HZ BOOKS

Linux
与自由软件资源丛书

Apache Server

Apache Server Commentary

Greg Holden
(美) Nicholas Wells 著
Matthew Keller

朱珂 尹刚 姚熠 陈永盛 等译
章文嵩 审校

附 CD-ROM 1张

源代码分析

机械工业出版社
China Machine Press

COROLIS

目　录

第1章 访存控制模块

本章描述了采用不同方式来控制对文件、目录、位置和虚拟主机进行访问的六个模块。第一个模块是mod_access，它提供以IP地址、域名、子网掩码和环境变量为基础的访存控制，其他五个模块处理不同形式的用户授权。

Apache缺省的配置已经包含了mod_access和mod_auth模块。尽管Apache的文档说明了缺省条件下并不包括mod_auth_anon、mod_auth_db、mod_auth_dbm和mod_digest模块，但Red Hat Linux 6.0的Apache已经缺省包含这些模块，而对于其他的系统，如果想使用这些模块，则需要把这些模块重新编译进服务器，或者（对于Apache 1.3或以后的版本）把它们作为动态共享对象（Dynamic Shared Object, DSO）加到服务器中。

1.1 mod_access模块

缺省时，mod_access模块已编译进Apache，它提供通过Allow、Deny、Allow from env=、Deny from env=和Order指令来控制访存。这些指令可以放在任何包容器中（例如Directory、Files、Location或者VirtualHost），并且如果没有被覆盖，它们能够提供包容器所指定范围内的访存控制。

Allow指令允许特定的主机访存特定目录，而Deny指令拒绝某特定主机访存特定目录。这两条指令有相同的操作参数，操作参数All用于允许或拒绝所有的主机访存，而且可以指定完整域名（Fully Qualified Domain Name, FQDN）、部分域名、完整或部分IP地址等操作参数。

下面的这个例子，除了.somedomain.com域中的所有主机，其他主机均可进行访存。

```
Order allow, deny
Allow from all
Deny from .somedomain.com
```

Allow from env= 和Deny from env= 这两个指令根据指定的变量是否存在而允许或拒绝主机访存。

下面这个例子允许已授权的用户进行访存，因为变量REMOTE_USER是为授权用户设定的。

```
Order deny,allow
Deny from all
Allow from env = REMOTE_USER
```

Order这个指令用来控制Allow和Deny指令的次序，关键字allow、deny说明在Deny指令之前评估Allow指令，关键字deny、allow说明在Allow指令之前评估Deny指令，而关键字mutual-failure用来说明次序是不相关的。

1.1.1 模块结构

当系统调用mod_access模块时，首先执行check_dir_access例程(11733行)，这个例程先获

得特定目录配置信息（11679行），接着检查Order指令是否设定为allow, deny(11683行)或为deny, allow（11690行）；否则，此例程假定Order指令为mutual-failure（11696行），最后，这个例程检查Satisfy核心指令，以确定是否应当拒绝访存(11704行~11706行)。如果访问请求被拒绝，该例程会在错误日志文件中记录一条出错信息(11707行~11710行)。模块结构见图1-1。

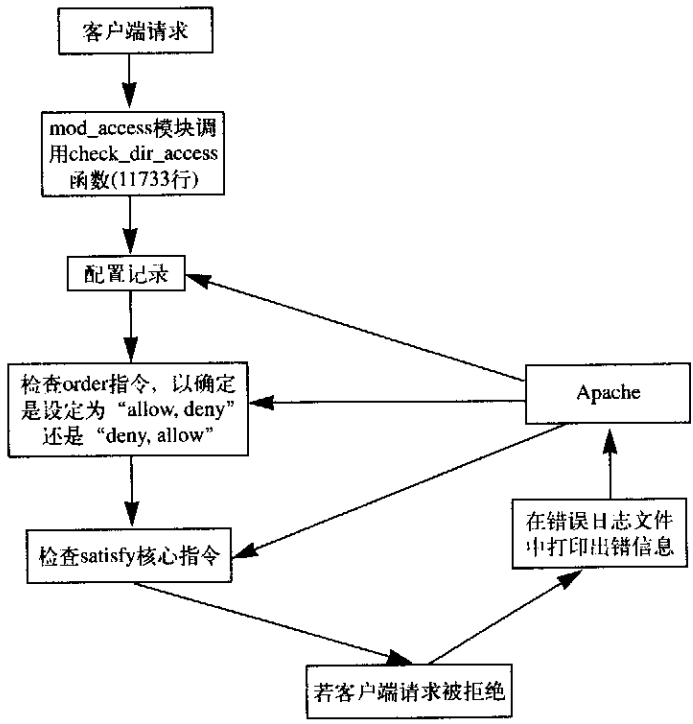

图1-1 mod_access模块限定了可访问的主机名或IP地址

1.1.2 定制

模块mod_access根据主机的IP信息控制访存资源，通过优化进程和扩大控制访存资源的范围，改变和增强对资源访问的控制能力：

- 通过以太网卡本身的固定物理地址对访问进行限制，增强安全性。
- 访问信息可以存放在一个文件或数据库中，以方便动态修改和查询，而不需要重新启动服务器。

1. mod_access

11371: allowdeny_type是一种枚举数据类型，它包含在allowdeny结构中，用于确定描述访存资源的数据类型。

11372: T_ENV标志符通过一个环境变量来设定allowdeny_type为允许或拒绝状态。

11373: T_ALL标志符用于设定allowdeny_type是允许或是拒绝所有主机。

11374: T_IP标志符根据IP地址、IP范围、IP子网或这几种形式的组合来设定allowdeny_type为允许或拒绝。

11375: T_HOST标志符根据主机名设定allow_type为允许或拒绝。

11376: 当配置文件有错误时（非法的IP地址或网络掩码），T_FAIL标志符设定allowdeny_type为失败状态。

11379: allowdeny结构存放描述访存资源的配置数据。

11391: 以下三行定义在配置文件中使用的Order指令的三种情况。

11396: access_dir_conf结构用于记录配置文件中mod_access模块使用的所有信息（包括Allow、Deny和Order指令，以及环境变量）。

11402: 此行定义Apache可以使用的模块。

2. create_access_dir_config

11404: create_access_dir_config例程是mod_access模块的初始化例程，此例程主要是创建mod_access模块使用的配置数据结构，函数返回一个access_dir_conf对象。

11407: 变量conf是一个类型为access_dir_conf的结构指针。

11412: 这个for循环用于初始化变量conf中的order数组（数据结构access_dir_conf的一个成员），设置order数组的初始值为DENY_THEN_ALLOW。

