Apache 源代码解析

--基于 Apache0.6.5

电子版(ver0.02)

前言

这本书主要针对在校计算机专业大三及以上年级学生阅读，具有同等学历或对 Apache 项目本身感兴趣者亦为潜在之读者。

每年新的毕业生找工作时最常被问到的问题是：“你有工作经验吗？”。即便不被问到这个问题，招聘者也首先会在应聘简历中搜寻此子之工作经验。而何谓工作经验。鄙人认为有二。

其一，合作经验。即与其他个体交流、合作之能力。做为社会人，每个个体因为各种原因难免要跟社会上的各式人等打交道，小到见面打招呼，上课占位置，大到管理集体，制定规划。老子曰：治大国若烹小鲜，此万事相通者也。

其二，项目经验。这对一个刚毕业的学生来说相对比较困难。也许有人不同意：“兄弟毕业时也有过毕业设计的”。不可否认，每人毕业时都几乎有毕业设计，且美 其名曰“某某系统”，然而，每年毕业生何止数十万，又有几个所谓的“系统”是真正从用户的需求出发，按照用户的要求做出来的？又有几个所谓的系统是一个真 正的“系统”功能？无非是答辩完成便束之高阁、无人问津罢了。

俗话说学以致用，实际上只有在用的过程中才能对所学加深理解，才能巩固所学。现在大学课程都是针对纯理论的东西进行教授，除了毕业设计时那个所谓的“系统”外，老师很少会指导学生去研究某个课题，尤其是专、本科教育。

这样带来的后果是什么呢？为了得到所谓的工作经验，好多学生毕业后去参加培训，花钱上班，甚至于上学的最后一年课程的内容就是花钱上班，何其悲哀！

这些年，随着网络的兴起，人们的目光开始聚焦在互联网上。做为将来或现任的程序员，都开始琢磨着怎么开发一个网站，都开始研究 PHP、ASP、JSP 甚至 于 JAVASCRIPT 了。这很好，不过也导致了一个问题，就是这样的程序员越来越多，且需要掌握的技术越来越多，从 J2EE 每年新出和淘汰的技术名词上 就可见一斑。当然，待遇嘛，可能距离期望越来越远了。

其实，此时的你，有另外一个选择。

一个网站，除了前台页面的表现之外，还需要一个东西来支撑，把程序员做好的效果传递给客户端，并最终在客户端浏览器上体现，这个东西就是 Web 服务器。

Apache 是目前应用最广的 web 服务器，由于它是开源的，使用者可以根据自己的需要对源代码进行调整以适应自己的特定需求。

但并不是说任何人都可以对其进行优化，前提是你要读懂它。

本书要求读者具有 C 语言的基础，如果没有，建议您看看《C 程序设计语言》。另外需要知道在 Linux 系统下进行 C 语言开发的知识，至少看过一本相关的书籍。

本书从 Apache 的早期版本——0.6.5 源代码入手，以调用顺序为主线，以模块为单位，分十三个章节对 Apache 的源代码进行了讲解。对每个模块的 讲解分三部分：首先说明 Apache 这个模块达到的效果；然后说明通过怎样的设置让 Apache 达到这样的效果；最后通过对源代码的注释，让读者了解 Apache 怎样在源代码级实现这个功能。做到知其然，知其所以然。

第 1 章 概述.....1
1.1 编译环境.....1
1.2 代码结构.....1
1.3 编译并运行 Apache.....2
第 2 章 主程序.....4
2.1 守护进程.....4
2.1.1 什么是守护进程.....4
2.1.2 创建守护进程.....4
2.2 Apache 生命周期.....5
2.3 代码注释.....6
2.3.1 httpd.c.....6
2.4 小结.....12
第 3 章 自定义库函数.....13
3.1 背景知识.....13
3.1.1 URL 编码/解码.....13
3.1.2 时间格式.....13
3.1.3 夏令时.....13
3.1.4 BASE64 编码.....14
3.2 代码注释.....16
3.2.1 stream.h、stream.c.....16
3.2.2 util.c.....21
3.3 小结.....46
第 4 章日志和重定向.....47
4.1 概述.....47
4.2 背景知识.....47
4.2.1 配置 Apache.....47
4.2.2 HTTP 状态码.....48
4.2.3 重定向.....48
4.3 日志文件配置指令.....49
4.4 配置实践.....50
4.5 代码注释.....52
4.6 小结.....60
第 5 章 目录别名.....61
5.1 概述.....61
5.2 使用目录别名.....61
5.2.1 为什么使用目录别名.....61
5.2.2 如何使用目录别名.....61
5.3 目录别名实践.....62
5.3.1 指令实践.....62
5.3.2 进程中的数据.....63
5.4 代码注释.....64
5.6 小结.....69
第 6 章 MIME.....70
6.1 概述.....70

6.2 配置指令.....70
6.3 数据组织.....71
6.4 代码注释.....72
6.5 小结.....86
第 7 章 服务器端包含（SSI）.....87
7.1 概述.....87
7.2 背景知识.....87
7.2.1 字符实体.....87
7.2.2 环境变量.....88
7.3 配置 Apache 支持 SSI.....89
7.3.1 配置方法.....89
7.3.2 配置指令解析.....89
7.4 SSI 命令解说.....90
7.4.1 config 命令.....90
7.4.2 include 命令.....91
7.4.3 echo 命令.....91
7.4.4 fsize 命令.....92
7.4.5 flastmod 命令.....92
7.4.6 exec 命令.....92
7.4.7 一个简单的实例.....92
7.5 代码注释.....93
7.6 小结.....111
第 8 章 执行 CGI 脚本.....112
8.1 概述.....112
8.2 背景知识.....112
8.2.1 初识 CGI.....112
8.2.2 isindex 标签.....113
8.3 代码注释.....114
8.4 小结.....121
第 9 章 自动索引目录.....122
9.1 概述.....122
9.2 配置自动索引目录.....123
9.3 配置数据的获取.....125
9.4 代码注释.....127
9.5 小结.....144
第 10 章 存取控制.....145
10.1 概述.....145
10.2 存取控制设置.....146
10.2.1 限制访问和认证访问.....146
10.2.2 设置指令详解.....148
10.3 代码注释.....150
10.3.1 http_access.c.....151
10.3.2 http_auth.c.....158
10.4 小结.....161

第 11 章 读取配置文件.....162
11.1 概述.....162
11.2 代码注释.....162
11.3 小结.....189
第 12 章 HTTP 方法.....190
12.1 概述.....190
12.2 代码注释.....190
12.2.1 http_request.c.....190
12.2.2 http_put.c.....195
12.2.3 http_get.c.....195
12.2.4 http_post.c.....200
12.2.5 http_delete.c.....201
12.3 小结.....202
第 13 章 头文件.....203
13.1 概述.....203
13.2 代码注释.....203
13.3 小结.....212

第1章 概述

Apache 作为世界排名第一的 Web 服务器，运行在全球一半以上提供 Web 服务的服务器上。

最早的 Apache 版本（0.6.2）由 Apache group 于 1995 年 4 月发布。目前最新版本是 2.2.11。目前能从网上找到的最古老的版本是 0.6.5，也就是我们要讨论的版本。

1.1 编译环境

本书中提供的代码在 CentOS4.5 下成功编译通过。目前主流硬件设备都可以编译运行，如果您没有相应的硬件资源，可以使用虚拟机来编译测试。关于虚拟机的使用网上有很多介绍的文章，过程都比较简单，这里不再赘述。

在 Linux 系统中，可以通过 uname –a 命令来查看当前使用系统的内核版本，作者的服务器输出如下：

[devel@RIZI ~]$ uname -a

Linux RIZI 2.6.9-55.ELsmp #1 SMP Wed May 2 14:28:44 EDT 2007 i686 i686 i386 GNU/Linux

相对应的 gcc 版本：

[devel@RIZI ~]$ gcc -v

gcc version 3.4.6 20060404 (Red Hat 3.4.6-8)

[devel@RIZI ~]$

1.2 代码结构

本书中的代码是经过裁剪的 Apache0.6.5，我们一般接触的服务器是 Linux，所以和其它系统相关的部分裁剪掉了，这些部分包括对 IBM、HP 甚至于 SONYBSD 系统的适配，这些代码对初学者来说可能会造成困惑，以至于影响我们对整个服务器的学习。

另外删减了协商视图的部分，这部分不是 Web 服务器的核心功能，一般很少使用，并且目前最主流的浏览器 IE 不会在头信息里面给出有助于服务器提供该服务的信息。

代码中有一些错误，作者也进行了相应的修改，在代码的注释里面会提到。

本书介绍的代码可以从http://www.oldapache.org/sourcecode/apache0.6.5.rar获取。

把下载的文件解压缩后，可以看到里面包含了 6 个目录，分别是 src、conf、htdocs、cgi-bin、icons 和 logs。其中 src 目录中就是 Apache 的源代码，其它目录功能在 1.3 节中说明。

Apache0.6.5 源代码共包括 20 个文件，按功能可以分成几大部分，如表 1-1 所示：

表 1-1 Apache0.6.5 源代码功能分类

功能模块	包括代码	代码功能
编译辅助	Makefile	处理工程编译（类似批处理）。
日志记录	http_log.c	记录 Apache 运行中的各种日志。

工具集	util.c、stream.h、stream.c	util.c 定义实现了一些工具函数，这些函数供其它功能模块调用。 stream.h 和 stream.c 实现了通过数据流缓冲读写文件的功能
主程序	httpd.c	Apache 主流程
配置读取	http_config.c	读取 Apache 配置文件内容并初始化本地变量
处理请求	http_request.c、http_get.c、http_put.c 、 http_post.c http_delete.c	http_request.c 处理客户端请求的主流程，它会根据用户请求的方法种类调用其它几个文件中定义的方法来应答客户端的请求。
脚本处理	http_script.c	处理用户对 cgi 脚本的请求并返回执行结果
SSI 模块	http_include.c	处理客户端请求中的 SSI 代码，并返回执行结果
列表文件夹	http_dir.c	针对列表文件夹的处理
目录别名	http_alias.c	处理配置文件中设置的目录别名
存取控制和认证	http_access.c、http_auth.c	对客户端的访问请求进行验证
MIME	http_mime.c	针对 MIME 的处理

1.3 编译并运行Apache

根据 1.2 中给出的地址下载 Apache0.6.5。在您的服务中创建一个 devel 的用户（也可以使用其它非 root 用户，作者使用的是 devel 用户，书中内容针对 devel 用户，如果您使用的是其它用户，请做相应修改）。

在目录/home/devel 下创建 apache_0.6.5 目录，并将 1.2 节中下载的文件解压缩到该目录，此时的目录结构应该如下所示：

```
[devel@RIZI apache_0.6.5]$ pwd
/home/devel/apache_0.6.5
[devel@RIZI apache_0.6.5]$ ls
cgi-bin  conf  htdocs  icons  logs  src
[devel@RIZI apache_0.6.5]$
```

在目录/home/devel/apache_0.6.5/src 下执行 make，如果没有出错，输出应该类似于：

```
[devel@RIZI src]$ pwd
/home/devel/apache_0.6.5/src
[devel@RIZI src]$ make
gcc    -c -O2 http_config.c
…… …… ……
gcc    -c -O2 stream.c
gcc    -lcrypt -o httpd http_config.o httpd.o http_request.o util.o http_dir.o http_alias.o http_log.o http_mime.o http_access.o http_auth.o http_get.o http_post.o http_script.o http_include.o http_put.o http_delete.o stream.o
[devel@RIZI src]$ ls
```

```
http_access.c   http_auth.o ……  http_alias.o     httpd
[devel@RIZI src]$
```

编译成功，在目录下生成一堆.o 文件及 httpd 文件，httpd 文件即最终可执行的 Apache web 服务器程序。

Apache 运行需要一些配置文件，这些文件存放着 conf 目录中；logs 目录中存放 Apache 运行过程中产生的日志文件，htdocs 是您网站的根目录，cgi-bin 可以放置 cgi 脚本，icons 目录存放网站中可能使用到的图标，。这些目录名称可以和上面所列的名称不同，如果不同，需要重新设置 Apache 的配置文件并修改源代码中的相关配置信息，暂时不建议这样做。

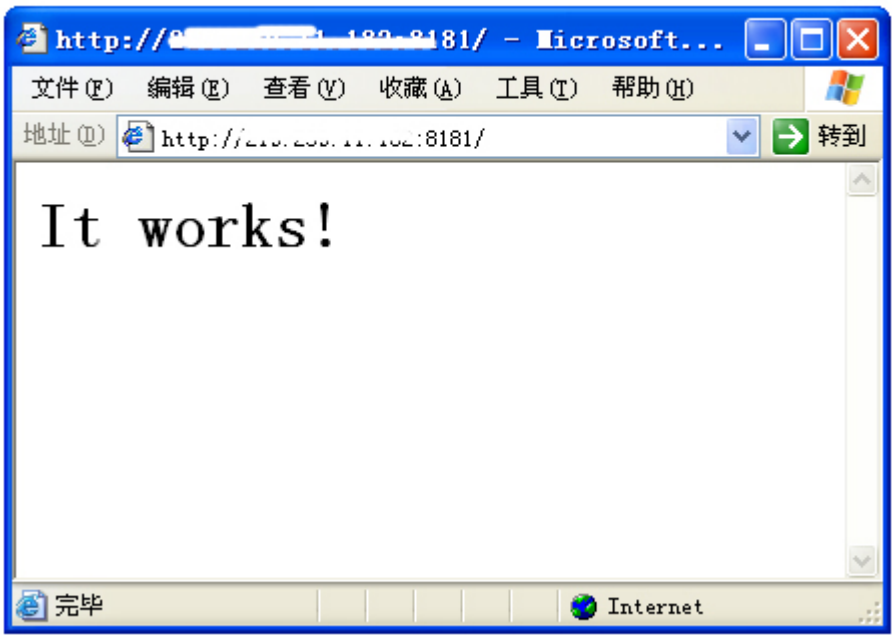
现在可以试着运行你的 Apache 了，在 src 目录下执行./httpd，可以看到结果类似于：

```
[devel@RIZI src]$ ./httpd
[devel@RIZI src]$ ps x
PID TTY      STAT   TIME COMMAND
 8739 ?        Ss     0:00 ./httpd
10248 pts/2    R+     0:00 ps x
……
[devel@RIZI src]$
```

此时Apache已经成功运行，您可以打开浏览器，在地址栏里面输入http://yourip:port/（如http://127.0.0.1:8181）来查看Apache的运行效果。

注：为了不合现有的服务冲突，在您下载的压缩包里，配置文件中配置的 Apache 的监听端口为 8181。

如果没有意外，将会看到这样的界面：



如果您没有看到上面的画面，请参照前面的步骤确定您的每个操作。

如果您看到了上面的画面，那么恭喜您，您的环境已经搭建成功，接下来，我们将要进入 Apache 源代码的神奇世界，下一张我们将要讲述 Apache 的主程序 httpd.c。Are you ready？Let’s go!

第 2 章 主程序

本章介绍了 httpd.c 程序的相关内容。这个程序是 Apache 的主程序，里面包括了 Apache 初始化的绝大部分调用，从这个文件里面我们能大概了解 Apache 前期准备工作的内容。

程序 httpd.c 里面涉及到 Linux 下 c 语言编程中的进程、守护进程、管道、信号量等几个重要概念，同时需要您理解 Socket 编程的一般流程。如果您对这些概念或流程比较陌生，请参考相关书籍。

2.1 守护进程

2.1.1 什么是守护进程

要理解守护进程，首先我们需要了解什么是终端。在 Linux 系统中，系统与用户进行交流的界面成为终端，每个从这个终端运行的程序会依附于这个终端，这个终端称为在其运行的程序的控制终端。当终端被关闭时，从这个终端运行的进程就会自动关闭。

终端是一个用户设备，它从用户接受键盘输入，并将这些输入发送到主机，主机处理用户输入的指令，并将处理的结果现实在这个终端的屏幕上。

Apache 运行起来的时候我们不希望它依赖于某个特定的终端，否则在这个终端关闭的时候我们的 Web 服务也就停止了，用户就再也不能访问我们的站点。

这时候我们就需要让 Apache 以守护进程的身份运行。那么什么是守护进程呢？

简单点说，守护进程就是脱离了特定终端，在后台运行的进程。

怎么让一个在终端上运行的程序成为守护进程呢？

2.1.2 创建守护进程

创建守护进行需要主要的 3 步：

1.创建子进程，父进程退出

通过 fork 调用创建子进程，父进程退出。这样新创建的子进程就成了孤儿进程，会被 init 进程收养，成为 init 进程的子进程。

从表现上看，程序立即返回，似乎程序已经运行完毕。

但是这个子进程并没有从实质上摆脱运行其“先父”进程的终端。

2.在子进程中创建新会话

这里又有两个比较重要的概念，进程组和会话。

进程组是一个或多个进程的集合，用唯一的进程组 ID 来标识，进程组开始时有一个组长进程，组长进程的进程 ID 就是组 ID，组长进程可能会退出，但它的退出并不影响进程组的组 ID。

会话是一个或多个进程组的组合，通常一个会话开始于用户登录，终止于用户退出这次登录，在用户退出登陆前运行的所有进程都属于这个会话。

或者，您是一个 web 程序员，那就一定用过 session 这个东东，其实一个会话可以理解成一个 session。巧的是会话也是译自 session 这个单词。

从步骤 1 里面我们知道，子进程从形式上脱离了其“先父”进程的终端，但实质上由

于 fork 调用继承了父进程的会话、进程组、控制终端，所以虽然父进程退出了，但会话、进程组、终端等并没有受到影响。这就需要我们的 setsid 函数出马了。

您可以通过在命令行中输入 man setsid 来查看对这个函数的最权威的描述。基本上这个函数有三重功效（够多的）：

- 让进程摆脱原会话的控制
- 让进程摆脱原进程组的控制
- 让进程摆脱原控制终端的控制

是的，这正是我们需要的。

是不是这样就完成了我们将程序置为守护进程的大任了呢？还没有！不只是因为我上面说需要三个步骤，主要还是因为 fork 调用惹的祸。

3.修改当前工作目录

这里需要说明一个概念，Linux 文件系统的挂载和卸载。

如果您习惯了使用 windows 系统，可能对这个概念没有直观的理解。比如您在 windows 系统里面，在光驱里面放了一张光盘，光驱开始读盘，你就可以在资源管理器里面点击光驱盘符来查看光盘里面的内容，当然，前提是光驱能正常把盘里面的内容读出来的话。

但是在 Linux 系统里面就没有这么风光了，你在光驱里面放上光盘之后，并不能直接读取光盘里面的内容，还需要把光驱设备挂载（mount）到某个挂载点上，然后通过这个挂载点来浏览光驱里面的内容。使用完了之后还需要卸载（unmount）光驱。

当然，当代的 Linux 会自动把光驱 mount 到固定的挂载点上（一般是/mnt/cdrom），这也是你能在放置完关盘后直接通过/mnt/cdrom 来访问光驱的原因。访问完了之后也可以通过图形界面里面右键菜单的弹出来结束对光驱的使用（此时系统调用了 unmount 指令）。

对其它外设的使用如 U 盘也是同样的操作，为什么要提到挂载和卸载呢？因为 fork 调用埋下了另外一个隐患，就是子进程继承了父进程的当前工作目录。

这可能造成一些问题，比如我们是通过 U 盘来启动的 Apache（也许您根本不会这么做），或者说 Apache 的工作目录在后继的操作中要卸载（这是无法预知的事情，谁也保不准哪块云彩下雨），很显然是卸载不掉的，因为“进程在运行时，当前目录所在的文件系统不能卸载”！那怎么办呢？其实方法很简单，修改当前工作目录。

我们可以通过调用 chdir 来修改当前的工作目录，这样就避免了上面提到的问题，修改到那个目录也是一个学问，可以像有人说的修改到/tmp 目录，这个目录不会卸载，但我还是建议您修改到根目录，也就是/目录，这个目录如果卸载了，整个系统就没法用了。

关于 chdir 的详细信息可以通过在命令行里面输入 man chdir 来获取。简单介绍如下：

原型：int chdir(const char *path);

功能：修改当前进程的工作目录到 path 目录。

举例：chdir("/");

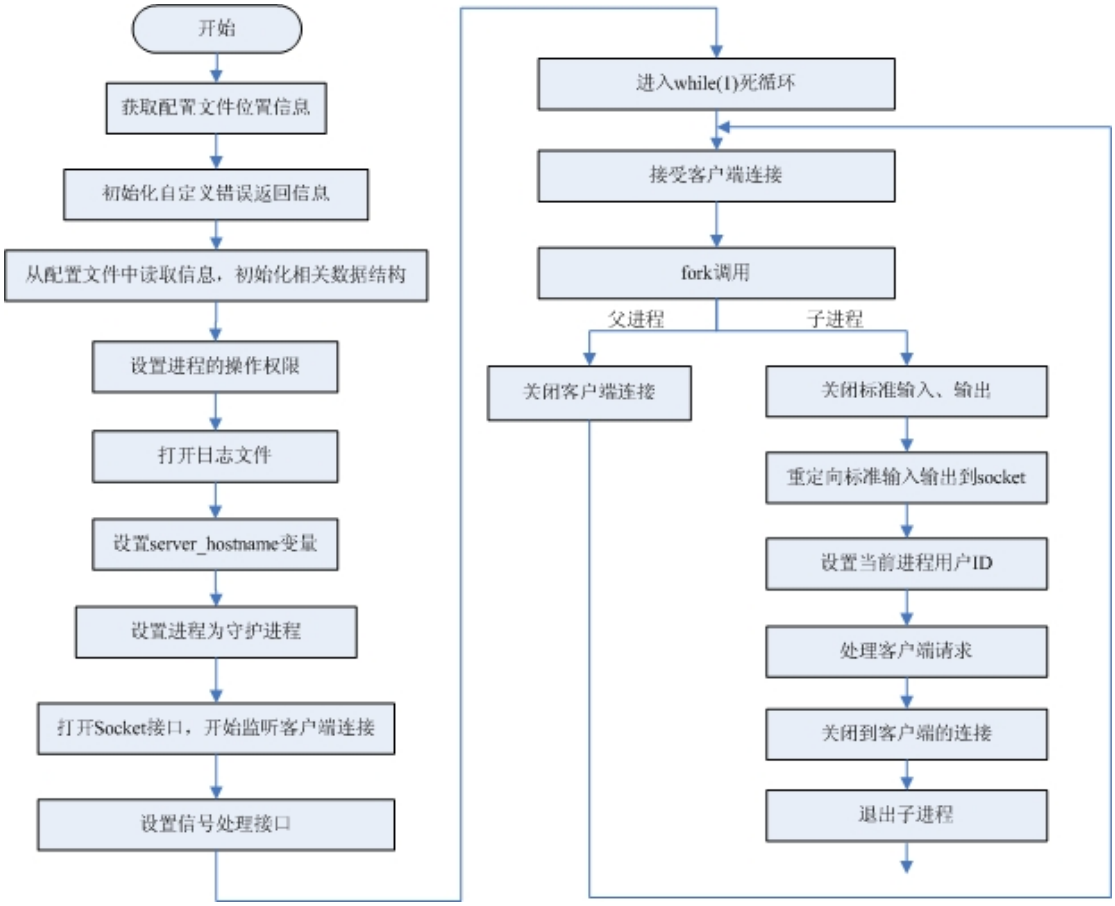
2.2 Apache生命周期

main 函数是整个 Apache 的主函数，以 main 函数为主线，通过跟踪 main 函数的调用，我们可以了解 Apache 的生命周期。

如果您跟着调用的层层深入，发现阅读代码有困难，可以暂时放下，在后继章节里面，我们会根据功能模块对 Apache 的各个部分做一个详细、深入、详细的讲解，等您看完全书后再回来看这个图，会有一种豁然开朗的感觉。

您可以先通过查看图 2-1 来对 Apache 的生命周期有一个感性的认识：

图 2-1 Apache 生命周期示意图



2.3 代码注释

从现在开始，我们将会接触到 Apache 的源代码。为了方便您和下载的源代码对照查看，作者在注释时提供了对应源代码的行号。为区分行号和正常代码，行号使用浅色字体显示。

2.3.1 httpd.c

```
     /* httpd.c: 简单的 http 守护进程，用来应答客户端请求 */
1    #include "httpd.h"
2     /* 显示程序执行参数提示 */
3    void usage(char*bin)
4    {
5          fprintf(stderr,"Usage: %s [-d directory] [-f file] [-v]\n",bin);
6          fprintf(stderr,"-d directory : specify an alternate initial ServerRoot\n");
7          fprintf(stderr,"-f file : specify an alternate ServerConfigFile\n");
8          exit(1);
9    }
```

```
10
11  void htexit(int status,FILE*out)
12  {
13      exit(status);
14  }
15
16  int sd ;        // 服务器监听 socket 描述符
17  pid_t pgrp ;    // 进程组 ID
18   /* 创建守护进程 */
19  void detach()
20  {
21      int x ;
22
23      chdir("/");             // 将工作目录修改成根目录
24      if((x=fork())>0)        // 父进程退出
25          exit(0);
26      else if(x==-1)          //fork 调用失败，给出提示
27      {
28          fprintf(stderr,"httpd: unable to fork new process\n");
29          perror("fork");
30          exit(1);
31      }
32      if((pgrp=setsid())==-1) //成为守护进程，参看 2.1.2 节描述
33      {
34          fprintf(stderr,"httpd: setsid failed\n");
35          perror("setsid");
36          exit(1);
37      }
38  }
39   /* 处理 SIGTERM 信号:记录日志、给进程组发送 SIGKILL 信号、关闭 socket */
40  void sig_term()
41  {
42      log_error("httpd: caught SIGTERM, shutting down");
43      killpg(pgrp,SIGKILL);
44      shutdown(sd,2);
45       /*关闭 socket 双方连接*/
46      close(sd);
47  }
48   /* 处理 SIGBUS 信号:记录日志、更改工作目录、退出程序。 */
49  void bus_error()
50  {
51      log_error("httpd: caught SIGBUS, dumping core");
52      chdir(server_root);
53      abort();
```

```
54  }
55   /* 处理 SIGSEGV 信号:记录日志、更改工作目录、退出程序 */
56  void seg_fault()
57  {
58      log_error("httpd: caught SIGSEGV, dumping core");
59      chdir(server_root);
60      abort();
61  }
62
63  void set_signals();
64
65   /* 设置组权限 */
66  static void set_group_privs()
67  {
68      if(!getuid())
69      {
70          char*name ;
             /* 如果从配置文件中得到的是 uid 的值，需要根据 UID 获取 username */
71          if(user_name[0]=='#')
72          {
73              struct passwd*ent ;
74              uid_t uid=atoi(&user_name[1]);
75
76              ent=getpwuid(uid);
77              if(ent==NULL)
78                  die(SERVER_ERROR,"couldn't determine user name from uid",
79                  stdout);
80              name=ent->pw_name ;
81          }
82          else name=user_name ;
83           /* 重设组属性:将 name 所属的所有组及 group_id 添加到当前运行进程的
                有效组
             */
84          if(initgroups(name,group_id)==-1)
85              die(SERVER_ERROR,"unable to setgroups",stdout);
86
87          if(setgid(group_id)==-1)      // 设置当前进程组 ID
88              die(SERVER_ERROR,"unable to change gid",stdout);
89      }
90  }
91   /* 重启服务 */
92  void restart()
93  {
94      log_error_noclose("httpd: caught SIGHUP, restarting"); // 记录日志
```

```
95          kill_mime();              // 清空 MIME 链表
96          kill_security();          // 清空目录存取权限控制数组
97          kill_indexing();          // 清空列表目录中初始化的链表
98          if(server_hostname)       // 确保 server_hostname 为空
99          {
100             free(server_hostname);
101             server_hostname=NULL ;
102         }
103         read_config(error_log);  // 读取配置文件
104         set_group_privs();        // 设置组权限
105         close_logs();             // 关闭日志文件
106         open_logs();              // 开启日志文件
107         log_error_noclose("httpd: successful restart"); // 记录重启成功日志
108         get_local_host();         // 获取本机 hostname
109         set_signals();            // 设置收到不同信号时的处理函数
110     }
111      /* 设置收到各种信号时的回调函数 */
112     void set_signals()
113     {
114         signal(SIGSEGV,(void(*)())seg_fault);
115         signal(SIGBUS,(void(*)())bus_error);
116         signal(SIGTERM,(void(*)())sig_term);
117         signal(SIGHUP,(void(*)())restart);
118         signal(SIGCHLD,SIG_IGN);
119     }
120      /* 记录启动的时间。但是仅限于这个函数内部知道 */
121     void speed_hack_libs()
122     {
123         time_t dummy_time_t=time(NULL);
124         struct tm*dummy_time=localtime(&dummy_time_t);
125         struct tm*other_dummy_time=gmtime(&dummy_time_t);
126         char buf[MAX_STRING_LEN];
127
128         strftime(buf,MAX_STRING_LEN,"%d/%b/%Y:%H:%M:%S",dummy_time);
129     }
130      /* 运行主函数 */
131     void standalone_main()
132     {
133         int csd,clen,pid,one=1 ;
134         struct sockaddr_in sa_server ;
135         struct sockaddr sa_client ;
136
137         detach();  // 创建守护进程
138
```

```
139         if((sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))==-1)
140         {
141             fprintf(stderr,"httpd: could not get socket\n");
142             perror("socket");
143             exit(1);
144         }
145
146         if((setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,
147             (const char*)&one,sizeof(one))) ==-1)
148         {
149             fprintf(stderr,"httpd: could not set socket option\n");
150             perror("setsockopt");
151             exit(1);
152         }
153         bzero((char*)&sa_server,sizeof(sa_server));
154         sa_server.sin_family=AF_INET ;
155         sa_server.sin_addr.s_addr=htonl(INADDR_ANY);
156         sa_server.sin_port=htons(port);
157         if(bind(sd,(struct sockaddr*)&sa_server,sizeof(sa_server))==-1)
158         {
159             fprintf(stderr,"httpd: could not bind to port %d\n",port);
160             perror("bind");
161             exit(1);
162         }
163         listen(sd,128);       // 监听端口，最多允许 128 个排队请求
164
165         speed_hack_libs(); // 记录启动时间
166         set_signals();        // 设置信号处理回调函数
167         log_pid();            // 将当前进程的进程 ID 写入指定文件
168
169         while(1)
170         {
171     retry :
172         clen=sizeof(sa_client);
173         if((csd=accept(sd,&sa_client,&clen))==-1)
174         {
175             if(errno==EINTR)  // 中断错误
176             {
177                 goto retry ;
178             }
179             log_error("socket error: accept failed");
180             goto retry ;
181         }
182         if((pid=fork())==-1)
```

```
183             log_error("unable to fork new process");
184         else if(!pid)           // 子进程
185         {
186             struct passwd*pwent ;
187             close(0);       // 关闭标准输入
188             close(1);       // 关闭标准输出
189             dup2(csd,0);    // 重定向标准输入到 csd
190             dup2(csd,1);    // 重定向标准输出到 csd
191             close(sd);
192             (void)fcntl(csd,F_SETFD,(int)FD_CLOEXEC);
193             if(!getuid())   // 只有以 root 用户运行的时候才执行下列代码
194             {
195                 if(setuid(user_id)==-1)
196                     die(SERVER_ERROR,"unable to change uid",stdout);
197             }
198             process_request(0,stdout);    // 处理请求
199             fclose(stdin);
200             fclose(stdout);
201             shutdown(csd,2);              // 不允许继续发送和接收数据
202             close(csd);
203             exit(0);
204         }
205         close(csd);
206         }
207     }
208
209     extern char*optarg ;
210      /* Apache 主函数 */
211     int main(int argc,char*argv[])
212     {
213         int c,init ;
214         strcpy(server_root,HTTPD_ROOT);
              /* 获取服务器配置文件位置 */
215         make_full_path(server_root,SERVER_CONFIG_FILE,server_confname);
              /* 根据不同参数运行，参数含义参看 usage()函数 */
216         while((c=getopt(argc,argv,"d:f:v"))!=-1)
217         {
218             switch(c)
219             {
220             case 'd' :
221                 strcpy(server_root,optarg);
222                 make_full_path(server_root,SERVER_CONFIG_FILE,
                                        server_confname);
223                 break ;
```

```
224                 case 'f' :
225                     strcpy(server_confname,optarg);
226                     break ;
227                 case 'v' :
228                     printf("Apache httpd version %s.\n",SERVER_VERSION);
229                     exit(1);
230                 case '?' :
231                     usage(argv[0]);
232                 }
233         }
             // 初始化自定义错误代码(如 404)信息
234         for(init=0;init<=RESPONSE_CODES;init++)
235             response_code_strings[init]=NULL ;
236
237         read_config(stderr);      // 读取配置
238         set_group_privs();        // 设置组权限
239         open_logs();              // 打开日志文件
240         get_local_host();         // 获取 server_hostname
241
242         standalone_main();        // 调用运行主函数
243         fclose(stdin);
244         fclose(stdout);
245         exit(0);
246     }
```

2.4 小结

本章代码完整展示了 Apache 在整个生命周期中的所作所为，当然，如果您要了解细节就不得不跟着一层层的函数调用去探寻究竟。如果您真的能够一直走下去，那当然好，这本书对您也就没有价值了，如果您调了几层之后有些迷糊或者觉得焦躁不安，没有关系，这很正常，这也是本书的价值所在。

我们在后继章节里面会详细描述 Apache 的各个模块和工具函数库，不要着急，不要着急，休息，休息一会儿。

俗话说，工欲善其事，必先利其器。在第三章，我们会先拿工具函数开刀，一个是因为这些自定义函数个体都比较简单，没有很多对其他自定义函数的深层调用，另外一个是在了解了工具函数提供的功能之后，我们再遇到对他们的调用的时候，仅知道他们实现什么功能即可，不必再深层探讨他们的实现了。

第 3 章 自定义库函数

本章描述 Apache0.6.5 的自定义函数库。这些函数供实现 Apache 功能模块的代码调用。本章代码可以按照功能分成两类，一类实现对文件的读操作进行缓冲，包括 stream.h 和 stream.c；另外一类由单独的 util.c 文件组成，实现了一些供其它函数调用的工具函数。

3.1 背景知识

本章涉及到 4 个相关知识点：url 编/解码，时间格式，夏令时以及 BASE64 编码，了解这些知识点是您能够理解代码的前提，如果您具备了相关知识，可以跳过相关的小节。下面我们对这几个背景知识进行简单说明。

3.1.1 URL编码/解码

发送给服务器的URL请求中，如果含有非数字字母数据的，浏览器会编码后传递，比如您在注册个人信息的时候，需要填写您的中文名和公司英文名两项数据，如果您的中文名字是张三，公司英文名字是Microsoft Corporation，请求方法是GET，表单中中文名字的输入框的name属性是cname，公司英文名输入框的name属性是corpname，服务器端处理表单数据的文件是 /user/reg.php ，那么浏览器发送的请求将类似于 http://www.server.com/user/reg.php?cname=%D5%C5%C8%FD&corpname=Microsoft%20Corporation

其中%D5%C5%C8%FD 就是“张三”两个汉字 URL 编码后的效果，而公司名称中的%20 就是空格字符 URL 编码后的效果。

由于浏览器的这个特性，服务器需要在接收数据之后进行 URL 解码操作，同时服务器提供了编码操作。

编码和解码操作是通过函数 escape_url 和 unescape_url 完成的。

编码的方法是：不是字母或数字的字符将被替换成百分号（%）后跟其 16 进制 ASCII 编码。由于历史原因，将加号（+）编码成%20（这点和 RFC1738 不兼容）。

解码就是编码的逆向工程。

3.1.2 时间格式

在 HTTP 协议里面用到了几个时间格式，在 Apache0.6.5 里面除了用户自定义时间格式外，还涉及到一下三种：

ctime 格式时间：Fri Mar 13 13:58:46 2009

RFC860 格式时间：Monday, 15-Aug-05 15:52:01

RFC822 格式时间：Mon, 15 Aug 2005 15:52:01

3.1.3 夏令时

夏令时是一种为节约能源而人为规定地方时间的制度。在这一制度实行期间所采用的统

一时间称为“夏令时间”。一般在天亮早的夏季人为将时间提前一小时，可以使人早起早睡，减少照明量，以充分利用光照资源，从而节约照明用电。中国曾在 1986 年到 1991 年实行过夏令时，80 前的同学们应该还有印象。

在 C 语言中，结构 tm 中的域 tm_isdst 来标识夏令时，实行夏令时的时候，tm_isdst 为正；不实行夏令时的进候，tm_isdst 为 0；不了解情况时，tm_isdst()为负。

结构 tm 的详细组成及其含义可以在/usr/include/time.h 中找到。在此不再赘述。

3.1.4 BASE64 编码

BASE64 编码最早使用在邮件传输上，由于历史原因，Email 被设计成只允许传送 ASCII 字符，这样一来，如果您发送了一封带有非 ASCII 字符的 Email 通过有“历史原因”的网关就可能出现问题：问题网关有可能将每个字节的最高位置为 0。

此时我们就需要对邮件中传输的信息进行编码，使之全部用 ASCII 字符表示。

举例说明，您可以给自己发一封邮件，标题是 test，内容简单些，只有两个字“张三”，这时邮箱收到的数据是什么呢？

我是通过 Firefox 接收的邮件（OutLook 类似），从 Firefox 里面看起来一切正常，显示发件人，收件人，邮件内容，但是网关上传输的数据并不是您看起来这么简单。

有一个方法可以知道网关上传输的数据内容都有那些，在 Firefox 里面选中这封邮件，从菜单中选择“文件”，从子菜单里面选择“导出邮件”，格式选择“OutLook 邮件（*.eml）”，随便起一个名字，比如 abc.eml。用文本编辑器打开这个邮件，你会发现内容有点“变态”：

X-Kaspersky: Original server data starting here: +OK 2303 octets

……

Subject: test

From: Lee tsingien <***@***.com>

To: ***@***.com

Content-Type: text/plain; charset=GB2312

Content-Transfer-Encoding: base64

……

1cXI/Q==

从高亮部分可以看到，传输的编码类型是 BASE64，那邮件内容呢？就是“1cXI/Q==”，这就是“张三”两个汉字的 BASE64 编码结果。看到了，全是 ASCII 字符，这样就跟有“历史问题”的网关兼容了。

很神奇是不是，您也许要问，它是怎么“编”的呢？

其实很简单，将需要编码的二进制数据每次取 3 个 byte，顺序放入一个 24bit 的缓冲区中，不足 3byte，将缓冲区中剩余部分补 0。然后每次从缓冲区中取出 6 个 bit，根据 RFC2045 中定义的码表得到对应的字符即可。

比如我们的张三，16 进制数据就是 D5C5C8FD，转换成二进制就是 11010101110001011100100011111101。每次取 6 个 bit，取出的二进制序列转换成十进制就是 53（110101）、28（011100）、23（010111）、8（001000）、63（111111）、16（010000），在表 3-1 中找到对应的字符就是 1cXI/Q

你也看到了，似乎不对，因为邮件里面的数据比我们操作的结果多出两个等号（=），这是怎么回事呢？

因为上面的编码规则说的并不完整，相信您也看到了，规则里面是“每次取 3 个 byte”，而需要编码的数据的字节数不一定是 3 的倍数啊，就像我们的字节数是 4 字节，不满足条件，

这该怎么处理呢？

补充规则：在编码的字符串后面添加“3-(原文字节数 MOD 3)”个等号（=）。在我们的例子里面，就是添加 3-(4mod3)=2 个等号。

所以最终编码结果为 1cXI/Q==，也就是您在邮件里面看到的内容。

表 3-1 The Base64 Alphabet

值	编码	值	编码	值	编码	值	编码
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

说了这一通，全是和邮件相关的，我们在讨论的是 Web 服务器啊。别忙，如果您看一下解码函数 uudecode 调用位置您就明白了，它是为第九章将要讲到的 Basic 认证方式服务的，这种认证方式从客户端传递过来的认证数据也是 BASE64 编码的。

说点题外话，其实在 IE 浏览器里面还有一个地方用到了 BASE64 编码了，在 IE6 之后的版本里面提供了一个功能，就是把网页保存成 mht 格式。

您可以用IE打开百度首页http://www.baidu.com，然后在“文件”菜单中选择“另存为”，在“保存类型”中选择“Web档案，单一文件（*.mht）”选项，然后指定一个文件名，保存成功后您会发现，虽然百度的首页里面包含的图片，IE只生成了一个.mht文件。然后您可以断开网络连接，在脱机的情况用IE打开这个mht文件，您会发现百度的LOGO图片也能正常显示，很奇妙？其实您可以用文本编辑器打开这个mht文件，其中有类似这样的一段信息：

------=_NextPart_000_0000_01C9C495.CF9C2DA0

Content-Type: image/gif

Content-Transfer-Encoding: base64

Content-Location: http://www.baidu.com/img/baidu_logo.gif

R0lGODlhDgGBALMAAPHs98zK91RM5JGM7nBq6O1raPfBv+lEQfSdmyMZ3OEGAf///wAAAAAAAAAA

……

2rIJEQAAOw==

看到了熟悉的==？再看看上面，注明了Content-Type是image/gif，编码方式也是BASE64，聪明的您一定想到了，原来LOGO文件http://www.baidu.com/img/baidu_logo.gif 的

二进制信息已经通过BASE64 编码保存在mht文件中了，所以即便是脱机，您也可以看到这个图片。

如果您有兴趣，可以把这部分内容解码成二进制，然后把二进制信息写入到一个文件中，并把这个命名为 baidu_logo.gif，使用图片查看工具查看，看看是不是您当初保存的百度的 LOGO 图片。

3.2 代码注释

3.2.1 stream.h、stream.c

您可能听说过包裹函数（或者叫包装函数），实际上就是对常用操作的一个封装，比如原子日志，在多进程或多线程同时操作一个日志文件的时候，可能会有竞争问题存在，为了避免竞争，可以设计一个原子日志系统，这个系统提供给调用者几个简单的接口（函数）来完成原子日志的记录，而竞争问题的解决、文件读写的错误、信号的处理等等都在原子日志系统里面完成，对调用者来说是透明的。

同样 stream.h 和 stream.c 就提供了几个包裹函数，主要用来对文件读操作进行缓冲，这对 socket 这种特殊文件来说是很有意义的。

1. stream.h

```
1   #ifndef STREAM_INC
2   #define STREAM_INC
3    /*推荐的 buffer 大小 */
4   #define STREAM_BUFSIZE (5840) // 4 个以太网包的大小 1460*4=5840
5
6   typedef struct
7   {
8       int cnt ;                   //   buffer 里面可用的字符数
9       unsigned char*ptr ;         // 缓冲区当前读取位置指针
10      unsigned char*base ;        // 缓冲区基地址（缓冲区开始位置指针）
11      int size ;                  // 缓冲区大小
12      int file ;                  // UNIX 系统文件描述符
13      int isnl ;                  // 是新的一行吗？
14  }
15  STREAM ;
16   /* 初始化和释放结构体实例 */
17  STREAM*mkstream(size_t buffsize);
18  void freestream(STREAM*sp);
19   /* 把一个文件描述符和一个结构体实例关联起来 */
20  void sdopen(int fd,STREAM*sp);
21   /* 提供读操作的包裹函数，从流里面读取一行 */
22  int sgets(unsigned char*s,int n,unsigned int timeout,STREAM*sp);
23  #endif
```

