接受请求，这里使用“*”代表的所有地址.
NameVirtualHost *:80
<VirtualHost *:80>                  #为现存的主机建造一个<VirtualHost>定义块，取消中心主机，这段要放在最上面
    ServerName *                    #与全局的 ServerName 参数相同
    DocumentRoot /usr/local/apache2/htdocs/   #与全局的 DocumentRoot 参数相同
</VirtualHost>
<VirtualHost *:80>                  #配置第一个虚拟主机，使用 www.company1.com 访问
    ServerName www.company1.com     #设置了该虚拟机用于辨识自己的主机名
    DocumentRoot /www/company1/     #设置了该虚拟机存放网页文件的根目录
   # 你可以在这里添加其他指令，例如访问日志、错误日志，以及管理员邮箱等
    <Directory "/www/company1">     #设置/www/company1/目录的访问权限
        Options Indexes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
<VirtualHost *:80>                  #配置第二个虚拟主机，使用 www.company2.com 访问
    ServerName www.company2.com     #设置了该虚拟机用于辨识自己的主机名
    DocumentRoot /www/company2/     #设置了该虚拟机存放网页文件的根目录
    # 你可以在这里添加其他指令，例如访问日志、错误日志，以及管理员邮箱等
    <Directory "/www/company2">     #设置/www/company2/目录的访问权限
        Options Indexes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
<VirtualHost *:80>                  #配置第三个虚拟主机，使用 www.company3.com 访问
    ServerName www.company3.com     #设置了该虚拟机用于辨识自己的主机名
    DocumentRoot /www/company3/     #设置了该虚拟机存放网页文件的根目录
    # 你可以在这里添加其他指令，例如访问日志、错误日志，以及管理员邮箱等
    <Directory "/www/company3">     #设置/www/company3/目录的访问权限
        Options Indexes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

当然，你可以在上面的配置文件中用一个固定的 IP 地址来代替 NameVirtualHost 和<VirtualHost>指令中的“*”号，以达到一些特定的目的。比如说，你可能会希望在一个 IP 地址上运行一个基于域名的虚拟主机，而在另外一个 IP 地址上运行一个基于 IP 的或是另外一套基于域名的虚拟主机。

4． 测试每个虚拟主机

打开浏览器分别输入每个虚拟主机对应的主机名，访问其域名所对应的网页根目录，如果出现自己主机文档根目录下面的文件列表，即基于域名的虚拟主机配置成功。具体操作如图 8 至图 10 所示。

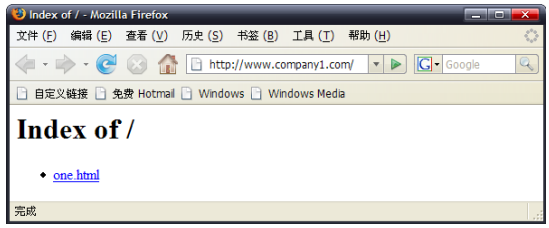

图 8　请求 www.company1.com 的结果

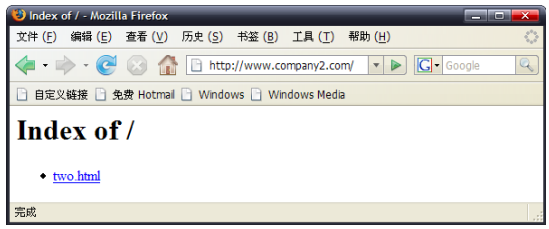

图 9　请求 www.company2.com 的结果

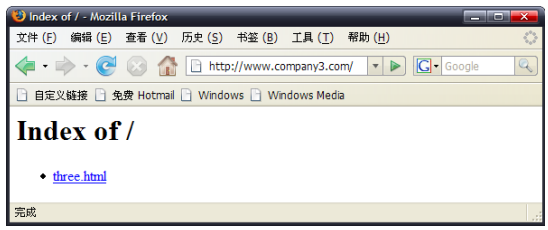

图 10　请求 www.company3.com 的结果

8 总结

网站必须在 Web 服务器上发布，用户才能浏览和查看网站中的信息。Apache 是世界使用排名第一的 Web 服务器软件，本章主要介绍 Apache 服务器的基本原理，重点介绍了 Apache 服务器的配置过程。本章主要以配置 Linux 下的 Apache 服务器为例，按照这一过程，读者可以一步步地完成一个 Apache 的配置。