11414: 设置allow数组。

11416: 设置deny数组。

11419: 返回类型为access_dir_conf的conf对象。

3. order

11422: 该例程是access_cmds(11575行)的一个回调例程，在mod_access模块处理order指令时调用此例程（11577行）。

11425: 该例程有一个驻留的paccess_dir_conf类型结构变量dv。

11428: 下面8行是比较该例程最后一个参数变量的值是否为order指令的几个可接受值（如：allow、deny deny、allow和mutual-fail）。

11435: 如果该变量值不符合任何一个order指令值，则例程返回“unknown order”字符串，如果要扩展order指令，就必须在这行上面加上相应的配置选项。

4. is_ip

11444: is_ip例程用于确定一个字符串是否为合法的IP地址、子网地址或网络掩码。

5. allow_cmd

11451: allow_cmd例程是access_cmds(11575行)的一个回调例程，mod_access模块处理allow指令时在11580行调用此例程，处理deny指令时在11584行调用此例程。

11458: allow和deny指令语法检查。

11466: 如果allow/deny是一个环境变量，则例程设置类型为T_ENV。

11471: 如果allow/deny是“all”，则例程设置类型为T_ALL。

11475: 如果allow/deny是network/netmask配对，则例程设置类型为T_IP。

11483: 如果allow/deny不是一个合法的IP地址，则例程设置类型为T_FAIL,且警告用户配置出错。

11575: 创建类型为command_rec的对象access_cmds。

11577: 调用order例程，根据配置文件中的Order指令进行赋值。

11580: 调用allow_cmd例程，根据配置文件中的Allow指令进行赋值。

11584: 调用allow_cmd例程，根据配置文件中的Deny指令进行赋值。

6. in_domain

11590: in_domain例程检查某一主机是否为某一域名内的主机。

7. find_allowdeny

11616: find_allowdeny例程搜索一个allowdeny对象数组(11379行)，它由该模块的处理例程check_dir_access调用（11674行）。

11629: 这条switch语句检查结构allowdeny的type属性。

8. check_dir_access

11674: check_dir_access例程在11733行被设置为mod_access模块的访问处理例程，它是在Apache请求过程中调用的。

11677: 创建一个access_dir_conf对象。

11679: ap_get_module_config例程是Apache内部例程，它的功能是为模块获取配置数据。

11683: 从这里到11702行，根据Order指令设置来决定调用find_allowdeny例程（11616行）的Order指令值和适当的动作。

11704: 这个if块根据mod_access模块已建立的规则和Satisfy核心指令来处理不允许访问的客户请求。

11718: 这是模块mod_access的标准输出，名称为access_module。

11722: 模块的初始化例程设置为access_dir_config。

11728: 设置access_cmds对象。

11733: 设置模块处理例程为check_dir_access。

1.2 mod_auth模块

缺省条件下，mod_auth模块已编译进Apache，它允许一个服务器用标准文本文件进行用户身份认证。该模块拥有三个基于文本文件设置的身份验证指令：AuthGroupFile、AuthUserFile和AuthAuthoritative。这三条指令可以放置在任何一个目录下的包容器中。

AuthGroupFile和AuthUserFile指令设定用户文件和群组文件的位置。用户文件的格式类似UNIX环境下的/etc/passwd文件格式；不同之处在于/etc/passwd文件有多个域，用户文件只有前面两个域，文件格式如下：

用户名称：加密密码

群组文件的格式不同于Unix环境下的Group文件，其格式如下：

组别：用户名称1用户名称2用户名称3

AuthAuthoritative指令设定On或Off参数来控制验证过程，设定指令值为On，则防止底层模块通过验证。该指令的缺省值为On。

1.2.1 模块结构

当Apache调用mod_auth模块时，首先执行authenticate_basic_user例程获取用户的ID并确定它的合法性，接着调用例程check_user_access检查requires指令的设定。同时，模块确定一个用户所属的组和用户的密码。模块的初始化例程为create_auth_dir_config 。模块结构如图1-2所示。

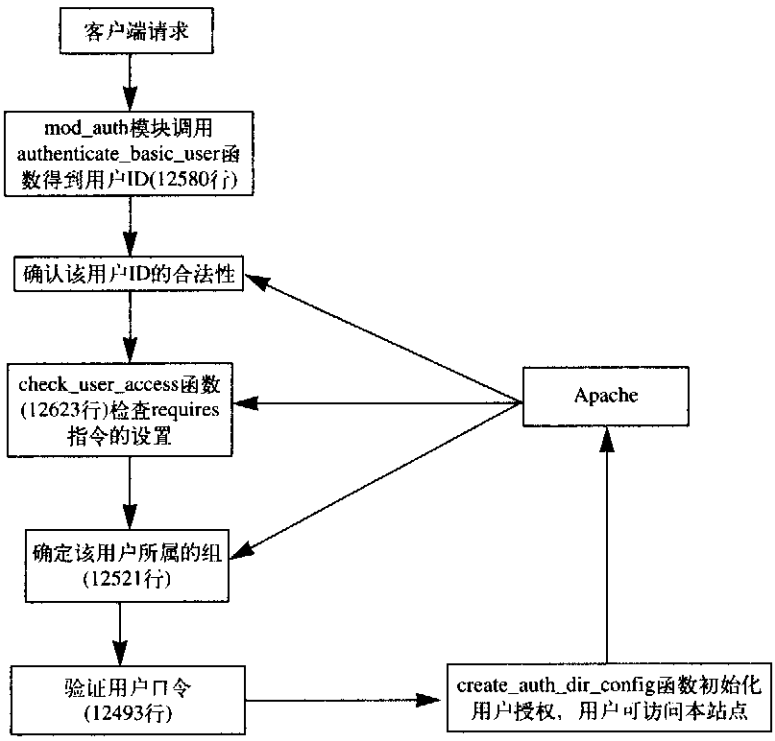

图1-2　mod_auth模块为基于文本的访问提供web资源

1.2.2 定制

模块mod_auth是开发用户身份验证模块的标准。它根据用户名称、密码和地域信息控制资源访存。如果想做的事情不仅仅是优化进程，则建议使用这个模块作为模板来创建自己的身份验证模块。例如，可以创建下面这些模块：

- 开发一个使用纯文本密码的身份鉴别模块。
- 开发一个使用自己公司用户数据库进行身份鉴别的模块（使用Microsoft Access、FoxPro或一个通用的ODBC或JDBC进行鉴别）。
- 开发一个只使用密码进行身份鉴别的模块，该模块的身份鉴别依赖于被请求的资源。