2. stream.c

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

```
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>

#include "stream.h"
 /*先声明，实现之前在 sget 中调用了，其实可以把实现放在这里，省掉声明部分 */
static int nread(int fd,unsigned char*buff,int n,int*isnl,
                        unsigned int timeout);
 /* 创建一个 stream 结构实例，成功返回只想这个实例的指针，失败返回 NULL */
STREAM*mkstream(size_t buffsize)
{
      STREAM*sp ;
       /* 未指定缓冲区大小则使用默认大小 */
      if(buffsize==0)buffsize=STREAM_BUFSIZE ;
       /* 给实例分配空间 */
      sp=malloc(sizeof(STREAM));
      if(sp==NULL)return NULL ;
       /* 给缓冲区分配空间 */
      sp->base=malloc(buffsize);
      if(sp->base==NULL)
      {
            free(sp);
            return NULL ;
      }
      sp->cnt=0 ;          //缓冲区剩余 0 字节
      sp->ptr=sp->base ; //当前读取位置为缓冲区开始位置
      sp->size=buffsize ; //缓冲区大小为指定或默认大小
      sp->file=-1 ;        //未关联任何文件
      sp->isnl=1 ;         //是新行

      return sp ;
}
 /* 释放指定的 stream 实例 */
void freestream(STREAM*sp)
{
      free(sp->base);
      free(sp);
}
 /* 将给定的文件描述和指定的 stream 实例挂钩 */
```

```
46  void sdopen(int fd,STREAM*sp)
47  {
48      sp->cnt=0 ;
49      sp->ptr=sp->base ;
50      sp->file=fd ;
51      sp->isnl=1 ;
52  }
53
    /*
     * 从 stream 实例缓冲区中读取数据，返回读取的数据字节数。
     * 下面任何一个条件满足则停止读取
     * 1.读取到了 n 个数据或者 2.读到了一行数据。
     * 如果缓冲区中没有数据可读，则调用 nread 刷新缓冲区后重新读取
     * 读取或刷新缓冲区同时更新 stream 实例的各个属性。
     * 返回值：-1：从文件中读取数据失败；其他值：读取到的字节数
     */
54  int sgets(unsigned char*buff,int n,unsigned int timeout,STREAM*sp)
55  {
56      int i,j,r,done,ch ;
57
58      j=0 ;       //目的缓冲区当前写入位置
59      done=0 ; //是否已经满足的读取结束条件
60
61      r=0 ;       //从 stream 实例指定的文件中读取到的字节数
62
63      for(;;)
64      {
65          for(i=0;i<sp->cnt;)                    // 缓冲区中有数据
66          {
67              ch=sp->ptr[i++];                   // 缓冲区中的当前位置字符
68              if(ch=='\012')                     // 是\n
69              {
70                  if(j==0)buff[j++]='\n' ;       // 是个空行
                     /* 前一个字符是\r，把它替换成\n */
71                  else if(buff[j-1]=='\015')buff[j-1]='\n' ;
72                  else if(j<n-1)buff[j++]='\n' ; // 未读取到指定字节数
73                  else i--;                      // 读取完成
74                  done=1 ;
75                  break ;
76              }
77              if(j==n-1)          // 读取到了指定字节数
78              {
79                  i--;
80                  done=1 ;
```

```
81                          break ;
82                      }
83
84                      buff[j++]=ch ;      // 把读到的字符放入用户指定的缓冲区中
85                  }
86                  sp->cnt-=i ;            // 更新剩余字符数量
87                  sp->ptr+=i ;            // 移动当前读取位置指针
88                  if(done)break ;         // 如果满足读取要求，就退出循环
89
90                  sp->ptr=sp->base ;      // 重置当前读取位置指针到缓冲区开始位置
91                  /* 从文件（socket）中读取 sp->size 个字符 */
92                  r=nread(sp->file,sp->base,sp->size,&sp->isnl,timeout);
93                  if(r== -1)break ;       // 读取错误
94                  sp->cnt=r ;             // 更新剩余字符数量为读到缓冲区中字符数量
95                  if(r==0)break ;         // 读到了 EOF
96          }
97
98          buff[j]='\0' ;                  // 设置目标缓冲区最后一个字符为字符串结束符
99          if(r== -1)return-1 ;            // 读取文件描述符错误
100         else return j ;                 // 返回读取到的字节数
101     }
102
        /*
         * 将文件描述符 fd 中的数据读取 n 个字节到 buffer 里面。如果读取到一个空行或
         * 者遇到一个 EOF 标志则停止读取。
         * isnl 非零的话说明刚读到一个(CR)LF。
         * 发生错误的时候返回-1，遇到 EOF 的时候返回 0，正常返回读取的字节数。
         */
103     static int nread(int fd,unsigned char*buff,int n,int*isnl,unsigned int timeout)
104     {
105         fd_set readfds ;
106         struct timeval timeouts ;
107         int i,j,r ;
108
109         FD_ZERO(&readfds);              //将文件描述符集 readfds 清空
110
111         do                              //等待数据到达
112         {
113             timeouts.tv_sec=timeout ;   //设置超时时间
114             timeouts.tv_usec=0 ;
115             FD_SET(fd,&readfds);
116             r=select(fd+1,&readfds,NULL,NULL,&timeouts);
117         }
118         while(r==-1&&errno==EINTR);
```

```
119         if(r==-1)return-1 ;
120
121         if(r==0)   // 等待超时
122         {
123             errno=ETIMEDOUT ;
124             return-1 ;
125         }
126         do r=recv(fd,(char*)buff,n,MSG_PEEK); // 检测 fd 上到达的数据
127         while(r==-1&&errno==EINTR);
128         if(r==-1)return-1 ;
129          /* 从到达的数据中查找 LFLF 或 LFCRLF */
130         if(isnl)j=-1 ;
131         else j=-3 ;
132
133         for(i=0;i<r;i++)
134         {
135             if(buff[i]=='\012')   // 当前数据是\n
136             {
137                 if(i-j<=2&&(i-j==1||buff[i-1]=='\015'))
138                 {
139                     i++;
140                     break ;
141                 }
142                 else
143                     j=i ;
144             }
145         }
146         for(j=0;j<i;)
147         {
148             r=read(fd,&buff[j],i-j);
149             if(r==-1)
150             {
151                 if(errno!=EINTR)return-1 ;
152             }
153             else if(r==0)
154                 break ;
155             else
156                 j+=r ;
157         }
158         if(j>0)*isnl=(buff[j-1]=='\012');
159         return j ;
160     }
```

3.2.2 util.c

```
1   #include "httpd.h"
2
3   #include "stream.h"
4
    /*
     * 功能:获取当前时间
     * 返回值:类似于 Fri Mar 13 13:58:46 2009
     */
5   char*get_time()
6   {
7       time_t t ;
8       char*time_string ;
9
10      t=time(NULL);
11      time_string=ctime(&t);
12      time_string[strlen(time_string)-1]='\0' ;
13      return(time_string);
14  }
15
    /*
     * 功能:格式化时间成 http 格林尼治时间格式
     * 参数:sec->需要被格式化的时间
     * 返回值:格式化后的时间字符串
     */
16  char*gm_timestr_822(time_t sec)
17  {
18      return ht_time(sec,HTTP_TIME_FORMAT,1);
19  }
20
    /*
     * 功能:根据给定的格式，格式化给定的时间
     * 参数:t->需要格式化的时间
     *      fmt->格式化字符串
     *      gmt->是否格式化成格林尼治时间
     * 返回值:格式化后的字符串
     */
21  char*ht_time(time_t t,char*fmt,int gmt)
22  {
23      static char ts[MAX_STRING_LEN];
24      struct tm*tms ;
25
26      tms=(gmt?gmtime(&t):localtime(&t));
```

```
27
28          strftime(ts,MAX_STRING_LEN,fmt,tms);
29          return ts ;
30      }
31
         /*
         * 功能:根据时区以及夏令时修正当前时间
         * 参数:修正前的时间
         * 返回值:修正后的时间
         */
32      struct tm*get_gmtoff(long*tz)
33      {
34          time_t tt ;
35          struct tm*t ;
36
37          tt=time(NULL);
38          t=localtime(&tt);
39          *tz=-timezone ;
40          if(t->tm_isdst)   //夏令时调整，参看 3.1.3 介绍
41              *tz+=3600 ;
42          return t ;
43      }
44       /*月份简称数组  */
45      static char*months[]=
46      {
47          "Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"
48      }
49      ;
50
        /*
         * 功能:根据给定的月份简称，获取月份的整数值
         * 参数:月份简称，如 Mar
         * 返回值:给定简称对应的月份数值，如给定 Mar 返回 2
         */
51      int find_month(char*mon)
52      {
53          register int x ;
54
55          for(x=0;x<12;x++)
56              if(!strcmp(months[x],mon))
57                  return x ;
58              return-1 ;
59      }
60
```

```
        /*
        * 功能: 判断给定的时间 lms 是否比 ims 字符串表示的时间更晚
        * 参数: lsm->给定要比较的时间
        *       ims=If-modified-since，时间字符串
        * 返回值:0-> lsm > ims
        *         1-> lsm <= ims
        */
61      int later_than(struct tm*lms,char*ims)
62      {
63          char*ip ;
64          char mname[MAX_STRING_LEN];
65          int year=0,month=0,day=0,hour=0,min=0,sec=0,x ;
            /* 下面涉及到几种时间格式，但是不论那种格式
             * 都是以星期开始。 提到的时间格式包括:
             * ctime:Fri Mar 13 13:58:46 2009
             * RFC860:Monday, 15-Aug-05 15:52:01
             * RFC822:Mon, 15 Aug 2005 15:52:01
             * 下面四行的操作是跳过前面的星期及其后的空格。
             */
66          if(!(ip=strchr(ims,' ')))
67              return 0 ;
68          else
69              while(isspace(*ip))
70                  ++ip ;
71
72              if(isalpha(*ip))
73              {
                    // ctime 格式:Mar 13 13:58:46 2009
74                  sscanf(ip,"%s %d %d:%d:%d %*s %d",
                    mname,&day,&hour,&min,&sec,&year);
75              }
76              else if(ip[2]=='-')
77              {
78                  char t[MAX_STRING_LEN];
                    // rfc850 时间格式:15-Aug-05 15:52:01
79                  sscanf(ip,"%s %d:%d:%d",t,&hour,&min,&sec);
80                  t[2]='\0' ;
81                  day=atoi(t);
82                  t[6]='\0' ;
83                  strcpy(mname,&t[3]);
84                  x=atoi(&t[7]);
85                  if(x<70)
86                      x+=100 ;
87                  year=1900+x ;
```

```
88              }
89              else    // RFC 822 时间格式:15 Aug 2005 15:52:01
90              {
91                  sscanf(ip,"%d %s %d %d:%d:%d",
                    &day,mname,&year,&hour,&min,&sec);
92              }
93              month=find_month(mname); //对应的整数月份
94              if((x=(1900+lms->tm_year)-year))
95                  return x<0 ;
96              if((x=lms->tm_mon-month))
97                  return x<0 ;
98              if((x=lms->tm_mday-day))
99                  return x<0 ;
100             if((x=lms->tm_hour-hour))
101                 return x<0 ;
102             if((x=lms->tm_min-min))
103                 return x<0 ;
104             if((x=lms->tm_sec-sec))
105                 return x<0 ;
106
107             return 1 ;
108     }
109
        /*
        * 功能: 判断字符串 str 是否符合 exp 表达式
        * 参数:str->要判断的字符串
        *       exp->检测表达式
        * 返回值:0->符合
        *          1->不符合
        *          -1->出错
        */
110     int strcmp_match(char*str,char*exp)
111     {
112         int x,y ;
113
114          for(x=0,y=0;exp[y];++y,++x)
115          {
116              if((!str[x])&&(exp[y]!='*'))
117                  return-1 ;
118             if(exp[y]=='*')
119             {
120                 while(exp[++y]=='*');
121                 if(!exp[y])
122                     return 0 ;
```

```
123                     while(str[x])
124                     {
125                         int ret ;
126                         if((ret=strcmp_match(&str[x++],&exp[y]))!=1)
127                             return ret ;
128                     }
129                     return-1 ;
130                 }
131                 else
132                     if((exp[y]!='?')&&(str[x]!=exp[y]))
133                         return 1 ;
134             }
135             return(str[x]!='\0');
136         }
137
            /*
             * 功能:判断给定的字符串中是否含有*或者?
             * 参数:str->需要判断的字符串
             * 返回值: 1->含有字符*或?
             *          0->不含有字符*或?
             */
138         int is_matchexp(char*str)
139         {
140             register int x ;
141
142             for(x=0;str[x];x++)
143                 if((str[x]=='*')||(str[x]=='?'))
144                     return 1 ;
145                 return 0 ;
146         }
147
            /*
             * 功能:将目标字符串中前 start 个字符去除，然后将源字符串中的字符附加到目标
             * 字符串剩余的字符前面。组成新的目标字符串
             * 参数:start->开始删除位置
             *      dest->目标字符串，即需要处理的字符串
             *      src->附加字符串
             */
148         void strsubfirst(int start,char*dest,char*src)
149         {
150             int src_len,dest_len,i ;
151
152             if((src_len=strlen(src))<start)
153             {
```

```
154             for(i=0;dest[i]=src[i];i++);
155             for(i=src_len;dest[i]=dest[i-src_len+start];i++);
156         }
157         else
158         {
159             for(dest_len=strlen(dest),i=dest_len+src_len-start;i>=src_len;i--)
160                 dest[i]=dest[i-src_len+start];
161             for(i=0;i<src_len;i++)dest[i]=src[i];
162         }
163     }
164
    /*
     * 功能:如果目录中含有..这样的路径，则变换成实际路径。
     *       如/0/1/../11/abc.html 经过转换后成为/0/11/abc.html
     * 参数:需要转换的路径名称
     */
165  void getparents(char*name)
166  {
167      int l=0,w=0 ;
168      const char*lookfor=".." ;
169
170      while(name[l]!='\0')
171      {
172          if(name[l]!=lookfor[w])(w>0?(l-=(w-1),w=0):l++);
173          else
174          {
175              if(lookfor[++w]=='\0')
176              {
177                  if((name[l+1]=='\0')||(name[l+1]=='/')&&
178                      (((l>3)&&(name[l-2]=='/'))||(l<=3)))
179                  {
180                      register int m=l+1,n ;
181
182                      l=l-3 ;
183                      if(l>=0)
184                      {
185                          while((l!=0)&&(name[l]!='/'))--l ;
186                      }
187                      else l=0 ;
188                      n=l ;
189                      while(name[n]=name[m])(++n,++m);
190                      w=0 ;
191                  }
192                  else w=0 ;
```

```
193                     }
194                     else++l ;
195                 }
196         }
197     }
198
        /*
         * 功能:将字符串(路径)中//替换成/
         * 参数:需要替换的字符串
         */
199     void no2slash(char*name)
200     {
201         register int x,y ;
202
203         for(x=0;name[x];x++)
204             if(x&&(name[x-1]=='/')&&(name[x]=='/'))
205                 for(y=x+1;name[y-1];y++)
206                     name[y-1]=name[y];
207     }
208
        /*
         * 功能:将源路径的前 n 个目录填充到目标字符串里面。
         *       如 make_dirstr("/usr/local/apache", 3, dest);
         *       执行后 dest = /usr/local
         * 参数:s->源路径;n->前几个;d->目标字符串
         */
209     void make_dirstr(char*s,int n,char*d)
210     {
211         register int x,f ;
212
213         for(x=0,f=0;s[x];x++)
214         {
215             if((d[x]=s[x])=='/')
216                 if((++f)==n)
217                 {
218                     d[x]='\0' ;
219                     return ;
220                 }
221         }
222         d[x]='\0' ;
223     }
224
        /*
         * 功能:计算给定路径目录层数(深度)
```

```
         * 参数:指定的路径
         * 返回值:路径参数(深度)
         */
225      int count_dirs(char*path)
226      {
227          register int x,n ;
228
229          for(x=0,n=0;path[x];x++)
230              if(path[x]=='/')n++;
231              return n ;
232      }
233
         /*
         * 功能:将源路径赋值给目标路径，如果源路径最后一个字符不为/，则在目标路
         * 径后面添加一个/
         * 参数:s->源路径;d->目标路径
         */
234      void strcpy_dir(char*d,char*s)
235      {
236          register int x ;
237
238          for(x=0;s[x];x++)
239              d[x]=s[x];
240
241          if(s[x-1]!='/')d[x++]='/' ;
242          d[x]='\0' ;
243      }
244
         /*
         * 功能:将工作目录设置成 file 所在的路径
         * 参数:file->要新的工作目录或者其下的一个文件路径
         */
245      void chdir_file(char*file)
246      {
247          int i ;
248
249          if((i=rind(file,'/'))==-1)
250              return ;
251          file[i]='\0' ;
252          chdir(file);
253          file[i]='/' ;
254      }
255
         /*
```

```
        * 功能:将给定字符串转换成大写并在前面增加 HTTP_。如果字符串中含有'-'，则
        * 转换成'_'。
        * 参数:需要操作的字符串
        */
256     void http2cgi(char*w)
257     {
258         register int x ;
259
260         for(x=strlen(w);x!=-1;--x)
261             w[x+5]=(w[x]=='-'?'_':toupper(w[x]));
262         strncpy(w,"HTTP_",5);
263     }
264
        /*
        * 功能:记录错误信息，关闭标准输入、输出，退出程序执行
        */
265     void getline_timed_out()
266     {
267         char errstr[MAX_STRING_LEN];
268
269         sprintf(errstr,"timed out waiting for %s",remote_name);
270         log_error(errstr);
271         fclose(stdin);
272         fclose(stdout);
273         exit(0);
274     }
275
276     static STREAM getlinestr ;
277     static unsigned char getlinebuff[8192];
278
       /*
        * 功能:使用文件描述符 fd 初始化 getlinestr 结构。
        * 参数:fd->和 STREAM 结构 getlinestr 挂钩文件的文件描述符
        * 注意: 该函数必须在 getlines()调用前调用
        */
279     void getlineinit(int fd)
280     {
281         getlinestr.base=getlinebuff ;
282         getlinestr.size=8192 ;
283         sdopen(fd,&getlinestr);
284     }
285
        /*
        * 功能:在超时时间 timeout 到达之前，从和 getlinestr 关联的文件里面读取 n 个字
```

```
         * 节放到 s 里面。
         * 参数:s->接收读取数据的 buffer
         *       n->计划读取的字符个数
         *       f->很遗憾，这个参数没有使用
         *       timeout->超时时间
         * 返回值:0 正确读写，1，没有读到数据或发生错误
         */
286      int getline(char*s,int n,int f,unsigned int timeout)
287      {
288          int rv ;
289
290          rv=sgets((unsigned char*)s,n,timeout,&getlinestr);
291          if(rv==-1)
292          {
293              if(errno==ETIMEDOUT)getline_timed_out();
294              s[0]='\0' ;
295              return 1 ;
296          }
297          if(rv==0)return 1 ;
298          if(s[rv-1]=='\n')s[rv-1]='\0' ;
299          return 0 ;
300      }
301
         /*
         * 功能:从源字符串里面读取给定字符前的部分赋值给目标字符串，后面部分覆盖
         * 源字符串
         * 参数:word->目标字符串
         *       line->源字符串
         *       stop->分割字符串的字符
         */
302      void getword(char*word,char*line,char stop)
303      {
304          int x=0,y ;
305
306          for(x=0;((line[x])&&(line[x]!=stop));x++)
307              word[x]=line[x];
308
309          word[x]='\0' ;
310          if(line[x])++x ;
311          y=0 ;
312
313          while(line[y++]=line[x++]);
314      }
315
```

```
        /*
         * 功能:将源字符串里面第一个单词赋值给目标字符串剩余的内容去掉前置空格
         * 覆盖源字符串
         * 该函数为分析配置文件行使用，如 Port 80。
         * 参数:word->目标字符串，line 源字符串
         */
316     void cfg_getword(char*word,char*line)
317     {
318         int x=0,y ;
319
320         for(x=0;line[x]&&isspace(line[x]);x++);
321         y=0 ;
322         while(1)
323         {
324             if(!(word[y]=line[x]))
325                 break ;
326             if(isspace(line[x]))
327                 if((!x)||(line[x-1]!='\\'))
328                     break ;
329                 if(line[x]!='\\')++y ;
330                 ++x ;
331         }
332         word[y]='\0' ;
333         while(line[x]&&isspace(line[x]))++x ;
334         for(y=0;line[y]=line[x];++x,++y);
335     }
336
        /*
         * 功能:读取文件一行或 n 个字节，跳过前置后继空格
         * 参数:s->读取数据保存 buffer
         *       n->读取字符数
         *       f->要读取的文件
         * 返回值:1->到达文件结尾;0->未到文件结尾
         * 返回值指定是否需要继续读取
         */
337     int cfg_getline(char*s,int n,FILE*f)
338     {
339         register int i=0,c ;
340
341         s[0]='\0' ;
342         do
343         {
344             c=getc(f);
345         }
```

```
346         while(c=='\t'||c==' ');
347
348         while(1)
349         {
350             if((c=='\t')||(c==' '))
351             {
352                 s[i++]=' ' ;
353                 while((c=='\t')||(c==' '))
354                     c=getc(f);
355             }
356             if(c==CR)
357             {
358                 /* \r*/
359                 c=getc(f);
360             }
361             if(c==EOF||c==0x4||c==LF||i==(n-1))
362             {
363                 while(i&&(s[i-1]==' '))--i ;
364                 s[i]='\0' ;
365                 return(feof(f)?1:0);
366             }
367             s[i]=c ;
368             ++i ;
369             c=getc(f);
370         }
371     }
372
        /*
         * 功能:将给定命令行里面出现的特定字符前添加斜杠'\'
         * 参数:cmd->需要处理的命令行
         */
373     void escape_shell_cmd(char*cmd)
374     {
375         register int x,y,l ;
376
377         l=strlen(cmd);
378         for(x=0;cmd[x];x++)
379         {
380             if(ind("&;`'\"|*?~<>^()[]{}$\\",cmd[x])!=-1)
381             {
382                 for(y=l+1;y>x;y--)
383                     cmd[y]=cmd[y-1];
384                 l++;
385                 cmd[x]='\\' ;
```

```
386                     x++;
387                 }
388             }
389     }
390
        /*
         * 功能:将给定字符串中的'+'字符替换成空格' '
         * 参数:str->需要操作的字符串
         */
391     void plustospace(char*str)
392     {
393         register int x ;
394
395         for(x=0;str[x];x++)if(str[x]=='+')str[x]=' ' ;
396     }
397
        /*
         * 功能:将给定字符串中的空格' '字符替换成'+'
         * 参数:str->需要操作的字符串
         */
398     void spacetoplus(char*str)
399     {
400         register int x ;
401
402         for(x=0;str[x];x++)if(str[x]==' ')str[x]='+' ;
403     }
404
        /*
         * 功能:解码 URL 里的特殊字符。
         * 参数:URL 特殊字符的十六进制编码，如%5c
         * 返回值:解码后的字符，如\
         */
405     char x2c(char*what)
406     {
407         register char digit ;
408
409         digit=((what[0]>='A')?((what[0]&0xdf)-'A')+10:(what[0]-'0'));
410         digit*=16 ;
411         digit+=(what[1]>='A'?((what[1]&0xdf)-'A')+10:(what[1]-'0'));
412         return(digit);
413     }
414
       /*
        * 功能:解码给定的 URL
```

```
     * 参数:解码前的 URL
     */
415  void unescape_url(char*url)
416  {
417      register int x,y ;
418
419      for(x=0,y=0;url[y];++x,++y)
420      {
421          if((url[x]=url[y])=='%')
422          {
423              url[x]=x2c(&url[y+1]);
424              y+=2 ;
425          }
426      }
427      url[x]='\0' ;
428  }
429   //编码特殊字符，主要有%?+&
430  #define c2x(what,where) sprintf(where,"%%%2x",what)
431
     /*
      * 功能:给指定的 URL 编码
      * 参数:编码前的 URL
      */
432  void escape_url(char*url)
433  {
434      register int x,y ;
435      register char digit ;
436      char*copy ;
437
438      copy=strdup(url);
439
440      for(x=0,y=0;copy[x];x++,y++)
441      {
442          if(ind("% ?+&",url[y]=copy[x])!=-1)
443          {
444              c2x(copy[x],&url[y]);
445              y+=2 ;
446          }
447      }
448      url[y]='\0' ;
449      free(copy);
450  }
451   //此函数功能和 escape_url 函数完全一样
452  void escape_uri(char*url)
```

```
453     {
454         register int x,y ;
455         register char digit ;
456         char*copy ;
457
458         copy=strdup(url);
459
460         for(x=0,y=0;copy[x];x++,y++)
461         {
462             if(ind(":% ?+&",url[y]=copy[x])!=-1)
463             {
464                 c2x(copy[x],&url[y]);
465                 y+=2 ;
466             }
467         }
468         url[y]='\0' ;
469         free(copy);
470     }
471
        /*
         * 功能:将两个源路径合并(相加)赋值给 dst
         * 参数:src1->源路径 1，目标路径的前半部分
         *       src2->源路径 2，目标路径的后半部分
         */
472     void make_full_path(char*src1,char*src2,char*dst)
473     {
474         register int x,y ;
475
476         for(x=0;dst[x]=src1[x];x++);
477
478         if(!x)dst[x++]='/' ;
479         else if((dst[x-1]!='/'))
480             dst[x++]='/' ;
481
482         for(y=0;dst[x]=src2[y];x++,y++);
483     }
484
         /*
         * 功能:判断一个给定的路径是否是一个目录
         * 参数:path->要判断的路径
         * 返回值:1->是一个目录;0:出错或不是一个目录
         */
485     int is_directory(char*path)
486     {
```

```
487         struct stat finfo ;
488
489         if(stat(path,&finfo)==-1)
490             return 0 ;
491
492         return(S_ISDIR(finfo.st_mode));
493     }
494
        /*
         * 功能:判断一个给定的字符串是否是一个 URL
         *       URL 标准='  字母://任意字符串'
         * 参数:u->要判断的字符串
         * 返回值:1->是一个 URL;0:不是一个 URL
         */
495     int is_url(char*u)
496     {
497         register int x ;
498
499         for(x=0;u[x]!=':';x++)
500             if((!u[x])||(!isalpha(u[x])))
501                 return 0 ;
502
503             if((u[x+1]=='/')&&(u[x+2]=='/'))
504                 return 1 ;
505             else return 0 ;
506     }
507
        /*
         * 功能:将给定的参数和它的值组成一个环境变量表达式'变量名=变量值'
         * 参数:name->参数名称
         *       value->参数值
         *       out->错误输出位置
         * 返回值:环境变量结果字符串
         */
508     char*make_env_str(char*name,char*value,FILE*out)
509     {
510         char*t,*tp ;
511
512         if(!(t=(char*)malloc(strlen(name)+strlen(value)+2)))
513             die(NO_MEMORY,"make_env_str",out);
514
515         for(tp=t;*tp=*name;tp++,name++);
516         for(*tp++='=';*tp=*value;tp++,value++);
517         return t ;
```

```
518     }
519
        /*
         * 功能:根据环境变量名替换已有的同名环境变量值
         * 参数:env->环境变量列表
         *       name->要替换的环境变量名
         *       value->要替换的环境变量名对应的值
         *       out->错误输出位置
         * 返回值:无(虽然给定的类型是 char*)
         */
520     char*replace_env_str(char**env,char*name,char*value,FILE*out)
521     {
522         char*p ;
523         register int i,len ;
524
525         for(i=0,len=strlen(name);env[i];i++)
526             if(strncmp(env[i],name,len)==0)
527             {
528                 free(env[i]);
529                 env[i]=make_env_str(name,value,out);
530                 break ;
531             }
532     }
533
        /*
         * 功能:在目前的环境变量列表里面增加指定个数的环境变量位置，并把
         * in_headers_env 指向环境变量列表开始位置。
         * 参数:env->环境变量列表指针
         *       to_add->要添加的环境变量的个数
         *       pos->环境变量列表中新增位置指针
         * 返回值:环境变量列表起始位置
         */
534     char**new_env(char**env,int to_add,int*pos)
535     {
536         char**newenv ;
537
538         if(!env)
539         {
540             *pos=0 ;
541             newenv=(char**)malloc((to_add+1)*sizeof(char*));
542             newenv[to_add]=NULL ;
543         }
544         else
545         {
```

```
546             int x ;
547
548             for(x=0;env[x];x++);
549             if(!(newenv=(char**)malloc((to_add+x+1)*(sizeof(char*)))))
550                 return NULL ;
551             for(x=0;env[x];x++)
552                 newenv[x]=env[x];
553             newenv[to_add+x]=NULL ;
554             *pos=x ;
555             free(env);
556         }
557         return(in_headers_env=newenv);
558     }
559
        /*
         * 功能:释放指定的环境变量列表空间
         * 参数:env->环境变量列表指针
         */
560     void free_env(char**env)
561     {
562         int x ;
563
564         for(x=0;env[x];x++)
565             free(env[x]);
566         free(env);
567     }
568
        /*
         * 功能:根据给定的文件属性，判定该文件是否可以被 apache 执行
         * 参数:finfo->文件属性结构指针
         * 返回值:1->可执行; 0->不能执行
         */
569     int can_exec(struct stat*finfo)
570     {
571         if(user_id==finfo->st_uid)
572             if(finfo->st_mode&S_IXUSR)
573                 return 1 ;
574             if(group_id==finfo->st_gid)
575                 if(finfo->st_mode&S_IXGRP)
576                     return 1 ;
577                 return(finfo->st_mode&S_IXOTH);
578     }
579
        /*
```

```
 * 功能:从字符串 s 里面找到第一次出现字符 c 的位置
 * 参数:s->被查找的字符串;c->需要查找的字符
 * 返回值:-1->在给定字符串中没找到指定字符
 *          >=0->给定字符第一次出现在字符串中的位置
 */
580  int ind(const char*s,char c)
581  {
582      register int x ;
583
584      for(x=0;s[x];x++)
585          if(s[x]==c)return x ;
586
587          return-1 ;
588  }
589
     /*
 * 功能:从字符串里最后一次出现字符 c 的位置
 * 参数:s->被查找字符串;c->需要查找的字符
 * 返回值:-1->在字符串里面没找到指定字符
 *          >=0 ->字符串中最后一次出现字符 c 的位置
 */
590  int rind(char*s,char c)
591  {
592      register int x ;
593
594      for(x=strlen(s)-1;x!=-1;x--)
595          if(s[x]==c)return x ;
596
597          return-1 ;
598  }
599
     /*
 * 功能:将给定字符串中的所有字符转换成小写
 * 参数:str->打算转换的字符串
 */
600  void str_tolower(char*str)
601  {
602      while(*str)
603      {
604          *str=tolower(*str);
605          ++str ;
606      }
607  }
608
```

```
        /*
         * 功能:根据用户名获取用户 ID
         * 参数:name->需要获得用户 ID 的用户名
         * 返回值:用户名对应的用户 ID
         */
609     uid_t uname2id(char*name)
610     {
611         struct passwd*ent ;
612
613         if(name[0]=='#')
614             return(atoi(&name[1]));
615
616         if(!(ent=getpwnam(name)))
617         {
618             fprintf(stderr,"httpd: bad user name %s\n",name);
619             exit(1);
620         }
621         else return(ent->pw_uid);
622     }
623
        /*
         * 功能:根据组名获取组 ID
         * 参数:name->需要获得组 ID 的组名
         * 返回值:组名对应的组 ID
         */
624     gid_t gname2id(char*name)
625     {
626         struct group*ent ;
627
628         if(name[0]=='#')
629             return(atoi(&name[1]));
630
631         if(!(ent=getgrnam(name)))
632         {
633             fprintf(stderr,"httpd: bad group name %s\n",name);
634             exit(1);
635         }
636         else return(ent->gr_gid);
637     }
638
         /*
         * 功能:根据 socket 文件描述符获取该 socket 使用的端口
         * 参数:sd->socket 文件描述符,out->错误输出位置
         * 返回值:给定的 socket 文件描述符使用的端口号
```

```
 */
int get_portnum(int sd,FILE*out)
{
    struct sockaddr addr ;
    int len ;

    len=sizeof(struct sockaddr);
    if(getsockname(sd,&addr,&len)<0)
        die(SERVER_ERROR,"could not get port number",out);
    return ntohs(((struct sockaddr_in*)&addr)->sin_port);
}

/*
 * 功能:根据 hostent 结构指针获取全域名
 * 参数:指向初始化完成的 hostent 结构的指针
 * 返回值:全域名
 */
char*find_fqdn(struct hostent*p)
{
    int x ;

    if(ind(p->h_name,'.')==-1)
    {
        for(x=0;p->h_aliases[x];++x)
        {
            if((ind(p->h_aliases[x],'.')!=-1)&&
                (!strncmp(p->h_aliases[x],p->h_name,strlen(p->h_name))))
                return strdup(p->h_aliases[x]);
        }
        return NULL ;
    }
    else return strdup(p->h_name);
}

/*
 * 根据给定的 socket 连接描述符，获取客户端 ip/host/name。
 * 如果需要，管理名称服务器缓冲
 */
void get_remote_host(int fd)
{
    struct sockaddr addr ;
    int len,i ;
    struct in_addr*iaddr ;
    struct hostent*hptr ;
```

```
673         extern int h_errno ;
674
675         len=sizeof(struct sockaddr);
676
677         if((getpeername(fd,&addr,&len))<0)
678         {
679             remote_host=NULL ;
680             remote_ip=NULL ;
681             remote_name="UNKNOWN_HOST" ;
682             return ;
683         }
684         iaddr=&(((struct sockaddr_in*)&addr)->sin_addr);
685         remote_ip=inet_ntoa(*iaddr);
686         remote_host=NULL ;
687          /* default values 默认值 */
688         remote_name=remote_ip ;
689
690         hptr=gethostbyaddr((char*)iaddr,sizeof(struct in_addr),AF_INET);
691         if(hptr!=NULL)
692         {
693             char**haddr ;
694
695             remote_host=strdup(hptr->h_name);
696             str_tolower(remote_host);
697         }
698         if(remote_host!=NULL)remote_name=remote_host ;
699     }
700      /*根据配置文件或函数操作获取本机的 hostname */
701     void get_local_host()
702     {
703         char str[128];
704         int len=128 ;
705
706         if(!server_hostname)
707         {
708             struct hostent*p ;
709             gethostname(str,len);
710             if((!(p=gethostbyname(str)))||(!(server_hostname=find_fqdn(p))))
711             {
712                 fprintf(stderr,"httpd: cannot determine local host name.\n");
713                 fprintf(stderr,"Use ServerName to set it manually.\n");
714                 exit(1);
715             }
716         }
```

```
717     }
718      /*根据服务器名/端口号/请求地址构建请求 URL */
719     void construct_url(char*d,char*s)
720     {
721           if(port==80)
722           {
723                 sprintf(d,"http://%s%s",server_hostname,s);
724           }
725           else
726           {
727                 sprintf(d,"http://%s:%d%s",server_hostname,port,s);
728           }
729     }
730      /*下面这个整型数组和函数实现了 BASE64 的解码操作  */
731     const int pr2six[256]=
732     {
733           64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,
734                 64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,62,64,64,64,63,
735                 52,53,54,55,56,57,58,59,60,61,64,64,64,64,64,64,64,0,1,2,3,4,5,6,7,8,9,
736                 10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,64,64,64,64,64,64,26,27,
737                 28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,
738                 64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,
739                 64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,
740                 64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,
741                 64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,
742                 64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,
743                 64,64,64,64,64,64,64,64,64,64,64,64,64,64
744     }
745     ;
746     /*使用 BASE64 编码方式解码 bufcoded 里面的数据，写入到 bufplain 里面。*/
747     void uudecode(char*bufcoded,unsigned char*bufplain,int outbufsize)
748     {
749           int nbytesdecoded,j ;
750           register unsigned char*bufin ;
751           register unsigned char*bufout=bufplain ;
752           register int nprbytes ;
753
754           /*  跳过前导空格  */
755
756           while(*bufcoded==' '||*bufcoded=='\t')bufcoded++;
757
758           bufin=(unsigned char*)bufcoded ;
759           while(pr2six[*(bufin++)]<=63);
760           nprbytes=(char*)bufin-bufcoded-1 ;
```

```
761         nbytesdecoded=((nprbytes+3)/4)*3 ;
762         if(nbytesdecoded>outbufsize)
763         {
764             nprbytes=(outbufsize*4)/3 ;
765         }
766
767         bufin=(unsigned char*)bufcoded ;
768
769         while(nprbytes>0)
770         {
771             *(bufout++)=
772                 (unsigned char)(pr2six[*bufin]<<2|pr2six[bufin[1]]>>4);
773             *(bufout++)=
774                 (unsigned char)(pr2six[bufin[1]]<<4|pr2six[bufin[2]]>>2);
775             *(bufout++)=
776                 (unsigned char)(pr2six[bufin[2]]<<6|pr2six[bufin[3]]);
777             bufin+=4 ;
778             nprbytes-=4 ;
779         }
780
781         if(nprbytes&03)
782         {
783             if(pr2six[bufin[-2]]>63)
784                 nbytesdecoded-=2 ;
785             else
786                 nbytesdecoded-=1 ;
787         }
788         bufplain[nbytesdecoded]='\0' ;
789     }
790
        /*
         * 根据错误号获取错误号在错误信息里面的位置下标
         * 如 200->0, 302->1, 304->3 等等.
         */
791     int index_of_response(int err_no)
792     {
793         char*cptr,err_string[4];
794         static char*response_codes=RESPONSE_CODE_LIST ;
795         int index_number ;
796
797         sprintf(err_string,"%3d",err_no);
798
799         cptr=response_codes ;
800         cptr++;
```

```
801         index_number=0 ;
802         while(*cptr&&strncmp(cptr,err_string,3))
803         {
804             cptr+=4 ;
805             index_number++;
806         }
807         return index_number ;
808     }
809
        /* 重命名已经存在的环境变量使得他们具有 REDIRECT_前缀
         */
810     void rename_original_env(char**env,FILE*fd)
811 {
812         int x=0 ;
813         char*cptr ;
814
815         if(env==NULL)return ;
816
817         while(env[x]!=NULL)
818         {
819
820             if(!(cptr=(char*)malloc((strlen(env[x])+1+9)*sizeof(char))))
821                 die(NO_MEMORY,"make_env_str",fd);
822
823             sprintf(cptr,"REDIRECT_%s",env[x]);
824             free(env[x]);
825             env[x]=cptr ;
826             x++;
827         }
828
829         env[x]=NULL ;
830     }
831
        /*
        * 本函数使用一个环境变量记录了一个错误或问题的原始状态。并作为全局变量
        * 提供给重定向的错误或问题来使用
        */
832     char**add_redirection_status_env(char**env,FILE*out,int status)
833     {
834         char t[HUGE_STRING_LEN],*env_path ;
835         int x ;
836
837         if(!(env=new_env(env,1,&x)))
838             die(NO_MEMORY,"add_redirection_status_env",out);
```

```
839
840             sprintf(t,"%d",status);
841             env[x++]=make_env_str("REDIRECT_STATUS",t,out);
842             original_status=status ;
843
844             env[x]=NULL ;
845             in_headers_env=env ;
846             return env ;
847         }
848
            /*
             * 我们在重定向前做更多有用的事情来代替重新设置所有的东西。我们会保持尽
             * 可能多的信息以使被调用的 URL 获得好处。
             */
849         void prepare_for_redirect(FILE*fd,int orig_status,char*new_url)
850         {
851
852             content_type[0]='\0' ;
853             content_length=-1 ;
854             auth_line[0]='\0' ;
855             content_encoding[0]='\0' ;
856             status_line=NULL ;
857             status=0 ;
858             out_headers=NULL ;
859             location[0]='\0' ;
860             sprintf(the_request,"GET %s",new_url);
861
862             rename_original_env(in_headers_env,fd);
863
864             add_redirection_status_env(in_headers_env,fd,orig_status);
865         }
```

3.3 小结

本章对 Apache 的自定义函数库和包裹函数进行了较为详细的说明，这些函数供 Apache 核心模块调用，如果您在自己的项目中有类似的需求，可以将它们放在您自己项目的自定义函数库中使用。

除了对背景知识的详细解说外，对自定义函数和包裹函数没有进行逐行的注解，一个是因为这些包裹函数本身比较简单，另外一个是这些函数仅仅作为工具供其它部分调用，不会对您理解 Apache 本身造成障碍。

第 4 章日志和重定向

4.1 概述

我们开启 Web 服务后，我们可能关心几个事情，比如都有那些用户访问过我们的网站，都是在哪些时间段访问的，用户对我们的哪些页面感兴趣，我们网站的访问量和流量如何，Apache 运行过程中是否一切正常，都出过什么错误，当前 Apache 进程的进程号是多少等等。我们从何而知呢？这就需要 Apache 在生命周期中对这些情况做一些记录。这就是 Apache 的日志文件。

Apache0.6.5 共有三种日志：访问日志、错误日志和进程 ID 日志。其中访问日志以固定格式记录客户端的访问记录；错误日志记录了 Apache 运行过程中发生的各种错误信息；进程 ID 日志记录了最后一次运行 Apache 时，Apache 进程的进程 ID。

日志文件的相关操作通过源代码文件 http_log.c 完成。

4.2 背景知识

4.2.1 配置Apache

作为最为流行的 Web 服务器，Apache 提供了丰富的配置选项方便管理员定制自己的 Web 服务器。这些配置选项默认保存在 conf 目录下的配置文件中。

本章以及后继几个章节围绕 Apache 的各个功能模块展开，而每个功能模块都和相应的配置指令相关，每个配置指令存在于约定俗成的配置文件中。

早期的（包括 0.6.5 版本）Apache 拥有 4 个配置文件，各个配置文件的名字和功能如下所示：

httpd.conf 主配置文件，提供了最基本的服务器设置，如监听的端口，运行服务器所使用的用户角色等内容。

mime.types 用来标识不同文件对用的 MIME 类型。更多内容请参考第六章的描述。

srm.conf 服务器资源映射文件。设置如站点根目录位置等内容。

access.conf 服务器访问权限配置文件，用来控制不同用户和计算机对指定资源的访问控制。

这些配置文件内容可以分成两个部分，其中以井号（#）开头的行是注释行，Apache 读取配置文件的时候会忽略这些行的内容；其它行为配置行，一般以配置指令开始。

一个典型的配置文件可能包含这样的内容：

```
# 我们配置 Apache 使用 8181 端口
Port 8181
```

现代的 Apache 把 srm.conf、access.conf 和 httpd.conf 三个配置文件的内容合并到了一个 httpd.conf 文件中，并在原有的配置指令基础上增加了很多新的配置指令，同时也兼容本书中描述的所有配置指令。

我们在这里对这些配置文件的功能先有一个大致的了解，在相关的章节里面会对各个配置文件的配置指令进行详细的描述。

4.2.2 HTTP状态码

当客户端向 Apache 请求资源的时候，Apache 会给客户端一个 3 位数字的响应状态代码。称作 HTTP 状态码。状态码由 RFC2616 规范定义。

我们常见的状态码是 200，表示服务器成功处理了请求。目前已经定义的状态码主要有 5 类，每类状态码的含义及其典型代表如下所示：

1××（临时响应）

如 100，服务器返回此代码表示已经收到请求的一部分，正在等待其余部分

2××（成功）

如 200，表示服务器成功处理了请求。

3××（重定向）

如 304。一般浏览器都有缓存，如果浏览器请求的资源自上次请求后没有更改过，服务器会返回 304 状态码，浏览器使用本地缓存显示返回 304 状态码的资源。

4××（请求错误）

最常见的是 401 和 404，401 表示客户端请求的资源需要身份验证；404 表示服务器未找到客户端请求的资源。

5××（服务器错误）

如 500，表示服务器内部错误。

状态码的完整描述请参看：http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Apache0.6.5 可能返回 9 种状态码，它们的含义如下所示：

状态码	含义
200	服务器理解客户端请求意义并正确处理
302	客户端要请求的资源临时放到一个新地方了。应答中包含了资源的新地址
304	上次客户端获取数据后，数据没有改变，可以使用客户端本地缓存
400	错误请求 。请求的语法错误，服务器无法理解
401	未授权。客户端访问指定资源以前，必须通过服务器的授权
403	禁止访问 。不允许访问客户端请求的资源
404	找不到 找不到客户端请求的内容
500	服务器内部错误
501	服务器不支持实现请求所需要的功能。
503	服务器由于维护或者负载过重未能应答。

4.2.3 重定向

如果您做过网站开发，一定对重定向不陌生，您可以通过脚本语言来设置，让浏览器在不需要用户干预的情况下跳转到某个新的页面。

Apache 也提供了针对重定向的设置选项（下节中将要描述的 ErrorDocument 指令），以便在特定的情况下让浏览器直接跳转到新的页面或者直接发送新的页面给浏览器。

网站管理员设置重定向有什么意义呢？

我们可以做一个实验，使用浏览器访问我们网站上并不存在的页面，这时服务器会发送一个 404 状态码给客户端浏览器，此时浏览器会按照自己预先定义的样式来显示一个错误页面，这个页面千篇一律，毫无新意，也没有太多的提示性，典型的样子类似于：

这个页面简单地说明了用户操作中发生的问题，但以现代的眼光连看，对用户不够友好，其实我们可以做出来更漂亮、更有个性或更有提示性的 404 错误页面给用户。如下图就是一个个性化的 404 页面：

这个 404 页面属于怀旧型，好像让我们有看到了经典的 Windows 蓝屏。

Apache 在 die 函数中完成对重定向的处理，这个过程如下所述：

1. 管理员通过修改配置文件 srm.conf 设置在特定条件下要跳转的新位置
2. Apache 读取配置文件并初始化在 http_config.c 中定义的 response_code_strings 字符串数组。
3. 在特定条件发生的时候（如用户请求的页面不存在的时候），Apache 查看 response_code_strings 对应的错误信息是否是一个 URL
4. 如果对应的信息是一个内部 URL（指向本站内部资源的 URL，以字符'/'开头，如 /res/404.html），则 Apache 直接将客户端的请求转换成对这个内部 URL 的请求，将其内容发送给客户端。
5. 如果对应的信息是一个外部 URL（指向本站外的资源，如 http://www.otherwebsit.com/some.html），则向客户端发送 302 状态码，同时给出这个外部 URL 的地址。浏览器接收到 302 状态码和对应的 URL 地址时，一般会直接向服务器发送一个新的、针对给定 URL 的资源请求。
6. 如果对应的信息是一个错误的 URL，则给客户端发送服务器错误状态码
7. 如果 response_code_strings 中的信息不是一个 URL，而是错误提示信息，则将错误提示信息作为页面内容发送给客户端。

Apache 还有一个重定向指令 Redirect，我们将在第五章中讲解。

4.3 日志文件配置指令

这里面涉及到两个配置文件中的 4 个配置指令。分别是 srm.conf 文件中的 ErrorDocument

指令；httpd.conf 文件中的 ErrorLog 指令、TransferLog 指令和 PidFile 指令。下面分别介绍它们：

1. ErrorLog 指令

语法：ErrorLog *file*

功能：指定错误日志文件的存放位置。一般设置成相对于 SERVER_ROOT 的路径

示例：ErrorLog logs/error_log

上例说明，错误日志文件为 SERVER_ROOT/logs/error_log。

2. TransferLog 指令

语法：TransferLog *file*

功能：指定访问日志文件的存放位置。一般设置成相对于 SERVER_ROOT 的路径

示例：TransferLog logs/access_log

上例说明，访问日志文件为 SERVER_ROOT/logs/ access _log。

3. PidFile 指令

语法：PidFile *filename*

功能：指定进程 ID 记录文件的位置

示例：PidFile logs/httpd.pid

上例说明，进程 ID 日志文件为 SERVER_ROOT/logs/httpd.pid。

4. ErrorDocument 指令

这个指令和服务器日志没有什么关系，但是作为 http_log.c 文件处理的一部分，我们也在这里说明一下。

语法：ErrorDocument *错误代码文档*

功能：定义了当遇到错误的时候服务器将给客户端什么样的回应

示例：ErrorDocument 500 "The server made a boo boo.