1. mod_auth

12437: ap_md5.h文件中包含一些加密用的例程。如果想使用自己的加密例程，将该文件替换掉就行了。

12439: 数据结构auth_config_struct包含三个属性：密码文件名称（12440行），组文件名称（12441行）和一个整型的标志，用于指定本模块是否有权进行身份验证（12442行）。该结构命名为auth_config_rec。

2. create_auth_dir_config

12445: create_auth_dir_config例程在模块初始化期间被调用，该例程创建一个名为sec的auth_config_rec对象。在12716行设定它为模块初始化例程。密码文件名和组文件名初始值设为空值，缺省标志值设定为1。

12470: 模块的Command_rec，命名为auth_cmds在这里设置。

12491: 设定和输出模块的名称。

3. get_pw

12493: get_pw例程从特定的密码文件中获取特定用户的密码。

4. groups_for_user

12521: groups_for_user例程从特定的组文件中获取特定用户所属的组列表。

5. authenticate_basic_user

12580: authenticate_basic_user例程调用get_pw和其他例程进行用户身份验证。该例程通过返回一个整型值来通知Apache如何处理该用户。

6. check_user_access

12623: check_user_access例程通过调用groups_for_user例程来确定一个合法的用户是否可以访存所请求的资源。该例程也处理AuthAuthoritative指令，并且如果AuthAuthoritative值设定为false时，例程拒绝处理身份验证。

12712: 定义模块mod_auth的标准输出：auth_module。

12716: 设定create_auth_dir_config为初始化验证配置例程。

12722: 设定auth_cmds为命令列表回调例程。

12725: 设定authenticate_basic_user为模块身份验证例程。

12726: 设定check_user_access为检查目录访存权限例程。

1.3 mod_auth_anon模块

缺省条件下，mod_auth_anon模块没有编译进Apache。它提供允许匿名用户访存特定区域（通常情况下，该区域只有通过身份验证的用户方可访存）的方法。类似一个FTP服务器，它允许通过email地址来追踪用户。该模块包含六个指令：Anonymous、Anonymous_Authoritative、Anonymous_LogEmail、Anonymous_MustGiveEmail、Anonymous_NoUserID和Anonymous_VerifyEmail，主要用于匿名用户追踪和身份验证。这些指令可以放置在任何一个包容器中。

Anonymous指令把允许访问的列表中的用户名当作匿名用户。另一些用户名称（如：test、anonymous和www）代表用户数据库中不存在的用户。

Anonymous_Authoritative指令与mod_auth模块的Auth_Authoritative指令相似。它通过设定On或Off参数来控制验证。设定指令为On，则防止底层模块通过验证。然而，该指令不同于Auth_Authoritative，其缺省值为Off。

如果Anonymous_LogEmail指令设定为on（缺省值），则Apache在错误日志文件中记录email地址。

如果Anonymous_MustGiveEmail指令设定为on（缺省值），则Apache不允许有空密码。

Anonymous_NoUserID指令用于控制访问用户是否需要指定一个用户名称。该指令缺省为off，如果指令设定为on，则用户名称是不相关的，甚至不需要打印出来。

如果Anonymous_VerifyEmail与Anonymous_LogEmail和Anonymous_must GiveEmail一道设定为On，将执行一项检查以保证一个周期后至少有一个@标记。

1.3.1 模块结构

当Apache调用mod_auth_anon模块时，首先执行anon_authenticate_basic_user例程获取用户的ID，并根据所使用的指令来检查合法性；接着调用例程check_anon_access检查本模块控制的请求资源，并根据这些请求资源来接受或拒绝验证。同时，模块确定一个用户所属的组和用户的密码。模块的初始化例程为create_anon_auth_dir_config（12961行）。模块结构如图1-3所示。

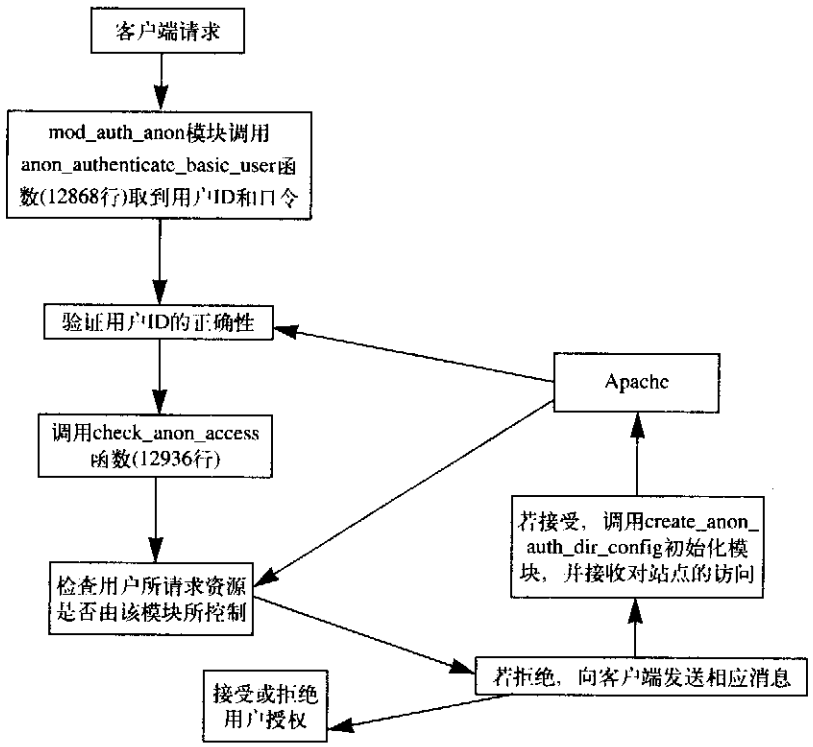

图1-3　利用mod_auth_anon模块可以设定允许用户匿名访问的范围

1.3.2 定制

由于模块mod_auth_anon是基于模块mod_auth的，因此建议用户不要定制这个模块。

1. mod_auth_anon

12743：声明auth_anon节点类型。

12748：声明数据结构anon_auth_config_rec。

2. create_anon_auth_dir_config

12759：create_anon_auth_dir_config例程为模块mod_auth_anon创建配置信息，12770行到12776行设置本模块指令的缺省值。

3. anon_set_passwd_flag

12780: anon_set_passwd_flag例程设定auth_anon_mustemail标志。

4. anon_set_userid_flag

12787: anon_set_userid_flag例程设定auth_anon_nouserid标志。

5. anon_set_logmail_flag

12793: anon_set_logmail_flag例程设定auth_anon_logmail标志。

6. anon_set_verifyemail_flag

12799: anon_set_verifyemail_flag例程设定auth_anon_verifyemail标志。

7. anon_set_authoritative_flag

12805: anon_set_authoritative_flag例程设定auth_anon_authoritative标志。

12841: 定义模块的command_rec为anon_auth_cmds.

12866: 设定和输出模块的名称anon_auth_module。

8. anon_authenticate_basic_user

12868: anon_authenticate_basic_user例程根据设定的指令进行获取和校验用户名和密码。

9. check_anon_access

12936: check_anon_access例程确定请求的资源真正受mod_auth_anon模块保护。

12957: 该模块的标准输出，anon_auth_module被定义。

12961: 设定create_anon_auth_dir_config为创建访问指令配置例程。

12966: 设定命令列表的回调例程为anon_auth_cmds。

12969: 设定authenticate_basic_user为模块身份验证例程。

12971: 设定check_anon_access为检查控制访问例程。

1.4 mod_auth_db模块

缺省条件下，模块mod_auth_db没有编译进Apache。它提供的一种方法是通过Berkeley DB数据库文件进行用户身份验证。与mod_auth模块相似，该模块也有三条指令：AuthDBGroupFile、AuthDBUserFile、AuthDBAuthoritative；这三条指令的功能是指定用户和组的数据库位置，并且确定是否需要模块验证用户身份。这些指令可以放置在任何一个包容器中。

AuthDBGroupFile和AuthDBUserFile指令分别指定组数据库和用户数据库位置，尽管它们可以是同一数据库。这两个数据库主要由许多“键名/键值”对组成，且把用户名称作为键名。如果组数据库和用户数据库是分开的，则组文件包含有一系列组，这些组包含用户名和组名，两者之间通过逗号隔开。如果组数据库和用户数据库为同一文件，则其键值包括一个加密过的密码，跟随一个冒号，其后是由逗号隔开的组名。

AuthDBAuthoritative指令设定On或Off参数来控制验证通过，设定指令值为On则防止底层模块通过验证。该指令的缺省值为On。

1.4.1 模块结构

当Apache调用模块mod_auth_db时，首先执行db_authenticate_basic_user例程，这个例程使用一个Berkeley DB数据库处理用户身份验证，接着调用例程db_check_auth，此例程根据配

置的指令来确定是否允许用户访问请求资源。流程如图1-4所示。

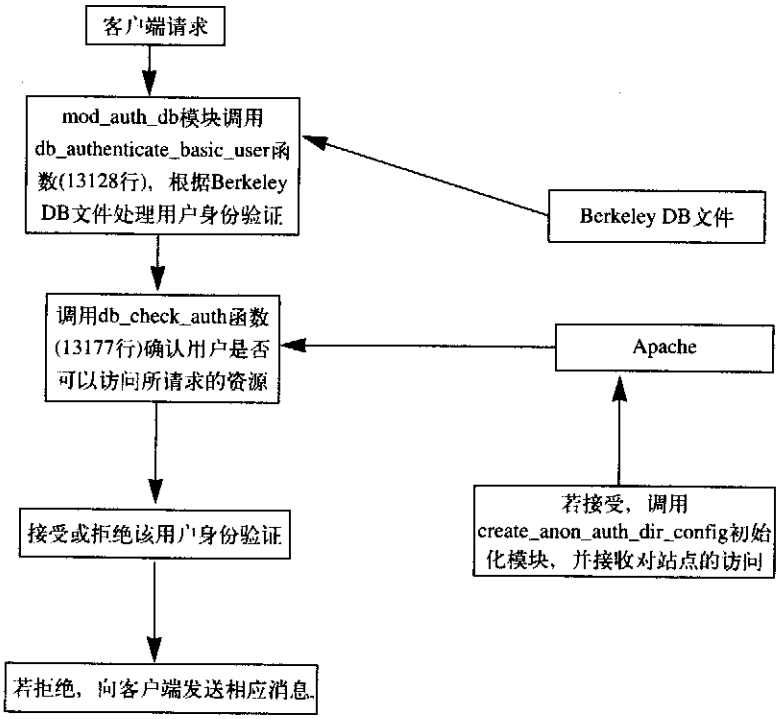

图1-4　通过mod_auth_db模块，Apache可以根据Berkeley DB数据库文件验证用户身份

1.4.2 定制

相对于标准密码文件格式的身份验证表，使用一个正式的数据库来定制mod_auth_db模块是一个很好的练习。定制模块mod_auth_db主要集中在数据库格式的用户名称和密码的验证上：

- 开发一个使用不同于MD5的加密算法进行身份验证的模块。
- 开发一个使用用户钟爱的数据库来获取用户名称、密码和群组的验证模块。

1. mod_auth_db

12986: 包含头文件db.h，如果没有使用Berkeley DB数据库，去掉这一行或用别的头文件代替此文件。

12987: 包含头文件ap_md5.h，如果没有使用MD5加密算法，去掉这一行或用别的头文件代替此文件。

12993: 声明数据结构db_auth_config_rec。这个结构包含密码和组数据库文件以及模块的授权标志。

2. create_db_auth_dir_config

13000: create_db_auth_dir_config例程创建模块结构并设定结构的缺省值。

13021: 定义command_rec对象: db_auth_cmds。

3. get_db_pw

13049: 设定模块名称。

13051: get_db_pw例程从特定的密码数据库获取用户的密码。

4. get_db_grp

13109: get_db_grp例程在某一特定的组数据库中查找特定用户的组别。

5. db_authenticate_basic_user

13128: db_authenticate_basic_user例程是模块的身份验证的处理例程。它基于密码数据库文件检查用户名称和密码的合法性。

6. db_check_auth

13177: db_check_auth例程是根据配置的指令确保允许已验证的用户访问特定的资源。

13245: 定义本模块的标准模块列表。

13249: 设定create_db_auth_dir_config为模块初始化例程。

13255: 设定命令列表结构db_auth_cmds。

13258: 设定身份验证处理例程db_authenticate_basic_user。

13259: 设定访存控制处理例程db_check_auth。

1.5 mod_auth_dbm模块

缺省条件下，模块mod_auth_dbm并没有编译进Apache。它提供的一种方法是通过DBM格式的数据库文件进行用户身份验证。与mod_auth模块相似，该模块也有三条指令：AuthDBMGroupFile、AuthDBMUserFile、AuthDBMAuthoritative，这三条指令的功能是指定用户和组的数据库位置，并且确定是否需要模块验证用户身份。这些指令可以放置在任何一个包容器包含语句中。