上例说明，当服务器发生 500 错误的时候，客户端收到的信息里面将包含 The server made a boo boo 这段文字（也会有一些服务器已经定义的错误信息）。

当用户的请求遇到问题或错误的时候，Apache 可以通过以下四种方式应答用户：

1. 使用服务器本身的错误信息回应用户。
2. 在 1.的基础上附加配置文件定义的信息回应用户。
3. 重定向到一个本地 URL
4. 重定向到一个外部 URL

后面三种情况都是通过 ErrorDocument 指令来配置的。如：

```
ErrorDocument 500 "The server made a boo boo %s
ErrorDocument 404 /error/404.html
ErrorDocument 503 http://www.yourdomain.com/error/503.html
```

在"The server made a boo boo %s 中%s 将被设置成失败原因

4.4 配置实践

下面我们通过实例来分别验证上节中提到的 4 个设置指令。

1. ErrorLog

此指令在 httpd.conf 文件中设置：

```
ErrorLog logs/error_log
```

这条设置指令表示 Apache 的错误日志文件为/home/devel/apache_0.6.5/logs/error_log。运行以下指令集来查看该日志文件的输出：

```
[devel@RIZI src]$ pwd
/home/devel/apache_0.6.5/src
[devel@RIZI src]$ ps x
  PID TTY      STAT   TIME COMMAND
 5010 ?        S      0:00 sshd: devel@pts/1
 5011 pts/1    Ss     0:00 -bash
 5178 ?        Ss     0:00 ./httpd
 5204 pts/1    R+     0:00 ps x
[devel@RIZI src]$ kill 5178
[devel@RIZI src]$ tail /home/devel/apache_0.6.5/logs/error_log
[Mon May  4 15:59:32 2009] httpd: caught SIGTERM, shutting down
[devel@RIZI src]$
```

从上面我们可以看到，当我们 Kill 掉 Apache 的时候，Apache 接受到了系统发给它的 SIGTERM 信号，关闭服务并记录了一条错误日志。

2. TransferLog

此指令在 httpd.conf 文件中设置：

```
TransferLog logs/access_log
```

这条指令表示 Apache 的访问日志文件为/home/devel/apache_0.6.5/logs/ access_log，运行您的 Apache，打开浏览器，在地址栏输入您的 Apache 提供服务的站点首页，如 http://11.22.33.44:81/index.html。可以在 Linux 命令行中查看该日志文件的内容：

```
[devel@RIZI apache_0.6.5]$ tail /home/devel/apache_0.6.5/logs/ access_log
222.131.235.66 - - [04/May/2009 16:02:17 +0800] "GET /index.html HTTP/1.1" 200 44
[devel@RIZI apache_0.6.5]$
```

从上面的操作我们可以看出，当我们访问站点首页的时候，Apache 在给客户端发送其请求内容的同时记录了一条访问日志。包括客户端的 IP、访问时间、访问方法、访问内容、使用的协议版本、状态码和发送的字节数等信息。

3. PidFile

此指令在 httpd.conf 文件中设置：

```
PidFile logs/httpd.pid
```

这条指令表示 Apache 的进程 ID 日志文件为/home/devel/apache_0.6.5/logs/httpd.pid，运行您的 Apache，在 Linux 命令行中输入如下的指令序列：

```
[devel@RIZI apache_0.6.5]$ tail /home/devel/apache_0.6.5/logs/ httpd.pid
2905
[devel@RIZI apache_0.6.5]$ps x
PID TTY STAT TIME COMMAND
2905 ? Ss 0:00 ./httpd
10248 pts/2 R+ 0:00 ps x
……
```

4. ErrorDocument

该指令在 srm.conf 文件中设置，我们在 srm.conf 文件最后添加如下内容：

```
ErrorDocument 404 http://www.google.cn
```

打开浏览器，在地址栏输入http://www.yourdomain.com:port/~usernotexit，我们可以看到，浏览器实际打开的地址是我们设定的http://www.google.cn。

注意：请将www.yourdomain.com替换成您服务器的域名或IP，port部分填写您的Apache

服务器监听的端口号。

4.5 代码注释

本章所列功能的实现文件是 http_log.c。下面是它的代码及注释：

```
1   #include "httpd.h"
     /* 错误文件 */
2   FILE*error_log ;
     /* 访问日志文件描述符 */
3   static int xfer_log ;
     /* 访问日志文件打开方式:读写| 附加| 创建 */
4   static int xfer_flags=(O_WRONLY|O_APPEND|O_CREAT);
     /* 访问日志文件存取权限:所有者可读写/组成员和其它用户可读 */
5   static mode_t xfer_mode=(S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
6    /* 以下几个变量在重定向时记录客户端原始请求信息 */
7   char original_url[HUGE_STRING_LEN];
8   char original_args[HUGE_STRING_LEN];
9   char original_method[20];
10  int original_status ;
11  char auth_string[MAX_STRING_LEN];
12   /*打开日志文件并设置他们的属性(主要是访问日志文件属性) */
13  void open_logs()
14  {
15      if(!(error_log=fopen(error_fname,"a")))
16      {
17          fprintf(stderr,"httpd: could not open error log file %s.\n",
18              error_fname);
19          perror("fopen");
20          exit(1);
21      }
22      if((xfer_log=open(xfer_fname,xfer_flags,xfer_mode))<0)
23      {
24          fprintf(stderr,"httpd: could not open transfer log file %s.\n",
25              xfer_fname);
26          perror("open");
27          exit(1);
28      }
29      if(fcntl(xfer_log,F_SETFD,(int)FD_CLOEXEC)==-1)
30      {
31          perror("fcntl");
32          fprintf(stderr,
33              "httpd: could not set close-on-exec on transfer log file.\n",xfer_fname);
34          exit(1);
35      }
```

```
36
37      }
38       /* 关闭日志文件 */
39      void close_logs()
40      {
41            close(xfer_log);
42            fclose(error_log);
43      }
44       /* 将标准错误输出到错误日志文件中。*/
45      void error_log2stderr()
46      {
47            if(fileno(error_log)!=STDERR_FILENO)
48                  dup2(fileno(error_log),STDERR_FILENO);
49      }
50       /* 将当前进程的进程 ID 写入配置(默认或配置文件)里面给定的记录文件中去。*/
51      void log_pid()
52      {
53            FILE*pid_file ;
54
55            if(!(pid_file=fopen(pid_fname,"w")))
56            {
57                  fprintf(stderr,"httpd: could not log pid to file %s\n",pid_fname);
58                  exit(1);
59            }
60            fprintf(pid_file,"%ld\n",(long)getpid());
61            fclose(pid_file);
62      }
63
64      char the_request[HUGE_STRING_LEN]; // 请求内容
65      int status ;                       // 状态码
66      int bytes_sent ;                   // 服务器发送给客户端的字节数
67       /* 记录客户端请求 */
68      void record_request(char*cmd_line)
69      {
70            status=-1 ;
71            bytes_sent=-1 ;
72
73            strcpy(the_request,cmd_line);
74      }
75
76      extern char log_user[MAX_STRING_LEN];
77       /* 增加一条访问日志 */
78      void log_transaction(int closeit)
79      {
```

```
80          char str[HUGE_STRING_LEN];
81          long timz ;
82          struct tm*t ;
83          char tstr[MAX_STRING_LEN],sign ;
84
85          t=get_gmtoff(&timz);
86          sign=(timz<0?'-':'+');
87          if(timz<0)
88                timz=-timz ;
89           //原本的格式：strftime(tstr,MAX_STRING_LEN,"%d/%b/%Y:%H:%M:%S",t);
90          strftime(tstr,MAX_STRING_LEN,"%d/%b/%Y %H:%M:%S",t);
91          sprintf(str,"%s %s %s [%s %c%02ld%02ld] \"%s\" ",
92                remote_name,
93                "-",
94                (log_user[0]?log_user:"-"),
95                tstr,
96                sign,
97                timz/3600,
98                timz%3600,
99                the_request);
100         if(status!=-1)
101               sprintf(str,"%s%d ",str,status);
102         else
103               strcat(str,"- ");
104
105         if(bytes_sent!=-1)
106               sprintf(str,"%s%d\n",str,bytes_sent);
107         else
108               strcat(str,"-\n");
109
110        write(xfer_log,str,strlen(str));
111        if(closeit)close(xfer_log);       // 只有在希望关闭的时候才关闭日志文件
112   }
113    /* 添加一条错误日志记录 */
114   void log_error(char*err)
115   {
116         fprintf(error_log,"[%s] %s\n",get_time(),err);
117         fclose(error_log);
118   }
119    /*添加一条错误日志记录，但并不关闭错误日志文件 */
120   void log_error_noclose(char*err)
121   {
122         fprintf(error_log,"[%s] %s\n",get_time(),err);
123         fflush(error_log);
```

```
124     }
125      /* 添加一条访问失败及其访问者、失败原因的错误日志记录 */
126     void log_reason(char*reason,char*file)
127     {
128           char t[MAX_STRING_LEN];
129
130           sprintf(t,
131                 "httpd: access to %s failed for %s, reason: %s",file,remote_name,reason);
132           log_error(t);
133     }
134      /* 创建 http 响应头信息 */
135     void begin_http_header(FILE*fd,char*msg)
136     {
137           if(original_status==AUTH_REQUIRED)
138           {
139                 fprintf(fd,
140                       "%s %d %s\015\012",
                            SERVER_PROTOCOL,AUTH_REQUIRED,"Unauthorized");
141                 fprintf(fd,"WWW-Authenticate: %s\015\012",auth_string);
142                 status=AUTH_REQUIRED ;
143           }
144           else
145                 fprintf(fd,"%s %s\015\012",SERVER_PROTOCOL,msg);
146            // 附加默认头部:时间和服务器版本
147           dump_default_header(fd);
148     }
149      /* 拼写错误头信息 */
150     void error_head(FILE*fd,char*err)
151     {
152           if(!assbackwards)
153           {
154                  //http1.0
155                 begin_http_header(fd,err);
156                 fprintf(fd,"Content-type: text/html\015\012\015\012");
157           }
158           if(!header_only)
159           {
160                 fprintf(fd,"<HEAD><TITLE>%s</TITLE></HEAD>%c",err,LF);
161                 fprintf(fd,"<BODY><H1>%s</H1>%c",err,LF);
162           }
163     }
164      /* 拼写 html 头信息 */
165     void title_html(FILE*fd,char*msg)
166     {
```

```
167         fprintf(fd,"<HEAD><TITLE>%s</TITLE></HEAD>%c",msg,LF);
168         fprintf(fd,"<BODY><H1>%s</H1>%c",msg,LF);
169     }
170      /* 出错处理，将出错信息发送给客户端*/
171     void die(int type,char*err_string,FILE*fd)
172     {
173         char t[MAX_STRING_LEN];
174         int error_index ;
             // 根据错误类型返回该类型在总错误列表中的数组下标，在 Util.c 中定义
175         error_index=index_of_response(type);
176         if(original_url[0]=='\0') // 原始请求为空
177         {
                /* 如果配置文件中定义了错误响应的错误信息不为空，我们认为是一个
                   重定向地址
                */
178             if(response_code_strings[error_index]!=NULL)
179             {
                    /* 只有以双引号(")开始的文本是错误提示信息。其它的任何内容
                       为是一个本地或外部 URL.
                    */
180                 if((char)*response_code_strings[error_index]!="")
181                 {     // 如果返回内容是一个外部 URL，当作是一个重定向处理
182                     if(is_url(response_code_strings[error_index]))
183                     {
184                         type=REDIRECT ;
185                         err_string=response_code_strings[error_index];
186                     }
187                     else if(*response_code_strings[error_index]=='/') // 内部 URL
188                     {
189                         status=type ;
190                         log_transaction(0);
191                         original_url[0]=' ' ;
                            /* 在重定向之前记录已经存在的信息 */
192                     prepare_for_redirect(fd,type,response_code_strings[error_index]);
                            /* 如果我们是从一个服务器错误重定向的，需要记录*/
193                         if(type==SERVER_ERROR)log_error_noclose(err_string);
                            /* 记住 AUTH_REQ 字符串*/
194                         if(type==AUTH_REQUIRED)strcpy(auth_string,err_string);
                            /* 重定向到新的 URL 上，不附带参数 */
195                       process_get(0,fd,"GET",response_code_strings[error_index],"");
196                         fflush(fd);
197                         htexit(0,fd);
198                     }
199                     else
```

```
200                     {       /* 用户给了我们一个错误的重定向 URL*/
201                         type=SERVER_ERROR ;
202                         err_string="Invalid attempt to redirect." ;
203                         log_error(response_code_strings[error_index]);
204                     }
205                 }
206             }
207         }
208
209         switch(type)                // 根据调用原因返回不同的错误内容给客户端
210         {
211         case REDIRECT :         //302 重定向
212             status=302 ;
213             if(!assbackwards)   // http1.0
214             {
215                 begin_http_header(fd,"302 Found");
216                 fprintf(fd,"Location: %s\015\012",err_string);
217                 fprintf(fd,"Content-type: text/html\015\012\015\012");
218             }
219             if(header_only)break ;
                 /* 如果我们定义了错误信息，就把它们返回给客户端 */
220             if(response_code_strings[error_index]!=NULL)
221             {
222                 fprintf(fd,response_code_strings[error_index]+1,err_string);
223                 break ;
224             }
225             title_html(fd,"Document moved");
226             fprintf(fd,"This document has moved <A HREF=\"%s\">here</A>.<P>%c",
227                 err_string,LF);
228             break ;
229         case USE_LOCAL_COPY :  // 请用户使用本地未过期文件。
230             status=USE_LOCAL_COPY ;
231             begin_http_header(fd,"304 Not modified\015\012");
232             header_only=1 ;
233             break ;
234         case AUTH_REQUIRED :    // 需要认证
235             status=401 ;
236             if(!assbackwards)
237             {
238                 begin_http_header(fd,"401 Unauthorized");
239                 fprintf(fd,"Content-type: text/html\015\012");
240                 fprintf(fd,"WWW-Authenticate: %s\015\012\015\012",err_string);
241             }
242             if(header_only)break ;
```

```
243             if(response_code_strings[error_index]!=NULL)
244             {
245                 fprintf(fd,response_code_strings[error_index]+1,err_string);
246                 break ;
247             }
248             title_html(fd,"Authorization Required");
249             fprintf(fd,"Browser not authentication-capable or %c",LF);
250             fprintf(fd,"authentication failed.%c",LF);
251             break ;
252         case BAD_REQUEST : // 错误请求
253             status=400 ;
254             error_head(fd,"400 Bad Request");
255             if(header_only)break ;
256             if(response_code_strings[error_index]!=NULL)
257             {
258                 fprintf(fd,response_code_strings[error_index]+1,err_string);
259                 break ;
260             }
261             fprintf(fd,"Your client sent a query that this server could not%c",LF);
262             fprintf(fd,"understand.<P>%c",LF);
263             fprintf(fd,"Reason: %s<P>%c",err_string,LF);
264             break ;
265         case FORBIDDEN :    // 不允许访问
266             status=403 ;
267             error_head(fd,"403 Forbidden");
268             if(header_only)break ;
269             if(response_code_strings[error_index]!=NULL)
270             {
271                 fprintf(fd,response_code_strings[error_index]+1,err_string);
272                 break ;
273             }
274             fprintf(fd,
275                 "Your client does not have permission to get URL %s ",err_string);
276             fprintf(fd,"from this server.<P>%c",LF);
277             break ;
278         case NOT_FOUND :    // 没找到请求的资源
279             status=404 ;
280             error_head(fd,"404 Not Found");
281             if(header_only)break ;
282             if(response_code_strings[error_index]!=NULL)
283             {
284                 if(original_url[0]&&*response_code_strings[error_index]=='/')
285                     fprintf(fd,
286                     "Tried to redirect to %s after some earlier problem with %s, but %s
```

```
                didn't exist\n",
287             response_code_strings[error_index],
288             original_url+1,
289             response_code_strings[error_index]);
290         else
291             fprintf(fd,response_code_strings[error_index]+1,err_string);
292         break ;
293     }
294     fprintf(fd,"The requested URL %s was not found on this server.<P>%c",
295         err_string,LF);
296     break ;
297 case SERVER_ERROR :             // 服务器错误
298     status=500 ;
299     error_head(fd,"500 Server Error");
300     log_error(err_string);
301     if(header_only)
302         break ;
303     if(response_code_strings[error_index]!=NULL)
304     {
305         fprintf(fd,response_code_strings[error_index]+1,err_string);
306         break ;
307     }
308     fprintf(fd,"The server encountered an internal error or%c",LF);
309     fprintf(fd,"misconfiguration and was unable to complete your%c",LF);
310     fprintf(fd,"request.<P>%c",LF);
311 fprintf(fd,"Please contact the server administrator,%c",LF);
312     fprintf(fd," %s ",server_admin);
313     fprintf(fd,"and inform them of the time the error occurred, and%c",LF);
314     fprintf(fd,"anything you might have done that may have caused%c",LF);
315     fprintf(fd,"the error.<P>%c",LF);
316     break ;
317 case NOT_IMPLEMENTED :      // 服务器无法执行请求的方法。
318     status=501 ;
319     error_head(fd,"501 Not Implemented");
320     if(header_only)break ;
321     if(response_code_strings[error_index]!=NULL)
322     {
323         fprintf(fd,response_code_strings[error_index],err_string);
324         break ;
325     }
326     fprintf(fd,"We are sorry to be unable to perform the method %s",
327         err_string);
328     fprintf(fd," at this time.<P>%c",LF);
329     fprintf(fd,"If you would like to see this capability in future%c",LF);
```

```
330                 fprintf(fd,"releases, send the method which failed, why you%c",LF);
331                 fprintf(fd,"would like to have it, and the server version %s%c",
                                SERVER_VERSION,LF);
332                 break ;
333             case NO_MEMORY :    // 内存不足
334                 log_error("httpd: memory exhausted");
335                 status=500 ;
336                 error_head(fd,"500 Server Error");
337                 if(header_only)break ;
338                 fprintf(fd,"The server has temporarily run out of resources for%c",LF);
339                 fprintf(fd,"your request. Please try again at a later time.<P>%c",LF);
340                 break ;
341             }
342             if(!header_only)
343                 fprintf(fd,"</BODY>%c",LF);
344             fflush(fd);
345             log_transaction(1);         // 添加一条访问日志
346             htexit(1,fd);
347     }
```

4.6 小结

本章介绍了 Apache 的日志处理模块。管理员通过对日志文件的分析，不但能了解 Apache 本身的运行情况，还可以从中得到统计信息，这些统计信息可以为网站的管理、规划、甚至于推广提供指导。

第 5 章 目录别名

5.1 概述

从第一章中我们知道，htdocs 目录是存放客户端可访问网站资源的地方（当然，您也可以设置成其它目录，本书中除特殊说明外，按照 Apache 的默认目录结构描述）。这样设计的初衷是为了方便管理，但是也带来了一个问题：网站无法对该目录外的资源进行访问。

本章讲述的目录别名为我们解决了这个问题，它提供了一种访问文档路径（htdocs 目录）外资源的一种方式。

Apache0.6.5 使用文件 http_alias.c 来处理和别名相关的操作。

Apache 提供一种方式，给不在文档目录下的文件在页面上引用提供了方便。Apache0.6.5 提供了三种类型的别名定义指令，分别是 Alias、ScriptAlias 和 Redirect。

另外一个指令 UserDir 也作为一个别名在 http_alias.c 中处理。

5.2 使用目录别名

5.2.1 为什么使用目录别名

如果你是一个程序员，可能不打算让你的文档路径给非程序员操作，但是有时候你又要使用其它人提供的资源，如图片资源，这些资源可能给专门的美工来维护，如果您把文档路径操作权限给美工，由于专业背景的不同，他们的误操作可能导致技术性问题。把这些图片放在您的文档目录之外是一个明智的做法。

Apache 的目录别名可以实现这个功能。

当然，使用目录别名的出发点可能有很多（比如提供 Linux 用户的个人站点服务），我们不能也不打算一一列举，我们在这里有一个大概的印象即可。

注意：从现在开始，本书中所有出现http://www.yourdomain.com的位置，都换成您自己的主机地址，如http://www.google.cn或http://11.22.33.44:8181。

5.2.2 如何使用目录别名

Apache0.6.5 提供了三种设置别名的指令。分别是 Alias、ScriptAlias 和 Redirect。

1. Alias

语法：Alias *URL-path file-path*|*directory-path*

功能：URL-path 路径开头的 URL 被映射到以 *directory-path* 开头的本地文件

示例：Alias /res/images /ftp/actor/images

上面的示例指定，对http://www.yourdomain.com/res/images/logo.gif的请求，服务器将返回/ftp/actor/images/logo.gif。

2. ScriptAlias

语法：ScriptAlias *URL-path file-path*|*directory-path*

功能：和 Alias 指令相同。但同时它又标明此目录中含有应该由 cgi-script 处理器处理的

CGI 脚本。

示例：ScriptAlias /cgi-bin/ /www/cgi-bin/

上面的示例指定，对 http://www.yourdomain.com/cgi-bin/test.cgi 的请求会引导服务器执行/www/cgi-bin/test.cgi 脚本

3. Redirect

语法：Redirect *URL-path URL*

功能：发送一个外部重定向使客户端重定向到一个不同的 URL

示例：Redirect /test http://www.oldapache.org

上面的示例指定，如果客户端请求http://www.yourdomain.com/test/hello.txt，则会被重定向到http://www.oldapache.org

4.UserDir

语法：UserDir *directory-filename*

功能：指定用户网站的目录

示例：UserDir /public_html

上面的示例指定，如果客户端请求http://www.yourdomain.com/~laoli/index.html，则服务器返回/home/laoli/public_html/index.html。

5.3 目录别名实践

目录别名的设置在 srm.conf 配置文件中完成。下面我们一一说明他们的配置方法和达到的效果。

注意：本章中描述的目录、配置文件信息以第一章中描述内容为准，如果您的目录信息和目录中的内容和第一章中的描述不符，请做相应修改。

5.3.1 指令实践

1. Alias 测试

在配置文件 srm.conf 中找到行

Alias /icons/ /home/devel/apache_0.6.5/icons/

如果前面有注释，去掉注释。

在 icons 目录下面有一个 back.gif 文件。在您的浏览器地址里面输入 http://www.yourdomain.com/icons/back.gif 。可以看到，服务器返回的图片即是/home/devel/apache_0.6.5/icons目录下的back.gif文件。

2. ScriptAlias 测试

在配置文件 srm.conf 中找到行

ScriptAlias /cgi-bin/ /home/devel/apache_0.6.5/cgi-bin/

如果前面有注释，去掉注释。

在cgi-bin目录下有一个脚本文件date。可以在您的浏览器路径里面输入 http://www.yourdomain.com/cgi-bin/date，以测试输出效果。测试之前请确认date文件有执行权限。如果不出意外，应该显示当前时间。

3. Redirect 测试

这是一个重定向配置，srm.conf 文件中默认没有这项配置。可以通过在 srm.conf 文件最后面增加这样一行：

Redirect /test http://www.google.cn/

在浏览器地址栏里面输入http://www.yourdomain.com/test/index.html，您的浏览器直接重定向到http://www.google.cn页面了。

4. UserDir 测试

在配置文件 srm.conf 里面。默认有一行 UserDir public_html。我们现在测试它的行为。

登录您的服务器，在您的默认工作目录下（如果您是以用户名devel登录的，这个目录应该是/home/devel）创建目录public_html。在这个目录下创建文件index.html，内容可以是简单的hello world。然后在您的浏览器地址栏里面输入http://www.yourdomain.com/~devel，如果没有意外，您应该能看到/home/devel/public_html/index.html的输出。

这种设置方式曾经在某些提供免费空间的网站上使用过。

5.3.2 进程中的数据

在做完上述修改后，您可以通过输出 http_alias.c 中定义的链表 a 和链表 v 的内容来查看此时进程中的数据。

可以在 http_config.c（功能：读取配置并赋值相应的变量）文件中的 read_config 函数最后加上一个对函数 output_alias 的调用。在 http_alias.c 中实现该函数，同时在 httpd.h 文件中声明该函数。

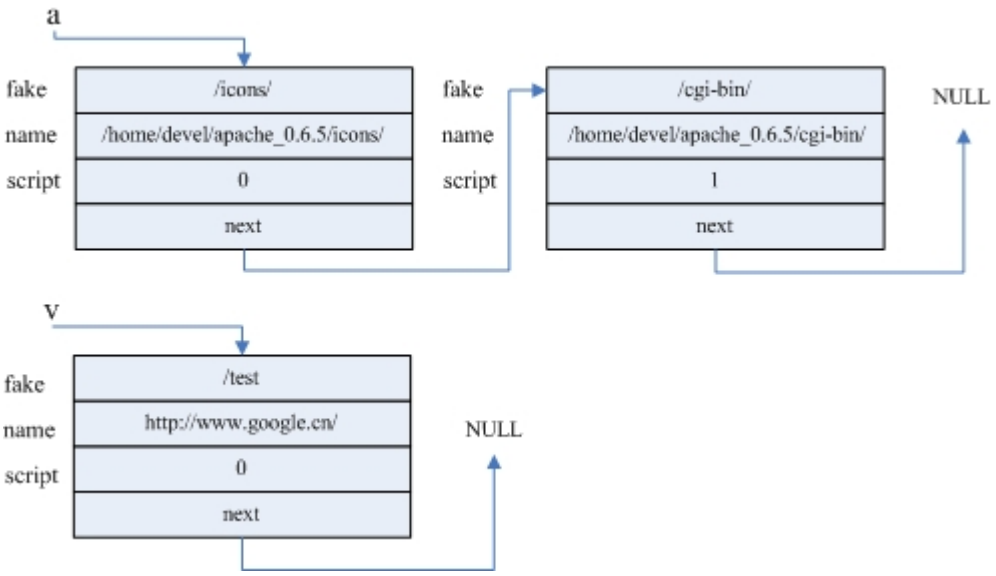
函数 output_alias 原型如下：

```
void output_alias()
{
    alias *p = a;
    int loopi = 0;
    while(p)
    {
        printf("a[%d]->fake = %s, a[%d]->real = %s, a[%d]->script = %d\n", loopi, p->fake, loopi, p->real, loopi, p->script);
        loopi ++;
     p = p->next;
    }
    p = v;
    loopi = 0;
    while(p)
    {
        printf("v[%d]->fake = %s, v[%d]->real = %s, v[%d]->script = %d\n", loopi, p->fake, loopi, p->real, loopi, p->script);
        loopi ++;
     p = p->next;
    }
}
```

重新编译 httpd。执行后可以看到输入类是这样：

```
a[0]->fake = /icons/, a[0]->real = /home/devel/apache_0.6.5/icons/, a[0]->script = 0
a[1]->fake = /cgi-bin/, a[1]->real = /home/devel/apache_0.6.5/cgi-bin/, a[1]->script = 1
v[0]->fake = /test, v[2]->real = http://www.google.cn/, v[0]->script = 0
```

这样看起来不够直观，此时内存中的链表数据类似于下图所示：



5.4 代码注释

```
1   #include "httpd.h"
2
3   typedef struct salias alias ;
4
5   struct salias
6   {
7       char*fake,*real ;    // 别名和真实目录
8       int script ;         // 脚本类型
9       alias*next ;         // 下一个别名，最后一个为 NULL
10  }
11  ;
12
     /* a 是 alias 链表首元素指针;v 是重定向 alias 链表首元素指针;*/
     /* last_a 指向 alias 链表的最后一个元素*/
     /* last_p 指向重定向 alias 链表最后一个元素*/
13  static alias*a=NULL,**last_a=&a ;
14  static alias*v=NULL,**last_v=&v ;
15   /* dirs_in_alias =  别名对应的真实目录的目录层数*/
16  int dirs_in_alias ;
17   /*  删除先前分配的别名列表  */
18  static void free_aliases(alias*p)
19  {
20      alias*next ;
21      while(p!=NULL)
22      {
23          free(p->fake);   // p->fake 分配了 fake 和 real 的空间，参看 mk_alias 函数
```

```
24              next=p->next ;
25              free(p);
26              p=next ;
27          }
28          ;
29      }
30
        /*
         * 功能:分配新的别名,如 fake=/cgi-bin/ real=/home/devel/apache-0.6.5/cgi-bin
         * 返回值: 成功返回指向新分配并赋值的空间，失败返回 NULL
         */
31      alias*mk_alias(const char*fake,const char*real,int is_script)
32      {
33          int fakelen,reallen ;
34          alias*p ;
35          /* 分配空间 */
36          p=malloc(sizeof(alias));
37          if(p==NULL)return NULL ;
38          fakelen=strlen(fake);
39          reallen=strlen(real);
40          p->fake=malloc(fakelen+reallen+2);
41          if(p->fake==NULL)
42          {
43              free(p);
44              return NULL ;
45          }
46           /* 拷贝数据 */
47          strcpy(p->fake,fake);
48          p->real=p->fake+fakelen+1 ;
49          strcpy(p->real,real);
50          p->script=is_script ;
51          p->next=NULL ;
52          return p ;
53      }
54       /* 重设(清空)别名链表 */
55      void reset_aliases()
56      {
57          free_aliases(a);
58          free_aliases(v);
59          last_a=&a ;
60          last_v=&v ;
61      }
62
         /* 根据给定的参数添加别名
```

```
 * 如 f=/cgi-bin/ r=/home/devel/apache-0.6.5/cgi-bin/
 */
63  int add_alias(char*f,char*r,int is_script)
64  {
65      alias*p ;
66
67      if(r[0]!='/') // real 不是以'/'开始的字符串，此时是个相对路径，要找到绝对路径
68      {
69          char real[MAX_STRING_LEN];
70          make_full_path((is_script?server_root:document_root),r,real);
71          p=mk_alias(f,real,is_script);
72      }
73      else
74          p=mk_alias(f,r,is_script);
75      if(p==NULL)return 0 ;
76      *last_a=p ;
77      last_a=&p->next ;
78      return 1 ;
79  }
80   /* 添加一个重定向 alias */
81  int add_redirect(char*f,char*url)
82  {
83      alias*p ;
84
85      p=mk_alias(f,url,0);
86      if(p==NULL)return 0 ;
87      *last_v=p ;
88      last_v=&p->next ;
89      return 1 ;
90  }
91

/*
* 功能:1.根据给定的 name 判断这个 name 是个什么类型的 URL
*       2.如果 URL 是个合法的 URL，则修改 URL 映射到具体的本地文件，
*          如 name=/cgi-bin/index.php。如果有 fake=/cgi-bin/
*          和 real=/home/devel/apache-0.6.5/php/这样的定义
*          那么执行后 name=/home/devel/apache-0.6.5/php/index.php
*          否则将 name 前面加上 DOCUMENT_ROOT
* 参数:
*          name:给定的 URL
*          fd:输出目的地。在这里是错误输出到哪里
* 返回值:
*    BAD_URL            错误的 URL
 *    REDIRECT_URL    一个重定向 URL
```

```
 *    SCRIPT_CGI          一个符合 ScriptAlias 定义的 URL
 *    STD_DOCUMENT    其他 URL
 *  注意:该函数可能会修改 name 的值。
 */
92  int translate_name(char*name,FILE*fd)
93  {
94      alias*p ;
95      register int l ;
96      char w[MAX_STRING_LEN];
97      struct passwd*pw ;
98
99      getparents(name);
100
101     if(name[0]!='/'&&name[0]!='\0')   // 如果 name 不是以'/'开头也不为空字符串
102         return BAD_URL ;
103
104     for(p=v;p!=NULL;p=p->next)      // 检查是不是重定向 URL
105     {
106         l=strlen(p->fake);
107         if(!strncmp(name,p->fake,l)&&(p->fake[l-1]=='/'
108             ||l==strlen(name)||name[l]=='/'))
109         {
110             strsubfirst(l,name,p->real);
111             return REDIRECT_URL ;
112         }
113     }
114
115     for(p=a;p!=NULL;p=p->next)          // 检查是否是一般的别名
116     {
117         l=strlen(p->fake);
118         if(!strncmp(name,p->fake,l)&&(p->fake[l-1]=='/'
119         ||l==strlen(name)||name[l]=='/'))
120         {
121             strsubfirst(l,name,p->real);
122             dirs_in_alias=count_dirs(p->real); // 计算 p->real 目录层数
123             return p->script ;
124         }
125     }
126
        /*用户目录，如 URL='/~devel/'。可以给每个 linux 用户分配一个站点。
        * 站点的根目录放在用户目录的 public_html 目录下。
        */
127     if((user_dir[0])&&(name[0]=='/')&&(name[1]=='~'))
128     {
```

```
129             char fake[MAX_STRING_LEN],
                real[MAX_STRING_LEN],dname[MAX_STRING_LEN];
130
131             strcpy(dname,&name[2]);      // dname='devel /'
132             getword(w,dname,'/');        // w='devel',dname=''
133             if(!(pw=getpwnam(w)))        //  没有这个用户？
134                 die(NOT_FOUND,name,fd);
135             fake[0]='/' ;
136             fake[1]='~' ;
137             strcpy(&fake[2],w);          // fake='/~devel'
138              /* user_dir 赋值参看 http_config.c 。默认设置是 public_html */
139             make_full_path(pw->pw_dir,user_dir,real);  // real=/home/devel/public_html/
140             if(!add_alias(fake,real,STD_DOCUMENT))//增加一个别名
141                 die(NO_MEMORY,"translate_name",fd);
142             strsubfirst(strlen(w)+2,name,real);
143             return STD_DOCUMENT ;
144         }
145          /*没有别名，在请求前添加 document_root*/
146         strsubfirst(0,name,document_root);
147         return STD_DOCUMENT ;
148     }
149
        /*
         * 功能:把一个对本地文件绝对目录的请求替换成相对目录的请求
         * 如把/home/devel/apache-0.6.5/htdocs/servinfo.php 替换成/servinfo.php
         * 或者把/home/devel/apache-0.6.5/cgi-bin/servinfo.php 替换成/cgi-bin/servinfo.php
         */
150     void unmunge_name(char*name)
151     {
152         register int l ;
153         alias*p ;
154
155         l=strlen(document_root);
156         if(!strncmp(name,document_root,l))
157         {
158             strsubfirst(l,name,"");
159             if(!name[0])
160             {
161                 name[0]='/' ;
162                 name[1]='\0' ;
163             }
164             return ;
165         }
166
```

```
167         for(p=a;p!=NULL;p=p->next)
168         {
169             l=strlen(p->real);
170             if(!strncmp(name,p->real,l))
171             {
172                 strsubfirst(l,name,p->fake);
173                 if(!name[0])
174                 {
175                     name[0]='/' ;
176                     name[1]='\0' ;
177                 }
178                 return ;
179             }
180         }
181     }
```

5.6 小结

目录别名原来最广泛的应用可能就是 UserDir 指令的使用了，在那个提供免费站点泛滥的年代，几乎每个网民都有一个~username 的“个人网站”，随着网络使用的成熟化，现代人又开始在博客上大做文章了，不过 Apache 提供的目录别名功能还是在很多场合和实践中被应用到。

第 6 章 MIME

6.1 概述

MIME 的英文全称是"Multipurpose Internet Mail Extensions"，即多目的 Internet 邮件扩展（协议）。它设计的最初目的是为了在发送电子邮件时附加多媒体数据，让邮件客户程序能根据其类型进行处理。

最早的 HTTP 协议中，并没有附加的数据类型信息，所有传送的数据都被客户程序解释为超文本标记语言 HTML 文档，而为了支持多媒体数据类型，HTTP 协议中就使用了附加在文档之前的 MIME 数据类型信息来标识数据类型。

例如，假设你要传送一个 Microsoft Excel 文件到客户端。那么这时的 MIME 类型就是“application/vnd.ms-excel”。在大多数实际情况中，这个文件然后将传送给 Execl 来处理（假设我们设定 Execl 为处理 application/vnd.ms-excel 类型的应用程序）。

每个 MIME 类型由两部分组成，从 mime.types 文件中能看到常见媒体格式的 MIME 定义：

text/html	html
text/plain	txt
text/richtext	rtx

服务器传输数据到客户端的时候，传输的是数据流，这样客户端并不知道接受的数据是什么 MIME 类型，所以服务器在发送真正的数据前，需要发送标识数据的 MIME 信息，这个信息使用 Content-type 关键字进行定义，例如对应 HTML 文档，服务器将首先发送下列两行 MIME 标识信息，这个标识并不是真正的数据文件的一部分：

Content-type: text/html

第二行是一个空行，使用空行的目的是将 MIME 信息与真正的数据内容分隔开。

配置文件 mime.types 中定义了一些常见的标准 MIME 类型。

本章中涉及的内容有些是为第 9 章内容服务，如果您觉得阅读本章内容比较乏味或不知道为什么要这样做，可以参考第 9 章的实际应用部分。

6.2 配置指令

配置文件 mime.types 中定义了一些标准 MIME 类型，一般不建议修改这些内容，如果你需要添加自定义的或扩展的 MIME 类型，可以通过 AddType 和 AddEncoding 指令设置。

建议自定义 MIME 添加指令添加在 srm.conf 文件中。

1. AddType 指令

语法：AddType *MIME-type extension* [*extension*] . .

功能：在给定的文件扩展名与特定的内容类型间建立映射

示例：AddType image/gif .gif

上例说明，以.gif 为扩展名的文件的 MIME 类型为 image/gif

2. AddEncoding 指令

语法：AddEncoding *MIME-enc extension* [*extension*] . . .

功能：在文件扩展名与特定的编码方式间建立映射关系
示例：AddEncoding x-gzip .gz
上例说明，以.gz 为扩展名的文件的编码方式为 x-gzip

6.3 数据组织

http_mime.c 文件中定义了三个 mime_ext 结构链表来存储标准和自定义 MIME，mime_ext 的结构如下：

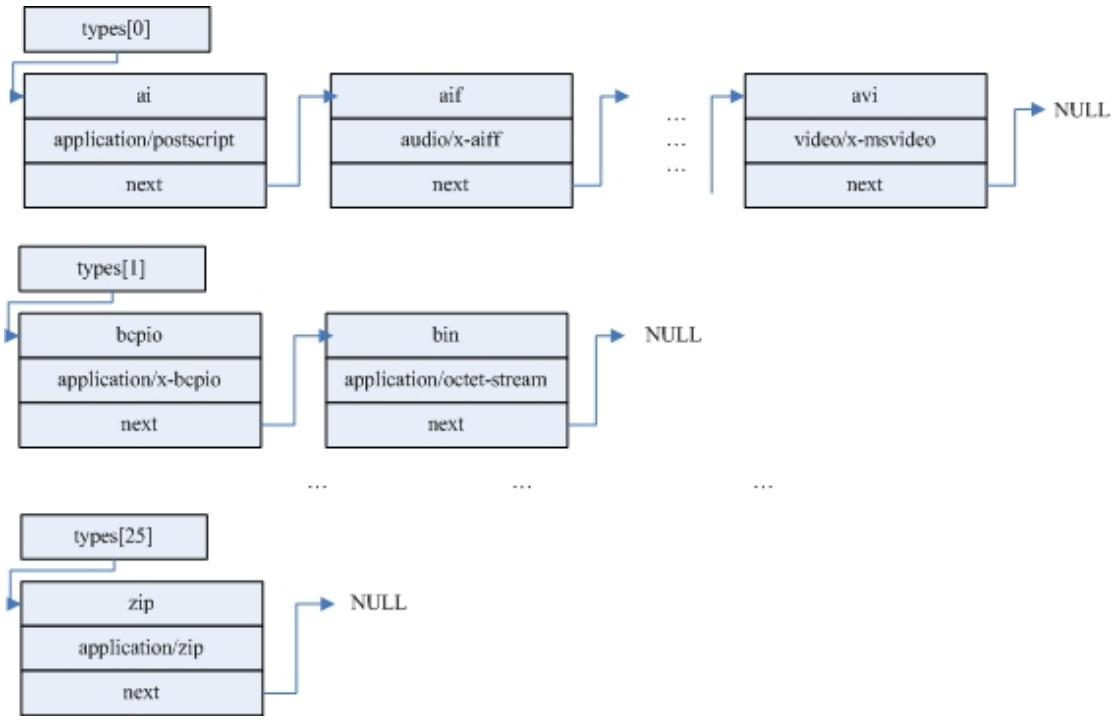
```
struct mime_ext {
    char *ext;                  //扩展名
    char *ct;                   //媒体类型
    struct mime_ext *next;      //下条记录
};
```

三个链表分别为：

1.系统默认已经定义的[扩展名/内容类型]映射关系链表数组

struct mime_ext *types[27];

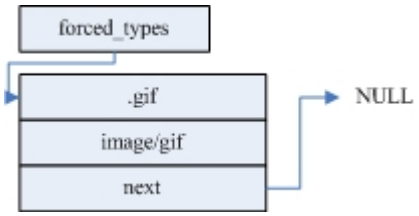
26 个字母，以开头字母为标记分组所有 MIME 记录，这个记录 conf/mime.types 文件中定义的 MIME，读取配置文件后，内存中的数据类似于下图所示：



2.用户自定义[扩展名/内容类型]映射关系链表。

struct mime_ext *forced_types;

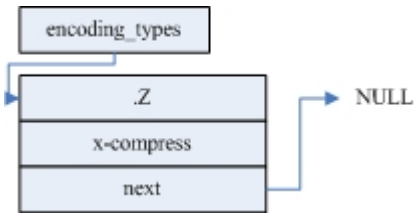
如果添加了 AddType image/gif .gif 这样的定义，内存中的数据类似于下图所示：



3.编码方式链表

struct mime_ext *encoding_types;