AuthDBMGroupFile和AuthDBMUserFile指令分别指定组数据库和用户数据库位置，尽管它们可以是同一数据库。这两个数据库都将用户名称作为键名。如果组数据库和用户数据库是分开的，则组数据库由许多组记录组成，这些组记录包含用户名和组名，两者之间通过逗号隔开。如果组数据库和用户数据库为同一文件，则其键值包括一个加密过的密码，跟随一个冒号，其后是由逗号隔开的组名。

AuthDBMAuthoritative指令设定On或Off参数来控制验证通过，设定指令值为On，则防止底层模块通过验证。该指令的缺省

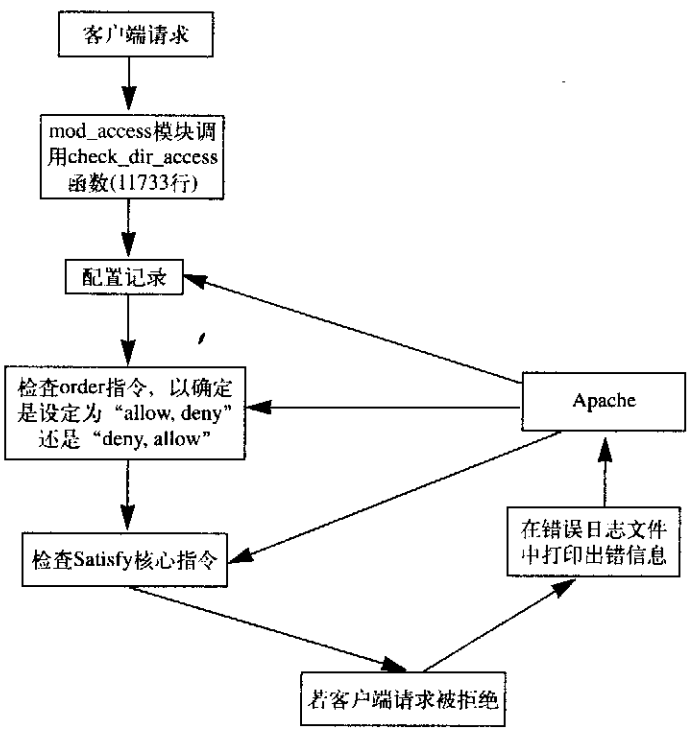

图1-5 mod_auth_dbm模块提供了基于DBM格式的数据库文件的用户身份验证

值为On。

1.5.1 模块结构

当Apache调用模块mod_ auth_dbm时，首先执行dbm_ authenticate_basic_user例程，这个例程使用一个DBM格式的数据库处理用户身份验证，接着调用例程dbm_check_auth，此例程根据配置的指令来确定是否允许用户访问请求资源。流程如图1-5所示。

1.5.2 定制

定制模块mod_auth_dbm与定制模块mod_auth_db非常相似，但相对于使用标准密码文件格式的身份验证表，使用一个正式的数据库来定制mod_auth_db模块是一个很好的练习。定制模块mod_auth_dbm也主要集中于数据库格式的用户名称和密码的验证：

- 开发一个使用不同于MD5的加密算法进行身份验证的模块。
- 开发一个使用用户所钟爱的数据库来获取用户名称、密码和群组的验证模块。

1. mod_auth_dbm

13316: 包含头文件ndbm.h，如果使用其他类型数据库，去掉这一行或用别的头文件代替此文件。

13317: 包含头文件ap_md5.h，如果加密算法不是MD5，去掉这一行或用别的头文件代替此文件。

13333: 声明数据结构dbm_auth_config_rec。这个结构包含密码名和组数据体文件以及模块授权标志。

2. create_dbm_auth_dir_config

13341: create_dbm_auth_dir_config例程创建模块结构并设定结构的缺省值。

13365: 定义command_rec对象: dbm_auth_cmds。

3. get_dbm_pw

13392: 定义模块。

13394: get_dbm_pw例程从特定的DBM密码文件中获取用户的密码。

4. get_dbm_grp

13443: get_dbm_grp例程在特定的DBM群组文件中查找特定用户的组别。

5. dbm_authenticate_basic_user

13464: dbm_authenticate_basic_user例程是模块的身份验证的处理例程。它从配置文件指定的DBM密码文件中获取用户名称和密码，并检查它们的合法性。

6. dbm_check_auth

13513: dbm_check_auth例程根据配置的指令确保允许已验证的用户访问特定的资源。

13582: 定义本模块的标准模块列表。

13586: 设定create_dbm_auth_dir_config为模块初始化例程。

13592: 设定命令列表结构dbm_auth_cmds。

13595: 设定身份验证处理例程dbm_authenticate_basic_user。

13596: 设定访存控制处理例程dbm_check_auth。

1.6 mod_digest模块

缺省条件下，模块mod_digest并没有编译进Apache。它允许Apache支持MD5分类验证，这种验证通过加密那些往Web服务器发送的用户名和口令来增加安全性。此模块只有一条指令：AuthDigestFile。

指令AuthDigestFile设定特定格式用户文件的路径，这些文件通常和Apache支持程序htdigest一起生成。

1.6.1 模块结构

当Apache调用模块mod_digest时，首先执行authenticate_digestc_user例程，这个例程校验用户名和口令，接着调用例程digest_check_auth，此例程也校验用户是否有相应的权限访问所请求的资源。流程如图1-6所示。