如果添加了 AddEncoding x-gzip .gz 这样的定义，内存中的数据类似于下图所示：



6.4 代码注释

本章代码主要包括 http_mime.c，还包括部分配置文件的读取，配置文件读取部分我们将在第 11 章中详细描述。

```
1   #include "httpd.h"
2
3   struct mime_ext
4   {
5       char*ext ;              // 扩展名
6       char*ct ;               // MIME 类型
7       struct mime_ext*next ;  //下条记录
8   }
9   ;
10
11  #define hash(i) (isalpha(i) ? (tolower(i)) - 'a' : 26)
     /*
     * 系统默认已经定义的[扩展名/内容类型]映射关系链表数组
     * 26 个字母，以开头字母为标记分组所有 MIME 记录，参看 init_mime 函数
     */
12
13  struct mime_ext*types[27];
     // 用户自定义[扩展名/内容类型]映射关系链表，如 AddType image/gif .gif
14  struct mime_ext*forced_types ;
     /*
     * 编码方式链表
     * encoding_types 主要为配置文件中类似于 AddEncoding x-gzip gz 的配置而设
     * 置的而 AddEncoding x-gzip gz 说明了.gz 扩展名的文件被认为是用 x-gzip
```

```
     * 方式编码的
     */
15   struct mime_ext*encoding_types ;
16
17   int content_length ;                              // 内容长度
18   char content_type[MAX_STRING_LEN];                // 内容类型
19   char content_encoding[MAX_STRING_LEN];            // 内容编码
20
21   char location[MAX_STRING_LEN];                    // 重定向位置
22   static char last_modified[MAX_STRING_LEN]; // 最后修改时间
23
24   char auth_line[MAX_STRING_LEN];                   // 认证行
25
26   char*out_headers ;                                // 外部头
27   char**in_headers_env ;                            // 头里面包含的环境变量
28   char*status_line ;                                // 状态行
29   char ims[MAX_STRING_LEN];                         // If-modified-since
30
     /* 将给定的结构放入类型链表的适当位置，其中:
     * 1.分组将由扩展名的第一个字母决定。
     * 2.位置将由扩展名的字符串大小排序。
     */
31   void hash_insert(struct mime_ext*me)
32   {
33       register int i=hash(me->ext[0]);   // i 是 me->ext[0]在字母表中的 index
34       register struct mime_ext*p,*q ;
35
36       if(!(q=types[i]))
37       {
38           types[i]=me ;
39           return ;
40       }
41       if((!(p=q->next))&&(strcmp(q->ext,me->ext)>=0))
42       {
43           types[i]=me ;
44           me->next=q ;
45           return ;
46       }
47       while(p)
48       {
49           if(strcmp(p->ext,me->ext)>=0)break ;
50           q=p ;
51           p=p->next ;
52       }
```

```
53          me->next=p ;
54          q->next=me ;
55      }
56       /*  清空类型链表 types/forced_types/encoding_types */
57      void kill_mime()
58      {
59          register struct mime_ext*p,*q ;
60          register int x ;
61
62          for(x=0;x<27;x++)
63          {
64              p=types[x];
65              while(p)
66              {
67                  free(p->ext);
68                  free(p->ct);
69                  q=p ;
70                  p=p->next ;
71                  free(q);
72              }
73          }
74          p=forced_types ;
75          while(p)
76          {
77              free(p->ext);
78              free(p->ct);
79              q=p ;
80              p=p->next ;
81              free(q);
82          }
83          p=encoding_types ;
84          while(p)
85          {
86              free(p->ext);
87              free(p->ct);
88              q=p ;
89              p=p->next ;
90              free(q);
91          }
92      }
93       /*  从配置文件中读取 MIME 类型记录。初始化 types 数组。*/
94      void init_mime()
95      {
96          char l[MAX_STRING_LEN],w[MAX_STRING_LEN],*ct ;
```

```
97          FILE*f ;
98          register struct mime_ext*me ;
99          register int x,y ;
             /* 一般 types_confname 是 conf/mime.types */
100         if(!(f=fopen(types_confname,"r")))
101         {
102             fprintf(stderr,
103                 "httpd: could not open mime types file %s\n",types_confname);
104             perror("fopen");
105             exit(1);
106         }
107         for(x=0;x<27;x++)
108             types[x]=NULL ;
109         forced_types=NULL ;
110         encoding_types=NULL ;
111          /* 读取一行配置文件，直到文件结束 */
112         while(!(cfg_getline(l,MAX_STRING_LEN,f)))
113         {
114             if(l[0]=='#')continue ;    // 跳过注释行(以#开头的行)
                /* 以空格分割行，w 中为类型(application/pdf)，l 中为扩展名(pdf)  */
115             cfg_getword(w,l);
116             if(!(ct=(char*)malloc(sizeof(char)*(strlen(w)+1))))
117                 die(NO_MEMORY,"init_mime",stderr);
118             strcpy(ct,w);
119             while(l[0])                 // 有多于 0 个类型扩展名？
120             {
121                 cfg_getword(w,l);
                     /* 分配一个新的 MIME 类型结构空间 */
122                 if(!(me=(struct mime_ext*)malloc(sizeof(struct mime_ext))))
123                     die(NO_MEMORY,"init_mime",stderr);
                     /* 给扩展名分配空间 */
124                 if(!(me->ext=(char*)malloc(sizeof(char)*(strlen(w)+1))))
125                     die(NO_MEMORY,"init_mime",stderr);
                     /* 将扩展名中的字母转换成小写 */
126                 for(x=0;w[x];x++)
127                     me->ext[x]=(islower(w[x])?w[x]:tolower(w[x]));
128                 me->ext[x]='\0' ;
            /* 给 me->ct 分配空间并将类型字符串(application/pdf)赋值给 mt->ct */
129                 if(!(me->ct=strdup(ct)))
130                     die(NO_MEMORY,"init_mime",stderr);
131                 me->next=NULL ;
132                 hash_insert(me); // 插入 types 链表中
133             }
134             free(ct);
```

```
135         }
136         fclose(f);
137     }
138
        /*
        * 显示所有(系统定义和用户定义的)[扩展名/内容类型]映射关系
        * 该函数可能是调试时使用的函数，并没有在系统的其它部分调用
        */
139     void dump_types()
140     {
141         struct mime_ext*p ;
142         register int x ;
143
144         for(x=0;x<27;x++)
145         {
146             p=types[x];
147             while(p)
148             {
149                 printf("ext %s: %s\n",p->ext,p->ct);
150                 p=p->next ;
151             }
152         }
153         p=forced_types ;
154         while(p)
155         {
156             printf("file %s: %s\n",p->ext,p->ct);
157             p=p->next ;
158         }
159     }
160
        /*
        * 根据文件名，在所有三个类型链表中找对应的文件类型，如果没找到，
        * 则使用第三个参数指定的 默认类型,
        * store_encoding 指定是否将找到的编码方式存储到 content_encoding
        * 变量中去。
        */
161     void find_ct(char*file,int store_encoding,char*default_type)
162     {
163         int i,l,l2 ;
164         struct mime_ext*p ;
165         char fn[MAX_STRING_LEN];
166
167         strcpy(fn,file);
168         if((i=rind(fn,'.'))>=0)      // 有扩展名
```

```
169         {
170             ++i ;
171             l=strlen(fn);
172             p=encoding_types ;
173              /* 有更多 encoding_type 的记录 */
174             while(p)
175             {       /* 比较记录中的扩展名和给定文件的扩展名，如果相同 */
176                 if(!strcmp(p->ext,&fn[i]))
177                 {
178                     fn[i-1]='\0' ;
179                     if(store_encoding)
180                     {
181                         if(content_encoding[0])
182                             sprintf(content_encoding,
                                "%s, %s",content_encoding,
183                             p->ct);
184                         else
185                             strcpy(content_encoding,p->ct);
186                     }
187                     if((i=rind(fn,'.'))<0)
188                         break ;
189                     ++i ;
190                     l=strlen(fn);
191                     p=encoding_types ;
192                 }
193                 else
194                     p=p->next ;
195             }
196         }
197         p=forced_types ;
198         l=strlen(fn);
199
200         while(p)
201         {
202             l2=l-strlen(p->ext);
203             if((l2>=0)&&(!strcasecmp(p->ext,&fn[l2])))
204             {
205                 strcpy(content_type,p->ct);
206                 return ;
207             }
208             p=p->next ;
209         }
210
211         if((i=rind(fn,'.'))<0)
```

```
212         {
213             strcpy(content_type,default_type);
214             return ;
215         }
216         ++i ;
217         p=types[hash(fn[i])];
218
219         while(p)
220         {
221             if(!strcasecmp(p->ext,&fn[i]))
222             {
223                 strcpy(content_type,p->ct);
224                 return ;
225             }
226             p=p->next ;
227         }
228         strcpy(content_type,default_type);
229     }
230      /* 检查文件的内容类型 */
231     void probe_content_type(char*file)
232     {
233         find_ct(file,0,default_type);
234     }
235      /* 设置文件的内容类型 */
236     void set_content_type(char*file)
237     {
238         find_ct(file,1,default_type);
239     }
240      /* 设置文件内容类型和参数 */
241     void set_content_type_and_parms(char*file)
242     {
243         find_ct(file,1,"");
244     }
245      /* 扫描脚本头部，赋值给对应的全局变量 */
246     int scan_script_header(FILE*f,FILE*fd,char*path)
247     {
248         char w[MAX_STRING_LEN];
249         char*l ;
250         int p ;
251
252         strcpy(content_type,default_type);
253
254         while(1)
255         {
```

```
256             if(fgets(w,MAX_STRING_LEN-1,f)==NULL)
257             {
258                 sprintf(w,"httpd: malformed header from script %s",path);
259                 die(SERVER_ERROR,w,fd);
260             }
261             p=strlen(w);
262             if(p>0&&w[p-1]=='\n')  // 去掉 w 后面的换行信息(\015='\r')
263             {
264                 if(p>1&&w[p-2]=='\015')w[p-2]='\0' ;
265                 else w[p-1]='\0' ;
266             }
267
268             if(w[0]=='\0')return 0 ;
269               /* 如果我们找到一个假冒的头信息(不含冒号':')，不要忽略它。*/
270             if(!(l=strchr(w,':')))
271             {
272                 while(fgets(w,MAX_STRING_LEN-1,f)!=NULL);
273                 fclose(f);
274                 sprintf(w,"httpd: malformed header from script %s",path);
275                 die(SERVER_ERROR,w,fd);
276                 return 0 ;  /* 毫无意义？我想是的，但是这样做并没有害处*/
277             }
278             *l++='\0' ; // 截断 w 使 w 等于冒号前的部分并使 l 指向冒号后的地方
279             if(!strcasecmp(w,"Content-type"))
280             {
281                 char*endp=l+strlen(l)-1 ;      // 删掉 l 结尾的空格
282                 while(endp>l&&isspace(*endp))*endp--='\0' ;
283                 while(*l==' ')l++;             // 删掉 l 前面的空格
284                 strcpy(content_type,l);        // 将 l 的值赋值给 content_type
285             }
286             else if(!strcasecmp(w,"Location"))
287             {
                    /* 如果我们已经有一个状态行，不要覆盖它，但是如果没有，
                    * 就创建一个
                    */
288                 if(!&status_line[0])
289                     status_line=strdup("302 Found");
290                 sscanf(l,"%s",location);
291             }
292             else if(!strcasecmp(w,"Status"))
293             {
294                 for(p=0;isspace(l[p]);p++);        // 去掉前置空格
295                 sscanf(&l[p],"%d",&status);        // 将状态值赋值给它
296                 if(!(status_line=strdup(&l[p])))   // 状态行内容就是状态值
```

```
297                     die(NO_MEMORY,"scan_script_header",fd);
298             }
299             else
300             {   // 恢复 w 的原始值，因为我们在上面把它从:位置分割成了 w 和 l
301                 *(--l)=':' ;
302                 p=strlen(w);
303                 w[p]=CR ;
304                 w[++p]=LF ;
305                 w[++p]='\0' ;
                     // 如果 out_headers 不存在
306                 if(!out_headers)
307                 {       // 把 w 的值填充到 out_headers 里面
308                     if(!(out_headers=strdup(w)))
309                         die(NO_MEMORY,"scan_script_header",fd);
310                 }
311                 else
312                 {
313                     int loh=strlen(out_headers); // out_headers 目前的长度
                         // 重新给 out_headers 分配足够的空间
314                     out_headers=(char*)
                                 realloc(out_headers,(loh+p+3)*sizeof(char));
315                     if(!out_headers)
316                         die(NO_MEMORY,"scan_script_header",fd);
                         // 把 w 的值粘帖在 out_headers 原值的后面
317                     strcpy(&out_headers[loh],w);
318                 }
319             }
320         }
321     }
322      /* 在外部头头里面添加一个属性值 */
323     void force_header(char*w,FILE*fd)
324     {
325         if(!out_headers)
326         {
327             if(!(out_headers=strdup(w)))
328                 die(NO_MEMORY,"force_header",fd);
329         }
330         else
331         {
332             int loh=strlen(out_headers);
333             int p=strlen(w);
334             out_headers=(char*)realloc(out_headers,
335                 (loh+p+1)*sizeof(char));
336             if(!out_headers)
```

```
337                     die(NO_MEMORY,"force_header",fd);
338             strcpy(&out_headers[loh],w);
339         }
340     }
341
        /*
        * 从配置文件中获取一条 AddType 信息澹(用户自定义[扩展名/内容类型]映
        * 射)，添加到[扩展名/内容类型]映射关系链表
        */
342     void add_type(char*fn,char*t,FILE*out)
343     {
344         struct mime_ext*n ;
345
346         if(!(n=(struct mime_ext*)malloc(sizeof(struct mime_ext))))
347             die(NO_MEMORY,"add_type",out);
348
349         if(!(n->ext=strdup(fn)))
350             die(NO_MEMORY,"add_type",out);
351         if(!(n->ct=strdup(t)))
352             die(NO_MEMORY,"add_type",out);
353         n->next=forced_types ;
354         forced_types=n ;
355     }
356      /* 从配置文件中获取一条编码方式信息，添加到编码方式信息链表中 */
357     void add_encoding(char*fn,char*t,FILE*out)
358     {
359         struct mime_ext*n ;
360
361         if(!(n=(struct mime_ext*)malloc(sizeof(struct mime_ext))))
362             die(NO_MEMORY,"add_encoding",out);
363
364         if(!(n->ext=strdup(fn)))
365             die(NO_MEMORY,"add_encoding",out);
366         if(!(n->ct=strdup(t)))
367             die(NO_MEMORY,"add_encoding",out);
368         n->next=encoding_types ;
369         encoding_types=n ;
370     }
371      /* 设置内容长度 */
372     void set_content_length(int l)
373     {
374         content_length=l ;
375     }
376
```

```
        /*
        * 设置最后一次修改时间为当前时间。如果这个时间小于等于客户端
        * 的 If-modified-since 时间，给客户端返回 304 状态码
        */
377     void set_last_modified(time_t t,FILE*out)
378     {
379         struct tm*tms ;
380         char ts[MAX_STRING_LEN];
381
382         tms=gmtime(&t);
383         strftime(ts,MAX_STRING_LEN,HTTP_TIME_FORMAT,tms);
384         strcpy(last_modified,ts);
385
386         if(!ims[0])
387             return ;
388
389         if(later_than(tms,ims))
390             die(USE_LOCAL_COPY,NULL,out);
391     }
392      /* 初始化头变量值 */
393     void init_header_vars()
394     {
395         content_type[0]='\0' ;
396         last_modified[0]='\0' ;
397         content_length=-1 ;
398         auth_line[0]='\0' ;
399         content_encoding[0]='\0' ;
400         location[0]='\0' ;
401         ims[0]='\0' ;
402         status_line=NULL ;
403         out_headers=NULL ;
404         in_headers_env=NULL ;
405     }
406
        /*
        * 合并头部同名变量内容。
        * 比如现有头部内容里面含有一条:HTTP_ACCEPT= image/gif
        * 而给定的 h='HTTP_ACCEPT', v='image/jpeg'
        * 那么合并后就成了:HTTP_ACCEPT= image/gif, image/jpeg
        * 找到并合并成功返回 1，否则返回 0
        */
407     int merge_header(char*h,char*v,FILE*out)
408     {
409         register int x,l,lt ;
```

```
410         char**t ;
411
412         for(l=0;h[l];++l);
413         h[l]='=' ;
414         h[++l]='\0' ;
415
416         for(t=in_headers_env;*t;++t)
417         {
418             if(!strncmp(*t,h,l))
419             {
420                 lt=strlen(*t);
421                 if(!(*t=(char*)realloc(*t,(lt+strlen(v)+3)*sizeof(char))))
422                     die(NO_MEMORY,"merge_header",out);
423                 (*t)[lt++]=',' ;
424                 (*t)[lt++]=' ' ;
425                 strcpy(&((*t)[lt]),v);
426                 return 1 ;
427             }
428         }
429         h[l-1]='\0' ;
430         return 0 ;
431     }
432
      /*
      *   根据请求里面的 MIME 信息设置本地相关变量值并更新 in_headers_env
      *   链表。
      *   MIME 头里面的信息类似于:
      *   Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
      *   application/x-shockwave-flash, application/vnd.ms-excel,
      *   application/vnd.ms-powerpoint, application/msword, * /*
      *   Accept-Language: zh-cn
      *   Accept-Encoding: gzip, deflate
      *   User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
      *   .NET CLR 1.1.4322)
      *   Host: 127.0.0.1
      *   Connection: Keep-Alive
      */
433     void get_mime_headers(int fd,FILE*out)
434     {
435         char w[MAX_STRING_LEN];
436         char l[MAX_STRING_LEN];
437         int num_inh,num_processed ;
438         char*t ;
439
```

```
440         num_inh=0 ;
441         num_processed=0 ;
442
443         while(!(getline(w,MAX_STRING_LEN-1,fd,timeout)))
444         {
445             if(!w[0])
446                 return ;
447             if((++num_processed)>MAX_HEADERS)
448                 die(BAD_REQUEST,"too many header lines",out);
449             if(!(t=strchr(w,':')))        // 没找到冒号
450                 continue ;
451             *t++='\0' ;
452             while(isspace(*t))++t ;   // 跳过空格
453             strcpy(l,t);                  // 如:l=zh-cn
454
455             if(!strcasecmp(w,"Content-type"))
456             {
457                 strcpy(content_type,l);
458                 continue ;
459             }
460             if(!strcasecmp(w,"Authorization"))
461             {
462                 strcpy(auth_line,l);
463                 continue ;
464             }
465             if(!strcasecmp(w,"Content-length"))
466             {
467                 sscanf(l,"%d",&content_length);
468                 continue ;
469             }
470             if(!strcasecmp(w,"If-modified-since"))
471                 strcpy(ims,l);
472              /*  修改 w 变量名称的值为'HTTP_'+UPPER(w.replaceAll("-","_")) */
473             http2cgi(w);
474             if(in_headers_env)  // in_headers_env 已经存在
475             {       /*合并里面已经有的相应变量的值,如果没找到，创建新变量  */
476                 if(!merge_header(w,l,out))
477                 {
478                     in_headers_env=(char**)realloc
                                    (in_headers_env,(num_inh+2)*sizeof(char*));
479                     if(!in_headers_env)
480                         die(NO_MEMORY,"get_mime_headers",out);
481                     in_headers_env[num_inh++]=make_env_str(w,l,out);
482                     in_headers_env[num_inh]=NULL ;
```

```
483                     }
484                 }
485                 else //不存在，就创建并把变量表达式写进 in_headers_env 链表
486                 {
487                     if(!(in_headers_env=(char**)malloc(2*sizeof(char*))))
488                         die(NO_MEMORY,"get_mime_headers",out);
489                     in_headers_env[num_inh++]=make_env_str(w,l,out);
490                     in_headers_env[num_inh]=NULL ;
491                 }
492             }
493     }
494      /* 附加默认头部 */
495     void dump_default_header(FILE*fd)
496     {
497         fprintf(fd,"Date: %s\015\012",gm_timestr_822(time(NULL)));
498         fprintf(fd,"Server: %s\015\012",SERVER_VERSION);
499     }
500      /* 发送 http 头信息到 fd */
501     void send_http_header(FILE*fd)
502     {
503         if(!status_line)
504         {
505             if(location[0])
506             {
507                 status=302 ;
508                 status_line="302 Found" ;
509             }
510             else
511             {
512                 status=200 ;
513                 status_line="200 OK" ;
514             }
515         }
516         begin_http_header(fd,status_line);
517         if(content_type[0])
518             fprintf(fd,"Content-type: %s\015\012",content_type);
519         if(last_modified[0])
520             fprintf(fd,"Last-modified: %s\015\012",last_modified);
521         if(content_length>=0)
522             fprintf(fd,"Content-length: %d\015\012",content_length);
523         if(location[0])
524             fprintf(fd,"Location: %s\015\012",location);
525         if(content_encoding[0])
526             fprintf(fd,"Content-encoding: %s\015\012",content_encoding);
```

```
527             if(out_headers)
528                     fprintf(fd,"%s",out_headers);
529             fputs("\015\012",fd);
530     }
```

6.5 小结

最初的 HTML 只处理文本内容，当代的 HTML 标准已经包括了图片、音乐、视频等多媒体信息在内，对网站的使用者来说页面越来越丰富多彩，对 Web 服务器的开发者来说，就提出了相应的问题。

本章描述的方法和思路就是为解决多媒体信息而来的，它的解决方式和实现方法值得我们借鉴。

第 7 章 服务器端包含（SSI）

7.1 概述

现在的 Apache 本身除了提供静态页面服务之外，还可以通过加载第三方扩展模块方式支持 php、jsp 等脚本语言，提供动态页面效果。您也许不知道，其实即便不使用这些脚本语言，Apache 也支持在您的页面里面加入动态内容。

所谓的动态，就是在不同条件下显示不同的内容，比如当前时间的显示就属于动态内容显示。Apache0.6.5 对动态内容支持包括两部分，一部分是本章描述的 SSI，另一部分是将要在下一章描述的 cgi 脚本。

SSI 全称是 Server Side Include，即服务器端包含。SSI 允许在静态文件中显示一些系统变量甚至执行 shell 命令或 cgi 脚本。

服务器包含部分的功能在 http_include.c 中实现。

7.2 背景知识

7.2.1 字符实体

如果您做过页面或者对 HTML 比较熟悉，一定对字符实体这个概念不陌生。

我们在做页面的时候，有时候会遇到这样的情况：我想在某个位置加入几个连续的空格，如果我在源代码里面输入空格，不论输入多少个，在页面显示的时候都是显示一个空格，因为 HTML 会自动截去多余的空格。不管你加多少空格，都被看做一个空格。

这时候我们一般的做法是用 来代替空格，也许您觉得 不够专业，明眼人一看就知道是空格，那好，我们还有一个替代方案，使用 来代替空格。在源代码中连续输入几个 序列或 序列，用浏览器打开，是不是显示出几个空格呢？

这个 或 就是 HTML 字符实体（HTML Character Entities）。

一个字符实体分成三部分：第一部分是一个&符号；第二部分是实体(Entity)名字或者是#加上实体(Entity)编号；第三部分是一个分号。

用实体名字的好处是比较好理解，一看 ，大概就猜出是 none-break space 的意思，但是其劣势在于并不是所有的浏览器都支持最新的 Entity 名字。而实体(Entity)编号，各种浏览器都能处理。

注意：Entity 是区分大小写的。

最常用的字符实体

显示结果	说明	Entity Name	Entity Number
	显示一个空格		
<	小于	&lt;	&#60;
>	大于	&gt;	&#62;
&	&符号	&amp;	&#38;

其他常用的字符实体

更多 HTMLver4 的字符实体定义参看：http://www.w3.org/TR/html4/sgml/entities.html

7.2.2 环境变量

做 Java 开发的人一定对环境变量这个概念不陌生，相信每个 Java 程序员在成功输出 Hello World 之前都对环境变量进行了一些或艰苦或顺利的设置。

其实每个程序运行的时候都使用了环境变量，只是你没有察觉而已。

我们知道，如果想在 Linux 下执行一个可执行文件，一般需要给出这个可执行文件的绝对或相对位置，比如我们启动 Apache 的时候，就需要类似下面这样的方式：

```
[devel@RIZI src]$ pwd
/home/devel/apache_0.6.5/src
[devel@RIZI src]$ httpd
[devel@RIZI src]$
```

如果我们在/home/devel/apache_0.6.5/src 目录外的任何位置执行下面的指令，是不能启动 Apache 的：

```
[devel@RIZI ~]$ pwd
/home/devel
[devel@RIZI ~]$ httpd
-bash: httpd: command not found
[devel@RIZI ~]$
```

这时因为系统“没找到”httpd 这个可执行文件。那么系统都会在哪里“找”呢？答案是系统变量 PATH 中。

Bash（或任何您当前在使用的 shell）根据您输入的命令在 PATH 系统变量指定的位置找对应的可执行文件，如果找到了，就根据您的输入参数执行这个程序，如果没找到，就会给出“命令没找到”提示。

系统常用命令都在 PATH 系统变量指定的目录中，这也是我们可以在任何目录执行系统命令如 ls 的原因。

我们可以通过 env 命令来查看当前登录用户的系统变量：

```
[devel@RIZI ~]$ env
SHELL=/bin/bash
HISTSIZE=1000
…… ……
PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/devel/bin
…… ……
OLDPWD=/home/devel/apache_0.6.5
[devel@RIZI ~]$
```

而 ls 命令的位置一般在/bin 目录下：

```
[devel@RIZI ~]$ which ls
/bin/ls
[devel@RIZI ~]$
```

Apache 维护一个自定义的环境变量数据结构供 SSI 使用。

7.3 配置Apache支持SSI

如果您要使用 SSI，需要修改两个配置文件：srm.conf 和 access.conf

7.3.1 配置方法

1.在 srm.conf 中找到这样的行：
#AddType text/x-server-parsed-html .shtml
如果没有，增加这样一行，并去掉前面的注释(#)。
AddType 的含义请参看 7.3.2
2.在 access.conf 中找到您要使用 SSI 的目录，如您的站点根目录
<Directory /home/devel/apache-0.6.5//htdocs>
Options Indexes FollowSymLinks
</Directory>
在 Opetion 后面增加 Includes 选项。

7.3.2 配置指令解析

1.AddType 指令
语法：AddType *MIME-type extension* [*extension*] ...
功能：在给定的文件扩展名与特定的内容类型间建立映射
示例：AddType text/x-server-parsed-html .shtml
上例说明，.shtml 为扩展名的文件是 text/x-server-parsed-html 类型的文件
2.Option 指令
语法：Options *可选项* [*可选项*] ...
功能：配置在特定目录使用哪些特性，该指令出现在<Directory>指令内部
示例：Options Indexes FollowSymLinks
上例说明，在指定的目录（<Directory>指令制定的目录）下列表文件，可以使用符号链接。具体参数及含义如下：
Indexes：如果服务器收到了一个映射到该目录的 URL 请求，而目录下又没有 DirectoryIndex(如 index.html)，那么服务器返回一个格式化目录列表。
Includes：允许服务器端包含
IncludesNOEXEC：允许服务器端包含，但禁用#exec 命令和#exec CGI
FollowSymLinks：服务器会在此目录中使用符号连接
SymLinksIfOwnerMatch：符号连接仅当与它的目的文件或目录具有相同的用户 ID 时才使用。简单理解就是符号链接的所有者和它指向的文件或目录是同一个人
execCGI：允许执行 CGI 脚本
MultiViews：这是一个被诟病的功能。允许你访问页面时不需要文件的扩展名。比如，你有一个叫 "evaluation.txt" 的文件，在启用 MultiViews 的站点，你可以用 "example.com/evaluation"来访问到这个文件，该指令在我们要讲解的版本中不予支持
None：都不允许
All：除 MultiViews 之外的所有特性

7.4 SSI命令解说

SSI 命令是嵌入在正常的 HTML 源代码中的。

SSI 在使用时遵循以下格式：

<!--#directive parameter="value"-->

其中，directive 是向服务器发送的指令名称，parameter 是指令的操作对象，而 value 则是用户希望得到的指令处理结果。

所有的 SSI 命令都是以"<!--#"开始，其中"<!-"和"#"之间不能有任何空格，否则服务器会把 SSI 命令当称普通的文件注释处理，不会显示出任何结果，也不会产生错误提示。此外，SSI 命令中的"="两边不能有空格，右边的值必须包含在双引号内，后面可以跟空格，最后是结束标签"-->"。

Apache0.6.5 支持 6 种类型的 SSI 命令，如下所示：

Config errmsg, timefmt, sizefmt
include virtual, file
echo var
fsize file
flastmod file
exec cmd, cgi

下面我们根据实例，一一解说。

7.4.1 config命令

Config 命令主要用于修改 SSI 的默认设置。主要有一个内容设置和两个格式设置：

1. errmsg

内容设置，设置错误信息的内容，在里面你可以加上自己的邮件地址之类的个人自定义信息。如：

<!--#config errmsg="Error! Please email webmaster@mydomain.com -->

2. timefmt

格式设置，可以重新设置 SSI 输出时间的格式，需要在显示某个时间前面设置，如：

<!--#config timefmt="%A, %B %d, %Y"-->
<!--#echo var="LAST_MODIFIED" -->

显示结果为：

Thursday, March 26, 2009

具体的时间格式参看附录。

3. sizefmt

格式设置，可以定义文件的大小以什么为单位显示。是以字节、千字节还是兆字节为单位表示。如果以字节为单位，参数值为"bytes"；对于千字节和兆字节可以使用缩写形式。同样，sizefmt 参数必须放在 fsize 命令的前面才能使用。

<!--#config sizefmt="bytes" -->
<!--#fsize file="index.html" -->

7.4.2 include命令

有时候我们在所有的页面里面会引用一些内容，比如会有几个固定链接，如“版权信息 | 关于我们 | 友情链接”等。如果我们在每个页面里面都直接插入这样的信息，一旦有相关信息的变动，如关于我们页面的地址从/include/aboutus.html 改到了/res/aboutus.html，我们就需要修改每个页面。这样的工作是繁杂且容易出错的。这时候我们可以使用 include 指令来解决这个问题。这个命令也许是 SSI 里面最有价值且使用频率最高的命令了

Include 命令可以有两个不同的参数：

1.Virtual：给出到服务器端某个文档的虚拟路径。例如：

<!--#include virtual="/res/readme.html" -->

2.File：给出到当前目录的相对路径，其中不能使用"../"，也不能使用绝对路径。例如：

<!--#include file="readme.html" -->

这就要求每一个目录中都包含一个 readme.html 文件。

7.4.3 echo命令

Echo 命令可以显示以下各环境变量：

1. DOCUMENT_NAME
   显示当前文档的名称。
   <!--#echo var="DOCUMENT_NAME" -->
   显示结果为：hello.shtml
2. DOCUMENT_URL
   显示当前文档的虚拟路径。例如：
   <!--#echo var="DOCUMENT_URI" -->
   显示结果为：/hello.shtml
3. DATE_LOCAL
   显示服务器设定时区的日期和时间。
   <!--#echo var="DATE_LOCAL" -->
   显示结果为：Thursday, 26-Mar-09 16:59:31 CST
4. DATE_GMT
   功能与 DATE_LOCAL 一样，只不过返回的是以格林尼治标准时间为基准的日期
   <!--#echo var="DATE_GMT" -->
   显示结果为：Thursday, 26-Mar-09 08:59:31 GMT
5. LAST_MODIFIED
   显示当前文档的最后更新时间。
   <!--#echo var="LAST_MODIFIED" -->
   显示结果为：Thursday, 26-Mar-09 16:57:27 CST
6. CGI 环境变量
   PATH：服务器 PATH 信息
   SERVER_SOFTWARE：服务器软件版本信息
   SERVER_NAME：服务器名
   SERVER_PORT：服务端口号
   REMOTE_HOST：客户端名称

REMOTE_ADDR：客户端地址
DOCUMENT_ROOT：站点根目录
SERVER_ADMIN：站点管理员邮件地址
REMOTE_USER：远程用户
AUTH_TYPE：认证类型
REDIRECT_QUERY_STRING：重定向参数信息
REDIRECT_URL：重定向 URL

最后两项和重定向有关，REMOTE_USER 和 AUTH_TYPE 跟认证有关。这四项只有在特定情况下才能有值。

7.4.4 fsize命令

显示指定文件的大小。如：
<!--#fsize file="hello.zip" -->
输出为：1K
可以结合 config 命令的 sizefmt 参数定制输出格式，如：
<!--#config sizefmt="bytes" -->
<!--#fsize file="hello.zip" -->
输出为：2
说明这个文件只有 2 字节。

7.4.5 flastmod命令

显示指定文件的最后修改日期。
<!--#flastmod file="hello.zip" -->
输出为：Friday, 20-Mar-09 16:33:12 CST

7.4.6 exec命令

exec 命令可以执行 CGI 脚本或者 shell 命令。使用方法如下：

cmd：使用/bin/sh 执行指定的字串。如果服务器配置 Option 时使用了 IncludesNOEXEC 选项，则该命令将被屏蔽。

cgi：可以用来执行 CGI 脚本。例如，下面这个例子中使用服务端 cgi-bin 目录下的 date 脚本程序显示当前时间：

<!--#exec cgi="/cgi-bin/date" -->

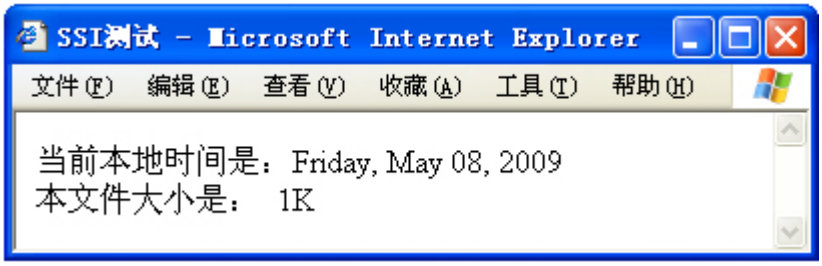
7.4.7 一个简单的实例

本小节我们提供一个简单的，使用 SSI 功能的 HTML 页面，来检测 Apache 对 SSI 的支持。该 HTML 文件的源代码如下：

<html xmlns="http://www.w3.org/1999/xhtml">
<head>

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>SSI 测试</title>
</head>
<body>
<!--#config timefmt="%A, %B %d, %Y" -->
当前本地时间是：<!--#echo var="DATE_LOCAL" -->
<br/>
本文件大小是：<!--#fsize file="ssi.shtml" -->
</body>
</html>
```

您可以在http://www.oldapache.org/example/chapter7/ssi.rar下载该代码。下载后解压缩，将解压缩后得到的ssi.shtml文件放到您的htdocs目录，按照 7.3 节中的说明配置Apache，重新启动Apache使新配置生效。从浏览器中访问页面http://www.yourdomain.com/ssi.shtml，在我的浏览器中得到的结果如下图所示：



7.5 代码注释

```
1   #include "httpd.h"
2
3   #define STARTING_SEQUENCE "<!--#"    // ssi 命令起始标识
4   #define ENDING_SEQUENCE "-->"        // ssi 命令结束标识
     /* 默认错误信息 */
5   #define DEFAULT_ERROR_MSG "[an error occurred while processing this directive]"
6   #define DEFAULT_TIME_FORMAT "%A, %d-%b-%y %T %Z" // 默认时间格式
7   #define SIZEFMT_BYTES 0
8   #define SIZEFMT_KMG 1
9
10  static time_t date,lm ;
11
12  static void decodehtml(char*s);
13  static char*get_tag(FILE*in,char*tag,int dodecode);
14  static int get_directive(FILE*in,char*d);
15   /* ----------------------- 环境变量函数 ------------------------- */
16  #define NUM_INCLUDE_VARS 5
17
     /*
     * 增加一个 include 环境变量
```

```
 * 包括设置时间格式，和给定文件的名称、URI 以及最后修改实践
 */
char**add_include_vars(char**env,char*file,char*path_args,
                                    char*args,char*timefmt,FILE*out)
{
    int x ;
    struct stat finfo ;
    char ufile[HUGE_STRING_LEN];
    char*t ;

    if(!(env=new_env(env,NUM_INCLUDE_VARS,&x)))
        die(NO_MEMORY,"add_include_vars",out);
    date=time(NULL);
    env[x++]=make_env_str("DATE_LOCAL",ht_time(date,timefmt,0),out);
    env[x++]=make_env_str("DATE_GMT",ht_time(date,timefmt,1),out);

    if(stat(file,&finfo)!=-1)
    {
        lm=finfo.st_mtime ;
        env[x++]=make_env_str("LAST_MODIFIED",ht_time(lm,timefmt,0),out);
    }
    strcpy(ufile,file);
    unmunge_name(ufile);
    env[x++]=make_env_str("DOCUMENT_URI",ufile,out);
    if(t=strrchr(ufile,'/'))
        ++t ;
    else
        t=ufile ;
    env[x++]=make_env_str("DOCUMENT_NAME",t,out);
    env[x]=NULL ;
    return env ;
}

#define GET_CHAR(f,c,r) \
  { \
    int i = getc(f); \
    if(feof(f) || ferror(f) || (i == -1)) { \
        fclose(f); \
        return r; \
    } \
    c = (char)i; \
  }
  /* -------------------------- 解析函数 -------------------------- */
int find_string(FILE*in,char*str,FILE*out)
```

```
60  {
61      int x,l=strlen(str),p ;
62      char c ;
63
64      p=0 ;
65      while(1)
66      {
67          GET_CHAR(in,c,1);
68          if(c==str[p])
69          {
70              if((++p)==l)
71                  return 0 ;
72          }
73          else
74          {
75              if(out)
76              {
77                  if(p)
78                  {
79                      for(x=0;x<p;x++)
80                      {
81                          putc(str[x],out);
82                          ++bytes_sent ;
83                      }
84                  }
85                  putc(c,out);
86                  ++bytes_sent ;
87              }
88              p=0 ;
89          }
90      }
91  }
92
     /*
     * 解码一个包含 html 实体字符的字符串
     * 参数 s 会被解码后的字符串覆盖(替换)
     * 如果有语法错误，则做如下修正:
     * 未知的元字符将被从解码字符串中删除，如&rizixiu;
     * 引用了一个未使用的数字字符将会被删除，如&#26;
     * 特殊的，&#00; 将不会被解码，但是会被删除
     */
93  #define MAXENTLEN (6)    // 任何一个 ISO-LATIN-1 HTML 实体名字长度
94  typedef const char*cchar ;
95
```

```
96  static void
97  decodehtml(char*s)
98  {
99      int val,i,j ;
100     char*p=s ;
101     cchar ents ;
102     static const cchar entlist[MAXENTLEN+1]=
103     {
104         NULL,    /* 0 */
105         NULL,    /* 1 */
106         "lt\074gt\076", /* 2 */
107         "amp\046ETH\320eth\360", /* 3 */
108         "quot\042Auml\304Euml\313Iuml\317Ouml\326Uuml\334auml
            \344euml\353\
109         iuml\357ouml\366uuml\374yuml\377", /* 4 */
110     "Acirc\302Aring\305AElig\306Ecirc\312Icirc\316Ocirc\324Ucirc\333\
111     THORN\336szlig\337acirc\342aring\345aelig\346ecirc\352icirc\356ocirc\364\
112     ucirc\373thorn\376", /* 5 */
113     "Agrave\300Aacute\301Atilde\303Ccedil\307Egrave\310Eacute\311\
114     Igrave\314Iacute\315Ntilde\321Ograve\322Oacute\323Otilde\325Oslash\330\
115     Ugrave\331Uacute\332Yacute\335agrave\340aacute\341atilde\343ccedil\347\
116     egrave\350eacute\351igrave\354iacute\355ntilde\361ograve\362oacute\363\
117     otilde\365oslash\370ugrave\371uacute\372yacute\375" /* 6 */
118 }
119 ;
120
121     for(;*s!='\0';s++,p++)
122     {
123         if(*s!='&')
124         {
125             *p=*s ;
126             continue ;
127         }
128          /* 找到字符实体结束位置 */
129         for(i=1;s[i]!=';'&&s[i]!='\0';i++);
130
131         if(s[i]=='\0')
132         {
133             *p=*s ;
134             continue ;
135         }
136          /* 是数字吗？ */
137         if(s[1]=='#')
138         {
```

```
139                     for(j=2,val=0;j<i&&isdigit(s[j]);j++)
140                         val=val*10+s[j]-'0' ;
141                     s+=i ;
142                     if(j<i||val<=8||(val>=11&&val<=31)||
143                         (val>=127&&val<=160)||val>=256)
144                         p--;        // 没有数据要输出了
145                     else
146                         *p=val ;
147                 }
148                 else
149                 {
150                     j=i-1 ;
151                     if(i-1>MAXENTLEN||entlist[i-1]==NULL)
152                     {
153                         *p='&' ;
154                         continue ;// 跳过
155                     }
156                     for(ents=entlist[i-1];*ents!='\0';ents+=i)
157                         if(strncmp(s+1,ents,i-1)==0)break ;
158
159                         if(*ents=='\0')
160                             *p='&' ;
161                         else
162                         {
163                             *p=((const unsigned char*)ents)[i-1];
164                             s+=i ;
165                         }
166                 }
167             }
168
169             *p='\0' ;
170         }
171
            /*
             * 获取下一个标签的名称和值
             * 如果没有下一个标签，设置标签的名称为'done'
             * 如果 dodecode 非零，标签的值需要解码处理
             */
172         static char*get_tag(FILE*in,char*tag,int dodecode)
173         {
174             char*t=tag,*tag_val,c,term ;
175             int n ;
176
177             n=0 ;
```

```
178
179         do   // 跳过空格
180         {
181             GET_CHAR(in,c,NULL);
182         }
183         while(isspace(c));
184          // 标签不能以-开头
185         if(c=='-')
186         {
187             GET_CHAR(in,c,NULL);
188             if(c=='-')
189             {
190                 do
191                 {
192                     GET_CHAR(in,c,NULL);
193                 }
194                 while(isspace(c));
195                 if(c=='>')
196                 {
197                     strcpy(tag,"done");
198                     return tag ;
199                 }
200             }
201             return NULL ; //  失败
202         }
203
204         while(1)
205         {
206             if(++n==MAX_STRING_LEN)
207             {
208                 t[MAX_STRING_LEN-1]='\0' ;
209                 return NULL ;
210             }
211             if(c=='='||isspace(c))break ;
212             *(t++)=tolower(c);
213             GET_CHAR(in,c,NULL);
214         }
215
216         *t++='\0' ;
217         tag_val=t ;
218
219         while(isspace(c))GET_CHAR(in,c,NULL);
220         if(c!='=')return NULL ;
221
```

```
222         do   // 跳过等号（=）后的空格
223         {
224             GET_CHAR(in,c,NULL);
225         }
226         while(isspace(c));
227 
228         if(c!=""&&c!='\")return NULL ;
229         term=c ;
230         while(1)
231         {
232             GET_CHAR(in,c,NULL);
233             if(++n==MAX_STRING_LEN)
234             {
235                 t[MAX_STRING_LEN-1]='\0' ;
236                 return NULL ;
237             }
238             if(c==term)break ;
239             *(t++)=c ;
240         }
241         *t='\0' ;
242         if(dodecode)decodehtml(tag_val);
243         return tag_val ;
244     }
245      /* 获取指令 */
246     static int get_directive(FILE*in,char*d)
247     {
248         char c ;
249 
250         while(1)
251         {
252             GET_CHAR(in,c,1);
253             if(!isspace(c))
254                 break ;
255         }
256         while(1)
257         {
258             *d++=tolower(c);
259             GET_CHAR(in,c,1);
260             if(isspace(c))
261                 break ;
262         }
263         *d='\0' ;
264         return 0 ;
265     }
```

```
266
267     void send_parsed_content(char*file,FILE*f,FILE*fd,
268                                         char*path_args,char*args,
269                                         char**env,int noexec);
270      /*  发送 include 文件到客户端  */
271     int send_included_file(char*file,FILE*out,char**env,char*fn)
272     {
273           FILE*f ;
274           struct stat finfo ;
275           int allow ;
276           char op,i ;
277
278           if(stat(file,&finfo)==-1)
279                 return-1 ;
280           evaluate_access(file,&finfo,M_GET,&allow,&op,out);
281           if(!allow)
282                 return-1 ;
283           set_content_type(file);
284           if((op&OPT_INCLUDES)&&(!strcmp(content_type,INCLUDES_MAGIC_TYPE)
285                 ||!strcmp(content_type,INCLUDES_MAGIC_TYPE3)))
286           {
287                 if(!(f=fopen(file,"r")))
288                       return-1 ;
289                 send_parsed_content(file,f,out,"","",env,op&OPT_INCNOEXEC);
290                 chdir_file(fn);
291           }
292           else if(!strcmp(content_type,CGI_MAGIC_TYPE))
293                 return-1 ;
294           else
295           {
296                 if(!(f=fopen(file,"r")))
297                       return-1 ;
298                 send_fd(f,out,NULL);
299                 fclose(f);
300           }
301           return 0 ;
302     }
303      /*  处理 include 指令  */
304     int handle_include(FILE*in,FILE*out,char*fn,char**env,char*error)
305     {
306           char tag[MAX_STRING_LEN],errstr[MAX_STRING_LEN];
307           char*tag_val ;
308
309           while(1)
```

```
310         {
311         if(!(tag_val=get_tag(in,tag,1)))
312                 return 1 ;
313             if(!strcmp(tag,"file"))          // 处理 file 指令
314             {
315                 char dir[MAX_STRING_LEN],to_send[MAX_STRING_LEN];
316
317                 getparents(tag_val);
318                 getcwd(dir,MAX_STRING_LEN);
319                 make_full_path(dir,tag_val,to_send);
320                 if(send_included_file(to_send,out,env,fn))
321                 {
322                     sprintf(errstr,
323                         "unable to include %s in parsed file %s",tag_val,fn);
324                     log_error_noclose(errstr);
325                     bytes_sent+=fprintf(out,"%s",error);
326                 }
327             }
328             else if(!strcmp(tag,"virtual")) // 处理 virtual 指令
329             {
330                 if(translate_name(tag_val,out)!=STD_DOCUMENT)
331                 {
332                     bytes_sent+=fprintf(out,"%s",error);
333                     log_error_noclose(errstr);
334                 }
335                 else if(send_included_file(tag_val,out,env,fn))
336                 {
337                     sprintf(errstr,
338                         "unable to include %s in parsed file %s",tag_val,fn);
339                     log_error_noclose(errstr);
340                     bytes_sent+=fprintf(out,"%s",error);
341                 }
342             }
343             else if(!strcmp(tag,"done"))   // 结束指令
344                 return 0 ;
345             else                           // 未知指令
346             {
347                 sprintf(errstr,"unknown parameter %s to tag echo in %s",tag,fn);
348                 log_error_noclose(errstr);
349                 bytes_sent+=fprintf(out,"%s",error);
350             }
351         }
352     }
353      /* 处理 echo 指令，主要输出自定义的环境变量 */
```

```
354  int handle_echo(FILE*in,FILE*out,char*file,char*error,char**env)
355  {
356      char tag[MAX_STRING_LEN];
357      char*tag_val ;
358
359      while(1)
360      {
361          if(!(tag_val=get_tag(in,tag,1)))
362              return 1 ;
363          if(!strcmp(tag,"var"))
364          {
365              int x,i,len ;
366
367              len=strlen(tag_val);
368              for(x=0;env[x]!=NULL;x++)
369              {
370                  i=ind(env[x],'=');
371                  if(i==len&&strncmp(env[x],tag_val,i)==0)
372                  {
373                      bytes_sent+=fprintf(out,"%s",&env[x][i+1]);
374                      break ;
375                  }
376              }
                 // 没有得到相应的值，输出(none)
377              if(!env[x])bytes_sent+=fprintf(out,"(none)");
378          }
379          else if(!strcmp(tag,"done"))
380              return 0 ;
381          else   // 非法指令。参看 echo 指令格式
382          {
383              char errstr[MAX_STRING_LEN];
384              sprintf(errstr,"unknown parameter %s to tag echo in %s",tag,file);
385              log_error_noclose(errstr);
386              bytes_sent+=fprintf(out,"%s",error);
387          }
388      }
389  }
390   /* 处理执行 cgi 脚本命令 */
391  int include_cgi(char*s,char*pargs,char*args,char**env,FILE*out)
392  {
393      char*argp,op,d[HUGE_STRING_LEN];
394      int allow,check_cgiopt ;
395      struct stat finfo ;
396
```

```
397         getparents(s);
398         if(s[0]=='/')
399         {
400             strcpy(d,s);
401             if(translate_name(d,out)!=SCRIPT_CGI)
402                 return-1 ;
403             check_cgiopt=0 ;
404         }
405         else
406         {
407             char dir[MAX_STRING_LEN];
408             getcwd(dir,MAX_STRING_LEN);
409             make_full_path(dir,s,d);
410             check_cgiopt=1 ;
411         }
412         if(stat(d,&finfo)==-1)
413             return-1 ;
414          /* 检查存取权限 */
415         evaluate_access(d,&finfo,M_GET,&allow,&op,out);
416         if((!allow)||(check_cgiopt&&(!(op&OPT_EXECCGI))))
417             return-1 ;
418          /* 执行脚本，详细操作参看第 8 章 */
419         if(cgi_stub("GET",d,pargs,args,env,&finfo,-1,out)==REDIRECT_URL)
420             bytes_sent+=fprintf(out,"<A HREF=\"%s\">%s</A>",location,location);
421         return 0 ;
422     }
423
424     static int ipid ;
425      /* 关闭处理命令子进程 */
426     void kill_include_child()
427     {
428         char errstr[MAX_STRING_LEN];
429         sprintf(errstr,"killing command process %d",ipid);
430         log_error_noclose(errstr);
431         kill(ipid,SIGKILL);
432         waitpid(ipid,NULL,0);
433     }
434
435     int include_cmd(char*s,char*pargs,char*args,char**env,FILE*out)
436     {
437         int p[2],x ;
438         FILE*f ;
439
440         if(pipe(p)==-1)
```

```
441             die(SERVER_ERROR,"httpd: could not create IPC pipe",out);
442         if((ipid=fork())==-1)
443             die(SERVER_ERROR,"httpd: could not fork new process",out);
444         if(!ipid)
445         {
446             char*argv0 ;
447
448             if(pargs[0]||args[0])
449             {
450                 if(!(env=new_env(env,4,&x)))
451                     return-1 ;
452                 if(pargs[0])
453                 {
454                     char p2[HUGE_STRING_LEN];
455
456                     escape_shell_cmd(pargs);
457                     env[x++]=make_env_str("PATH_INFO",pargs,out);
458                     strcpy(p2,pargs);
459                     if(translate_name(p2,out)!=BAD_URL)
460                         env[x++]=make_env_str("PATH_TRANSLATED",p2,out);
461                 }
462                 if(args[0])
463                 {
464                     env[x++]=make_env_str("QUERY_STRING",args,out);
465                     unescape_url(args);
466                     escape_shell_cmd(args);
467                     env[x++]=make_env_str("QUERY_STRING_UNESCAPED",
                                              args,out);
468                 }
469                 env[x]=NULL ;
470             }
471
472             close(p[0]);
473             if(p[1]!=STDOUT_FILENO)
474             {
475                 dup2(p[1],STDOUT_FILENO);
476                 close(p[1]);
477             }
478             error_log2stderr();
479             if(!(argv0=strrchr(SHELL_PATH,'/')))
480                 argv0=SHELL_PATH ;
481             if(execle(SHELL_PATH,argv0,"-c",s,(char*)0,env)==-1)
482             {
483                 fprintf(stderr,"httpd: exec of %s failed, errno is %d\n",
```

```
484                         SHELL_PATH,errno);
485                     exit(1);
486                 }
487         }
488         close(p[1]);
489         if(!(f=fdopen(p[0],"r")))
490         {
491             waitpid(ipid,NULL,0);
492             return-1 ;
493         }
494         send_fd(f,out,kill_include_child);
495         fclose(f);
496         waitpid(ipid,NULL,0);
497         return 0 ;
498     }
499      /*  处理执行命令(含 cmd 和 cgi 两个指令) */
500     int handle_exec(FILE*in,FILE*out,char*file,char*path_args,
501                         char*args,char*error,char**env)
502     {
503         char tag[MAX_STRING_LEN],errstr[MAX_STRING_LEN];
504         char*tag_val ;
505
506         while(1)
507         {
508             if(!(tag_val=get_tag(in,tag,1)))
509                 return 1 ;
510             if(!strcmp(tag,"cmd"))
511             {
512                 if(include_cmd(tag_val,path_args,args,env,out)==-1)
513                 {
514                     sprintf(errstr,"invalid command exec %s in %s",tag_val,file);
515                     log_error_noclose(errstr);
516                     bytes_sent+=fprintf(out,"%s",error);
517                 }
518                 chdir_file(file);
519             }
520             else if(!strcmp(tag,"cgi"))
521             {
522                 if(include_cgi(tag_val,path_args,args,env,out)==-1)
523                 {
524                     sprintf(errstr,"invalid CGI ref %s in %s",tag_val,file);
525                     log_error_noclose(errstr);
526                     bytes_sent+=fprintf(out,"%s",error);
527                 }
```

```
528                 chdir_file(file);
529             }
530             else if(!strcmp(tag,"done"))
531                 return 0 ;
532             else
533             {
534                 char errstr[MAX_STRING_LEN];
535                 sprintf(errstr,"unknown parameter %s to tag echo in %s",tag,file);
536                 log_error_noclose(errstr);
537                 bytes_sent+=fprintf(out,"%s",error);
538             }
539         }
540
541     }
542
        /*
        * 处理 congif 指令，主要是对错误消息格式、时间格式以及文件大小格式的
        * 设置
        */
543     int handle_config(FILE*in,FILE*out,char*file,char*error,
544                             char*tf,int*sizefmt,char**env)
545     {
546         char tag[MAX_STRING_LEN];
547         char*tag_val ;
548
549         while(1)
550         {
551             if(!(tag_val=get_tag(in,tag,0)))
552                 return 1 ;
553             if(!strcmp(tag,"errmsg"))
554                 strcpy(error,tag_val);
555             else if(!strcmp(tag,"timefmt"))
556             {
557                 strcpy(tf,tag_val);
558                 replace_env_str(env,"DATE_LOCAL",ht_time(date,tf,0),out);
559                 replace_env_str(env,"DATE_GMT",ht_time(date,tf,1),out);
560                 replace_env_str(env,"LAST_MODIFIED",ht_time(lm,tf,0),out);
561             }
562             else if(!strcmp(tag,"sizefmt"))
563             {
564                 decodehtml(tag_val);
565                 if(!strcmp(tag_val,"bytes"))
566                     *sizefmt=SIZEFMT_BYTES ;
567                 else if(!strcmp(tag_val,"abbrev"))
```

```
568                         *sizefmt=SIZEFMT_KMG ;
569                 }
570                 else if(!strcmp(tag,"done"))
571                     return 0 ;
572                 else
573                 {
574                     char errstr[MAX_STRING_LEN];
575                     sprintf(errstr,"unknown parameter %s to tag config in %s",
576                         tag,file);
577                     log_error_noclose(errstr);
578                     bytes_sent+=fprintf(out,"%s",error);
579                 }
580             }
581         }
582          /*  找到文件并获取它的 finfo */
583         int find_file(FILE*out,char*file,char*directive,char*tag,
584                       char*tag_val,struct stat*finfo,char*error)
585         {
586             char errstr[MAX_STRING_LEN],dir[MAX_STRING_LEN],
                    to_send[MAX_STRING_LEN];
587
588             if(!strcmp(tag,"file"))
589             {
590                 getparents(tag_val);
591                 getcwd(dir,MAX_STRING_LEN);
592                 make_full_path(dir,tag_val,to_send);
593                 if(stat(to_send,finfo)==-1)
594                 {
595                     sprintf(errstr,
596                         "unable to get information about %s in parsed file %s",to_send,file);
597                     log_error_noclose(errstr);
598                     bytes_sent+=fprintf(out,"%s",error);
599                     return-1 ;
600                 }
601                 return 0 ;
602             }
603             else if(!strcmp(tag,"virtual"))
604             {
605                 if(translate_name(tag_val,out)!=STD_DOCUMENT)
606                 {
607                     bytes_sent+=fprintf(out,"%s",error);
608                     log_error_noclose(errstr);
609                 }
610                 else if(stat(tag_val,finfo)==-1)
```

```
611                 {
612                     sprintf(errstr,
613                         "unable to get information about %s in parsed file %s",to_send,file);
614                     log_error_noclose(errstr);
615                     bytes_sent+=fprintf(out,"%s",error);
616                     return-1 ;
617                 }
618                 return 0 ;
619             }
620             else
621             {
622                 sprintf(errstr,
623                     "unknown parameter %s to tag %s in %s",tag,directive,file);
624                 log_error_noclose(errstr);
625                 bytes_sent+=fprintf(out,"%s",error);
626                 return-1 ;
627             }
628     }
629      /* 处理 fsize 指令 */
630     int handle_fsize(FILE*in,FILE*out,char*file,char*error,int sizefmt,char**env)
631     {
632         char tag[MAX_STRING_LEN];
633         char*tag_val ;
634         struct stat finfo ;
635 
636         while(1)
637         {
638             if(!(tag_val=get_tag(in,tag,1)))
639                 return 1 ;
640             else if(!strcmp(tag,"done"))
641                 return 0 ;
642             else if(!find_file(out,file,"fsize",tag,tag_val,&finfo,error))
643             {
644                 if(sizefmt==SIZEFMT_KMG)
645                 {
646                     send_size(finfo.st_size,out);
647                     bytes_sent+=5 ;
648                 }
649                 else
650                 {
651                     int l,x ;
652                     sprintf(tag,"%ld",finfo.st_size);
653                     l=strlen(tag);
654                     for(x=0;x<l;x++)
```

```
655                         {
656                             if(x&&(!((l-x)%3)))
657                             {
658                                 fputc(',',out);
659                                 ++bytes_sent ;
660                             }
661                             fputc(tag[x],out);
662                             ++bytes_sent ;
663                         }
664                     }
665                 }
666         }
667     }
668      /* 处理 flastmod 指令 */
669     int handle_flastmod(FILE*in,FILE*out,char*file,char*error,char*tf,char**env)
670     {
671         char tag[MAX_STRING_LEN];
672         char*tag_val ;
673         struct stat finfo ;
674
675         while(1)
676         {
677             if(!(tag_val=get_tag(in,tag,1)))
678                 return 1 ;
679             else if(!strcmp(tag,"done"))
680                 return 0 ;
681             else if(!find_file(out,file,"flastmod",tag,tag_val,&finfo,error))
682                 bytes_sent+=fprintf(out,"%s",ht_time(finfo.st_mtime,tf,0));
683         }
684     }
685
686      /* 主流程函数，处理 SSI 各种指令 */
687     void send_parsed_content(char*file,FILE*f,FILE*fd,char*path_args,
688                                 char*args,char**env,int noexec)
689     {
690         char directive[MAX_STRING_LEN],error[MAX_STRING_LEN],c ;
691         char timefmt[MAX_STRING_LEN],errstr[MAX_STRING_LEN];
692         int ret,sizefmt ;
693
694         strcpy(error,DEFAULT_ERROR_MSG);
695         strcpy(timefmt,DEFAULT_TIME_FORMAT);
696         sizefmt=SIZEFMT_KMG ;
697
698         chdir_file(file);
```

```
699
700         while(1)
701         {
702             if(!find_string(f,STARTING_SEQUENCE,fd))
703             {
704                 if(get_directive(f,directive))
705                     return ;
706                 if(!strcmp(directive,"exec"))
707                 {
708                     if(noexec)
709                     {
710                         sprintf(errstr,
711                         "httpd: exec used but not allowed in %s",file);
712                         log_error_noclose(errstr);
713                         bytes_sent+=fprintf(fd,"%s",error);
714                         ret=find_string(f,ENDING_SEQUENCE,NULL);
715                     }
716                     else
717                         ret=handle_exec(f,fd,file,path_args,args,error,env);
718                 }
719                 else if(!strcmp(directive,"config"))
720                     ret=handle_config(f,fd,file,error,timefmt,&sizefmt,env);
721                 else if(!strcmp(directive,"include"))
722                     ret=handle_include(f,fd,file,env,error);
723                 else if(!strcmp(directive,"echo"))
724                     ret=handle_echo(f,fd,file,error,env);
725                 else if(!strcmp(directive,"fsize"))
726                     ret=handle_fsize(f,fd,file,error,sizefmt,env);
727                 else if(!strcmp(directive,"flastmod"))
728                     ret=handle_flastmod(f,fd,file,error,timefmt,env);
729                 else
730                 {
731                     sprintf(errstr,
732                         "httpd: unknown directive %s in parsed doc %s",directive,file);
733                     log_error_noclose(errstr);
734                     bytes_sent+=fprintf(fd,"%s",error);
735                     ret=find_string(f,ENDING_SEQUENCE,NULL);
736                 }
737                 if(ret)
738                 {
739                     sprintf(errstr,"httpd: premature EOF in parsed file %s",file);
740                     log_error_noclose(errstr);
741                     return ;
742                 }
```

```
743                 }
744                 else
745                         return ;
746         }
747     }
748      /* 被 send_file 函数调用，发送解析过的 SSI 指令*/
749     void send_parsed_file(char*file,FILE*fd,char*path_args,char*args,int noexec)
750     {
751         FILE*f ;
752         char**env ;
753
754         if(!(f=fopen(file,"r")))
755         {
756                 log_reason("file permissions deny server access",file);
757                 unmunge_name(file);
758                 die(FORBIDDEN,file,fd);
759         }
760         strcpy(content_type,"text/html");
761         if(!assbackwards)
762                 send_http_header(fd);
763         if(header_only)
764                 return ;
765         fflush(fd);
766         assbackwards=1 ;
767         alarm(timeout);
768
769         env=add_include_vars(in_headers_env,file,path_args,args,
                                        DEFAULT_TIME_FORMAT,fd);
770         env=add_common_vars(env,fd);
771
772         send_parsed_content(file,f,fd,path_args,args,env,noexec);
773     }
```

7.6 小结

SSI 作为早期动态输出内容的一项技术，目前已经很少使用了，尤其是诸如 php、jsp 脚本语言大行其道之后。不过作为 Apache 的功能之一，我们有必要了解它的工作方式。并且不是所有的页面都有必要使用脚本语言，如果仅仅显示简单的动态内容，使用 Apache 自身提供的 SSI 功能，也是一个不错的选择。

第 8 章 执行CGI脚本

8.1 概述

CGI 是 Apache 生成动态交互式页面的一种方式，CGI 的全称是 Common Gateway Interface，即公共网关接口。

Apache 的 cgi-bin 目录是存放 CGI 程序的位置。

CGI 可以用任何语言来编写，这些语言包括 C/C++、Fortran、PERL、UNIX Shell 等。

Apache 对 CGI 的处理是通过文件 http_script.c 完成的。

本书不会对 CGI 程序本身进深入说明，有兴趣的读者可以参考相关的书籍。

8.2 背景知识

8.2.1 初识CGI

我们从 Apache0.6.5 提供的一个实例程序入手，来对 CGI 程序有一个大概的了解。

进入到 cgi-bin 目录，您可以找到 date 文件，这是一个 UNIX Shell 脚本，类似于 Windows（DOS）中的批处理，通过文本编辑器如 vi 打开这个文件，可以看到文件内容如下：

```
#!/bin/sh
DATE=/bin/date
echo Content-type: text/plain
echo
if [ -x $DATE ];    then
    $DATE
else
    echo Cannot find date command on this system.
fi
```

下面我们对这个脚本进行简单介绍：

第一行是注释行，说明执行这个 Shell 的是/bin/sh。

第二行是定义变量，意思是 DATE 变量的值是/bin/date，这个在不同的系统里面可能会不一样，您可以通过 which 命令来查看 date 命令的位置：

```
[devel@RIZI cgi-bin]$ which date
/bin/date
[devel@RIZI cgi-bin]$
```

第三行出现了一个 echo 指令，意思是直接输出后面的内容到客户端浏览器。因为 CGI 的输出可能是一个 HTML 页面，也可能是一张图片或一段音乐，所以我们要通过 Content-type 提示浏览器后继输出内容的类型。

if 和 fi 是成对出现的，if/else 和其它语言中语法一致，then 在本例中说明如果 if 后的条件成立的话要执行下面的语句，而 fi 说明对应 if 的结束。

整个 if-fi 的意思是，如果给定的命令存在，就输出执行结果到客户端，否则输出提示信息“Cannot find date command on this system”到客户端。

我们可以通过浏览器来查看输出结果，在我的浏览器里面的输出结果如下图所示：

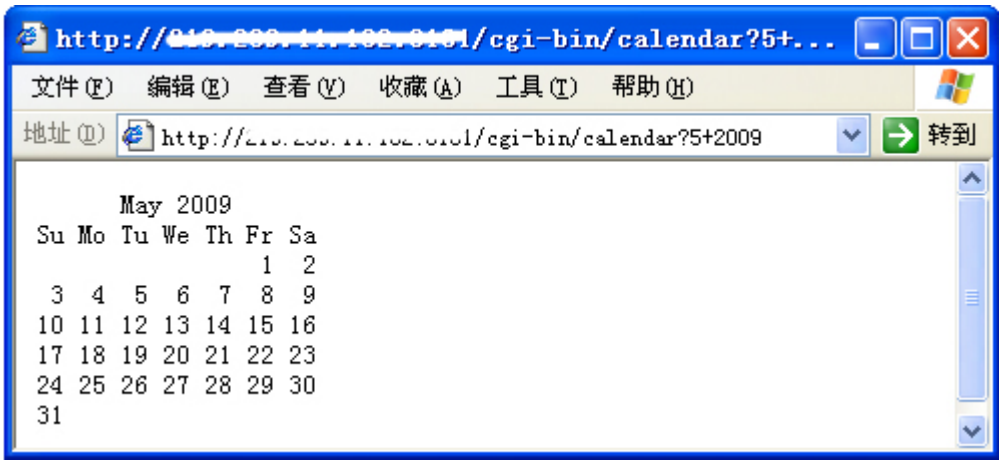
8.2.2 isindex标签

isindex 是 HTML 语言的一个标签，因为 Apache 处理 CGI 部分的源代码中涉及到了针对它的处理，并且很少有针对它的介绍，我们在这里做一个简单的说明，这也是理解源代码中 create_argv 函数的关键。

我们可以在 cgi-bin 目录中的 calendar 脚本中看到这个标签的使用。在您的浏览器里面调用这个脚本，可以看到输出如下（图片经过处理，剔除了提示内容）：

其中两个分割线及其之间的内容就是一个 isindex 标签产生的效果，对应的源代码就是<isindex>。该标签单独出现，已经不建议使用。

您可以在输入框里面输入一个月份加空格加一个年份，如“5 2009”，然后回车，可以看到 2009 年 5 月的日历输出：



我们可以从请求的 URL 里面看到，客户端请求的地址是/cgi-bin/calendar?5+2009。这个 URL 是一个经过编码处理，其中加号（+）即空格的编码字符。

Apache 针对这个 URL 的处理调用了 create_argv 函数，其中参数 av0=calendar，args=5+2009。

8.3 代码注释

```
1   #include "httpd.h"
2
3   int pid ;
4    /* 终止子进程执行 */
5   void kill_children()
6   {
7         char errstr[MAX_STRING_LEN];
8         sprintf(errstr,"killing CGI process %d",pid);
9         log_error_noclose(errstr);
10
11        kill(pid,SIGTERM);      // 先发送可捕获的终止信号
12        sleep(3);               // 给子进程结束一点时间
13        kill(pid,SIGKILL);      // 强制子进程终止
14        waitpid(pid,NULL,0);    // 等待子进程结束
15  }
16
     /*
     * 将 av0, args 组建成执行命令二维字符数组。
     * 如 av0 = 'calendar'; args='3+1993'，组建后的返回值为
     * char[][] = ['calendar','3','1993'];
     */
17  char**create_argv(char*av0,char*args,FILE*out)
18  {
19        register int x,n ;
20        char**av ;
21        char w[HUGE_STRING_LEN];
22        char l[HUGE_STRING_LEN];
23
24        for(x=0,n=2;args[x];x++)
25              if(args[x]=='+')++n ;
26
27        if(!(av=(char**)malloc((n+1)*sizeof(char*))))
28              die(NO_MEMORY,"create_argv",out);
29        av[0]=av0 ;
30        strcpy(l,args);
31        for(x=1;x<n;x++)          // 循环获得每个参数
32        {
33              getword(w,l,'+');     // w 里面保存当前参数名称
34              unescape_url(w);    // 解码参数里面的特殊字符，如:%5c=>\
35              escape_shell_cmd(w);    // 将 w 里面出现的特殊字符前加'\'，如'&'=>'\&'
36              if(!(av[x]=strdup(w)))    // 将处理后的参数赋值给 av[x]
37                    die(NO_MEMORY,"create_argv",out);
```

```
38          }
39          av[n]=NULL ;
40          return av ;
41      }
42       /* 根据给定的全路径名称，获取 cgi 所在路径的信息赋值给 finfo 变量。*/
43      void get_path_info(char*path,char*path_args,
44                                  struct stat*finfo)
45      {
46          char*cp ;
47          char*end=&path[strlen(path)];
48          char*last_cp=NULL ;
49          int rv ;
50
51          path_args[0]='\0' ;
52
53          for(cp=end;cp>path&&cp[-1]=='/';--cp)
54              continue ;
55          while(cp>path)
56          {
57              *cp='\0' ;              // 路径是否在这里结束
58              rv=stat(path,finfo);
59              if(cp!=end)*cp='/' ; // 恢复 path 的原样
60
61              if(!rv)                 // stat 调用成功
62              {
63                  if(S_ISDIR(finfo->st_mode)&&last_cp)
64                  {
65                      cp=last_cp ;
66                  }
67                  strcpy(path_args,cp);
68                  *cp='\0' ;
69                  return ;
70              }
71              else
72              {
73                  last_cp=cp ;
74
75                  while(--cp>path&&*cp!='/')
76                      continue ;
77              }
78          }
79      }
80
81      #define MAX_COMMON_VARS 13
```

```
82  #define MAX_CGI_VARS (MAX_COMMON_VARS+13)
83   /* 设置 CGI 程序执行可能需要的环境变量 */
84  char**add_cgi_vars(char**env,
85                     char*method,char*path,char*path_args,char*args,
86                     int*content,
87                     FILE*out)
88  {
89      int x ;
90      char t[HUGE_STRING_LEN],t2[HUGE_STRING_LEN];
91
92      if(!(env=new_env(env,MAX_CGI_VARS,&x)))
93          die(NO_MEMORY,"add_cgi_vars",out);
94
95      env[x++]=make_env_str("GATEWAY_INTERFACE","CGI/1.1",out);
96
97      env[x++]=make_env_str("SERVER_PROTOCOL",
98          (assbackwards?"HTTP/0.9":"HTTP/1.0"),out);
99      env[x++]=make_env_str("DOCUMENT_ROOT",document_root,out);
100     env[x++]=make_env_str("SERVER_ADMIN",server_admin,out);
101     env[x++]=make_env_str("REQUEST_METHOD",method,out);
102
103     strcpy(t,path);
104     unmunge_name(t);   // 将路径转换成相对于 document_root 的相对路径
105     env[x++]=make_env_str("SCRIPT_NAME",t,out);
106     if(path_args[0])
107     {
108         env[x++]=make_env_str("PATH_INFO",path_args,out);
109         strcpy(t2,path_args);
              /* t2 是合法 URL 则找出 t2 的绝对路径替换 t2 内容 */
110         if(translate_name(t2,out)!=BAD_URL)
111             env[x++]=make_env_str("PATH_TRANSLATED",t2,out);
112     }
113     env[x++]=make_env_str("QUERY_STRING",args,out);
114      /* 如果我们重定向了，设置两个相关变量 */
115     if(original_url[0])
116     {
117         env[x++]=make_env_str("REDIRECT_QUERY_STRING",
                                  original_args,out);
118         env[x++]=make_env_str("REDIRECT_URL",original_url+1,out);
119     }
120
121     if(content)
122     {
123         *content=0 ;
```

```
124             if((!strcmp(method,"POST"))||(!strcmp(method,"PUT")))
125             {
126                 *content=1 ;
127                 sprintf(t,"%d",content_length);
128                 env[x++]=make_env_str("CONTENT_TYPE",content_type,out);
129                 env[x++]=make_env_str("CONTENT_LENGTH",t,out);
130             }
131         }
132         env[x]=NULL ;
133         return env ;
134     }
135      /* 设置常用环境变量的值 */
136     char**add_common_vars(char**env,FILE*out)
137     {
138         char t[HUGE_STRING_LEN],*env_path ;
139         int x ;
140 
141         if(!(env=new_env(env,MAX_COMMON_VARS,&x)))
142             die(NO_MEMORY,"add_common_vars",out);
143 
144         if(!(env_path=getenv("PATH")))
145             env_path=DEFAULT_PATH ;
146         env[x++]=make_env_str("PATH",env_path,out);
147         env[x++]=make_env_str("SERVER_SOFTWARE",SERVER_VERSION,out);
148         env[x++]=make_env_str("SERVER_NAME",server_hostname,out);
149         sprintf(t,"%d",port);
150         env[x++]=make_env_str("SERVER_PORT",t,out);
151         env[x++]=make_env_str("REMOTE_HOST",remote_name,out);
152         env[x++]=make_env_str("REMOTE_ADDR",remote_ip,out);
153         env[x++]=make_env_str("DOCUMENT_ROOT",document_root,out);
154         env[x++]=make_env_str("SERVER_ADMIN",server_admin,out);
155         if(user[0])
156             env[x++]=make_env_str("REMOTE_USER",user,out);
157         if(auth_type)
158             env[x++]=make_env_str("AUTH_TYPE",auth_type,out);
159         if(original_url[0])
160         {
161             env[x++]=make_env_str("REDIRECT_QUERY_STRING",
                                      original_args,out);
162             env[x++]=make_env_str("REDIRECT_URL",original_url+1,out);
163         }
164 
165         env[x]=NULL ;
166         return env ;
```

```
167     }
168      /* 真正执行一个 CGI 脚本，并把脚本的执行结果输出给客户端 */
169     int cgi_stub(char*method,char*path,char*path_args,char*args,
170                          char**env,struct stat*finfo,int in,FILE*out)
171     {
172           int p[2];
173           int content,nph ;
174           char*argv0 ;
175           FILE*psin ;
176           register int x ;
177            /* 如果指定脚本不能被 apache 执行，给客户端提示 403 错误并退出*/
178           if(!can_exec(finfo))
179           {
180                 unmunge_name(path);
181                 die(FORBIDDEN,path,out);
182           }
183            // 获取要执行的 cgi 程序
184           if((argv0=strrchr(path,'/'))!=NULL)
185                 argv0++;
186           else argv0=path ;
187            // 修改工作目录
188           chdir_file(path);
189
190           if(pipe(p)<0)
191                 die(SERVER_ERROR,"httpd: could not create IPC pipe",out);
192           if((pid=fork())<0)
193                 die(SERVER_ERROR,"httpd: could not fork new process",out);
194
195           nph=(strncmp(argv0,"nph-",4)?0:1); // 脚本名字是否以 nph-开头
196           if(!pid)                // 子进程
197           {
198                 close(p[0]);    // 关闭管道读取端
199                 env=add_cgi_vars(env,method,path,path_args,args,&content,out);
200                 if(content)
201                       if(in!=STDIN_FILENO)
202                       {
203                             dup2(in,STDIN_FILENO);
204                             close(in);
205                       }
206                       if(nph)    // 脚本名字以 nph-开头
207                       {
208                             if(fileno(out)!=STDOUT_FILENO)
209                             {
210                                   dup2(fileno(out),STDOUT_FILENO);
```

```
211                         fclose(out);
212                     }
213                 }
214                 else                    // 将子进程的标准输出到写管道
215                 {
216                     if(p[1]!=STDOUT_FILENO)
217                     {
218                         dup2(p[1],STDOUT_FILENO);
219                         close(p[1]);
220                     }
221                 }
222                 error_log2stderr();  // 错误输出到错误日志
223                  // 表单方式提交的数据
224                 if((!args[0])||(ind(args,'=')>=0))
225                 {
226                     if(execle(path,argv0,(char*)0,env)==-1)
227                     {
228                         fprintf(stderr,"httpd: exec of %s failed, errno is %d\n",
229                             path,errno);
230                         exit(1);
231                     }
232                 }
233                 else // isindex 方式提交的数据
234                 {
235                     if(execve(path,create_argv(argv0,args,out),env)==-1)
236                     {
237                         fprintf(stderr,"httpd: exec of %s failed, errno is %d\n",
238                             path,errno);
239                         exit(1);
240                     }
241                 }
242         }
243         else
244         {
245             close(p[1]);    //父进程关闭写管道
246         }
247
248         if(!nph)                // 脚本名字不是以 nph-开头。
249         {
250             if(!(psin=fdopen(p[0],"r")))   // 打开管道读取脚本输出
251                 die(SERVER_ERROR,"could not read from script",out);
252             scan_script_header(psin,out,path); // 根据脚本输出设置头部变量的值
253
254             if(location[0]=='/') // 如果是重定向
```

```
255             {
256                 char t[HUGE_STRING_LEN],a[HUGE_STRING_LEN],*argp ;
257
258                 a[0]='\0' ;
259                 fclose(psin);
260                 waitpid(pid,NULL,0);
261                 strcpy(t,location);   // t=重定向的目标地址
262                 if(argp=strchr(t,'?'))
263                 {
264                     *argp++='\0' ;
265                     strcpy(a,argp); // a=参数
266                 }
267                 status=REDIRECT ;
268                 log_transaction(0);
269
270                 original_url[0]=' ' ;
271                  // 在重定向之前保存已有的信息
272                 prepare_for_redirect(out,REDIRECT,t);
                     // 使用重定向后的地址、参数调用处理 get 请求的函数
273                 process_get(in,out,"GET",t,a);
274                 return 0 ;
275             }
276             content_length=-1 ;
277             if(!assbackwards)               // HTTP 1.0
278                 send_http_header(out);  // 发送 http 头
279             if(!header_only)                // 需要发送 body
280             {
281                 if(!send_fd(psin,out,NULL)&&location[0])
282                 {
283                     title_html(out,"Document moved");
284                     fprintf(out,
285                         "This document has moved
                             <A HREF=\"%s\">here</A>.<P>%c",
286                         location,LF);
287                 }
288             }
289             else
290                 kill_children();
291             fclose(psin);          // 关闭读管道
292         }
293         else bytes_sent=-1 ;
294         waitpid(pid,NULL,0);    // 等待子进程完成
295         return 0 ;
296     }
```

```
297     /* 处理 cgi 脚本主流程 */
298   void exec_cgi_script(char*method,char*path,char*args,int in,FILE*out)
299   {
300         struct stat finfo ;
301         char path_args[HUGE_STRING_LEN];
302         char**env ;
303         int m=0 ;
304
305         get_path_info(path,path_args,&finfo);
306
307         if(S_ISDIR(finfo.st_mode))
308         {
309               unmunge_name(path);
310               die(NOT_FOUND,path,out);
311         }
312
313         if((!strcmp(method,"GET"))||(!strcmp(method,"HEAD")))m=M_GET ;
314         else if(!strcmp(method,"POST"))m=M_POST ;
315         else if(!strcmp(method,"PUT"))m=M_PUT ;
316         else if(!strcmp(method,"DELETE"))m=M_DELETE ;
317          // 检查存取权限
318         evaluate_access(path,&finfo,m,&allow,&allow_options,out);
          // 不允许，发送 403 错误
319         if(!allow)
320         {
321               log_reason("client denied by server configuration",path);
322               unmunge_name(path);
323               die(FORBIDDEN,path,out);
324         }
325         if(!(env=add_common_vars(in_headers_env,out)))
326               die(NO_MEMORY,"exec_cgi_script",out);
327
328         bytes_sent=0 ;
329          // 真正去执行 cgi 脚本
330         if(cgi_stub(method,path,path_args,args,env,&finfo,in,out)==REDIRECT_URL)
331               die(REDIRECT,location,out);
          // 记录日志
332         log_transaction(1);
333   }
```

8.4 小结

本章描述了 Apache 处理 CGI 脚本部分的源代码。为了更好理解这部分的源代码，作者对 CGI 程序和 HTML 的 isindex 标签进行了简单介绍。

第 9 章 自动索引目录

9.1 概述

Apache 除了可以把客户端请求的页面内容发送给客户端之外，还可以在没有页面的时候根据配置自己创建页面。

小王同学是做音乐的，有一些自己的原创音乐，会不定时上传一些音乐到网上供大家下载、试听、交流，但是小王同学不懂得网页制作，让我给他做一个个人音乐的展示、下载站点。要求在我不参与的情况下更新页面显示。

根据他的要求，我给他配置了一个站点出来，仅仅找了几个图标，修改了 Apache 的配置文件，就达到了他的要求。

配置文件及相关资源可以通过http://www.oldapache.org/example/chapter9/wang.rar下载。下载后解压缩，可以看到有三个目录：icons、conf和htdocs。您需要把原有的htdocs目录和conf目录下的所有文件删除，使用下载的两个目录内容替换原有的同名目录；把icons目录下的 3 个图片复制到您原来的icons目录下即可。

启动您的Apache，打开浏览器，在地址栏输入http://www.yourdomain.com/，如果没有意外，页面显示效果类似于下图：

如果您打开 htdocs 目录，可以看到，在这个目录里面并没有类似于 index.html 这样的“主页”文件，直接在这个里面增加新的文件，会自动显示在“主页”里面。这就是 Apache 的自动索引目录功能产生的效果。

学术的定义：自动索引目录提供了一种在目录下没有默认主文件（如 index.html）的时

候显示目录下面所有非隐藏文件及其属性的功能。

那么我们需要怎么配置这个功能呢？

9.2 配置自动索引目录

Apache0.6.5 提供了 IndexIgnore、IndexOptions、AddIcon、AddIconByType、AddIconByEncoding、AddAlt、AddAltByType、AddAltByEncoding、DefaultIcon、ReadmeName、HeaderName 和 AddDescription 共计 12 个指令来配置是否使用以及如何使用自动索引目录功能。

下面我们详细说明这 11 个指令的格式以及所达到的功效

1. IndexIgnore 指令

语法：IndexIgnore *file* [*file*] …

功能：当自动索引目录的时候隐藏列表中指定的文件

示例：IndexIgnore .html

上面这个示例指令表示，在自动索引目录下内容的时候，不要显示该文件夹下的 HTML 文件。

2. IndexOptions 指令

语法：IndexOptions [+|-]*option* [[+|-]*option*] ...

功能：自动索引目录时的各种配置选项

示例：IndexOptions SuppressSize

上面这个示例表示，在自动索引目录中显示文件大小列。

Apache0.6.5 支持 7 个可选的值，分别为：

FancyIndexing：对每种类型的文件前加上一个小图标以示区别。

IconsAreLinks：图标成为一个链接。默认情况下，目录下的文件名作为一个链接可供人们点击进行下载。但是人们可能对更直观图标有爱好。这个选项开启后，人们可以通过点击文件名前的图标来下载相应的文件，开启这个选项需要同时开启 FancyIndexing。

ScanHTMLTitles:如果您的列表目录下有 html 文件,开启这个选项后,Apache 会从这些 html 文件里面找到它们的 title,并把 title 的内容作为文件的介绍来显示。

SuppressLastModified：如果打开这个选项，文件的最后修改日期将作为一个列显示。

SuppressSize：如果打开这个选项，文件的大小将作为一个列显示

SuppressDescription：如果打开这个选项，文件的描述将作为一个列显示，之后可以通过 AddDescription 指令指定特定的文件（或文件类型）的描述。

None：仅显示一个文件名及其链接

3. AddDescription 指令

语法：AddDescription *string file* [*file*] ...

功能：对 file 类型（名字）的文件设置描述内容为 string

示例：AddDescription "Zip file" *.zip

上面的示例说明，对所有以.zip 作为扩展名的文件，它的描述属性为“Zip file”。

4. AddIcon 指令

语法：AddIcon *icon name* [*name*] ...

功能：对 name 指定的文件类型使用 icon 图标
示例：AddIcon /icons/dir.gif ^^DIRECTORY^^
上面的示例说明，如果列表中的文件是一个目录，则使用/icons/dir.gif 作为它的图标显示在文件（目录也是一种文件）名前面。

5. AddIconByType 指令

语法：AddIconByType *icon MIME-type* [*MIME-type*] ...
功能：用来指定与图标相关联的文件，将应用于那些没有使用 AddType 指定进行关联的文件。和名字含义一样，这条指令给特定 MINE 类型的文件指定显示图标。
示例：AddIconByType /icons/image3.gif image/*
上面的示例说明，对于 MIME 类型是 image/*的文件，它的图标将使用/icons/image3.gif。

6. AddIconByEncoding 指令

语法：AddIconByEncoding *icon MIME-encoding* [*MIME-encoding*] ...
功能：用一种编码类型与指定的图标关联。
示例：AddIconByEncoding (compressed, /icons/compressed.gif) x-compress
上例说明，用 x-compress 编码的文件，它前面的图标将会是/icons/compressed.gif。如果这个图标的 alt 属性将被设置成 compressed

7. DefaultIcon 指令

语法：DefaultIcon *url-path*
功能：自动索引的目录将在没有匹配任何 AddIcon 指令的图像位置显示 DefaultIcon。
示例：DefaultIcon /icon/unknown.gif
上例说明，如果有的文件类型没能通过 AddIcon 类指令匹配到相对应的 icon 的时候，显示 icon 的位置将会使用/icon/unknown.gif 文件。

8. AddAlt 指令

语法：AddAlt *string file* [*file*]
功能：将会对 file 类型的文件图标增加 alt 属性，属性值为 string
示例：AddAlt Compressed *.zip
上例说明，如果一个文件的扩展名是.zip，那么它的图标的 alt 属性值将会是 Compressed。

9. AddAltByType 指令

语法：AddAltByType *string MIME-type* [*MIME-type*] ...
功能：将会对特定 MIME 类型的文件图标的 alt 属性指定 string 值。
示例：AddAltByType ’plain text’ text/plain
上例说明，如果文件的 MIME 类型是 text/plain，那么它前面显示的图标 alt 属性的值将会是 plain text。

10. AddAltByEncoding 指令

语法：AddAltByEncoding *string MIME-encoding* [*MIME-encoding*] ...
功能：指定特定编码类型的文件前显示图标的 alt 属性值
示例：AddAltByEncoding gzip x-gzip
上例说明，对应 x-gzip 编码的文件，前面显示图标的 alt 属性值将会是 gzip。

11. ReadmeName 指令

语法：ReadmeName *filename*
功能：索引底部嵌入文件的文件名
示例：ReadmeName /FOOTER.html

上例说明，在索引目录的下面，将会显示/FOOTER.html 文件的内容

12. HeaderName 指令

语法：HeaderName *filename*

功能：索引顶部嵌入文件的文件名

示例：HeaderName /HEADER.html

上例说明，在索引目录的上面，将会显示/FOOTER.html 文件的内容

9.1 中提到的相关配置，您可以通过查看 srm.conf 文件获取。

配置有了，Apache 是怎么使这些配置生效的呢？

9.3 配置数据的获取

Apache 使用 http_dir.c 来实现自动索引目录的功能。在这个文件里面定义了两个重要结构：ent 和 item。其中 ent 是显示的每个文件（文件夹）的属性，元素如下：

```
struct ent {
    char *name;   // 文件（目录）名，如果是上级目录，显示"Parent Directory"
    char *icon;   // 文件类型相应的图标
    char *alt;    // 图标 alt 属性的值
    char *desc;   // 对文件的描述
    size_t size;  // 文件大小，如果是目录，显示'-'
    time_t lm;    // 文件最后修改时间
    struct ent *next;
};
```

从 9.2 节我们知道，某些条目是否显示可以配置的。可以对照 9.1 节中的效果图来认识这个结构。

另外一个重要的结构是 item，内容如下：

```
struct item {
    int type;
    char *apply_to;
    char *apply_path;
    char *data;
    struct item *next;
};
```

由于这个结构的每个部分在不同的实例链表中含义不同，我们无法整体定义每个属性的含义，具体到每个实例中每个属性的含义，参看下面调试信息的输出内容。

Apache 定义了 item 的几个链表：

- icon_list :由配置文件中 AddIcon 系列指令定义的内容列表
- alt_list:由配置文件中 AddAlt 系列指令定义的内容列表
- desc_list:由配置文件中 AddDescription 指令定义的内容列表(也有可能是 HTML 文件的 title)
- ign_list:由配置文件中 IndexIgnore 系列指令定义的内容列表
- hdr_list:header 文件列表
- rdme_list:readme 文件列表
- opts_list:由配置文件中 IndexOptions 指令定义的内容列表

这些链表在读取配置文件后会相应赋值，它们的值我们可以通过自己设计一个调试函数来显示。我设计的自定义函数的输出类似于：

1.icon_list:
icon_list[0]->type = 0 icon_list[0]->apply_to = *^^BLANKICON^^
icon_list[0]->apply_path = /* icon_list[0]->data = /icons/blank.gif
icon_list[1]->type = 0 icon_list[1]->apply_to = *^^DIRECTORY^^
icon_list[1]->apply_path = /* icon_list[1]->data = /icons/forder.gif
…… ……
icon_list[16]->type = 1 icon_list[16]->apply_to = text/*
icon_list[16]->apply_path = /* icon_list[16]->data = /icons/text.gif
2.alt_list:
alt_list[0]->type = 1 alt_list[0]->apply_to = audio/*
alt_list[0]->apply_path = /* alt_list[0]->data = SND
…… ……
alt_list[2]->type = 1 alt_list[2]->apply_to = text/*
alt_list[2]->apply_path = /* alt_list[2]->data = TXT

3.desc_list:
desc_list[0]->type = 0 desc_list[0]->apply_to = *.zip
desc_list[0]->apply_path = /* desc_list[0]->data = Zip file

4.ign_list:
ign_list[0]->type = 0 ign_list[0]->apply_to = * /README*
ign_list[0]->apply_path = /* ign_list[0]->data = (null)
…… ……
ign_list[4]->type = 0 ign_list[4]->apply_to = * /.??*
ign_list[4]->apply_path = /* ign_list[4]->data = (null)

5.hdr_list:
hdr_list[0]->type = 0 hdr_list[0]->apply_to = (null)
hdr_list[0]->apply_path = /* hdr_list[0]->data = /HEADER.html
6.rdme_list:
rdme_list[0]->type = 0 rdme_list[0]->apply_to = (null)
rdme_list[0]->apply_path = /* rdme_list[0]->data = /FOOTER.html

7.opts_list:
opts_list[0]->type = 61 opts_list[0]->apply_to = (null)
opts_list[0]->apply_path = /* opts_list[0]->data = (null)
…… ……
opts_list[2]->type = 0 opts_list[2]->apply_to = (null)
opts_list[2]->apply_path = /* opts_list[2]->data = (null)

9.4 代码注释

```
1   #include "httpd.h"
2    /* 目录下每个文件（文件夹）属性*/
3   struct ent
4   {
5         char*name ;   // 文件（目录）名，如果是上级目录，显示"Parent Directory"
6         char*icon ;   // 文件类型相应的图标
7         char*alt ;    // 图标 alt 属性的值
8         char*desc ;   // 对文件的描述
9         size_t size ;   // 文件大小
10        time_t lm ;   // 文件最后一次修改的时间
11        struct ent*next ;// 下一条记录
12  }
13  ;
14   /* 配置信息 */
15  struct item
16  {
17        int type ;    // 按什么类型区分 BY_PATH、BY_TYPE、BY_ENCODING
18        char*apply_to ;
19        char*apply_path ;
20        char*data ;
21        struct item*next ;
22  }
23  ;
24   /* 每个实例链表的含义参看 9.3 节描述，在此不再赘述 */
25  static struct item*icon_list,*alt_list,*desc_list,*ign_list ;
26  static struct item*hdr_list,*rdme_list,*opts_list ;
27
28  static int dir_opts ;
29   /* 初始化列表为空 */
30  void init_indexing()
31  {
32        icon_list=NULL ;
33        alt_list=NULL ;
34        desc_list=NULL ;
35        ign_list=NULL ;
36
```

```
37          hdr_list=NULL ;
38          rdme_list=NULL ;
39          opts_list=NULL ;
40      }
41       /* 清空指定的链表 */
42      void kill_item_list(struct item*p)
43      {
44          struct item*q ;
45
46          while(p)
47          {
48              if(p->apply_to)free(p->apply_to);
49              if(p->apply_path)free(p->apply_path);
50              if(p->data)free(p->data);
51              q=p ;
52              p=p->next ;
53              free(q);
54          }
55      }
56       /* 清空所有的链表 */
57      void kill_indexing()
58      {
59          kill_item_list(icon_list);
60          kill_item_list(alt_list);
61          kill_item_list(desc_list);
62          kill_item_list(ign_list);
63
64          kill_item_list(hdr_list);
65          kill_item_list(rdme_list);
66          kill_item_list(opts_list);
67      }
68       /* 根据给定的参数创建一个新的条目，返回新条目地址 */
69      struct item*new_item(int type,char*to,char*path,char*data,FILE*out)
70      {
71          struct item*p ;
72
73          if(!(p=(struct item*)malloc(sizeof(struct item))))
74              die(NO_MEMORY,"new_item",out);
75
76          p->type=type ;
77          if(data)
78          {
79              if(!(p->data=strdup(data)))
80                  die(NO_MEMORY,"new_item",out);
```

```
81          }
82          else
83              p->data=NULL ;
84
85          if(to)
86          {
87              if(!(p->apply_to=(char*)malloc(strlen(to)+2)))
88                  die(NO_MEMORY,"new_item",out);
89              if((type==BY_PATH)&&(!is_matchexp(to)))
90              {
91                  p->apply_to[0]='*' ;
92                  strcpy(&p->apply_to[1],to);
93              }
94              else
95                  strcpy(p->apply_to,to);
96          }
97          else
98              p->apply_to=NULL ;
99
100         if(!(p->apply_path=(char*)malloc(strlen(path)+2)))
101             die(NO_MEMORY,"new_item",out);
102         sprintf(p->apply_path,"%s*",path);
103
104         return p ;
105     }
106      /* 增加一条新的 alt 条目 */
107     void add_alt(int type,char*alt,char*to,char*path,FILE*out)
108     {
109         struct item*p ;
110
111         if(type==BY_PATH)
112         {
113             if(!strcmp(to,"**DIRECTORY**"))
114                 strcpy(to,"^^DIRECTORY^^");
115         }
116         p=new_item(type,to,path,alt,out);
117         p->next=alt_list ;
118         alt_list=p ;
119     }
120      /* 增加一条新的 icon 条目 */
121     void add_icon(int type,char*icon,char*to,char*path,FILE*out)
122     {
123         struct item*p ;
124         char iconbak[MAX_STRING_LEN];
```