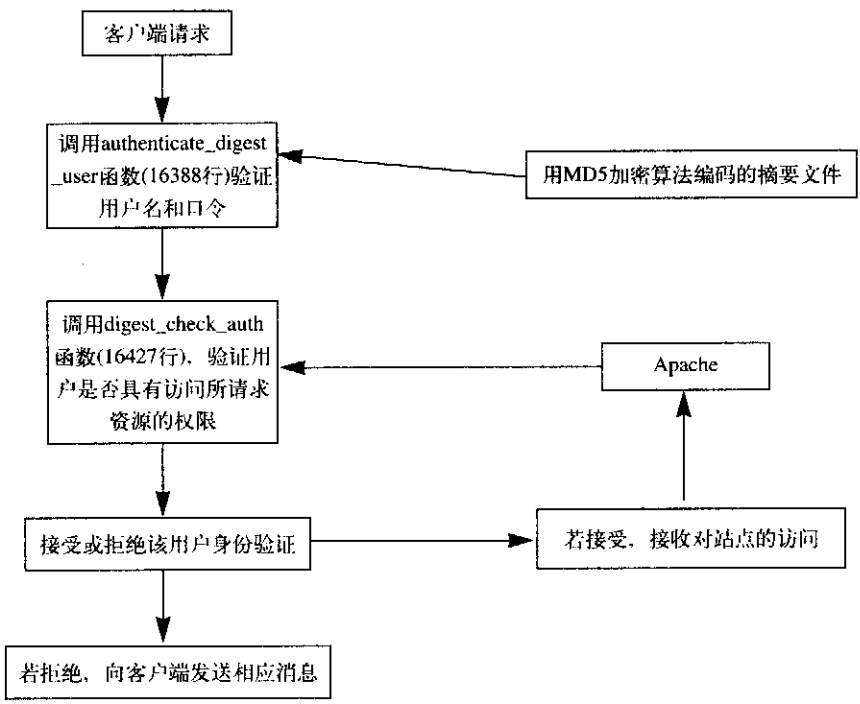

图1-6 mod_digest模块使用一个编码的摘要文件进行用户身份验证

1.6.2 定制

由于模块mod_digest要求浏览器与Web服务器之间用MD5加密算法进行通信，因此，定制此模块的技术集中在服务器端。

- 设计一个使用用户钟爱的数据库进行MD5查找的模块。

1. mod_digest

16147：声明结构digest_config_rec，此结构包含密码分类文件名称。

16151：声明结构digest_header_rec，此结构包含从Apache传给此模块的请求头信息。

2. create_digest_dir_config

16159: create_digest_dir_config为模块结构创建例程。

16175: 定义command_rec对象: digest_cmds。

16183: 输出模块名称: digest_module。

3. get_hash

16185: get_hash例程从特定的密码分类文件中获取特定用户的密码哈希。

4. get_digest_rec

16218: get_digest_rec例程使用digest_header_rec结构分析所请求的资源、用户名等，同时根据分析的结果执行相应的工作。

5. find_digest

16358: find_digest例程根据digest_header_rec结构返回MD5分类。

6. authenticate_digest_user

16388: authenticate_digest_user例程根据MD5分类检验用户的合法性。

7. digest_check_auth

16427: digest_check_auth例程的功能是确保已授权的用户访问特定的资源。

16479: 定义digest_module列表。

16483: 设定模块的初始化例程为create_digest_dir_config。

16489: 设定命令列表结构digest_cmds。

16492: 设定用户身份验证处理例程。

16493: 设定访存控制处理例程digest_check_auth。

第2章 别名和重定向模块

本章描述四个复杂且功能强大的模块，这些模块用于控制文件的虚拟位置，使用户的浏览器指向正确的文件位置并且映射Apache-served站点的一个URL到另外一个URL。用户有时会意识到他们的浏览器被重定向，但一般情况下，Web站点管理人员只要控制一个Web站点的结构和外观，用户是不会意识到发生过重定向的。

第一个模块是mod_alias，其主要功能是处理系统别名和重定向浏览器到其他文件系统或者完全不同的URL。缺省时模块mod_alias已编译进Apache。

为控制服务器端的图像映射，缺省时模块mod_alias已编译进Apache。它允许定义一图形可点击的区域，并且能够根据用户点击图像的区域使用户的浏览器进入不同的位置。这个模块并不影响客户端的图像映射（客户端地图是嵌入网页中的地图）。

在互联网经济全球化的时代，通常要求一个Web站点能支持多种语言。模块mod_negotiation实现Web服务器根据浏览器所设置的语言、字符集、编码等因素重定向客户端的请求。利用这个模块，那些提供多语种功能网页的公司能保证客户根据他们喜爱的语言浏览网站。缺省时模块mod_negotiation已编译进Apache。

最后一个模块是mod_rewrite，这个模块的功能非常强大，它允许Web站点管理人员根据自己设定好的规则集来重写浏览器的URL。

2.1 mod_alias模块

模块mod_alias提供的功能是链接Web服务器文件系统的不同部分到服务器的逻辑路径中，以及进行URL重定向。这个模块提供的访问控制指令有Alias、AliasMatch、ScriptAlias、ScriptAliasMatch、Redirect、RedirectMatch、RedirectTemp和RedirectPermanent。这些指令可以放置在任何一个包容器中。

Alias指令和AliasMatch指令将一路径从Web服务器空间的某一位置映射到机器文件系统的任意位置。

Alias指令用于配置文件时，其后必须跟随别名名称和物理路径。例如：如果希望客户通过/stuff 访问Web服务器中的路径/usr/mydir，Alias语句如下：

```
Alias  /stuff  /usr/mydir/
```

除了需要一个规则表达式代替逻辑路径名以外，AliasMatch指令与Alias指令的工作方法基本相同。前面的例子可以改为：

```
AliasMatch ^/stuff(.*) /usr/mydir$1
```

ScriptAlias和ScriptAliasMatch指令除了把文件夹标记为 CGI脚本目录以外，其功能与Alias和AliasMatch指令几乎相同。这两条指令与上面两条指令的语法也相同。

Redirect指令允许Web服务器依赖客户请求的URL而指向不同的URL。例如，如果有一文件在如下URL：

```
/zippy/mypage.html
```

2.1.2 定制

由于本模块主要功能是处理别名和重定向的，因此不需要用户进行定制。

1. Declarations

11920: 定义结构alias_entry。该结构具有五个描述指令动作的属性。

11929: 定义结构alias_server_conf。该结构对每个服务器别名和重定向进行跟踪。

11934: 定义结构alias_dir_conf。该结构跟踪每个目录的重定向。目录的别名不需要跟踪，因为它们没有意义。（别名的功能是透明地将一请求映射到某一目录，如果目录配置文件的一个别名是无用的，则本次请求肯定不可能获得该目录的映射。）

11938: 声明标准模块命名输出。

2. create_alias_config

11940: create_alias_config例程的功能是从conf文件中获取服务器的别名和重定向信息。

11947: 创建别名列表。

11949: 创建重定向列表。

3. create_alias_dir_config

11954: create_alias_dir_config例程的功能是从conf文件中获取目录的重定向信息。

11959: 创建重定向列表。

4. merge_alias_config

11964: merge_alias_config例程的功能是合并服务器配置信息。

5. merge_alias_dir_config

11980: merge_alias_dir_config例程的功能是合并目录配置信息。

6. add_alias_internal

11993: add_alias_internal例程的功能是增加一个alias_entry（11920行）。该例程只由本模块内的例程调用（因此，例程名有_internal）。