```
125
126         strcpy(iconbak,icon);
127         if(icon[0]=='(') // 类似于 AddIconByType (IMG,/icons/image2.gif) image/*
128         {
129             char alt[MAX_STRING_LEN];
130             getword(alt,iconbak,',');
131             add_alt(type,&alt[1],to,path,out);
132             iconbak[strlen(iconbak)-1]='\0' ;
133         }
134         if(type==BY_PATH)
135         {
136             if(!strcmp(to,"**DIRECTORY**"))
137                 strcpy(to,"^^DIRECTORY^^");
138         }
139         p=new_item(type,to,path,iconbak,out);
140         p->next=icon_list ;
141         icon_list=p ;
142     }
143      /* 增加一条新的 desc 条目 */
144     void add_desc(int type,char*desc,char*to,char*path,FILE*out)
145     {
146         struct item*p ;
147
148         p=new_item(type,to,path,desc,out);
149         p->next=desc_list ;
150         desc_list=p ;
151     }
152      /* 增加一条新的 ignore 条目 */
153     void add_ignore(char*ext,char*path,FILE*out)
154     {
155         struct item*p ;
156
157         p=new_item(0,ext,path,NULL,out);
158         p->next=ign_list ;
159         ign_list=p ;
160     }
161      /* 增加一条新的 header 条目 */
162     void add_header(char*name,char*path,FILE*out)
163     {
164         struct item*p ;
165
166         p=new_item(0,NULL,path,name,out);
167         p->next=hdr_list ;
168         hdr_list=p ;
```

```
169     }
170      /* 增加一条新的 readme 条目 */
171     void add_readme(char*name,char*path,FILE*out)
172     {
173         struct item*p ;
174
175         p=new_item(0,NULL,path,name,out);
176         p->next=rdme_list ;
177         rdme_list=p ;
178     }
179      /* 增加一条新的 opts 条目 */
180     void add_opts_int(int opts,char*path,FILE*out)
181     {
182         struct item*p ;
183
184         p=new_item(0,NULL,path,NULL,out);
185         p->type=opts ;
186         p->next=opts_list ;
187         opts_list=p ;
188     }
189      /* 增加 opts 条目 */
190     void add_opts(char*optstr,char*path,FILE*out)
191     {
192         char w[MAX_STRING_LEN];
193         int opts=0 ;
194         while(optstr[0])
195         {
196             cfg_getword(w,optstr);
197             if(!strcasecmp(w,"FancyIndexing"))
198                 opts|=FANCY_INDEXING ;
199             else if(!strcasecmp(w,"IconsAreLinks"))
200                 opts|=ICONS_ARE_LINKS ;
201             else if(!strcasecmp(w,"ScanHTMLTitles"))
202                 opts|=SCAN_HTML_TITLES ;
203             else if(!strcasecmp(w,"SuppressLastModified"))
204                 opts|=SUPPRESS_LAST_MOD ;
205             else if(!strcasecmp(w,"SuppressSize"))
206                 opts|=SUPPRESS_SIZE ;
207             else if(!strcasecmp(w,"SuppressDescription"))
208                 opts|=SUPPRESS_DESC ;
209             else if(!strcasecmp(w,"None"))
210                 opts=0 ;
211         }
212         add_opts_int(opts,path,out);
```

```
213     }
214
        /*
        * 根据给定的 path 在 list 里面找到 apply_path 和 path 相同的项，并返回该项
        *   data 域的值，如果没找到，返回 NULL
        */
215     char*find_item(struct item*list,char*path,int path_only)
216     {
217         struct item*p=list ;
218         char*t ;
219
220         while(p)
221         {     /* 特殊情况包括^^DIRECTORY^^ 和 ^^BLANKICON^^*/
222             if((path[0]=='^')||(!strcmp_match(path,p->apply_path)))
223             {
224                 if(!(p->apply_to))
225                     return p->data ;
226                 else if(p->type==BY_PATH)
227                 {
228                     if(!strcmp_match(path,p->apply_to))
229                         return p->data ;
230                 }
231                 else if(!path_only)
232                 {
233                     char pathbak[MAX_STRING_LEN];
234                     strcpy(pathbak,path);
235                     content_encoding[0]='\0' ;
236                     set_content_type(pathbak);
237                     if(!content_encoding[0])
238                     {
239                         if(p->type==BY_TYPE)
240                         {
241                             if(!strcmp_match(content_type,p->apply_to))
242                                 return p->data ;
243                         }
244                     }
245                     else
246                     {
247                         if(p->type==BY_ENCODING)
248                         {
249                             if(!strcmp_match(content_encoding,p->apply_to))
250                                 return p->data ;
251                         }
252                     }
```

```
253                     }
254                 }
255                 p=p->next ;
256         }
257         return NULL ;
258     }
259
260     #define find_icon(p,t) find_item(icon_list,p,t)
261     #define find_alt(p,t) find_item(alt_list,p,t)
262     #define find_desc(p) find_item(desc_list,p,0)
263     #define find_header(p) find_item(hdr_list,p,0)
264     #define find_readme(p) find_item(rdme_list,p,0)
265      /* 检查文件是否在 ignore 列表中 */
266     int ignore_entry(char*path)
267     {
268         struct item*p=ign_list ;
269
270         while(p)
271         {
272             if(!strcmp_match(path,p->apply_path))
273                 if(!strcmp_match(path,p->apply_to))
274                     return 1 ;
275                 p=p->next ;
276         }
277         return 0 ;
278     }
279
         /*
         * 从 opts 列表中找到 apply_path 属性和 path 值匹配的，返回它的 type 属性
         * 如果没找到返回 0
         */
280     int find_opts(char*path)
281     {
282         struct item*p=opts_list ;
283
284         while(p)
285         {
286             if(!strcmp_match(path,p->apply_path))
287                 return p->type ;
288             p=p->next ;
289         }
290         return 0 ;
291     }
292
```

```
 /*
 * 把 readme 文件内容输出到客户端，放在文件列表底部。
 * 找到并输出了，返回 1，否则返回 0
 */
293 int insert_readme(char*name,char*readme_fname,int rule,FILE*fd)
294 {
295     char fn[MAX_STRING_LEN];
296     FILE*r ;
297     struct stat finfo ;
298     int plaintext=0 ;
299
300     make_full_path(name,readme_fname,fn); // 找到 readme 文件的全路径
301     strcat(fn,".html");  // 在配置的 readme 文件后面添加.html 扩展名
302     if(stat(fn,&finfo)==-1) //可能是因为添加了.html 扩展名造成的文件找不到
303     {
304         fn[strlen(fn)-5]='\0' ;     // 去掉添加的.html 扩展名重新查找
305         if(stat(fn,&finfo)==-1)  // 还是没找到，返回 0
306             return 0 ;
307         plaintext=1 ;
308         if(rule)bytes_sent+=fprintf(fd,"<HR>%c",LF);
309         bytes_sent+=fprintf(fd,"<PRE>%c",LF);
310     }
311     else if(rule)bytes_sent+=fprintf(fd,"<HR>%c",LF);
312     if(!(r=fopen(fn,"r")))
313         return 0 ;
314     send_fd(r,fd,NULL);          // 将 readme 文件内容发送到客户端
315     fclose(r);
316     if(plaintext)
317         bytes_sent+=fprintf(fd,"</PRE>%c",LF);
318     return 1 ;
319 }
320  /*
   * 从文件里面找到 title 的值，在描述文件时，会把 title 做为文件的描述
   * 返回
   */
321 char*find_title(char*filename)
322 {
323     char titlebuf[MAX_STRING_LEN],*find="<TITLE>" ;
324     char filebak[MAX_STRING_LEN];
325     FILE*thefile ;
326     int x,y,n,p ;
327
328     content_encoding[0]='\0' ;
329     strcpy(filebak,filename);
```

```
330         set_content_type(filebak);
             // 文件类型是 text/html 并且有编码信息
331         if((!strcmp(content_type,"text/html"))&&(!content_encoding[0]))
332         {
333             if(!(thefile=fopen(filename,"r")))
334                 return NULL ;
335             n=fread(titlebuf,sizeof(char),MAX_STRING_LEN-1,thefile);
336             titlebuf[n]='\0' ;
337             for(x=0,p=0;titlebuf[x];x++)
338             {
339                 if(titlebuf[x]==find[p]) // 找到了起始的<TITLE>
340                 {
341                     if(!find[++p])
342                     {    // 找到下一个<字符，认为是</TITLE>
343                         if((p=ind(&titlebuf[++x],'<'))!=-1)
344                             titlebuf[x+p]='\0' ;
345                         for(y=x;titlebuf[y];y++)
346                             if((titlebuf[y]==CR)||(titlebuf[y]==LF))
347                                 titlebuf[y]=' ' ;
348                         return strdup(&titlebuf[x]);
349                     }
350                 }
351                 else p=0 ;
352             }
353             return NULL ;
354         }
355         content_encoding[0]='\0' ;
356         return NULL ;
357     }
358
          /*
          * 编码 html，将 fn 中出现字符'<','>','&'的地方编码成'&lt','&gt','&amp'
          * 实际上这个函数没有被使用
          */
359     void escape_html(char*fn)
360     {
361         register int x,y ;
362         char copy[MAX_STRING_LEN];
363
364         strcpy(copy,fn);
365         for(x=0,y=0;copy[y];x++,y++)
366         {
367             if(copy[y]=='<')
368             {
```

```
369                     strcpy(&fn[x],"&lt;");
370                     x+=3 ;
371                 }
372                 else if(copy[y]=='>')
373                 {
374                     strcpy(&fn[x],"&gt;");
375                     x+=3 ;
376                 }
377                 else if(copy[y]=='&')
378                 {
379                     strcpy(&fn[x],"&amp;");
380                     x+=4 ;
381                 }
382                 else
383                     fn[x]=copy[y];
384         }
385         fn[x]='\0' ;
386     }
387
        /*
        * 根据给定的路径、文件(子目录)名，创建并填充一个 ent 结构
        * 并返回结构的地址
        */
388     struct ent*make_dir_entry(char*path,char*name,FILE*out)
389     {
390         struct ent*p ;
391         struct stat finfo ;
392         char t[MAX_STRING_LEN],t2[MAX_STRING_LEN];
393
394         if((name[0]=='.')&&(!name[1]))
395             return(NULL);
396
397         make_full_path(path,name,t);// 拼起来整个目录
398
399         if(ignore_entry(t))             // 检查是否需要忽略
400             return(NULL);
401
402         if(!(p=(struct ent*)malloc(sizeof(struct ent))))
403             die(NO_MEMORY,"make_dir_entry",out);
404         if(!(p->name=(char*)malloc(strlen(name)+2)))
405             die(NO_MEMORY,"make_dir_entry",out);
406
407         if(dir_opts&FANCY_INDEXING)
408         {
```

```
409             if((!(dir_opts&FANCY_INDEXING))||stat(t,&finfo)==-1)
410             {
411                 strcpy(p->name,name);
412                 p->size=-1 ;
413                 p->icon=NULL ;
414                 p->alt=NULL ;
415                 p->desc=NULL ;
416                 p->lm=-1 ;
417             }
418             else
419             {
420                 p->lm=finfo.st_mtime ;
421                 if(S_ISDIR(finfo.st_mode))  // 如果 path+name 是个目录
422                 {
423                     if(!(p->icon=find_icon(t,1)))
424                         p->icon=find_icon("^^DIRECTORY^^",1);
425                     if(!(p->alt=find_alt(t,1)))
426                         p->alt="DIR" ;
427                     p->size=-1 ;
428                     strcpy_dir(p->name,name);
429                 }
430                 else // path+name 是文件
431                 {
432                     p->icon=find_icon(t,0);
433                     p->alt=find_alt(t,0);
434                     p->size=finfo.st_size ;
435                     strcpy(p->name,name);
436                 }
437             }
438             if(p->desc=find_desc(t))// 如果有描述
439                 p->desc=strdup(p->desc);
                 // 没有描述且允许扫描 HTML 文件的 TITLE 信息
440             if((!p->desc)&&(dir_opts&SCAN_HTML_TITLES))
441                 p->desc=find_title(t);
442
443         }
444         else
445         {
446             p->icon=NULL ;
447             p->alt=NULL ;
448             p->desc=NULL ;
449             p->size=-1 ;
450             p->lm=-1 ;
451             strcpy(p->name,name);
```

```
452         }
453         return(p);
454     }
455      /* 显示文件大小列 */
456     void send_size(size_t size,FILE*fd)
457     {
458         char schar ;
459
460         if(size==-1)
461         {
462             fputs("       -",fd);
463             bytes_sent+=5 ;
464         }
465         else
466         {
467             if(!size)
468             {
469                 fputs("      0K",fd);
470                 bytes_sent+=5 ;
471             }
472             else if(size<1024)
473             {
474                 fputs("      1K",fd);
475                 bytes_sent+=5 ;
476             }
477             else if(size<1048576)
478                 bytes_sent+=fprintf(fd,"%4dK",size/1024);
479             else
480                 bytes_sent+=fprintf(fd,"%4dM",size/1048576);
481         }
482     }
483      /* 截断描述，描述子段最多允许 23 个字节 */
484     void terminate_description(char*desc)
485     {
486         int maxsize=23 ;
487         register int x ;
488
489         if(dir_opts&SUPPRESS_LAST_MOD)maxsize+=17 ;
490         if(dir_opts&SUPPRESS_SIZE)maxsize+=7 ;
491
492         for(x=0;desc[x]&&maxsize;x++)
493         {
494             if(desc[x]=='<')
495             {
```

```
496                     while(desc[x]!='>')
497                     {
498                         if(!desc[x])
499                         {
500                             maxsize=0 ;
501                             break ;
502                         }
503                         ++x ;
504                     }
505                 }
506                 else--maxsize ;
507             }
508             if(!maxsize)
509             {
510                 desc[x]='>' ;
511                 desc[x+1]='\0' ;
512             }
513
514     }
515
            /*
            * 输出路径下的文件内容到客户端
            * 将 ar 里面的内容编写成 html 源代码的方式输出到客户端
            */
516     void output_directories(struct ent**ar,int n,char*name,FILE*fd)
517     {
518         int x,pad ;
519         char anchor[HUGE_STRING_LEN],t[MAX_STRING_LEN],
                t2[MAX_STRING_LEN];
520         char*tp ;
521
522         if(name[0]=='\0')
523         {
524             name[0]='/' ;
525             name[1]='\0' ;
526         }
527         fflush(fd);
528
529         if(dir_opts&FANCY_INDEXING)
530         {
531             fputs("<PRE>",fd);
532             bytes_sent+=5 ;
533             if(tp=find_icon("^^BLANKICON^^",1))
534                 bytes_sent+=fprintf(fd,"<IMG SRC=\"%s\" ALT=\"        \"> ",tp);
```

```
535             bytes_sent+=fprintf(fd,"Name                    ");
                 //这里原文是 if(!(dir_opts & SUPPRESS_LAST_MOD))
536             if((dir_opts&SUPPRESS_LAST_MOD))
537                 bytes_sent+=fprintf(fd,"Last modified    ");
                 //这里原文是 if(!(dir_opts & SUPPRESS_SIZE))，正好意愿相反
538             if((dir_opts&SUPPRESS_SIZE))
539                 bytes_sent+=fprintf(fd,"Size    ");
                 // 这里原文是 if(!(dir_opts & SUPPRESS_DESC))
540             if((dir_opts&SUPPRESS_DESC))
541                 bytes_sent+=fprintf(fd,"Description");
542             bytes_sent+=fprintf(fd,"%c<HR>%c",LF,LF);
543         }
544         else
545         {
546             fputs("<UL>",fd);
547             bytes_sent+=4 ;
548         }
549
550         for(x=0;x<n;x++)
551         {       // 上级目录
552             if((!strcmp(ar[x]->name,"../"))||(!strcmp(ar[x]->name,"..")))
553             {
554                 make_full_path(name,"../",t); // 获取上级目录名称
555                 getparents(t);
556                 if(t[0]=='\0')
557                 {
558                     t[0]='/' ;
559                     t[1]='\0' ;
560                 }
561                 escape_uri(t);                  // 编码目录名称
562                 sprintf(anchor,"<A HREF=\"%s\">",t);
563                 strcpy(t2,"Parent Directory</A>");
564             }
565             else
566             {
567                 strcpy(t,ar[x]->name);
568                 strcpy(t2,t);
569                 if(strlen(t2)>21)
570                 {
571                     t2[21]='>' ;
572                     t2[22]='\0' ;
573                 }
574                 strcat(t2,"</A>");
575                 escape_uri(t);
```

```
576             sprintf(anchor,"<A NAME=\"%s\" HREF=\"%s\">",t,t);
577         }
578         escape_url(t);
579
580         if(dir_opts&FANCY_INDEXING)
581         {
582             if(dir_opts&ICONS_ARE_LINKS)
583                 bytes_sent+=fprintf(fd,"%s",anchor);
584             if((ar[x]->icon)||default_icon[0])   // 有对应的图标或默认图标
585             {
586                 bytes_sent+=fprintf(fd,"<IMG SRC=\"%s\" ALT=\"[%s]\">",
587                     ar[x]->icon?ar[x]->icon:default_icon,
588                     ar[x]->alt?ar[x]->alt:"      ");
589             }
590             if(dir_opts&ICONS_ARE_LINKS)
591             {
592                 fputs("</A>",fd);
593                 bytes_sent+=4 ;
594             }
595             bytes_sent+=fprintf(fd," %s",anchor);
596             bytes_sent+=fprintf(fd,"%-27.27s",t2);
                 // 这里原本是 if(!(dir_opts & SUPPRESS_LAST_MOD))
597             if((dir_opts&SUPPRESS_LAST_MOD))
598             {
599                 if(ar[x]->lm!=-1)
600                 {
601                     struct tm*ts=localtime(&ar[x]->lm);
602                     strftime(t,MAX_STRING_LEN,
                                 "%d-%b-%y %H:%M    ",ts);
603                     fputs(t,fd);
604                     bytes_sent+=strlen(t);
605                 }
606                 else
607                 {
608                     fputs("                          ",fd);
609                     bytes_sent+=17 ;
610                 }
611             }
                 // 这里原本是 if(!(dir_opts & SUPPRESS_SIZE))
612             if((dir_opts&SUPPRESS_SIZE))
613             {
614                 send_size(ar[x]->size,fd);
615                 fputs("    ",fd);
616                 bytes_sent+=2 ;
```

```
617                     }
                         // 这里原本是 if(!(dir_opts & SUPPRESS_DESC))
618                     if((dir_opts&SUPPRESS_DESC))
619                     {
620                         if(ar[x]->desc)
621                         {
622                             terminate_description(ar[x]->desc);
623                             bytes_sent+=fprintf(fd,"%s",ar[x]->desc);
624                         }
625                     }
626                 }
627                 else
628                     bytes_sent+=fprintf(fd,"<LI> %s %s",anchor,t2);
629                 fputc(LF,fd);
630                 ++bytes_sent ;
631             }
632             if(dir_opts&FANCY_INDEXING)
633             {
634                 fputs("</PRE>",fd);
635                 bytes_sent+=6 ;
636             }
637             else
638             {
639                 fputs("</UL>",fd);
640                 bytes_sent+=5 ;
641             }
642     }
643      /* 目录排序，根据目录名 */
644     int dsortf(struct ent**s1,struct ent**s2)
645     {
646         return(strcmp((*s1)->name,(*s2)->name));
647     }
648      /* 列表 name 路径下的文件 */
649     void index_directory(char*name,FILE*fd)
650     {
651         DIR*d ;
652         struct DIR_TYPE*dstruct ;
653         int num_ent=0,x ;
654         struct ent*head,*p,*q ;
655         struct ent**ar ;
656         char unmunged_name[MAX_STRING_LEN];
657         char*tmp ;
             // unmunged_name = name + (name[strlen(name)-1] == '/'?':'/')
658         strcpy_dir(unmunged_name,name);
```

```
                // 绝对路径转换成相对 DOCUMENT_ROOT 的相对路径
659             unmunge_name(unmunged_name);
660             // 如果打开失败，返回 403 错误代码给客户端
661             if(!(d=opendir(name)))
662                 die(FORBIDDEN,unmunged_name,fd);
663             // content_type 为 text/html
664             strcpy(content_type,"text/html");
665             status=200 ;            // STATUS_OK
666             if(!assbackwards)   // 是否需要发送头，0.9 版本的 http 协议不支持消息头
667                 send_http_header(fd);
668
669             if(header_only)return ;
670
671             bytes_sent=0 ;
672             dir_opts=find_opts(name);
673             /* 输出 HTML 头部 */
674             bytes_sent+=fprintf(fd,
675                 "<HEAD><TITLE>Index of %s</TITLE></HEAD><BODY>",
676                 unmunged_name);
677             fputc(LF,fd);
678             ++bytes_sent ;
679
680             if((!(tmp=find_header(name)))||(!(insert_readme(name,tmp,0,fd))))
681                 bytes_sent+=fprintf(fd,"<H1>Index of %s</H1>%c",
                                        unmunged_name,LF);
682
                /*
                 * 因为我们不知道有多少目录/文件入口(子目录/及包含的文件)，
                 * 把它们放到一个链表中，然后数组化处理，这样就可以用 qsort 排序了
                 */
683             head=NULL ;
684             while(dstruct=readdir(d))       // 查找目录下的文件/目录
685             {
686                 if(p=make_dir_entry(name,dstruct->d_name,fd))
687                 {
688                     p->next=head ;
689                     head=p ;
690                     num_ent++;
691                 }
692             }
693             if(!(ar=(struct ent**)malloc(num_ent*sizeof(struct ent*))))
694                 die(NO_MEMORY,"index_directory",fd);
695             p=head ;
696             x=0 ;
```

```
697         while(p)
698         {
699             ar[x++]=p ;
700             p=p->next ;
701         }
             // 排序 ar
702         qsort((void*)ar,num_ent,sizeof(struct ent*),
703             (int(*)(const void*,const void*))dsortf);
704         output_directories(ar,num_ent,unmunged_name,fd); // 输出目录内容
             // 输出目录内容结束后释放 ar 和 head
705         free(ar);
706         q=head ;
707         while(q)
708         {
709             p=q->next ;
710             free(q->name);
711             if(q->desc)
712                 free(q->desc);
713             free(q);
714             q=p ;
715         }
716         closedir(d);    // 关闭目录
717         if(dir_opts&FANCY_INDEXING)
718             if(tmp=find_readme(name))
719                 insert_readme(name,tmp,1,fd);
720             else
721             {
722                 fputs("</UL>",fd);
723                 bytes_sent+=5 ;
724             }
              // 输出 body 并记录日志
725             fputs("</BODY>",fd);
726             bytes_sent+=7 ;
727             log_transaction(1);
728     }
```

9.5 小结

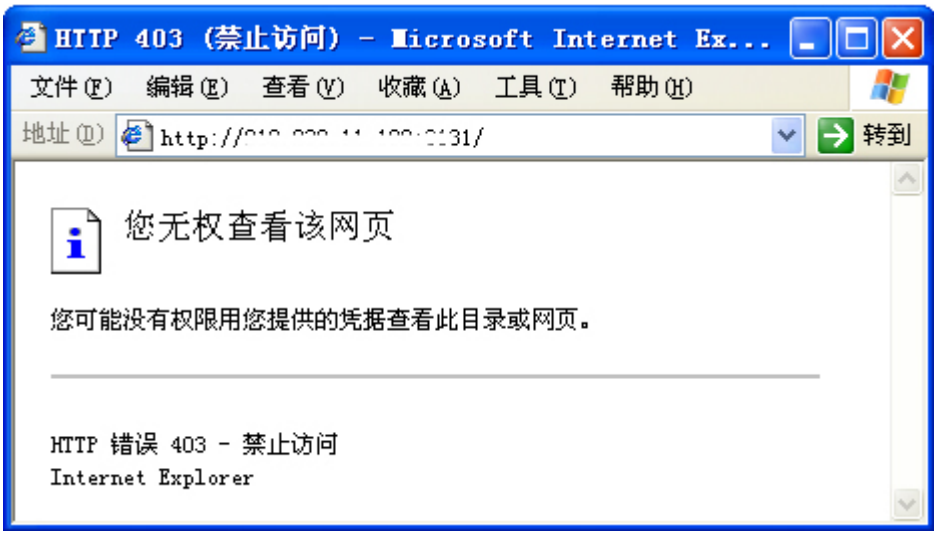
虽然现在的常用脚本语言可以完成我们在 9.1 中描述的任务，但是不得不加上数据库和后台管理系统的支持，如果您也是仅仅提供下载服务，并且对页面的美观没有特别的要求，还是可以使用 Apache 的自动索引目录功能而无需做相应的开发。

除去应用之外，Apache 中自动索引目录的实现方式也是很值得我们学习的。

第 10 章 存取控制

10.1 概述

您也许碰到过这样的情况，当您访问某个网站的时候，会收到 403 状态号，意思是禁止您查看请求的页面：



或者收到 401 状态号，表示对该页面的访问需要提供帐号、密码：

实现上述两个功能，就需要设置 Apache 的存取控制指令。

存取控制可以让网站管理员对某些资源的受众进行控制，比如给某些人看，不给某些人看，或者需要提供密码才能查看等等。

Apache 的存取控制是通过对配置文件 access.conf 的修改完成的。对存取控制的实际操作是通过源文件 http_access.c 和 http_auth.c 来实现的。

下面我们来配置 access.conf 文件来达到上面提到的两个功能。

10.2 存取控制设置

在本节中我们通过配置 Apache 的 access.conf 文件来实现 10.1 节中提到的两个功能。其中第一小节讲解怎么设置这两种状态，第二小节对 Apache 所有有关存取控制的设置指令进行详细说明。

10.2.1 限制访问和认证访问

限制访问

简单的做法是让 Apache 拒绝所有的客户端访问。打开配置文件 access.conf，找到<Limit GET>行，内容如下：

```
<Limit GET>
order allow,deny
allow from all
</Limit>
```

把其中的 allow from all 修改成 deny from all。保存文件，重启 Apache，通过浏览器访问您的站点首页，不出意外的话，您的浏览器应该会收到 403 状态号，即禁止访问。

禁止访问一般是禁止某些人的访问，在实际应用中，不会禁止所有人的访问，这样的站点是没有意义的，如果要禁止某些人的访问，可以参考 10.2.2 小节中的介绍。在这里我们仅仅重现 10.1 中实现的效果。

认证访问

首先还是修改 access.conf 文件中的<Limit GET>部分，修改后的内容类似于：

```
<Limit GET>
order allow,deny
allow from all
require valid-user
</Limit>
```

仅仅这样做是不够的。还需要创建认证文件和密码文件，认证文件中确定认证的方式、密码文件的位置、认证的提示语等内容；密码文件由 Apache 提供的工具创建，放在认证文件中指定的位置。

认证文件需要放在需要认证的目录下，一般以.htaccess 做为文件名（这个名字可以通过配置文件设置）。我们在这里对整个站点认证，所以把认证文件放在站点根目录。相关信息如下所示：

```
[devel@RIZI htdocs]$ pwd
/home/devel/apache_0.6.5/htdocs
[devel@RIZI htdocs]$ tail .htaccess
AuthName "Private Level 9"
AuthType Basic
AuthUserFile /home/devel/apache_0.6.5/htdocs/.htpasswd
[devel@RIZI htdocs]$
```

.htpasswd 文件即认证密码文件，如果您的电脑上安装了最近版本的 Apache，可以通过

新版本 Apache 提供的工具生成：

[devel@RIZI bin]$ pwd --我的新版 Apache 目录

/usr/local/apache/bin

[devel@RIZI bin]$ httpd –v --我的新版 Apache 版本

Server version: Apache/2.2.8 (Unix)

Server built: May 30 2008 15:51:42

[devel@RIZI bin]$ htpasswd -cb /home/devel/apache_0.6.5/htdocs/.htpasswd tsingien 123456

Adding password for user tsingien

[devel@RIZI bin]$

如果您没有较新版本的 Apache，也没有关系，这个密码文件是一个文本文件，您可以手动创建。

该文件每行一条记录，每个记录格式是“用户名：加密密码”。我使用 tsingien 做为用户名，123456 做为密码生成的密码文件内容为：

tsingien:SreeIymMXXp0U

设置好上面的步骤后，重启您的 Apache，在浏览器里面输入您的站点地址，即可看到了 10.1 节中展示的登录对话框了。

组认证访问

除了针对某个用户的认证之外，还有针对组的认证。Apache 没有提供创建认证组文件的命令，它只是一个文本文件，用户可以使用任何的文本编辑器创建并修改此文件。该文件中每一行的格式如下：

组名：用户名 用户名 ……

在认证组文件中指定的用户名，必须先添加到认证口令文件中：

[devel@RIZI bin]$ pwd

/usr/local/apache/bin

[devel@RIZI bin]$ htpasswd -cb /home/devel/apache_0.6.5/htdocs/.htpasswd tsingien 123456

Adding password for user tsingien

[devel@RIZI bin]$ htpasswd -b /home/devel/apache_0.6.5/htdocs/.htpasswd laoli 123456

Adding password for user laoli

[devel@RIZI bin]$ tail /home/devel/apache_0.6.5/htdocs/.htpasswd

tsingien:SreeIymMXXp0U

laoli:YbHbUeJaxCkaE

[devel@RIZI bin]$

修改 access.conf 文件中的<Limit GET>部分为：

<Limit GET>

order allow,deny

allow from all

require group goodmen

</Limit>

创建/home/devel/apache_0.6.5/htdocs/.htgroup 文件（组认证文件），内容为：

goodmen:tsingien laoli

badmen:xitele

修改文件/home/devel/apache_0.6.5/htdocs/.htaccess 文件内容为：

AuthName "Private Level 9"

AuthType Basic

AuthUserFile /home/devel/apache_0.6.5/htdocs/.htpasswd

AuthGroupFile /home/devel/apache_0.6.5/htdocs/.htgroup

上面的设置说明，我们允许 goodmen 组里面的 tsingien 和 laoli 两个用户通过认证访问站点，不允许其它用户访问。

设置完成后，重新启动 Apache，用浏览器访问站点，同样可以看到 10.1 节中展示的登陆对话框，使用 tsingien 或 laoli 帐号及其对应的密码即可登录。

下面，我们完整介绍一下 Apache 关于存取控制的设置指令。

10.2.2 设置指令详解

本章涉及到的指令包括 AllowOverride、Option、AuthName、AuthType、AuthUserFile、AuthGroupFile 以及 Limit 类指令。另外还包括列表目录类指令，列表目录类指令参考第九章的描述。

1. AllowOverride 指令

语法：AllowOverride All|None|*指令类型* [*指令类型*] ...

功能：允许存在于.htaccess 文件中的指令类型

示例：AllowOverride All

上例说明，所有具有.htaccess 上下文的指令都允许出现在.htaccess 文件中。

AllowOverride 指令可以有 7 个不同的值，名称和含义如下所示：

Limit：允许使用主机访问控制指令(Allow,Deny 和 Order)

Options：允许使用控制指定目录功能指令(Options)

FileInfo：允许使用控制文档类型指令(ErrorDocument 等)

AuthConfig：允许使用鉴权指令(AuthName 等)

Indexes：允许使用目录索引指令(AddIcon,HeaderName 等)

None：.htaccess 文件将被完全忽略

All：所有具有.htaccess 上下文的指令都允许出现在.htaccess 文件中

2. Option 指令

语法：Options [+|-]可选项 [[+|-]可选项] ...

功能：配置在特定目录使用哪些特性。

示例：Options All

上例说明，将使用所有特性。

Options指令可以有8个不同的值，名称和含义如下：

Indexes：如果服务器收到了一个映射到该目录的URL请求，而目录下又没有DirectoryIndex(如index.html)，那么服务器返回一个格式化目录列表。

Includes：允许服务器端包含。

IncludesNOEXEC：允许服务器端包含，但禁用#exec命令和#exec CGI。

FollowSymLinks：服务器会在此目录中使用符号连接。

SymLinksIfOwnerMatch：符号连接仅当与它的目的文件或目录具有相同的用户ID时才使用。简单理解就是符号链接的所有者和它指向的文件或目录是同一个人。

execCGI：允许执行CGI脚本。
None：都不允许。
All：所有特性。

3. AuthName 指令

语法：AuthName *auth-domain*
功能：为目录的验证域设置名字
示例：AuthName ″九级权限″
上例说明，“九级权限”字符串将出现在浏览器要求验证的对话框中。

4. AuthType 指令

语法：AuthType Basic|Digest
功能：用户验证的类型
示例：AuthType Basic
上例说明，将使用 Basic 验证类型验证。
Digest 认证更安全，但是并非所有的浏览器都支持。0.6.5 版本的 Apache 也不支持 Digest 类型的认证。

5. AuthUserFile 指令

语法：AuthUserFile *文件路径*
功能：指定含有认证用户的用户名、密码列表的文件位置
示例：AuthUserFile /home/devel/apache-0.6.5/htdocs/.htpasswd
上例说明，含有认证用户的用户名、密码列表的文件是/home/devel/apache-0.6.5/htdocs/.htpasswd。

6. AuthGroupFile 指令

语法：AuthGroupFile *文件路径*
功能：指定包含用来执行用户认证的用户组列表文本文件
示例：AuthGroupFile /home/devel/apache-0.6.5/htdocs/.htgroup
上例说明，包含用来执行用户认证的用户组列表文本文件是/home/devel/apache-0.6.5/htdocs/. htgroup。

7. <Limit>指令

语法：〈Limit *方法* [*方法*] ... 〉... 〈/Limit〉
功能：限定访问控制仅施用于某些特定的 HTTP 方法
示例：<Limit GET>…</Limit>
上例说明，…部分的访问控制指令仅限于 GET 方法。
该类指令包括：order、allow、require、deny。下面我们一一加以说明：

7.1 order 指令

语法：Order *ordering*
功能：控制缺省的访问状态和 Allow 与 Deny 指令被评估的顺序
示例：Order Deny,Allow
上例说明，Deny 指令在 Allow 指令之前被评估。缺省允许所有访问。 任何不匹配 Deny 指令或者匹配 Allow 指令的客户都被允许访问服务器。

7.2 allow 指令

语法：Allow from all|*host*
功能：控制哪些主机能够访问服务器的一个区域
示例：Allow from 10.1.2.3
上例说明，允许 IP 为 10.1.2.3 的主机访问

7.3 require 指令

语法：Require *entity-name* [*entity-name*] ...

功能：选择哪个认证用户能访问某个资源

示例：Require valid-user

上例说明，所有有效用户都可以访问这个资源。

Require 必须伴随 AuthName 和 AuthType 指令，以及诸如 AuthUserFile 和 AuthGroupFile 以确保其能够正确工作。

7.4 deny 指令

语法：Deny from all|*host*

功能：哪些主机被禁止访问服务器

示例：Deny from 10.2.3.4

上例说明，禁止 IP 为 10.2.3.4 的主机访问指定资源。

您可以通过修改配置文件来测试上面提到的 7 个指令及其子指令的效果，我们在下一节中通过 Apache 源代码来说明这些设置指令是怎么生效的。

10.3 代码注释

HTML1.0 版本支持对资源的 5 种不同请求方法，它们分别是 HEAD、GET、POST、PUT 和 DELETE。关于这 5 种请求方法的不同，可以参看 11.1 节的描述。

由于 HEAD 请求方法不需要服务器返回响应体，所以在存取控制的时候只处理其余的 4 种方法。

本章代码包括了http_access.c、http_auth.c和http_config.c中的一部分（parse_access_dir 函数调用），由于http_config.c主要是读取配置信息的功能，所以针对它不在本章描述，请参考本书的第11章。

针对存取控制的结构体security_data在httpd.h定义：

```
typedef struct {
char *d;                                    // 存取控制信息管理的路径
char opts;                                  // Options的值
char override;                              // AllowOverride的值
int order[METHODS];                         // 四种方法每种方法的order值
int num_allow[METHODS];                     // 四种方法允许访问的主机个数
char *allow[METHODS][MAX_SECURITY];         // 四种方法允许访问的主机列表
int num_auth[METHODS];                      // 四种方法可访问目录认证用户个数
char *auth[METHODS][MAX_SECURITY];          // 四种方法，可访问目录认证用户列表
char *auth_type;                            // 目录认证方式
char *auth_name;                            // 认证用户名
char *auth_pwfile;                          // 认证用户文件名
char *auth_grpfile;                         // 认证组文件名
int num_deny[METHODS];                      // 四种方法，不允许访问的主机个数
char *deny[METHODS][MAX_SECURITY];          // 四种方法，不允许访问的主机列表
} security_data;
```

在实际应用的时候，使用一个数组来存储管理员对每个目录的设置信息：

```
security_data sec[MAX_SECURITY];
```

10.3.1 http_access.c

```
1   #include "httpd.h"
2
     /*
     * 检查某个子域名 what 是否在域 domain 里面
     * 如检查 dl.oldapache.org 是否在域 oldapache.org 里面
     */
3   int in_domain(char*domain,char*what)
4   {
5       int dl=strlen(domain);
6       int wl=strlen(what);
7
8       if((wl-dl)>=0)
9           return(!strcmp(domain,&what[wl-dl]));
10      else
11      return 0 ;
12  }
13
     /*
     * 检查某个 IP 段 what 是否是 domain 的子段
     * 如 11.22.33.44 是 11.22.33 的子段
     */
14  int in_ip(char*domain,char*what)
15  {
16      return(!strncmp(domain,what,strlen(domain)));
17  }
18   /* 检查客户端 ip 或域名是否在允许访问的范围 */
19  int find_allow(int x,int m)
20  {
21      register int y ;
22
23      if(sec[x].num_allow[m]<0)
24          return 1 ;
25
26      for(y=0;y<sec[x].num_allow[m];y++) // 在允许列表中查找
27      {
28          if(!strcmp("all",sec[x].allow[m][y]))
29              return 1 ;
30          if(remote_host&&isalpha(remote_host[0]))
31              if(in_domain(sec[x].allow[m][y],remote_host))
32                  return 1 ;
33              if(in_ip(sec[x].allow[m][y],remote_ip))
34                  return 1 ;
```

```
35          }
36          return 0 ;
37      }
38       /* 检查客户端 ip 或域名是否在禁止访问的范围 */
39      int find_deny(int x,int m)
40      {
41          register int y ;
42
43          if(sec[x].num_deny[m]<0)
44              return 1 ;
45
46          for(y=0;y<sec[x].num_deny[m];y++) // 在禁止列表中查找
47          {
48              if(!strcmp("all",sec[x].deny[m][y]))
49                  return 1 ;
50              if(remote_host&&isalpha(remote_host[0]))
51                  if(in_domain(sec[x].deny[m][y],remote_host))
52                      return 1 ;
53                  if(in_ip(sec[x].deny[m][y],remote_ip))
54                      return 1 ;
55          }
56          return 0 ;
57      }
58
         /*
         * 检查某个目录的存取权限
         *   x:入口参数，目录在 sex 中的序列（index）
         *   m:入口参数，针对方法 m 的存取权限
         *   w:出口参数，是否允许访问
         *   n:出口参数,如果允许访问的用户个数不为 0，则 n = x
         */
59      void check_dir_access(int x,int m,int*w,int*n)
60      {
61          if(sec[x].auth_type)
62              auth_type=sec[x].auth_type ;
63          if(sec[x].auth_name)
64              auth_name=sec[x].auth_name ;
65          if(sec[x].auth_pwfile)
66              auth_pwfile=sec[x].auth_pwfile ;
67          if(sec[x].auth_grpfile)
68              auth_grpfile=sec[x].auth_grpfile ;
69
70          if(sec[x].order[m]==ALLOW_THEN_DENY)
71          {
```

```
72              *w=0 ;
73              if(find_allow(x,m))
74                  *w=1 ;
75              if(find_deny(x,m))
76                  *w=0 ;
77          }
78          else if(sec[x].order[m]==DENY_THEN_ALLOW)
79          {
80              if(find_deny(x,m))
81                  *w=0 ;
82              if(find_allow(x,m))
83                  *w=1 ;
84          }
85          else
86              *w=find_allow(x,m)&&(!find_deny(x,m));
87
88          if(sec[x].num_auth[m])
89              *n=x ;
90      }
91
        /*
         * 评估存取权限
         * 入口参数：
         *    p:文件全路径，如：/home/devel/apache_0.6.5/htdocs/hello.html
         *    finfo:p 的 stat 信息
         *    m:方法，如 M_GET
         *    out:标准错误输出文件描述符
         * 出口参数：
         *    allow:是否允许
         *    allow_options:允许选项
         */
92      void evaluate_access(char*p,struct stat*finfo,int m,int*allow,
93                                  char*allow_options,FILE*out)
94      {
95          int will_allow,need_auth,num_dirs ;
96          char opts[MAX_STRING_LEN],override[MAX_STRING_LEN];
97          char path[MAX_STRING_LEN],d[MAX_STRING_LEN];
98          char errstr[MAX_STRING_LEN];
99          register int x,y,z,n ;
100
101         if(S_ISDIR(finfo->st_mode))strcpy_dir(path,p);
102         else strcpy(path,p);
103
104         no2slash(path);                 // 双反斜杠替换成单反斜杠
```

```
105
106         num_dirs=count_dirs(path);  // 目录层级
107         will_allow=1 ;
108         need_auth=-1 ;
            /* 初始化认证相关的信息 */
109         auth_type=NULL ;
110         auth_name=NULL ;
111         auth_pwfile=NULL ;
112         auth_grpfile=NULL ;
113         user[0]='\0' ;
114         for(x=0;x<num_dirs;x++)
115         {
116             opts[x]=OPT_ALL ;
117             override[x]=OR_ALL ;
118         }
119
120         n=num_dirs-1 ;                  // 要检查的文件所在目录的上层目录的目录层数
121         for(x=0;x<num_sec;x++)      // 在所有设置的目录中找
122         {
123             if(is_matchexp(sec[x].d))                   // 目录中含有通配符
124             {
125                 if(!strcmp_match(path,sec[x].d))   // path 符合 sec[x].d 的定义
126                 {
127                     for(y=0;y<num_dirs;y++)
128                     {
129                         if(!(sec[x].opts&OPT_UNSET))   // 如果 option 设置了
130                             opts[y]=sec[x].opts ;
131                         override[y]=sec[x].override ;
132                     }
133                 }
134                 check_dir_access(x,m,&will_allow,&need_auth);
135             }
136             else if(!strncmp(path,sec[x].d,strlen(sec[x].d)))    //目录中不含通配符
137             {
138                 for(y=count_dirs(sec[x].d)-1;y<num_dirs;y++)
139                 {
140                     if(!(sec[x].opts&OPT_UNSET))          // 如果设置了 option 属性
141                         opts[y]=sec[x].opts ;
142                     override[y]=sec[x].override ;
143                 }
144                 check_dir_access(x,m,&will_allow,&need_auth);
145             }
146         }
147
```

```
            /* 如果 path 所在目录 override 属性不为 0，或者 option 属性不允许链接或者
            * 允许 OWNER 方式的符号连接
            * 关于 option 属性的介绍请参看第 9 章
            */
148         if((override[n])||(!(opts[n]&OPT_SYM_LINKS))||(opts[n]&OPT_SYM_OWNER))
149         {
150             for(x=0;x<num_dirs;x++)     // 针对每级目录
151             {
152                 y=num_sec ;
153                 make_dirstr(path,x+1,d);
154                  /* 不允许符号链接，或者允许 OWNER 方式的符号链接 */
155                 if((!(opts[x]&OPT_SYM_LINKS))||(opts[x]&OPT_SYM_OWNER))
156                 {
157                     struct stat lfi,fi ;
158                      /* 如果 lstat 调用失败，很显然没有符号链接，所以我们不需要
                        * 再检查它
                        */
159                     if(lstat(d,&lfi)>=0&&!(S_ISDIR(lfi.st_mode)))
160                     {
161                         if(opts[x]&OPT_SYM_OWNER) // 允许所有者的符号链接
162                         {
163                             char realpath[512];
164                             int bsz ;
165                              // 读取符号链接的实际路径
166                             if((bsz=readlink(d,realpath,256))==-1)
167                                 goto bong ;
168                             realpath[bsz]='\0' ;
169                             if(realpath[0]!='/')
170                             {
171                                 char t[256];
172                                 strcpy(t,"../");
173                                 strcpy(&t[3],realpath);
174                                 make_full_path(d,t,realpath);
175                                 getparents(realpath);
176                             }
177                             lstat(realpath,&fi);
178                             if(fi.st_uid!=lfi.st_uid) // 不是所有者
179                                 goto bong ;
180                         }
181                         else
182                         {
183                             bong :
184                             sprintf(errstr,"httpd: will not follow link %s",d);
185                             log_error(errstr);
```

```
186                                 *allow=0 ;
187                                 *allow_options=OPT_NONE ;
188                                 return ;
189                             }
190                         }
191                     }
192                     if(override[x])
193                     {
194                         parse_htaccess(d,override[x],out); //  分析 access 信息
195                          /*  分析 access 信息  */
196                         if(num_sec!=y)
197                         {
198                             for(z=count_dirs(sec[y].d)-1;z<num_dirs;z++)
199                             {
200                                 if(!(sec[y].opts&OPT_UNSET))
201                                     opts[z]=sec[y].opts ;
202                                 override[z]=sec[y].override ;
203                             }
204                             check_dir_access(y,m,&will_allow,&need_auth);
205                         }
206                     }
207                 }
208             }
209              /*  不是目录并且(不允许符号链接或允许 OWNER 方式的符号链接) */
210             if((!(S_ISDIR(finfo->st_mode)))&&
                    ((!(opts[n]&OPT_SYM_LINKS))||(opts[n]&OPT_SYM_OWNER)))
211             {
212                 struct stat fi,lfi ;
213                 lstat(path,&fi);
214                 if(!(S_ISREG(fi.st_mode)))
215                 {
216                     if(opts[n]&OPT_SYM_OWNER)
217                     {
218                         char realpath[512];
219                         int bsz ;
220
221                         if((bsz=readlink(path,realpath,256))==-1)
222                             goto gong ;
223                         realpath[bsz]='\0' ;
224                         if(realpath[0]!='/')
225                         {
226                             char t[256];
227                             strcpy(t,"../");
228                             strcpy(&t[3],realpath);
```

```
229                             make_full_path(path,t,realpath);
230                             getparents(realpath);
231                         }
232                         lstat(realpath,&lfi);
233                         if(fi.st_uid!=lfi.st_uid)
234                             goto gong ;
235                     }
236                     else
237                     {
238                         gong :
239                         sprintf(errstr,"httpd: will not follow link %s",path);
240                         log_error(errstr);
241                         *allow=0 ;
242                         *allow_options=OPT_NONE ;
243                         return ;
244                     }
245                 }
246             }
247             *allow=will_allow ;
248             if(will_allow)
249             {
250                 *allow_options=opts[num_dirs-1];
251                 if(need_auth>=0)
252                     check_auth(&sec[need_auth],m,out);
253             }
254             else*allow_options=0 ;
255         }
256          /* 清空目录存取权限控制数组 */
257         void kill_security()
258         {
259             register int x,y,m ;
260
261             for(x=0;x<num_sec;x++)
262             {
263                 free(sec[x].d);
264                 for(m=0;m<METHODS;m++)
265                 {
266                     for(y=0;y<sec[x].num_allow[m];y++)
267                         free(sec[x].allow[m][y]);
268                     for(y=0;y<sec[x].num_deny[m];y++)
269                         free(sec[x].deny[m][y]);
270                     for(y=0;y<sec[x].num_auth[m];y++)
271                         free(sec[x].auth[m][y]);
272                 }
```

```
273                 if(sec[x].auth_type)
274                     free(sec[x].auth_type);
275                 if(sec[x].auth_name)
276                     free(sec[x].auth_name);
277                 if(sec[x].auth_pwfile)
278                     free(sec[x].auth_pwfile);
279                 if(sec[x].auth_grpfile)
280                     free(sec[x].auth_grpfile);
281         }
282     }
283
```

10.3.2 http_auth.c

```
1   #include "httpd.h"
2   #include <unistd.h>
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <unistd.h>
6   #include <string.h>
7
8   char user[MAX_STRING_LEN];
9    /* 记录认证输出 */
10  void auth_bong(char*s,FILE*out)
11  {
12      char errstr[MAX_STRING_LEN];
         /* 调试信息 */
13      if(s)
14      {
15          sprintf(errstr,"%s authorization: %s",remote_name,s);
16          log_error(errstr);
17      }
18      if(!strcasecmp(auth_type,"Basic"))// Basic 认证方式
19      {
20          sprintf(errstr,"Basic realm=\"%s\"",auth_name);
21          die(AUTH_REQUIRED,errstr,out);
22      }
23      else                                    // 未知认证方式
24      {
25          sprintf(errstr,"Unknown authorization method %s",auth_type);
26          die(SERVER_ERROR,errstr,out);
27      }
28  }
29
```

```
30  extern char log_user[MAX_STRING_LEN];
31   /* 认证检查 */
32  void check_auth(security_data*sec,int m,FILE*out)
33  {
34      char at[MAX_STRING_LEN];
35      char ad[MAX_STRING_LEN];
36      char sent_pw[MAX_STRING_LEN];
37      char real_pw[MAX_STRING_LEN];
38      char t[MAX_STRING_LEN];
39      char w[MAX_STRING_LEN];
40      char errstr[MAX_STRING_LEN];
41      register int x,y ;
42      int grpstatus ;
43
44      if(!auth_type)                  // auth_type 为空
45      {
46          sprintf(errstr,
47              "httpd: authorization required for %s but not configured",
48              sec->d);
49          die(SERVER_ERROR,errstr,out);
50      }
51
52      if(!strcasecmp(auth_type,"Basic"))// Basic 认证方式
53      {
54          if(!auth_name)
55          {
56              sprintf(errstr,"httpd: need AuthName for %s",sec->d);
57              die(SERVER_ERROR,errstr,out);
58          }
59          if(!auth_line[0])
60              auth_bong(NULL,out);
61          if(!auth_pwfile)            // 未指定认证文件
62          {
63              sprintf(errstr,"httpd: need AuthUserFile for %s",sec->d);
64              die(SERVER_ERROR,errstr,out);
65          }
66          sscanf(auth_line,"%s %s",at,t);
67          if(strcmp(at,auth_type))    // 认证方式不匹配
68              auth_bong("type mismatch",out);
            // 使用 BASE64 编码方式解密认证串
69          uudecode(t,(unsigned char*)ad,MAX_STRING_LEN);
70          getword(user,ad,':');       // 获取用户名
71          if(log_user[0]=='\0')
72          {
```

```
73                  strcpy(log_user,user);
74              }
75              ;
76              strcpy(sent_pw,ad);             // 保存密码
77
78              if(!get_pw(user,real_pw,out)) // 获取本地文件中对应用户的密码
79              {
80                  sprintf(errstr,"user %s not found",user);
81                  auth_bong(errstr,out);
82              }
83               /* 比较密码。有关 crypt 函数功能及 DES 算法请参考相关资料 */
84              if(strcmp(real_pw,(char*)crypt(sent_pw,real_pw)))
85              {
86                  sprintf(errstr,"user %s: password mismatch",user);
87                  auth_bong(errstr,out);
88              }
89          }
90          else
91          {
92              sprintf(errstr,"unknown authorization type %s for %s",auth_type,
93                  sec->d);
94              auth_bong(errstr,out);
95          }
96
97          if(auth_grpfile)
98              grpstatus=init_group(auth_grpfile,out);
99          else
100             grpstatus=0 ;
101
102         for(x=0;x<sec->num_auth[m];x++)
103         {
104             strcpy(t,sec->auth[m][x]);
105             getword(w,t,' ');
106             if(!strcmp(w,"valid-user"))
107                 goto found ;
108             if(!strcmp(w,"user"))           // 特定用户认证
109             {
110                 while(t[0])
111                 {
112                     if(t[0]=='\"')
113                     {
114                         getword(w,&t[1],'\"');
115                         for(y=0;t[y];y++)
116                             t[y]=t[y+1];
```

```
117                         }
118                         getword(w,t,' ');
119                         if(!strcmp(user,w))
120                             goto found ;
121                     }
122                 }
123                 else if(!strcmp(w,"group"))    // 特定组认证
124                 {
125                     if(!grpstatus)
126                     {
127                         sprintf(errstr,"group required for %s, bad groupfile",
128                             sec->d);
129                         auth_bong(errstr,out);
130                     }
131                     while(t[0])                 // 获取可以认证的组序列
132                     {
133                         getword(w,t,' ');
134                         if(in_group(user,w)) // 检查客户端输入的用户是否在设置的组中
135                             goto found ;
136                     }
137                 }
138                 else
139                     auth_bong("require not followed by user or group",out);
140         }
141         if(grpstatus)kill_group();
142         sprintf(errstr,"user %s denied",user);
143         auth_bong(errstr,out);
144         found :
145         if(grpstatus)
146             kill_group();
147     }
```

10.4 小结

通过存取控制，您可以杜绝某些不受欢迎的人访问您的网站，也可以给网站划分出“私密地带”给某些持有密码的人浏览。

到此为止，我们已经介绍完了 Apache0.6.5 的所有功能模块，这些功能模块的实现都或多或少的使用了配置文件中的配置信息，下一章，我们将讨论这些配置信息的获取过程。即 http_config.c 文件。

第 11 章 读取配置文件

11.1 概述

我们在前几章里面描述的所有几乎所有配置的初始化都是在文件 http_config.c 中完成的。我们之所以把它放在这里是为了让您在了解了所有的配置功能以后才查看这些配置是怎么读取的，这样更容易理解这部分代码。

11.2 代码注释

```
1   #include "httpd.h"
2    /*  服务器全局配置信息  */
3   int port ;                                  //  服务器开启的端口
4   uid_t user_id ;                             //  服务器以哪个用户 ID 启动
5   char user_name[LOGNAME_MAX+12];             //  对应 user_id 的用户名
6   gid_t group_id ;                            //  服务器以哪个组权限启动
7   char server_root[MAX_STRING_LEN];           //  服务器目录
8   char error_fname[MAX_STRING_LEN];           //  错误日志
9   char xfer_fname[MAX_STRING_LEN];            //  访问日志
10  char pid_fname[MAX_STRING_LEN];             //  服务器运行时进程 ID 记录文件
11  char server_admin[MAX_STRING_LEN];          //  出错时提示接受意见的邮箱
12  char*server_hostname ;                      //  站点域名
13  char srm_confname[MAX_STRING_LEN];          // srm.conf 文件位置
14  char server_confname[MAX_STRING_LEN];       // httpd.conf 文件位置
15  char access_confname[MAX_STRING_LEN];       // access.conf 文件位置
16  char types_confname[MAX_STRING_LEN];        // mime.types 文件位置
17   /*  发生各种错误时(如 404),用户自定义的错误信息*/
18  char*response_code_strings[RESPONSE_CODES+1];
19  int timeout ;
20
     /*  处理服务器配置
     * 1.确定各个配置文件的路径
     * 2.确定各个日志文件的路径
     * 3.根据文件 httpd.conf 的内容设置相关参数
     */
21  void process_server_config(FILE*errors)
22  {
23        FILE*cfg ;
24        char l[MAX_STRING_LEN],w[MAX_STRING_LEN];
25        int n=0 ;
26         /*  初始化服务器全局配置信息  */
```

```
27          port=DEFAULT_PORT ;
28          strcpy(user_name,DEFAULT_USER);
29          user_id=uname2id(DEFAULT_USER);
30          group_id=gname2id(DEFAULT_GROUP);
31          make_full_path(server_root,DEFAULT_ERRORLOG,error_fname);
32          make_full_path(server_root,DEFAULT_XFERLOG,xfer_fname);
33          make_full_path(server_root,DEFAULT_PIDLOG,pid_fname);
34          strcpy(server_admin,DEFAULT_ADMIN);
35          server_hostname=NULL ;
36          make_full_path(server_root,RESOURCE_CONFIG_FILE,srm_confname);
37          make_full_path(server_root,ACCESS_CONFIG_FILE,access_confname);
38          make_full_path(server_root,TYPES_CONFIG_FILE,types_confname);
39
40          timeout=DEFAULT_TIMEOUT ;
41           // 打开 httpd.conf 文件
42          if(!(cfg=fopen(server_confname,"r")))
43          {
44               fprintf(errors,
45                    "httpd: could not open server config. file %s\n",
46                    server_confname);
47               perror("fopen");
48               exit(1);
49          }
             /* 解析服务器配置文件 */
50          while(!(cfg_getline(l,MAX_STRING_LEN,cfg)))
51          {
52               ++n ;
53               if((l[0]!='#')&&(l[0]!='\0'))          // 当前行行不是注释，且含有配置信息
54               {
55                    cfg_getword(w,l);                // 获取配置指令
56
57                    if(!strcasecmp(w,"Port"))        // 设置端口
58                    {
59                         cfg_getword(w,l);
60                         port=atoi(w);
61                    }
62                    else if(!strcasecmp(w,"User")) // 设置启动 Apache 使用的用户帐号
63                    {
64                         cfg_getword(w,l);
65                         user_id=uname2id(w);
66                         strcpy(user_name,w);
67                    }
68                    else if(!strcasecmp(w,"Group"))// 设置启动 Apache 使用的组
69                    {
```

```
70                      cfg_getword(w,l);
71                      group_id=gname2id(w);
72                  }
73                  else if(!strcasecmp(w,"ServerAdmin")) // 出问题时给谁发邮件
74                  {
75                      cfg_getword(w,l);
76                      strcpy(server_admin,w);
77                  }
78                  else if(!strcasecmp(w,"ServerName"))  // 站点域名
79                  {
80                      cfg_getword(w,l);
81                      if(server_hostname)
82                          free(server_hostname);
83                      if(!(server_hostname=strdup(w)))
84                          die(NO_MEMORY,"process_resource_config",errors);
85                  }
86                  else if(!strcasecmp(w,"ServerRoot"))   // 服务起始位置
87                  {
88                      cfg_getword(w,l);
89                      if(!is_directory(w))
90                      {
91                          fprintf(errors,
92                              "Syntax error on line %d of %s:\n",
93                              n,server_confname);
94                          fprintf(errors,"%s is not a valid directory.\n",w);
95                          exit(1);
96                      }
97                      strcpy(server_root,w);
                         /* 根据 server_root 重新设置几个配置文件及日志文件位置 */
98                      make_full_path(server_root,DEFAULT_ERRORLOG,error_fname);
99                      make_full_path(server_root,DEFAULT_XFERLOG,xfer_fname);
100                     make_full_path(server_root,DEFAULT_PIDLOG,pid_fname);
101                     make_full_path(server_root,
                                RESOURCE_CONFIG_FILE,srm_confname);
102                     make_full_path(server_root,
                                ACCESS_CONFIG_FILE,access_confname);
103                     make_full_path(server_root,
                                TYPES_CONFIG_FILE,types_confname);
104                 }
105                 else if(!strcasecmp(w,"ErrorLog"))      // 错误日志位置
106                 {
107                     cfg_getword(w,l);
108                     if(w[0]!='/')
109                         make_full_path(server_root,w,error_fname);
```

```
110                     else
111                         strcpy(error_fname,w);
112                 }
113                 else if(!strcasecmp(w,"TransferLog"))   // 访问日志位置
114                 {
115                     cfg_getword(w,l);
116                     if(w[0]!='/')
117                     make_full_path(server_root,w,xfer_fname);
118                     else strcpy(xfer_fname,w);
119                 }
120                 else if(!strcasecmp(w,"PidFile"))         // 记录服务器 pid 文件位置
121                 {
122                     cfg_getword(w,l);
123                     if(w[0]!='/')
124                         make_full_path(server_root,w,pid_fname);
125                     else strcpy(pid_fname,w);
126                 }
127                 else if(!strcasecmp(w,"AccessConfig"))// access.conf 文件位置
128                 {
129                     cfg_getword(w,l);
130                     if(w[0]!='/')
131                         make_full_path(server_root,w,access_confname);
132                     else strcpy(access_confname,w);
133                 }
134                 else if(!strcasecmp(w,"ResourceConfig"))    // srm.conf 文件位置
135                 {
136                     cfg_getword(w,l);
137                     if(w[0]!='/')
138                         make_full_path(server_root,w,srm_confname);
139                     else strcpy(srm_confname,w);
140                 }
141                 else if(!strcasecmp(w,"TypesConfig"))  // mime.conf 文件位置
142                 {
143                     cfg_getword(w,l);
144                     if(w[0]!='/')
145                         make_full_path(server_root,w,types_confname);
146                     else strcpy(types_confname,w);
147                 }
148                 else if(!strcasecmp(w,"Timeout"))         // 和客户端的连接超时时间
149                     timeout=atoi(l);
150
151                 else   // 错误的配置行
152                 {
153                     fprintf(errors,
```

```
154                             "Syntax error on line %d of %s:\n",
155                             n,server_confname);
156                         fprintf(errors,"Unknown keyword %s.\n",w);
157                         exit(1);
158                     }
159                 }
160         }
161         fclose(cfg);
162     }
163      /* 文档全局配置信息 */
164     char user_dir[MAX_STRING_LEN];
         // 可能有几个类型默认主页，如 index.html index.htm
165     char index_names[HUGE_STRING_LEN];
166     char access_name[MAX_STRING_LEN];
167     char document_root[MAX_STRING_LEN];
168     char default_type[MAX_STRING_LEN];
169     char default_icon[MAX_STRING_LEN];
170
         /*
         * 处理资源配置
         * 主要处理 srm.conf 配置文件的内容
         */
171     void process_resource_config(FILE*errors)
172     {
173         FILE*cfg ;
174         char l[MAX_STRING_LEN],w[MAX_STRING_LEN];
175         int init,n=0 ;
176
177         strcpy(user_dir,DEFAULT_USER_DIR);
178         strcpy(index_names,DEFAULT_INDEX);
179         strcpy(access_name,DEFAULT_ACCESS_FNAME);
180         strcpy(document_root,DOCUMENT_LOCATION);
181         strcpy(default_type,DEFAULT_TYPE);
182         default_icon[0]='\0' ;
183          // 增加一个新的 opt 条目，参看第九章“自动索引目录”
184         add_opts_int(0,"/",errors);
185          // 打开 srm.conf 配置文件
186         if(!(cfg=fopen(srm_confname,"r")))
187         {
188             fprintf(errors,"httpd: could not open document config. file %s\n",
189                 srm_confname);
190             perror("fopen");
191             exit(1);
192         }
```

```
        // 初始化或重新初始化自定义状态反馈信息，如果已经存在，则释放
193     for(init=0;init<=RESPONSE_CODES;init++)
194     {
195         if(response_code_strings[init]!=NULL)
196             free(response_code_strings[init]);
197         response_code_strings[init]=NULL ;
198     }
199      // 读取配置文件
200     while(!(cfg_getline(l,MAX_STRING_LEN,cfg)))
201     {
202         ++n ;
203         if((l[0]!='#')&&(l[0]!='\0'))              // 含有配置信息
204         {
205             cfg_getword(w,l);
206             if(!strcasecmp(w,"ScriptAlias"))   // 脚本别名，参看第 5 章
207             {
208                 char w2[MAX_STRING_LEN];
209
210                 cfg_getword(w,l);
211                 cfg_getword(w2,l);
212                 if((w[0]=='\0')||(w2[0]=='\0'))
213                 {
214                     fprintf(errors,
215                         "Syntax error on line %d of %s:\n",
216                         n,srm_confname);
217                     fprintf(errors,
218                         "ScriptAlias must be followed by a fakename,
                            one space, then a realname.\n");
219                     exit(1);
220                 }
                    // w=/cgi-bin/, w2=/home/devel/apache_0.6.5/cgi-bin/
221                 if(!add_alias(w,w2,SCRIPT_CGI))
222                     die(NO_MEMORY,"process_resource_config",errors);
223             }
224             else if(!strcasecmp(w,"UserDir"))          // 用户目录，参看第 5 章
225             {
226                 cfg_getword(w,l);
227                 if(!strcmp(w,"DISABLED"))
228                     user_dir[0]='\0' ;
229                 else
230                     strcpy(user_dir,w);
231             }
232             else if(!strcasecmp(w,"DirectoryIndex"))   // 默认文件，如 index.html
233             {
```

```
234                     strcpy(index_names,l);
235                 }
                    // 默认 MIME 类型，如 text/plain
236                 else if(!strcasecmp(w,"DefaultType"))
237                 {
238                     cfg_getword(w,l);
239                     strcpy(default_type,w);
240                 }
                    // 每个目录的存取控制文件名，如.htaccess
241                 else if(!strcasecmp(w,"AccessFileName"))
242                 {
243                     cfg_getword(w,l);
244                     strcpy(access_name,w);
245                 }
                    // 站点起始位置，如/home/devel/apache_0.6.5/htdocs
246                 else if(!strcasecmp(w,"DocumentRoot"))
247                 {
248                     cfg_getword(w,l);
249                     if(!is_directory(w))            // 如果不是一个目录
250                     {
251                         fprintf(errors,"Syntax error on line %d of %s:\n",n,
252                             srm_confname);
253                         fprintf(errors,"%s is not a valid directory.\n",w);
254                         exit(1);
255                     }
256                     strcpy(document_root,w);
257                 }
                    // 自定义别名，如 Alias /icons/ /home/devel/apache_0.6.5/icons/
258                 else if(!strcasecmp(w,"Alias"))
259                 {
260                     char w2[MAX_STRING_LEN];
261
262                     cfg_getword(w,l);
263                     cfg_getword(w2,l);
264                     if((w[0]=='\0')||(w2[0]=='\0')) // 格式不对
265                     {
266                         fprintf(errors,"Syntax error on line %d of %s:\n",n,
267                             srm_confname);
268                         fprintf(errors,
269                             "Alias must be followed by a fakename,
                                one space, then a realname.\n");
270                         exit(1);
271                     }
272                     if(!add_alias(w,w2,STD_DOCUMENT)) // 增加一个别名
```

```
273                     die(NO_MEMORY,"process_resource_config",errors);
274                 }
                    // 参看第六章的描述，如：AddType image/gif .gif
275                 else if(!strcasecmp(w,"AddType"))
276                 {
277                     char w2[MAX_STRING_LEN];
278                     cfg_getword(w,l);
279                     cfg_getword(w2,l);
280                     if((w[0]=='\0')||(w2[0]=='\0'))
281                     {
282                         fprintf(errors,
283                             "Syntax error on line %d of %s:\n",n,srm_confname);
284                         fprintf(errors,
285                             "AddType must be followed by a type, one space,
                                then a file or extension.\n");
286                         exit(1);
287                     }
288                     add_type(w2,w,errors);
289                 }
                    // 参看第六章的描述，如：AddEncoding x-compress .Z
290                 else if(!strcasecmp(w,"AddEncoding"))
291                 {
292                     char w2[MAX_STRING_LEN];
293                     cfg_getword(w,l);
294                     cfg_getword(w2,l);
295                     if((w[0]=='\0')||(w2[0]=='\0'))      // 格式错误
296                     {
297                         fprintf(errors,
298                             "Syntax error on line %d of %s:\n",n,srm_confname);
299                         fprintf(errors,
300                             "AddEncoding must be followed by a type, one space,
                                then a file or extension.\n");
301                         exit(1);
302                     }
303                     add_encoding(w2,w,errors);
304                 }
                    // 参看第五章的描述，如：Redirect /test http://www.oldapache.org/
305                 else if(!strcasecmp(w,"Redirect"))
306                 {
307                     char w2[MAX_STRING_LEN];
308                     cfg_getword(w,l);
309                     cfg_getword(w2,l);
310                     if((w[0]=='\0')||(w2[0]=='\0')||(!is_url(w2)))
311                     {
```

```
312                         fprintf(errors,
313                             "Syntax error on line %d of %s:\n",n,srm_confname);
314                         fprintf(errors,"Redirect must be followed by a document,
                                    one space, then a URL.\n");
315                         exit(1);
316                     }
317                     if(!add_redirect(w,w2))
318                         die(NO_MEMORY,"process_resource_config",errors);
319                 }
                    // 参看第 9 章“自动索引目录”
320                 else if(!strcasecmp(w,"FancyIndexing"))
321                 {
322                     cfg_getword(w,l);
323                     if(!strcmp(w,"on"))
324                         add_opts_int(FANCY_INDEXING,"/",errors);
325                     else if(!strcmp(w,"off"))
326                         add_opts_int(0,"/",errors);
327                     else
328                     {
329                         fprintf(errors,"Syntax error on line %d of %s:\n",n,
330                             srm_confname);
331                         fprintf(errors,"FancyIndexing must be on or off.\n");
332                         exit(1);
333                     }
334                 }
                    // 参看第 9 章“自动索引目录”
335                 else if(!strcasecmp(w,"AddDescription"))
336                 {
337                     char desc[MAX_STRING_LEN];
338                     int fq ;
339                     if((fq=ind(l,'\"'))==-1)
340                     {
341                         fprintf(errors,
342                             "Syntax error on line %d of %s:\n",n,srm_confname);
343                         fprintf(errors,"AddDescription must have quotes around the
                                    description.\n");
344                         exit(1);
345                     }
346                     else
347                     {
348                         getword(desc,&l[++fq],'\"');
349                         cfg_getword(w,&l[fq]);
350                         add_desc(BY_PATH,desc,w,"/",errors);
351                     }
```

```
352                 }
                     // 自动索引目录时不显示的文件
353                 else if(!strcasecmp(w,"IndexIgnore"))
354                 {
355                     while(l[0])
356                     {
357                         cfg_getword(w,l);
358                         add_ignore(w,"/",errors);
359                     }
360                 }
                     // 参看第 9 章“自动索引目录”
361                 else if(!strcasecmp(w,"AddIcon"))
362                 {
363                     char w2[MAX_STRING_LEN];
364                     cfg_getword(w2,l);
365                     while(l[0])
366                     {
367                         cfg_getword(w,l);
368                         add_icon(BY_PATH,w2,w,"/",errors);
369                     }
370                 }
                     // 参看第 9 章“自动索引目录”
371                 else if(!strcasecmp(w,"AddIconByType"))
372                 {
373                     char w2[MAX_STRING_LEN];
374                     cfg_getword(w2,l);
375                     while(l[0])
376                     {
377                         cfg_getword(w,l);
378                         add_icon(BY_TYPE,w2,w,"/",errors);
379                     }
380                 }
                     // 参看第 9 章“自动索引目录”
381                 else if(!strcasecmp(w,"AddIconByEncoding"))
382                 {
383                     char w2[MAX_STRING_LEN];
384                     cfg_getword(w2,l);
385                     while(l[0])
386                     {
387                         cfg_getword(w,l);
388                         add_icon(BY_ENCODING,w2,w,"/",errors);
389                     }
390                 }
                     // 参看第 9 章“自动索引目录”
```

```
391                 else if(!strcasecmp(w,"AddAlt"))
392                 {
393                     char w2[MAX_STRING_LEN];
394                     cfg_getword(w2,l);
395                     while(l[0])
396                     {
397                         cfg_getword(w,l);
398                         add_alt(BY_PATH,w2,w,"/",errors);
399                     }
400                 }
                    // 参看第 9 章“自动索引目录”
401                 else if(!strcasecmp(w,"AddAltByType"))
402                 {
403                     char w2[MAX_STRING_LEN];
404                     cfg_getword(w2,l);
405                     while(l[0])
406                     {
407                         cfg_getword(w,l);
408                         add_alt(BY_TYPE,w2,w,"/",errors);
409                     }
410                 }
                    // 参看第 9 章“自动索引目录”
411                 else if(!strcasecmp(w,"AddAltByEncoding"))
412                 {
413                     char w2[MAX_STRING_LEN];
414                     cfg_getword(w2,l);
415                     while(l[0])
416                     {
417                         cfg_getword(w,l);
418                         add_alt(BY_ENCODING,w2,w,"/",errors);
419                     }
420                 }
                    // 参看第 9 章“自动索引目录”
421                 else if(!strcasecmp(w,"DefaultIcon"))
422                 {
423                     cfg_getword(w,l);
424                     strcpy(default_icon,w);
425                 }
                    // 参看第 9 章“自动索引目录”
426                 else if(!strcasecmp(w,"ReadmeName"))
427                 {
428                     cfg_getword(w,l);
429                     add_readme(w,"/",errors);
430                 }
```

```
                    // 参看第 9 章“自动索引目录”
431                 else if(!strcasecmp(w,"HeaderName"))
432                 {
433                     cfg_getword(w,l);
434                     add_header(w,"/",errors);
435                 }
                    // 参看第 9 章“自动索引目录”
436                 else if(!strcasecmp(w,"IndexOptions"))
437                     add_opts(l,"/",errors);
                    // 参第 4 章“日志和重定向”，出错的时候返回给客户端的附加信息
438                 else if(!strcasecmp(w,"ErrorDocument"))
439                 {
440                     int error_number,index_number ;
441                     char*cptr ;
442
443                     cfg_getword(w,l);
444                     error_number=atoi(w);  // 应该是我们已经定义的数字
                    // 第一个参数应该是 3 位数字
445                     if(error_number<100||error_number>999
446                         ||!strstr(RESPONSE_CODE_LIST,w))
447                     {
448                         fprintf(errors,
449                             "Illegal HTTP response code '%s' line %d of %s:\n",
450                             w,n,srm_confname);
451                         exit(1);
452                     }
                     // 获取这个数字在 RESPONSE_CODE_LIST 数组中的下标
453                     index_number=index_of_response(error_number);
                     /*
                     * 如果自定义回应是一个字符串("开头),它必须包含一个%s
                     * 位置来保存错误信息。
                     */
454                     if(l[0]==""&&!strstr(l,"%s"))
455                         strcat(l," Reason: %s");
456                     if((response_code_strings[index_number]=strdup(l))==NULL)
457                     {
458                         // 分配空间来保存自定义文本或 url
459                         fprintf(errors,
460                             "Unable to malloc space to hold custom error
                            string '%s'. Line %d of %s:\n",
461                             l,n,srm_confname);
462                         exit(1);
463                     }
464                 }
```

```
465                     else
466                     {
467                         fprintf(errors,"Syntax error on line %d of %s:\n",n,
468                             srm_confname);
469                         fprintf(errors,"Unknown keyword %s.\n",w);
470                         exit(1);
471                     }
472                 }
473             }
474             fclose(cfg);
475         }
476          /* 认证使用的全局变量 */
477         char*auth_type ;                    // 认证类型
478         char*auth_name ;                    // 认证用户名
479         char*auth_pwfile ;                  // 用户/密码文件名
480         char*auth_grpfile ;                 // 认证组文件名
481
482         int num_sec ;                       // 设置路径存取权限的路径个数
483         security_data sec[MAX_SECURITY]; // 每个路径的详细访问策略
484
        /* 读取 access.conf 文件错误处理函数 */
485         void access_syntax_error(int n,char*err,char*file,FILE*out)
486         {
487             if(!file)
488             {
489                 fprintf(out,"Syntax error on line %d of access config. file.\n",n);
490                 fprintf(out,"%s\n",err);
491                 exit(1);
492             }
493             else
494             {
495                 char e[MAX_STRING_LEN];
496                 sprintf(e,
497                     "httpd: syntax error or override violation in access control file %s,
                        reason: %s",
498                     file,err);
499                 die(SERVER_ERROR,e,out);
500             }
501         }
502
          /*
          * 功能:从 access.conf 配置文件获取对某个目录的存取控制信息。
          * 返回值说明已经读取到第几行了。
          */
```

```
503  int parse_access_dir(FILE*f,int line,char or,char*dir,
504                       char*file,FILE*out)
505  {
506      char l[MAX_STRING_LEN];
507      char t[MAX_STRING_LEN];
508      char w[MAX_STRING_LEN];
509      char w2[MAX_STRING_LEN];
510      int n=line ;
511      register int x,i ;
512       // num_sec 在 process_access_config 中被初始化成 0
513      x=num_sec ;
514
515      sec[x].opts=OPT_UNSET ;
516      sec[x].override=or ;
517      if(!(sec[x].d=(char*)malloc((sizeof(char))*(strlen(dir)+2))))
518          die(NO_MEMORY,"parse_access_dir",out);
519      if(is_matchexp(dir))                    // 如果给定目录中含有*或者?
520          strcpy(sec[x].d,dir);
521      else
522          strcpy_dir(sec[x].d,dir);
       // 先设置默认值
523      sec[x].auth_type=NULL ;
524      sec[x].auth_name=NULL ;
525      sec[x].auth_pwfile=NULL ;
526      sec[x].auth_grpfile=NULL ;
527      for(i=0;i<METHODS;i++)                  // 客户端 4 种请求类型
528      {
529          sec[x].order[i]=DENY_THEN_ALLOW ;
530          sec[x].num_allow[i]=0 ;
531          sec[x].num_deny[i]=0 ;
532          sec[x].num_auth[i]=0 ;
533      }
       // 开始读取配置文件
534      while(!(cfg_getline(l,MAX_STRING_LEN,f)))
535      {
536          ++n ;
537          if((l[0]=='#')||(!l[0]))continue ;      // 注释行或空行
538          cfg_getword(w,l);
539          /*
           * 当服务器发现了一个.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访问信
           * 息。
           */
540          if(!strcasecmp(w,"AllowOverride")) // 对目录的 AllowOverride 属性设设置
```

```
541             {
542                 if(file)
543                     access_syntax_error(n,"override violation",file,out);
544                 sec[x].override=OR_NONE ;
545                 while(l[0])
546                 {
547                     cfg_getword(w,l);
                        // 允许使用主机访问控制指令(Allow,Deny 和 Order)
548                     if(!strcasecmp(w,"Limit"))
549                         sec[x].override|=OR_LIMIT ;
                        // 允许使用控制指定目录功能指令(Options)
550                     else if(!strcasecmp(w,"Options"))
551                         sec[x].override|=OR_OPTIONS ;
                        // 允许使用控制文档类型指令(ErrorDocument 等)
552                     else if(!strcasecmp(w,"FileInfo"))
553                         sec[x].override|=OR_FILEINFO ;
                        // 允许使用鉴权指令(AuthName 等)
554                     else if(!strcasecmp(w,"AuthConfig"))
555                         sec[x].override|=OR_AUTHCFG ;
                        // 允许使用目录索引指令(AddIcon,HeaderName 等)
556                     else if(!strcasecmp(w,"Indexes"))
557                         sec[x].override|=OR_INDEXES ;
                        // .htaccess 文件将被完全忽略
558                     else if(!strcasecmp(w,"None"))
559                         sec[x].override=OR_NONE ;
                        // 所有具有.htaccess 上下文的指令都允许出现在.htaccess 文件
                        // 中
560                     else if(!strcasecmp(w,"All"))
561                         sec[x].override=OR_ALL ;
562                     else
563                     {
564                         access_syntax_error(n,
565                             "Unknown keyword in AllowOverride directive.",file,out);
566                     }
567                 }
568             }
                // 该指令控制了在特定目录中将使用哪些服务器特性
569             else if(!strcasecmp(w,"Options"))
570             {
571                 if(!(or&OR_OPTIONS))   // 对目录 Option 目录属性设置
572                     access_syntax_error(n,"override violation",file,out);
573                 sec[x].opts=OPT_NONE ;
574                 while(l[0])
575                 {
```

```
576                     cfg_getword(w,l);

                        // 如果服务器收到了一个映射到该目录的 URL 请求，而目录
                        // 下又没有 DirectoryIndex(如 index.html)，那么服务器返回一
                        // 个格式化目录列表。
577                     if(!strcasecmp(w,"Indexes"))
578                         sec[x].opts|=OPT_INDEXES ;
                        // 允许服务器端包含
579                     else if(!strcasecmp(w,"Includes"))
580                         sec[x].opts|=OPT_INCLUDES ;
                        // 允许服务器端包含，但禁用#exec 命令和#exec CGI
581                     else if(!strcasecmp(w,"IncludesNOEXEC"))
582                         sec[x].opts|=(OPT_INCLUDES|OPT_INCNOEXEC);
                        // 服务器会在此目录中使用符号连接
583                     else if(!strcasecmp(w,"FollowSymLinks"))
584                         sec[x].opts|=OPT_SYM_LINKS ;
                          // 符号连接仅当与它的目的文件或目录具有相同的用户 ID 时
                          // 才使用。简单理解就是符号链接的所有者和它指向的文件
                          // 或目录是同一个人
585                     else if(!strcasecmp(w,"SymLinksIfOwnerMatch"))
586                         sec[x].opts|=OPT_SYM_OWNER ;
                        // 允许执行 CGI 脚本
587                     else if(!strcasecmp(w,"execCGI"))
588                         sec[x].opts|=OPT_EXECCGI ;
                        // 都不允许
589                     else if(!strcasecmp(w,"None"))
590                         sec[x].opts=OPT_NONE ;
                        // 所有特性都允许
591                     else if(!strcasecmp(w,"All"))
592                         sec[x].opts=OPT_ALL ;
593                     else
594                     {
595                         access_syntax_error(n,
596                             "Unknown keyword in Options directive.",file,out);
597                     }
598                 }
599             }
                // 访问认证时弹出对话框中出现的提示信息
600             else if(!strcasecmp(w,"AuthName"))
601             {
602                 if(!(or&OR_AUTHCFG))
603                     access_syntax_error(n,"override violation",file,out);
604                 if(sec[x].auth_name)
605                     free(sec[x].auth_name);
```

```
606                 if(!(sec[x].auth_name=strdup(l)))
607                     die(NO_MEMORY,"parse_access_dir",out);
608             }
                 // 访问认证类型
609             else if(!strcasecmp(w,"AuthType"))
610             {
611                 if(!(or&OR_AUTHCFG))
612                     access_syntax_error(n,"override violation",file,out);
613                 cfg_getword(w,l);
614                 if(sec[x].auth_type)
615                     free(sec[x].auth_type);
616                 if(!(sec[x].auth_type=strdup(w)))
617                     die(NO_MEMORY,"parse_access_dir",out);
618             }
                 // 访问认证时使用的密码文件
619             else if(!strcasecmp(w,"AuthUserFile"))
620             {
621                 if(!(or&OR_AUTHCFG))
622                     access_syntax_error(n,"override violation",file,out);
623                 cfg_getword(w,l);
624                 if(sec[x].auth_pwfile)
625                     free(sec[x].auth_pwfile);
626                 if(!(sec[x].auth_pwfile=strdup(w)))
627                     die(NO_MEMORY,"parse_access_dir",out);
628             }
                 // 组访问认证时包含组信息的文件
629             else if(!strcasecmp(w,"AuthGroupFile"))
630             {
631                 if(!(or&OR_AUTHCFG))
632                     access_syntax_error(n,"override violation",file,out);
633                 cfg_getword(w,l);
634                 if(sec[x].auth_grpfile)
635                     free(sec[x].auth_grpfile);
636                 if(!(sec[x].auth_grpfile=strdup(w)))
637                     die(NO_MEMORY,"parse_access_dir",out);
638             }
                 // 自定义 MIME 信息，如：AddType application/x-httpd-php .php
639             else if(!strcasecmp(w,"AddType"))
640             {
641                 if(!(or&OR_FILEINFO))
642                     access_syntax_error(n,"override violation",file,out);
643                 cfg_getword(w,l);
644                 cfg_getword(w2,l);
645                 if((w[0]=='\0')||(w2[0]=='\0'))
```

```
646                 {
647                     access_syntax_error(n,
648                         "AddType must be followed by a type, one space,
                            then a file or extension."
649                         ,file,out);
650                 }
651                 add_type(w2,w,out);
652             }
                // 如果服务器不能判断一个文件的类型的时候，按照 DefaultType 指定的
                // 类型返回给客户端，如:DefaultType text/plain
653             else if(!strcasecmp(w,"DefaultType"))
654             {
655                 if(!(or&OR_FILEINFO))
656                     access_syntax_error(n,"override violation",file,out);
657                 cfg_getword(w,l);
658                 strcpy(default_type,w);
659             }
                // 如:AddEncoding x-compress .Z
660             else if(!strcasecmp(w,"AddEncoding"))
661             {
662                 if(!(or&OR_FILEINFO))
663                     access_syntax_error(n,"override violation",file,out);
664                 cfg_getword(w,l);
665                 cfg_getword(w2,l);
666                 if((w[0]=='\0')||(w2[0]=='\0'))
667                 {
668                     access_syntax_error(n,
669                         "AddEncoding must be followed by a type, one space,
                            then a file or extension.",
670                         file,out);
671                 }
672                 add_encoding(w2,w,out);
673             }
                // 如:DefaultIcon /icons/unknown.gif
674             else if(!strcasecmp(w,"DefaultIcon"))
675             {
676                 if(!(or&OR_INDEXES))
677                     access_syntax_error(n,"override violation",file,out);
678                 cfg_getword(w,l);
679                 strcpy(default_icon,w);
680             }
                // 在自动索引目录的时候使用，对某种文件类型的简单说明
                // 如:AddDescription "tar archive" .tar
681             else if(!strcasecmp(w,"AddDescription"))
```

```
682             {
683                 char desc[MAX_STRING_LEN];
684                 int fq ;
685
686                 if(!(or&OR_INDEXES))
687                     access_syntax_error(n,"override violation",file,out);
688                 if((fq=ind(l,'\'"'))==-1)
689                     access_syntax_error(n,"AddDescription must have quotes",
690                     file,out);
691                 else
692                 {
693                     getword(desc,&l[++fq],'\'"');
694                     cfg_getword(w,&l[fq]);
695                     add_desc(BY_PATH,desc,w,sec[x].d,out);
696                 }
697             }
                 // IndexIgnore 指定了一些在自动索引目录的时候不显示的文件名
                 // 如：IndexIgnore */.??* *~ *# */README*
698             else if(!strcasecmp(w,"IndexIgnore"))
699             {
700                 if(!(or&OR_INDEXES))
701                     access_syntax_error(n,"override violation",file,out);
702                 while(l[0])
703                 {
704                     cfg_getword(w,l);
705                     add_ignore(w,sec[x].d,out);
706                 }
707             }
                 // 在自动索引目录的时候，AddIcon 指定了什么类型的文件使用那个图
                 // 标。如：AddIcon /icons/binary.gif .bin .exe
708             else if(!strcasecmp(w,"AddIcon"))
709             {
710                 char w2[MAX_STRING_LEN];
711
712                 if(!(or&OR_INDEXES))
713                     access_syntax_error(n,"override violation",file,out);
714                 cfg_getword(w2,l);
715                 while(l[0])
716                 {
717                     cfg_getword(w,l);
718                     add_icon(BY_PATH,w2,w,sec[x].d,out);
719                 }
720             }
                 // 功能同 AddIcon
```

```
            // 如:AddIconByType (TXT,/icons/text.gif) text/*
721         else if(!strcasecmp(w,"AddIconByType"))
722         {
723             char w2[MAX_STRING_LEN];
724
725             if(!(or&OR_INDEXES))
726                 access_syntax_error(n,"override violation",file,out);
727             cfg_getword(w2,l);
728             while(l[0])
729             {
730                 cfg_getword(w,l);
731                 add_icon(BY_TYPE,w2,w,sec[x].d,out);
732             }
733         }
            // 功能同上
            // 如:AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
734         else if(!strcasecmp(w,"AddIconByEncoding"))
735         {
736             char w2[MAX_STRING_LEN];
737
738             if(!(or&OR_INDEXES))
739                 access_syntax_error(n,"override violation",file,out);
740             cfg_getword(w2,l);
741             while(l[0])
742             {
743                 cfg_getword(w,l);
744                 add_icon(BY_ENCODING,w2,w,sec[x].d,out);
745             }
746         }
            // 服务器查找的默认的 readme 文件名。会附加在目录列表后面
747         else if(!strcasecmp(w,"ReadmeName"))
748         {
749             if(!(or&OR_INDEXES))
750                 access_syntax_error(n,"override violation",file,out);
751             cfg_getword(w,l);
752             add_readme(w,sec[x].d,out);
753         }
            // 目录索引前面添加文件的文件名
754         else if(!strcasecmp(w,"HeaderName"))
755         {
756             if(!(or&OR_INDEXES))
757                 access_syntax_error(n,"override violation",file,out);
758             cfg_getword(w,l);
759             add_header(w,sec[x].d,out);
```

```
760             }
                // 控制服务器产生的目录列表的外观
761             else if(!strcasecmp(w,"IndexOptions"))
762             {
763                 if(!(or&OR_INDEXES))
764                     access_syntax_error(n,"override violation",file,out);
765                 add_opts(l,sec[x].d,out);
766             }
                // 原来在服务器上的文档转移了。
                // 这个选项允许您告诉客户端到哪里找到这个转移了的文档
767             else if(!strcasecmp(w,"Redirect"))
768             {
769                 if(!(or&OR_FILEINFO))
770                     access_syntax_error(n,"override violation",file,out);
771                 cfg_getword(w,l);
772                 cfg_getword(w2,l);
773                 if((w[0]=='\0')||(w2[0]=='\0')||(!is_url(w2)))
774                 {
775                     access_syntax_error(n,
776                         "Redirect must be followed by a document,
                            one space, then a URL.",
777                         file,out);
778                 }
779                 if(!file)
780                 {
781                     if(!add_redirect(w,w2))
782                         die(NO_MEMORY,"parse_access_dir",out);
783                 }
784                 else
785                     access_syntax_error(n,
786                     "Redirect no longer supported from .htaccess files.",file,out);
787             }
                // 限定访问控制仅施用于某些特定的 HTTP 方法
788             else if(!strcasecmp(w,"<Limit"))
789             {
790                 int m[METHODS];
791
792                 if(!(or&OR_LIMIT))
793                     access_syntax_error(n,"override violation",file,out);
794                 for(i=0;i<METHODS;i++)m[i]=0 ;
795                 getword(w2,l,'>');
796                 while(w2[0])   // 确定是适用于哪种 HTTP 方法的限定访问
797                 {
798                     cfg_getword(w,w2);
```

```
799                     if(!strcasecmp(w,"GET"))m[M_GET]=1 ;
800                     else if(!strcasecmp(w,"PUT"))m[M_PUT]=1 ;
801                     else if(!strcasecmp(w,"POST"))m[M_POST]=1 ;
802                     else if(!strcasecmp(w,"DELETE"))m[M_DELETE]=1 ;
803                 }
804                 while(1)
805                 {
806                     if(cfg_getline(l,MAX_STRING_LEN,f))
807                         access_syntax_error(n,"Limit missing /Limit",file,out);
808                     n++;
809                     if((l[0]=='#')||(!l[0]))continue ;
810
811                     if(!strcasecmp(l,"</Limit>"))
812                         break ;
813                     cfg_getword(w,l);
                         // 控制缺省的访问状态和 Allow 与 Deny 指令被评估的顺序
814                     if(!strcasecmp(w,"order"))
815                     {
                             // 任何不匹配 Deny 指令或者匹配 Allow 指令的客户都被
                             // 允许访问服务器。
816                         if(!strcasecmp(l,"allow,deny"))
817                         {
818                             for(i=0;i<METHODS;i++)
819                                 if(m[i])
820                                     sec[x].order[i]=ALLOW_THEN_DENY ;
821                         }
                             // 任何不匹配 Allow 指令或者匹配 Deny 指令的客户都将
                             // 被禁止访问服务器
822                         else if(!strcasecmp(l,"deny,allow"))
823                         {
824                             for(i=0;i<METHODS;i++)
825                                 if(m[i])
826                                     sec[x].order[i]=DENY_THEN_ALLOW ;
827                         }
                             // 具有和 ALLOW_THEN_DENY 同样效果，不赞成使用
828                         else if(!strcasecmp(l,"mutual-failure"))
829                         {
830                             for(i=0;i<METHODS;i++)
831                                 if(m[i])
832                                     sec[x].order[i]=MUTUAL_FAILURE ;
833                         }
834                         else
835                             access_syntax_error(n,"Unknown order.",file,out);
836                     }
```

```
                    // 允许哪些主机访问
837                 else if((!strcasecmp(w,"allow")))
838                 {
839                     cfg_getword(w,l);
840                     if(strcmp(w,"from"))
841                         access_syntax_error(n,
842                         "allow must be followed by from.",file,out);
843                     while(1)
844                     {
845                         cfg_getword(w,l);
846                         if(!w[0])break ;
847                         for(i=0;i<METHODS;i++)
848                             if(m[i])
849                             {
                                  // 允许的主机个数
850                                 int q=sec[x].num_allow[i]++;
                                  // 允许的主机列表
851                                 if(!(sec[x].allow[i][q]=strdup(w)))
852                                     die(NO_MEMORY,"parse_access_dir",out);
853                             }
854                     }
855                 }
                    // 哪个认证用户能访问一个目录
856                 else if(!strcasecmp(w,"require"))
857                 {
858                     for(i=0;i<METHODS;i++)
859                         if(m[i])
860                         {
                              //允许访问的用户个数
861                             int q=sec[x].num_auth[i]++;
                              //允许访问的用户
862                             if(!(sec[x].auth[i][q]=strdup(l)))
863                                 die(NO_MEMORY,"parse_access_dir",out);
864                         }
865                 }
                    // 阻止哪些主机访问
866                 else if((!strcasecmp(w,"deny")))
867                 {
868                     cfg_getword(w,l);
869                     if(strcmp(w,"from"))
870                         access_syntax_error(n,
871                         "deny must be followed by from.",file,out);
872                     while(1)
873                     {
```

```
874                                 cfg_getword(w,l);
875                                 if(!w[0])break ;
876                                 for(i=0;i<METHODS;i++)
877                                     if(m[i])
878                                     {
                                            // 阻止的主机个数
879                                         int q=sec[x].num_deny[i]++;
                                            // 阻止的主机列表
880                                         if(!(sec[x].deny[i][q]=strdup(w)))
881                                             die(NO_MEMORY,"parse_access_dir",out);
882                                     }
883                             }
884                         }
885                         else
886                             access_syntax_error(n,
887                             "Unknown keyword in Limit region.",file,out);
888                     }
889                 }
890                 else if(!strcasecmp(w,"</Directory>"))
891                     break ;
892                 else
893                 {
894                     char errstr[MAX_STRING_LEN];
895                     sprintf(errstr,"Unknown method %s",w);
896                     access_syntax_error(n,errstr,file,out);
897                     return 0 ;
898                 }
899         }
900         ++num_sec ;
901         return n ;
902     }
903      /* 分析 access 信息 */
904     void parse_htaccess(char*path,char override,FILE*out)
905     {
906         FILE*f ;
907         char t[MAX_STRING_LEN];
908         char d[MAX_STRING_LEN];
909         int x ;
910
911         strcpy(d,path);
912         make_full_path(d,access_name,t);
913
914         if((f=fopen(t,"r")))
915         {
```

```
916                 parse_access_dir(f,-1,override,d,t,out);
917                 fclose(f);
918             }
919         }
920          /* 获取目录存取控制配置信息，从 access.conf 文件获取 */
921         void process_access_config(FILE*errors)
922         {
923             FILE*f ;
924             char l[MAX_STRING_LEN];
925             char w[MAX_STRING_LEN];
926             int n ;
927
928             num_sec=0 ;
929             n=0 ;
930             if(!(f=fopen(access_confname,"r")))                 // 打开文件
931             {
932                 fprintf(errors,"httpd: could not open access configuration file %s.\n",
933                     access_confname);
934                 perror("fopen");
935                 exit(1);
936             }
937             while(!(cfg_getline(l,MAX_STRING_LEN,f)))  // 读取一行配置文件
938             {
939                 ++n ;
940                 if((l[0]=='#')||(!l[0]))continue ;      // 如果是#开头或者空行则继续
941                 cfg_getword(w,l);
                     // 正常的配置类似于<Directory /home/devel/apache_0.6.5/htdocs>
942                 if(strcasecmp(w,"<Directory"))
943                 {
944                     fprintf(errors,
945                         "Syntax error on line %d of access config. file.\n",n);
946                     fprintf(errors,"Unknown directive %s.\n",w);
947                     exit(1);
948                 }
949                 getword(w,l,'>'); // w=/home/devel/apache_0.6.5/htdocs; l = "
                     // 调用前 n 是说明在第几行开始出现配置信息
950                 n=parse_access_dir(f,n,OR_ALL,w,NULL,errors);
951             }
952             fclose(f);
953         }
954
          /*
          * 根据用户名 user 获取密码并赋值给 pw，如果成功返回 1，否则返回 0
          * 其中密码是根据 htpasswd 生成的加密串，如 tsingien:SreeIymMXXp0U
```

```
 */
int get_pw(char*user,char*pw,FILE*errors)
{
    FILE*f ;
    char errstr[MAX_STRING_LEN];
    char l[MAX_STRING_LEN];
    char w[MAX_STRING_LEN];
     // 打开密码文件
    if(!(f=fopen(auth_pwfile,"r")))
    {
        sprintf(errstr,"Could not open user file %s",auth_pwfile);
        die(SERVER_ERROR,errstr,errors);
    }
    while(!(cfg_getline(l,MAX_STRING_LEN,f)))
    {
        if((l[0]=='#')||(!l[0]))continue ; // 这也有注释……
        getword(w,l,':');

        if(!strcmp(user,w)) // 文件中定义的用户名和从客户端传递过来的一致
        {
            strcpy(pw,l);
            fclose(f);
            return 1 ;
        }
    }
    fclose(f);
    return 0 ;
}
 // 组认证访问使用，参考第 10 章“存取控制”
struct ge
{
    char*name ;         // 组名
    char*members;       // 组中用户，以空格区分
    struct ge*next ;    // 下一条记录
}
;

static struct ge*grps ;
 /* 读取本地组认证配置文件，填充 grps 链表。 */
int init_group(char*grpfile,FILE*out)
{
    FILE*f ;
    struct ge*p ;
    char l[MAX_STRING_LEN],w[MAX_STRING_LEN];
```

```
998             // 打开文件
999             if(!(f=fopen(grpfile,"r")))
1000                return 0 ;
1001
1002            grps=NULL ;
1003            while(!(cfg_getline(l,MAX_STRING_LEN,f))) // 读取一行配置
1004            {
1005                if((l[0]=='#')||(!l[0]))continue ;       // 空行或注释
1006                getword(w,l,':');                       // 组名:用户名 用户名 ……
1007                if(!(p=(struct ge*)malloc(sizeof(struct ge))))
1008                    die(NO_MEMORY,"init_group",out);
1009                if(!(p->name=strdup(w)))
1010                    die(NO_MEMORY,"init_group",out);
1011                if(!(p->members=strdup(l)))
1012                    die(NO_MEMORY,"init_group",out);
1013                p->next=grps ;
1014                grps=p ;
1015            }
1016            fclose(f);
1017            return 1 ;
1018        }
1019         /* 判断某个用户是否在某个组里面 */
1020        int in_group(char*user,char*group)
1021        {
1022            struct ge*p=grps ;
1023            char l[MAX_STRING_LEN],w[MAX_STRING_LEN];
1024
1025            while(p)
1026            {
1027                if(!strcmp(p->name,group))   // 组名相同
1028                {
1029                    strcpy(l,p->members);
1030                    while(l[0])                 // 该组的每个用户名
1031                    {
1032                        getword(w,l,' ');
1033                        if(!strcmp(w,user))
1034                            return 1 ;
1035                    }
1036                }
1037                p=p->next ;
1038            }
1039            return 0 ;
1040        }
1041         /* 释放 grps 链表 */
```

```
1042    void kill_group()
1043    {
1044        struct ge*p=grps,*q ;
1045
1046        while(p)
1047        {
1048            free(p->name);
1049            free(p->members);
1050            q=p ;
1051            p=p->next ;
1052            free(q);
1053        }
1054    }
1055    /* 读取配置文件 */
1056    void read_config(FILE*errors)
1057    {
            // 清空别名列表
1058        reset_aliases();
            // 获取服务器配置信息
1059        process_server_config(errors);
            // 初始化 mine 指针数组。从 conf/mime.types 文件获取配置信息
1060        init_mime(errors);
            // 清空列表目录中使用的链表
1061        init_indexing();
            // 配置资源设置，如用户目录/错误文档等，从 srm.conf 目录获取的配置信息
1062        process_resource_config(errors);
            // 获取目录存取控制配置信息，从 access.conf 文件获取
1063        process_access_config(errors);
1064    }
```

11.3 小结

一个成熟的、功能强大的系统，通常拥有大量的可配置选项供用户进行个性化设置。Apache 也不例外。

这些可配置选项涉及到了 Apache 的各个功能模块，了解功能模块的功能和配置选项的含义是理解本章代码的前提。

第 12 章 HTTP方法

12.1 概述

成书时 HTTP 最新版本是 1.1，广泛应用的是 1.0 版本。而 Apache0.6.5 可处理的 HTTP 版本包括 0.9 和 1.0。

其中 0.9 版本目前已经极少使用，它只接受一种 GET 方法，没有在通讯中指定版本号，不支持请求头。不支持 POST 方法，客户端无法向服务器传递太多信息。

目前广泛使用的是 1.0 版本。支持以下 5 种方法：

- GET
  向特定的资源发出请求。
- HEAD
  客户端向服务器索要与 GET 请求相一致的响应，只不过响应体将不会被返回。
- POST
  向指定资源提交数据进行处理请求（例如提交表单或上传文件）。数据被包含在请求体中。
- PUT
  向指定资源位置上传其最新内容。
- DELETE
  删除指定资源。

12.2 代码注释

本章代码包括 http_request.c、http_put.c、http_get.c、http_post.c、http_delete.c。其中 http_request.c 包含了处理请求的主流程。在里面根据请求内容和方法的不同调用其它几个源文件中相应的函数实现对用户请求的响应。

12.2.1 http_request.c

```
1   #include "httpd.h"
2
3   int assbackwards ;          // 区分 HTTP 版本 assbackwards ? "HTTP/0.9" : "HTTP/1.0"
4   char*remote_host ;
5   char*remote_ip ;
6   char*remote_name ;
7
8   static void(*exit_callback)();
9    /* 超时处理:记录日志，关闭输入输出，退出程序 */
10  void send_fd_timed_out()
11  {
12        char errstr[MAX_STRING_LEN];
```

```
13
14          if(exit_callback)(*exit_callback)();
15          sprintf(errstr,"httpd: send timed out for %s",remote_name);
16          log_error(errstr);
17          log_transaction(1);
18          fclose(stdin);
19          fclose(stdout);
20          exit(0);
21      }
22       /* 将 f 中的内容发送到 fd 中。返回发送的字节数 */
23      long send_fd(FILE*f,FILE*fd,void(*onexit)())
24      {
25          char buf[IOBUFSIZE];
26          long total_bytes_sent ;
27
28          register int n,o,w ;
29           /* 几个回调函数 */
30          exit_callback=onexit ;
31          signal(SIGALRM,(void(*)())send_fd_timed_out);
32          signal(SIGPIPE,(void(*)())send_fd_timed_out);
33
34          total_bytes_sent=0 ;
35          while(1)
36          {
37              alarm(timeout);
38              if((n=fread(buf,sizeof(char),IOBUFSIZE,f))<1) // 从 f 中读取数据
39              {
40                  break ;
41              }
42              o=0 ;
43              if(bytes_sent!=-1)
44                  bytes_sent+=n ;
45              while(n)
46              {
47                  w=fwrite(&buf[o],sizeof(char),n,fd);          // 写到 fd 中
48                  n-=w ;
49                  o+=w ;
50                  total_bytes_sent+=w ;
51              }
52          }
53          fflush(fd);                  // 刷新 fd
54          return total_bytes_sent ; // 返回发送的字节数
55      }
56       /* 找到并执行脚本，返回客户端执行的结果 */
```

```
57  int find_script(char*req,char*name,char*args,int in,FILE*out)
58  {
59      struct stat finfo ;
60      char pa[MAX_STRING_LEN];
61      char ct_backup[MAX_STRING_LEN];
62
63      get_path_info(name,pa,&finfo);
          // 查看存取权限
64      evaluate_access(name,&finfo,M_GET,&allow,&allow_options,out);
65      if(!allow)
66      {
67          log_reason("client denied by server configuration",name);
68          unmunge_name(name);
69          die(FORBIDDEN,name,out);
70      }
71      strcpy(ct_backup,content_type);
          // 检查文件的 content_type
72      probe_content_type(name);
          // 如果是 content_type 是 application/x-httpd-cgi
73      if(!strcmp(content_type,CGI_MAGIC_TYPE))
74      {
75          strcpy(content_type,ct_backup);
              // 发送 cgi 执行结果到客户端，参看 11.2.3
76          send_cgi(req,name,pa,args,&finfo,in,out);
77          return 1 ;
78      }
79      return 0 ;
80  }
81
82  char log_user[MAX_STRING_LEN];
83   /* 处理客户端请求，调用方式:process_request(0,stdout) */
84  void process_request(int in,FILE*out)
85  {
86      char m[HUGE_STRING_LEN];
87      char w[HUGE_STRING_LEN];
88      char l[HUGE_STRING_LEN];
89      char url[HUGE_STRING_LEN];
90      char args[HUGE_STRING_LEN];
91      int s,n ;
92
93      log_user[0]='\0' ;
94       // 初始化 4-6 行的 remote_*参数
95      get_remote_host(in);
96      exit_callback=NULL ;
97
```

```
98          signal(SIGPIPE,(void(*)())send_fd_timed_out);
99           // 准备使用 STREAM 结构实例
100         getlineinit(in);
101         l[0]='\0' ;
             // 从客户端获取一行数据，存入 L 中
102         if(getline(l,HUGE_STRING_LEN,in,timeout))
103              return ;
104         if(!l[0])
105              return ;
106          // 记录请求
107         record_request(l);
             /*****************************************************
             * 一个请求实例：
             * GET /index.php?password=1234&name=5678 HTTP/1.0
             * Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
             * application/x-shockwave-flash, application/vnd.ms-excel,
             * application/vnd.ms-powerpoint, application/msword, * /*
             * Accept-Language: zh-cn
             * Accept-Encoding: gzip, deflate
             * User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
             * .NET CLR 1.1.4322)
             * Host: 127.0.0.1
             * ****************************************************/
             // m=GET;l=/index.php?password=1234&name=5678 HTTP/1.0
108         getword(m,l,' ');
             // args=/index.php?password=1234&name=5678;l=HTTP/1.0
109         getword(args,l,' ');
             // url=/index.php;args=password=1234&name=5678
110         getword(url,args,'?');
             // 解码 url。如果含有%20 这样的字符
111         unescape_url(url);
             /*
             * Apache 自定义错误响应。每次新的请求到来的时候我们需要记录
             * url 和 args 参数，在发生错误或问题后重定向时会使用到这些信息。
             * 我们把 original_url[]的第一个字节当作标志使用。
             * original_url[0] == '\0'意味着我们还没有重定向
             */
112         original_url[0]='\0' ;
             /*
             * 在失败的时候保存一个原始请求 URL 的副本。我们可以在自定义错误日志
             * 里面使用这个 URL
             */
113         strcpy(&original_url[1],url);
114         strcpy(&original_args[0],args);
```

```
115         original_status=0 ;                 // 在遇到错误或问题的时候修改这个值
116         strcpy(original_method,m);          // 同样，保存请求方法
117
118         getword(w,l,'\0');
119         init_header_vars();                 // 初始化返回给客户端 header 中的信息
120         if(w[0]!='\0')                      // HTTPv0.9 之后的版本
121         {
122             assbackwards=0 ;
123             get_mime_headers(in,out);       // 处理请求头里面的 MIME 信息
124         }
             /*
             * HTTPV0.9 只接受 GET 一种请求方法，没有在通讯中指定版本号，且不支
             * 持请求头
             */
125         else
126             assbackwards=1 ;
127
128         if(!strcmp(m,"HEAD"))               // 客户端的请求方法是 HEAD
129         {
130             header_only=1 ;
131             process_get(in,out,m,url,args);
132         }
133         else if(!strcmp(m,"GET"))           // 客户端的请求方法是 GET
134         {
135             header_only=0 ;
136             process_get(in,out,m,url,args);
137         }
138         else if(!strcmp(m,"POST"))          // 客户端的请求方法是 POST
139         {
140             header_only=0 ;
141             post_node(url,args,in,out);
142         }
143         else if(!strcmp(m,"PUT"))           // 客户端的请求方法是 PUT
144         {
145             header_only=0 ;
146             put_node(url,args,in,out);
147         }
148         else if(!strcmp(m,"DELETE"))        // 客户端的请求方法是 DELETE
149         {
150             header_only=0 ;
151             delete_node(url,args,in,out);
152         }
153         else                                // 不支持的方法
154             die(BAD_REQUEST,"Invalid or unsupported method.",out);
```

```
155
156     }
```

12.2.2 http_put.c

```
1     #include "httpd.h"
2      /* 处理客户端的 PUT 请求 */
3     void put_node(char*name,char*args,int in,FILE*out)
4     {
5           int s ;
6            // 查看文档的类型（STD_DOCUMENT?）并将给定的 URL 转换成绝对路径
7           s=translate_name(name,out);
8
9           switch(s)
10          {
11          case STD_DOCUMENT :     // 普通文本
12                if(find_script("PUT",name,args,in,out))
13                      return ;
14                else
15                      die(NOT_IMPLEMENTED,"PUT to non-script",out);
16          case REDIRECT_URL :       // 重定向
17                die(REDIRECT,name,out);
18          case SCRIPT_CGI :         // 执行脚本
19                exec_cgi_script("PUT",name,args,in,out);
20                break ;
21          case BAD_URL :            // 错误的请求
22                die(BAD_REQUEST,name,out);
23          default :                 // 其它类型
24                die(NOT_IMPLEMENTED,"Apache script exeuction of delete",out);
25          }
26    }
```

12.2.3 http_get.c

```
1     #include "httpd.h"
2
3     int header_only ;
4     int num_includes ;
5     int allow ;
6     char allow_options ;
7      /* 发送请求的文件给客户端 */
8     void send_file(char*file,FILE*fd,struct stat*fi,
9                         char*path_args,char*args,int in)
```

```
10    {
11        FILE*f ;
12         // 设置 content_type
13        set_content_type(file);
14         // 是 cgi 文件
15        if(!strcmp(content_type,CGI_MAGIC_TYPE))
16        {
17            send_cgi("GET",file,path_args,args,fi,in,fd);
18            return ;
19        }
           // 允许服务器端包含且没有 content_encoding 信息
20        if((allow_options&OPT_INCLUDES)&&(!content_encoding[0]))
21        {
22            if(!strcmp(content_type,INCLUDES_MAGIC_TYPE)
23                ||!strcmp(content_type,INCLUDES_MAGIC_TYPE3)) // shtml
24            {
25                status=200 ;
26                bytes_sent=0 ;
                  // 发送解析过的 SSI 内容到客户端
27                send_parsed_file(file,fd,path_args,args,
                                  allow_options&OPT_INCNOEXEC);
                  // 增加一条访问日志且关闭访问日志文件
28                log_transaction(1);
29                return ;
30            }
31        }
32        if(!(f=fopen(file,"r"))) // 普通文件，打开失败
33        {
34            log_reason("file permissions deny server access",file);
35            unmunge_name(file);
36            die(FORBIDDEN,file,fd); // 我们已经确认文件是存在的
37        }
38        status=200 ;
39        bytes_sent=0 ;
40         // HTTP/1.0 且 content_type 不是 httpd/send-as-is
41        if(!assbackwards&&strcmp(content_type,ASIS_MAGIC_TYPE))
42        {
43            set_content_length(fi->st_size);          // 设置 content_length
44            set_last_modified(fi->st_mtime,fd);       // 设置 last_modified
45            send_http_header(fd);                     // 设置 header 内容
46        }
47        num_includes=0 ;
48
49        if(!strcmp(content_type,ASIS_MAGIC_TYPE)) // content_type 是 httpd/send-as-is
```

```
50          {
51              scan_script_header(f,fd,file);
52              if(!assbackwards)              // HTTP/1.0，需要发送 header
53                  send_http_header(fd);
54          }
55          if(!header_only)                   // 不仅仅发送头信息
56              send_fd(f,fd,NULL);            // 发送文件内容
57          log_transaction(1);                // 记录日志
58          fclose(f);                         // 关闭文件
59      }
60       /* 发送执行 cgi 结果到客户端 */
61      void send_cgi(char*method,char*file,char*path_args,
62                          char*args,struct stat*finfo,int in,FILE*fd)
63      {
64          char**env ;
65          int m ;
66           /* 确定请求的方法 */
67          if((!strcmp(method,"GET"))||(!strcmp(method,"HEAD")))
68              m=M_GET ;
69          else if(!strcmp(method,"POST"))
70              m=M_POST ;
71          else if(!strcmp(method,"PUT"))
72              m=M_PUT ;
73          else if(!strcmp(method,"DELETE"))
74              m=M_DELETE ;
75           // 检查存取权限
76          evaluate_access(file,finfo,m,&allow,&allow_options,fd);
             // 不允许访问或不允许执行 CGI
77          if((!allow)||(!(allow_options&OPT_EXECCGI)))
78          {
79              log_reason("client denied by server configuration",file);
80              unmunge_name(file);
81              die(FORBIDDEN,file,fd);
82          }
             // 添加常规环境变量
83          if(!(env=add_common_vars(in_headers_env,fd)))
84              die(NO_MEMORY,"send_cgi",fd);
85          bytes_sent=0 ;
             // 执行 CGI，如果是重定向，给客户端发送重定向信息
86          if(cgi_stub(method,file,path_args,args,env,finfo,in,fd)==REDIRECT_URL)
87              die(REDIRECT,location,fd);
             // 添加访问记录
88          log_transaction(0);
89      }
```

```
90      /* 发送 HTML 文件给客户端 */
91    void send_node(char*file,char*args,int in,FILE*fd)
92    {
93          struct stat finfo ;
94          char pa[MAX_STRING_LEN],*name_ptr,*end_ptr,
                temp_name[HUGE_STRING_LEN];
95           // 获取 HTML 文件所在路径的信息
96          get_path_info(file,pa,&finfo);
             // 检查存取权限
97          evaluate_access(file,&finfo,M_GET,&allow,&allow_options,fd);
98          if(!allow)
99          {
100               log_reason("client denied by server configuration",file);
101               unmunge_name(file);
102               die(FORBIDDEN,file,fd);
103         }
             // 是个目录
104         if(S_ISDIR(finfo.st_mode))
105         {
106               char ifile[MAX_STRING_LEN];
107                // 并不是浏览根目录
108               if(pa[0]!='/')
109               {
110                     char url[MAX_STRING_LEN];
111                     strcpy_dir(ifile,file);
112                     unmunge_name(ifile);          // 转换成相对路径
113                     construct_url(url,ifile);     // 构建 URL
114                     escape_url(url);              // 编码 url 中出现的特殊字符
115                     die(REDIRECT,url,fd);
116               }
117
118               strncpy(temp_name,index_names,HUGE_STRING_LEN-1);
119               end_ptr=name_ptr=temp_name ;
120
121               while(*name_ptr)                  // 定义了主页文件
122               {
                     // 跳过前置空格
123                     while(*name_ptr&&isspace(*name_ptr))++name_ptr ;
124                     end_ptr=name_ptr ;
125                     if(strchr(end_ptr,' '))       // 多个主页文件，取其中一个
126                     {
127                           end_ptr=strchr(name_ptr,' ');
128                           *end_ptr='\0' ;
129                           end_ptr++;
```

```
130                 }
131                 else
132                     end_ptr+=strlen(end_ptr);
133
134                 make_full_path(file,name_ptr,ifile);     // 主页文件位置
135                 if(stat(ifile,&finfo)==-1)               // 发生错误,没找到主页文件
136                 {
137                     // 如果设置了列表文件夹
138                     if(!*end_ptr&&(allow_options&OPT_INDEXES))
139                     {
140                         index_directory(file,fd);
141                         return ;
142                     }
                        // 没找到主页文件并且不允许列表文件夹，并且没有定义其它
                        // 的主页文件。给客户端发送 FORBIDDEN（403）错误
143                     else if(!*end_ptr)
144                     {
145                         log_reason("file permissions deny server access",file);
146                         unmunge_name(file);
147                         die(FORBIDDEN,file,fd);
148                     }
149                 }
150                 else
151                 {
152                     send_file(ifile,fd,&finfo,pa,args,in);// 发送找到的主页文件
153                     return ;
154                 }
155                 name_ptr=end_ptr ;
156             }
157         }
            // 请求内容不是一个目录
158         if(S_ISREG(finfo.st_mode))
159             send_file(file,fd,&finfo,pa,args,in);          // 发送请求的文件到客户端
            // 非常规文件类型
160         else
161         {
162             log_reason("improper file type",file);
163             unmunge_name(file);
164             die(FORBIDDEN,file,fd);
165         }
166     }
167
```

```
    /*
```

```
 * 处理 GET/HEAD 请求
 * 参数类似于：
 *   in=stdin
 *   out=stdout
 *   m=GET
 *   url=/index.php
 *   args=password=1234&name=5678
 */
168  void process_get(int in,FILE*out,char*m,char*url,char*args)
169  {
170      int s ;
171       // HTTP/0.9 不支持 HEAD 方法
172      if(assbackwards&&header_only)
173      {
174          header_only=0 ;
175          die(BAD_REQUEST,"Invalid HTTP/0.9 method.",out);
176      }
          // 查看文档的类型（STD_DOCUMENT?）并将给定的 URL 转换成绝对路径
177      s=translate_name(url,out);
178      switch(s)
179      {
180      case STD_DOCUMENT :    // 普通文本
181          send_node(url,args,in,out);
182          return ;
183      case REDIRECT_URL :    // 重定向
184          die(REDIRECT,url,out);
185      case SCRIPT_CGI :      // 执行脚本
186          exec_cgi_script(m,url,args,in,out);
187          return ;
188      case BAD_URL :         // 错误的请求
189          die(BAD_REQUEST,url,out);
190      }
191  }
```

12.2.4 http_post.c

```
1   #include "httpd.h"
2    /* 处理客户端的 POST 请求 */
3   void post_node(char*name,char*args,int in,FILE*out)
4   {
5       int s ;
6        // 查看文档的类型，并将给定的 URL 转换成绝对路径
7       s=translate_name(name,out);
8
```

```
9           switch(s)
10          {
11          case STD_DOCUMENT :     // 普通文本
12              if(find_script("POST",name,args,in,out))
13                  return ;
14              else
15                  die(NOT_IMPLEMENTED,"POST to non-script",out);
16          case REDIRECT_URL :     // 重定向
17              die(REDIRECT,name,out);
18          case SCRIPT_CGI :       // 执行脚本
19              exec_cgi_script("POST",name,args,in,out);
20              break ;
21
22          case BAD_URL :          // 错误的请求
23              die(BAD_REQUEST,name,out);
24          }
25      }
```

12.2.5 http_delete.c

```
1   #include "httpd.h"
2    /* 处理客户端的 DELETE 请求 */
3   void delete_node(char*name,char*args,int in,FILE*out)
4   {
5           struct stat finfo ;
6           int s ;
7            // 查看文档的类型，并将给定的 URL 转换成绝对路径
8           s=translate_name(name,out);
9
10          switch(s)
11          {
12          case STD_DOCUMENT :     // 普通文本
13              if(find_script("DELETE",name,args,in,out))
14                  return ;
15              else
16                  die(NOT_IMPLEMENTED,"DELETE to non-script",out);
17          case REDIRECT_URL :     // 重定向
18              die(REDIRECT,name,out);
19          case SCRIPT_CGI :       // 执行脚本
20              exec_cgi_script("DELETE",name,args,in,out);
21              break ;
22          case BAD_URL :          // 错误的请求
23              die(BAD_REQUEST,name,out);
24          default :               // 其它类型
```

```
25                  die(NOT_IMPLEMENTED,"Apache script exeuction of delete",out);
26              }
27      }
```

12.3 小结

从本章描述的几个文件中我们可以看到 Apache 对客户端请求的处理流程，具体的处理过程在前面的几个章节里面已经有了详细的介绍，您现在可以跟着每一个函数调用深入跟踪，一探每个请求的处理过程了。

至此，我们对 Apache 源代码的介绍已经接近尾声，剩下的就是一个声明全局变量和全局函数的 httpd.h 文件了，我们将在下章进行介绍。

第 13 章 头文件

13.1 概述

本章描述的文件 httpd.h 里面包含了大部分函数的声明以及头文件的引用，另外还有一些基本配置，如果配置文件里面没有定义这些配置信息，就使用 httpd.h 中定义的信息。

13.2 代码注释

```
1   #include <crypt.h>
2   #include <sys/types.h>
3
4   #include <dirent.h>
5   #define DIR_TYPE dirent
6    /* ----------------------------  配置目录  ------------------------------ */
     // Apache 安装目录，其它目录基本上以本目录为起点
7   #define HTTPD_ROOT "/home/devel/apache_0.6.5"
8    // Web 服务（站点）的起始目录
9   #define DOCUMENT_LOCATION "/home/devel/apache_0.6.5/htdocs"
10   // 安全设置的最大条数，存取控制部分使用
11  #define MAX_SECURITY 50
12   // 管理员 email 地址
13  #define DEFAULT_ADMIN "[no address given]"
14   // 服务默认开启端口
15  #define DEFAULT_PORT 80
16   // 运行服务的默认用户名和组名
17  #define DEFAULT_USER "#-1"
18  #define DEFAULT_GROUP "#-1"
19   // 日志文件
20  #define DEFAULT_XFERLOG "logs/access_log"
21  #define DEFAULT_ERRORLOG "logs/error_log"
22  #define DEFAULT_PIDLOG "logs/httpd.pid"
23   // 每个目录默认页
24  #define DEFAULT_INDEX "index.html"
25   // 是否列表目录，如果要列表目录，设置成 1
26  #define DEFAULT_INDEXING 0
27   // 无法获取 content_type 时返回的 content_type
28  #define DEFAULT_TYPE "text/html"
29   // 每个目录默认的存取控制文件
30  #define DEFAULT_ACCESS_FNAME ".htaccess"
31   // 服务器全局配置文件
```

```
32  #define SERVER_CONFIG_FILE "conf/httpd.conf"
33   // 文档（资源）配置文件
34  #define RESOURCE_CONFIG_FILE "conf/srm.conf"
35   // MIME 类型配置文件
36  #define TYPES_CONFIG_FILE "conf/mime.types"
37   // 存取控制配置文件
38  #define ACCESS_CONFIG_FILE "conf/access.conf"
39   // 用户 home 目录中的站点默认路径，参看第 5 章
40  #define DEFAULT_USER_DIR "public_html"
41   // CGI 脚本中用到的默认 PATH 环境变量
42  #define DEFAULT_PATH "/bin:/usr/bin:/usr/ucb:/usr/bsd:/usr/local/bin"
43   // Bourne shell 的位置
44  #define SHELL_PATH "/bin/sh"
45   // 默认字符串长度
46  #define MAX_STRING_LEN HUGE_STRING_LEN
47  #define HUGE_STRING_LEN 8192
48   // 等待消息的默认超时时间
49  #define DEFAULT_TIMEOUT 1200
50   // 服务器内部读写缓冲大小
51  #define IOBUFSIZE 8192
52   // 从客户端接收 header 时最多接收行数
53  #define MAX_HEADERS 200
54   // RFC1123 格式时间。也是 HTTP/1.0 需要的时间格式
55  #define HTTP_TIME_FORMAT "%a, %d %b %Y %T GMT"
56   /* ------------------------------ 错误类型 ------------------------------ */
     /* 服务器信息 */
57  #define SERVER_VERSION "Apache/0.6.5"
58  #define SERVER_PROTOCOL "HTTP/1.0"
59  #define SERVER_SUPPORT http://www.apache.org/
60
61   /* 状态码定义，详细信息参看第 4 章 */
62  #define DOCUMENT_FOLLOWS 200
63  #define REDIRECT 302
64  #define USE_LOCAL_COPY 304
65  #define BAD_REQUEST 400
66  #define AUTH_REQUIRED 401
67  #define FORBIDDEN 403
68  #define NOT_FOUND 404
69  #define SERVER_ERROR 500
70  #define NOT_IMPLEMENTED 501
71  #define SERVICE_UNAVAILABLE 503
72  #define NO_MEMORY 6992
73  #define RESPONSE_CODE_LIST " 200 302 304 400 401 403 404 500 503 501 "
74  #define RESPONSE_CODES 10
```

```
75      /* 按照以下 4 种方法处理客户端请求 */
76      #define METHODS 4
77      #define M_GET 0
78      #define M_PUT 1
79      #define M_POST 2
80      #define M_DELETE 3
81
82      /* 客户端请求对象的类型 */
83      #define REDIRECT_URL -1
84      #define STD_DOCUMENT 0
85      #define SCRIPT_CGI 1
86      #define BAD_URL 2
87      /* option 指令的可能值，参看第 10 章 */
88      #define OPT_NONE 0
89      #define OPT_INDEXES 1
90      #define OPT_INCLUDES 2
91      #define OPT_SYM_LINKS 4
92      #define OPT_EXECCGI 8
93      #define OPT_UNSET 16
94      #define OPT_INCNOEXEC 32
95      #define OPT_SYM_OWNER 64
96      #define OPT_ALL
                (OPT_INDEXES|OPT_INCLUDES|OPT_SYM_LINKS|OPT_EXECCGI)
97      /* AllowOverride 属性值 */
98      #define OR_NONE 0
99      #define OR_LIMIT 1
100     #define OR_OPTIONS 2
101     #define OR_FILEINFO 4
102     #define OR_AUTHCFG 8
103     #define OR_INDEXES 16
104     #define OR_ALL
                (OR_LIMIT|OR_OPTIONS|OR_FILEINFO|OR_AUTHCFG|OR_INDEXES)
105     /* 几种特殊的 content_type */
106     #define CGI_MAGIC_TYPE "application/x-httpd-cgi"
107     #define INCLUDES_MAGIC_TYPE "text/x-server-parsed-html"
108     #define INCLUDES_MAGIC_TYPE3 "text/x-server-parsed-html3"
109     #define MAP_FILE_MAGIC_TYPE "application/x-type-map"
110     #define ASIS_MAGIC_TYPE "httpd/send-as-is"
111     /* 列表目录时使用 */
112     #define BY_PATH 0
113     #define BY_TYPE 1
114     #define BY_ENCODING 2
115
116     #define FANCY_INDEXING 1
```

```
117     #define ICONS_ARE_LINKS 2
118     #define SCAN_HTML_TITLES 4
119     #define SUPPRESS_LAST_MOD 8
120     #define SUPPRESS_SIZE 16
121     #define SUPPRESS_DESC 32
122      /* 常用头文件 */
123     #include <stdio.h>
124     #include <stdlib.h>
125     #include <string.h>
126     #include <sys/stat.h>
127     #include <sys/file.h>
128     #include <sys/socket.h>
129     #include <ctype.h>
130     #include <netinet/in.h>
131     #include <netdb.h>
132     #include <sys/ioctl.h>
133     #include <arpa/inet.h>
134     #include <time.h>
135     #include <signal.h>
136     #include <errno.h>
137     #include <sys/wait.h>
138     #include <pwd.h>
139     #include <grp.h>
140     #include <fcntl.h>
141     #include <limits.h>
142
143     #ifndef LOGNAME_MAX
144     #define LOGNAME_MAX 25
145     #endif
146
147     #include <unistd.h>
148      // \n
149     #define LF 10
150      // \r
151     #define CR 13
152
153      /* 存取控制 */
154     #define DENY_THEN_ALLOW 0
155     #define ALLOW_THEN_DENY 1
156     #define MUTUAL_FAILURE 2
157      /* 存取控制和认证，参看第 10 章 */
158     typedef struct
159     {
160          char*d ;
```

```
161         char opts ;
162         char override ;
163         int order[METHODS];
164         int num_allow[METHODS];
165         char*allow[METHODS][MAX_SECURITY];
166         int num_auth[METHODS];
167         char*auth[METHODS][MAX_SECURITY];
168
169         char*auth_type ;
170         char*auth_name ;
171
172         char*auth_pwfile ;
173         char*auth_grpfile ;
174
175         int num_deny[METHODS];
176         char*deny[METHODS][MAX_SECURITY];
177     }
178     security_data ;
179      /* ----------------------------- 全局变量 ----------------------------- */
       /* 服务器相关 */
180     extern int port ;
181     extern uid_t user_id ;
182     extern char user_name[LOGNAME_MAX+12];
183     extern gid_t group_id ;
184     extern char server_root[MAX_STRING_LEN];
185     extern char error_fname[MAX_STRING_LEN];
186     extern char xfer_fname[MAX_STRING_LEN];
187     extern char pid_fname[MAX_STRING_LEN];
188     extern char server_admin[MAX_STRING_LEN];
189     extern char*server_hostname ;
190     extern char server_confname[MAX_STRING_LEN];
191     extern char srm_confname[MAX_STRING_LEN];
192     extern char access_confname[MAX_STRING_LEN];
193     extern char types_confname[MAX_STRING_LEN];
194     extern int timeout ;
195      /* 文档相关 */
196     extern char user_dir[MAX_STRING_LEN];
197     extern char index_names[HUGE_STRING_LEN];
198     extern char access_name[MAX_STRING_LEN];
199     extern char document_root[MAX_STRING_LEN];
200     extern char default_type[MAX_STRING_LEN];
201     extern char default_icon[MAX_STRING_LEN];
202     extern char blank_icon[MAX_STRING_LEN];
203     extern int fancy_indexing ;
```

```
204     extern char readme_fname[MAX_STRING_LEN];
205      /* 存取控制相关 */
206     extern int num_sec ;
207     extern security_data sec[MAX_SECURITY];
208      /* 认证相关 */
209     extern char*auth_type ;
210     extern char*auth_name ;
211
212     extern char*auth_pwfile ;
213     extern char*auth_grpfile ;
214     extern char user[MAX_STRING_LEN];
215      /* 请求相关信息 */
216     extern int assbackwards ;
217     extern int header_only ;
218     extern char*remote_host ;
219     extern char*remote_ip ;
220     extern char*remote_name ;
221     extern int allow ;
222     extern char allow_options ;
223     extern int num_includes ;
224     extern int dirs_in_alias ;
225      /* MIME */
226     extern char auth_line[MAX_STRING_LEN];
227     extern int content_length ;
228     extern char content_type[MAX_STRING_LEN];
229     extern char content_encoding[MAX_STRING_LEN];
230     extern char location[MAX_STRING_LEN];
231     extern char**in_headers_env ;
232     extern char*out_headers ;
233     extern char*status_line ;
234      /* http_log */
235     extern FILE*error_log ;
236     extern int bytes_sent ;
237     extern int status ;
238
239     extern char original_url[HUGE_STRING_LEN];
240     extern char original_args[HUGE_STRING_LEN];
241     extern int original_status ;
242     extern char original_method[20];
243     extern char the_request[HUGE_STRING_LEN];
244      /* ---------- 函数原型(具体含义及实现参考相关章节) ---------- */
       /* httpd.c */
245     void htexit(int status,FILE*out);
246      /* http_config */
```

```
247     void read_config();
248     void parse_htaccess(char*dir,char override,FILE*out);
249     int get_pw(char*user,char*pw,FILE*errors);
250     int in_group(char*user,char*group);
251     int init_group(char*grpfile,FILE*out);
252     void kill_group();
253     extern char*response_code_strings[RESPONSE_CODES+1];
254
255     /* http_alias */
256     void reset_aliases();
257     void dump_aliases();
258     int add_alias(char*f,char*r,int is_script);
259     int add_redirect(char*f,char*url);
260     int translate_name(char*name,FILE*fd);
261     void unmunge_name(char*name);
262
263     /* http_request */
264     void process_request(int in,FILE*out);
265     long send_fd(FILE*f,FILE*fd,void(*callback)());
266     void send_fd_timed_out();
267     int find_script(char*method,char*name,char*args,int in,FILE*out);
268
269     /* http_get */
270     void send_file(char*file,FILE*fd,struct stat*fi,
271                            char*path_args,char*args,int in);
272     void process_include(FILE*f,FILE*fd,char*incstring,char*args);
273     void send_node(char*name,char*args,int in,FILE*fd);
274     void send_cgi(char*method,char*file,char*path_args,char*args,
275                           struct stat*finfo,int in,FILE*fd);
276     void process_get(int in,FILE*out,char*m,char*url,char*args);
277
278     /* http_post */
279     void post_node(char*name,char*args,int in,FILE*out);
280
281     /* http_put */
282     void put_node(char*name,char*args,int in,FILE*out);
283
284     /* http_delete */
285     void delete_node(char*name,char*args,int in,FILE*out);
286
287     /* http_script */
288     void exec_cgi_script(char*method,char*path,char*args,int in,FILE*out);
289     int cgi_stub(char*method,char*path,char*path_args,char*args,
290                          char**env,struct stat*finfo,int in,FILE*out);
```

```
291     char**add_common_vars(char**env,FILE*out);
292     void get_path_info(char*path,char*path_args,
293                                     struct stat*finfo);
294
295     /* http_dir */
296     void index_directory(char*name,FILE*fd);
297     void add_icon(int type,char*icon,char*to,char*path,FILE*out);
298     void add_alt(int type,char*alt,char*to,char*path,FILE*out);
299     void add_desc(int type,char*desc,char*to,char*path,FILE*out);
300     void add_ignore(char*ext,char*path,FILE*out);
301     void add_header(char*name,char*path,FILE*out);
302     void add_readme(char*name,char*path,FILE*out);
303     void add_opts(char*optstr,char*path,FILE*out);
304     void add_opts_int(int opts,char*path,FILE*out);
305     void send_size(size_t size,FILE*fd);
306
307     void init_indexing();
308     void kill_indexing();
309
310     /* http_log */
311     void record_request(char*cmd_line);
312     void log_error(char*err);
313     void log_error_noclose(char*err);
314     void log_reason(char*reason,char*file);
315     void title_html(FILE*fd,char*msg);
316     void die(int type,char*err_string,FILE*fd);
317     void open_logs();
318     void close_logs();
319     void begin_http_header(FILE*fd,char*msg);
320     void error_log2stderr();
321     void log_pid(void);
322     void log_transaction(int closeit);
323
324     /* http_mime */
325     void get_mime_headers(int fd,FILE*out);
326     void init_header_vars();
327     void send_http_header(FILE*fd);
328     void set_content_type(char*fn);
329     void set_content_type_and_parms(char*file);
330     void set_last_modified(time_t t,FILE*out);
331     void probe_content_type(char*fn);
332     int scan_script_header(FILE*f,FILE*fd,char*path);
333     void force_header(char*hdr,FILE*fd);
334     void add_type(char*fn,char*t,FILE*out);
```

```
335     void add_encoding(char*fn,char*t,FILE*out);
336     void set_content_length(int l);
337     void dump_types();
338     void init_mime();
339     void kill_mime();
340     int is_content_type(char*type);
341     void dump_default_header(FILE*fd);
342
343     /* http_access */
344     void evaluate_access(char*path,struct stat*finfo,int m,int*allow,
345                                     char*op,FILE*out);
346     void kill_security();
347
348     /* http_auth */
349     void check_auth(security_data*s,int m,FILE*out);
350
351     /* http_include */
352     void send_parsed_file(char*file,FILE*fd,char*path_args,char*args,
353                                       int noexec);
354
355     /* util */
356     void chdir_file(char*file);
357     void http2cgi(char*w);
358     int later_than(struct tm*tms,char*i);
359     int strcmp_match(char*str,char*exp);
360     int is_matchexp(char*str);
361     void strsubfirst(int start,char*dest,char*src);
362     void make_full_path(char*src1,char*src2,char*dst);
363     int is_directory(char*name);
364     int is_url(char*u);
365     void getparents(char*name);
366     void no2slash(char*name);
367     uid_t uname2id(char*name);
368     gid_t gname2id(char*name);
369     void getlineinit(int fd);
370     int getline(char*s,int n,int f,unsigned int timeout);
371     int cfg_getline(char*s,int n,FILE*f);
372     void getword(char*word,char*line,char stop);
373     void cfg_getword(char*word,char*line);
374     void get_remote_host(int fd);
375     char*get_time();
376     char*gm_timestr_822(time_t t);
377     char*ht_time(time_t t,char*fmt,int gmt);
378     struct tm*get_gmtoff(long*tz);
```

```
379     void make_dirstr(char*s,int n,char*d);
380     int count_dirs(char*path);
381     void strcpy_dir(char*d,char*s);
382     void unescape_url(char*url);
383     void escape_url(char*url);
384     void escape_uri(char*url);
385     void escape_shell_cmd(char*cmd);
386     void plustospace(char*str);
387     void spacetoplus(char*str);
388     void str_tolower(char*str);
389     void uudecode(char*s,unsigned char*d,int dl);
390     char*make_env_str(char*n,char*v,FILE*out);
391     char**new_env(char**env,int to_add,int*pos);
392     void free_env(char**env);
393     int ind(const char*s,char c);
394     int rind(char*s,char c);
395     void construct_url(char*d,char*s);
396     void get_local_host();
397     int get_portnum(int sd,FILE*out);
398     int can_exec(struct stat*finfo);
399
400     int index_of_response(int err_no);
401     void prepare_for_redirect(FILE*fd,int orig_status,char*new_url);
```

13.3 小结

至此，整个 Apache0.6.5 的源代码我们已经完全介绍完毕了。但愿此时的您，可以发出一句感慨：原来 Apache 是这么处理客户端请求的。若此，吾愿足矣！

如果您有任何问题或建议，可以通过本书官网http://www.oldapache.org上介绍的联系方式和作者取得联系。