12002: 创建新的alias_entry。

12007: 如果这里有规则表达式，则系统将把它编译进去。如果这里有错误，则系统返回。

12015: 设定伪造的URL。

12016: 设定真实的URL。

12017: 改变新的别名句柄。

7. add_alias

12022: add_alias例程的功能是增加一个使用Alias和ScriptAlias指令定义的别名。

12025: 调用add_alias_internal例程。

8. add_alias_regex

12028: add_alias_regex例程的功能是增加一个使用AliasMatch和ScriptAliasMatch指令定义的别名。

12031: 调用add_alias_internal例程。

9. add_redirect_internal

12034: add_redirect_internal例程的功能是增加一个alias_entry。该例程只由本模块内的例程调用（因此，例程名有_internal）。

12040: 定义新的alias_entry变量。

12051: 本行开始的if-else语句通过检查状态字段来决定该字段是否匹配关键字gone、permanent、temp或seeother中的一个。

12066: 如果这里有规则表达式，则系统将把它编译进去。如果这里有错误，则系统返回。

12073: 执行错误检查。

12089: 设定伪造的URL。

12090: 设定真实的URL。

12091: 设定模式匹配规则表达式。

12092: 设定重定向状态。

10. add_redirect

12096: add_redirect例程用于增加一个由指令Redirect、RedirectTemp和RedirectPermanent定义的重定向命令。

12100: 调用add_redirect_internal例程。

11. add_redirect_regex

12104: add_redirect例程用于增加一个由指令RedirectMatch定义的重定向命令。

12108: 调用add_redirect_internal例程。

12. alias_cmds

12112: 设定模块的command_rec (alias_cmds)。

13. alias_matchs

12148: alias_matchs例程用于获取匹配一给定别名的字符个数。例程返回匹配的字符个数。

14. try_alias_list

12191: try_alias_list例程设法将一别名或重定向与请求资源匹配。

15. translate_alias_redir

12255: translate_alias_redir例程的功能是将本模块的其他部分组合在一起。这个例程根据其他例程的执行结果决定是返回一状态码给Apache还是拒绝处理请求。

16. fixup_redir

12289: fixup_redir例程用于在处理重定向后执行例程修正工作。

12311: 定义标准模块输出: alias_module。

12315: 设定目录配置创建例程（create_alias_dir_config)。

12317: 设定目录配置合并例程（merge_alias_dir_config)。

12319: 设定服务器配置创建例程（create_alias_config)。

12320: 设定服务器配置合并例程（merge_alias_config)。

12321: 设定命令列表: alias_cmds。

12323: 设定文件名转化例程(translate-alias-redir)。

12329: 设定修正例程fixup_redir。

2.2 mod_imap模块

模块mod_imap的功能是处理服务器端的图像映射。本模块的配置指令有ImapMenu、

ImapDefault和ImapBase。

如果用户点击一图像映射没有定义的部分，用户将会接收到服务器的反馈，而ImapMenu指令控制这些反馈。指令选项有none、formatted、semiformatted和unformatted:

```
ImapMenu formatted
```

ImapDefault指令用于设定用户点击一图像映射没有定义部分的缺省动作。指令选项有error、map、referrer、nocontent或URL:

```
ImapDefault nocontent
```

ImapBase指令为图像映射后的URL设定缺省的base URL。该指令的选项有map、referrer或URL:

```
ImapBase http://yourhost.com
```

2.2.1 模块结构

当Apache调用模块mod_imap时，首先执行create_imap_dir_config例程创建模块缺省配置结构(17490行)；接着，系统调用merge_imap_dir_config例程合并conf文件的配置与缺省的配置 (17504行)。模块mod_imap的命令列表为imap_cmds（在17524行定义），并且模块的处理列表为imap_handlers。模块结构如图2-2所示:

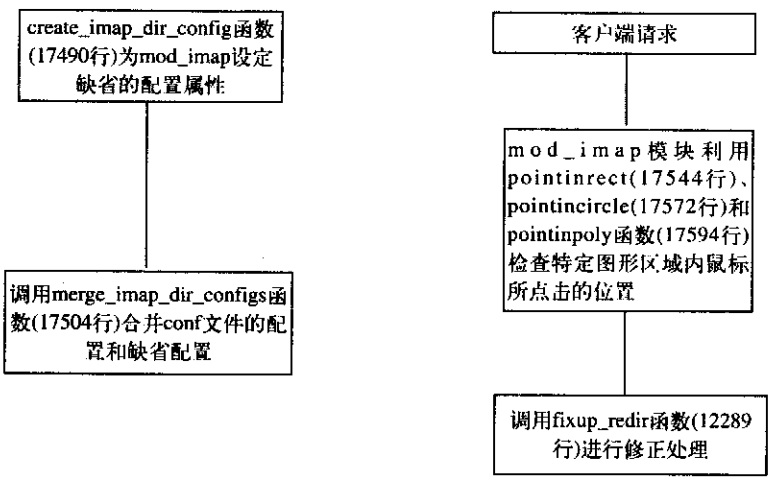

图2-2 mod_imap模块使Apahce可以处理服务器端的图像映射

2.2.2 定制

mod_imap模块的功能是处理服务器端的图像映射。定制这个模块主要集中在增强和扩充映射设计中所拥有的选项范围，例如:

- 增加用于定义图像映射空间的可用的地图形状。
- 提供多层映射互相覆盖的能力。
- 增强用户点击未定义部分映射时的错误响应能力。

1. imap_magic_type

17469: 这一行是定义映射的MIME类型。如果传统的映射格式改变了，则这里的MIME

类型定义也应改变。

17482: 声明标准模块输出: imap_module。

17484: 定义结构imap_conf_rec。

2. create_imap_dir_config

17490: create_imap_dir_config例程用于创建缺省配置信息。

3. merge_imap_dir_configs

17504: merge_imap_dir_configs例程用于合并配置信息。

4. imap_cmds

17524: 定义模块的command_rec: imap_cmds。

5. pointinrect

17544: pointinrect例程根据给定的坐标点是否在特定的矩形中来决定返回值为0或为1。

6. pointincircle

17572: pointincircle例程根据给定的坐标点是否在特定的圆中来决定返回值为0或为1。

7. pointinpoly

17594: pointinpoly例程根据给定的坐标点是否在特定的多边形中来决定返回值为0或为1。

8. is_closer

17626: is_closer例程根据一给定的点是否更接近一给定的坐标来决定返回值为0或为1。

9. get_x_coord

17653: get_x_coord例程从请求参数中获取x坐标并返回该值。如果有错误发生，则该例程返回-1。

10. get_y_coord

17682: get_y_coord例程从请求参数中获取y坐标并返回该值。如果有错误发生，则该例程返回-1。

11. read_quoted

17729: read_quoted例程从一带引号的字符串中分离出引号，并在该字符串尾部赋一空值。

12. imap_url

17763: 该例程根据用户选择映射的坐标识别URL。如果用户所选择的区域没有定义，则例程将返回空值。

13. imap_reply

17934: imap_reply例程根据redirect参数值返回HTTP代码。

17936: 返回一错误代码。

17940: 如果用户点击一映射没有定义的部分，则这行的条件将发生。

17944: 如果一重定向相对于用户点击的坐标，则这行的条件将总是发生。

14. menu_header

17953: menu_header例程输出菜单的开始部分。

15. menu_blank

17972: menu_blank例程向该菜单中插入不同类型的空行。

16. menu_comment

17986: menu_comment例程向该菜单插入一注释。

17. menu_default

18003: menu_default例程向该菜单插入链接。

18. menu_directive

18028: menu_directive例程向该菜单放置指令信息。

19. menu_footer

18053: menu_footer例程插入正确的HTML来结束一菜单。

20. imap_handler

18060: imap_handler例程接受图像映射和资源的请求，创建配置信息，调用适当的例程处理资源请求，并向Apache返回最后的响应。

21. imap_handlers

18399: 定义处理资源（imap_handlers）。

22. imap_module

18406: 定义本模块的标准输出。

18410: 设定create_imap_dir_config为配置创建例程。

18411: 设定merge_imap_dir_configs为配置合并例程。

18415: 设定命令列表imap_cmds。

18416: 设定处理列表imap_handlers。

2.3 mod_negotiation模块

模块mod_negotiation允许Web设计者根据用户的偏好提供不同的Web页面内容。该模块广泛使用的内容协商形式是面向多语言的选择。Web页面开发者使用此特征能够用多种语言制作网页，而且Apache根据用户浏览器的语言设定传回正确版本的页面。这个模块提供两条配置指令：CacheNegotiatedDocs和LanguagePriority。

CacheNegotiatedDocs指令控制一代理服务器是否能缓存那些经过内容协商后被显示过的Web文档。这条指令只对HTTP/1.0以下的版本有用，因为HTTP/1.1已经有内置的内容协商功能。

LanguagePriority指令允许Web站点选择语言的优先次序。例如：加拿大魁北克省的某一Web站点必须让讲英语和讲法语的赞助商都引起注意。但是该站点在设定语言优先次序时可能会让法语优先于英语（因为魁北克省大部分居民都讲法语），他们将按照如下设置：

```
LanguagePriority fr en
```

相反，在多伦多的一个能同时讲英语和讲法语的用户服务Web站点可能会让英语优先于法语，其设定如下所示：

```
LanguagePriority en fr
```

mod_negotiation功能非常强大，通过它，Apache能处理多种的客户偏爱。

2.3.1 模块结构

当Apache调用模块mod_negotiation时，首先执行create_neg_dir_config例程创建模块缺省配置结构(26328行)；接着，系统调用merge_neg_dir_config例程，合并conf文件的配置与缺省

的配置（26338行）。模块mod_negotiation命令列表为negotiation_cmds（在26381行定义）。模块的处理 列表为negotiation_handlers(29461行)，并且模块的类型检查例程为handle_multi（29293行）。模块结构如图2-3所示：

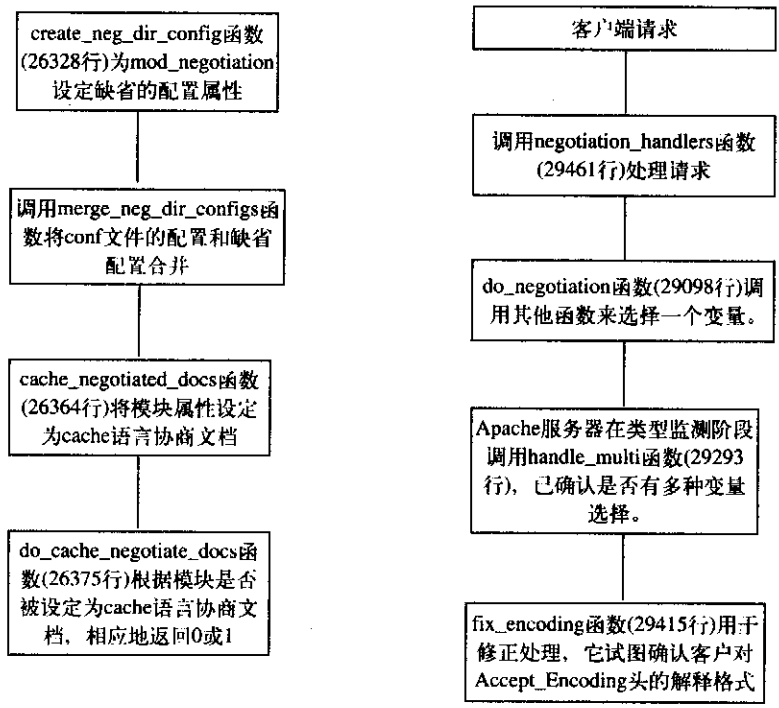

图2-3 通过mod-negotiation模块，Apache可以根据客户的偏好提供特定的内容

2.3.2 定制

模块mod_negotiation是一个庞大的模块，由于模块是按照内容协商的标准进行设计，因此定制此模块是一项艰难的工作。但此模块的代码从学习的角度来看是比较有意思的。

1. neg_dir_config

26322：定义结构neg_dir_config。

2. negotiation_module

26326：声明标准模块输出：negotiation_module。

3. create_neg_dir_config

26328：create_neg_dir_config例程用于创建缺省配置结构。

4. merge_neg_dir_configs

26338：merge_neg_dir_configs例程用于合并配置选项。

5. set_language_priority

26353：set_language_priority设定语言的优先次序。

6. cache_negotiated_docs

26364：cache_negotiated_docs例程设定模块配置信息，目的是为了缓存语言协商文档。

7. do_cache_negotiated_docs

26375: do_cache_negotiated_docs例程根据模块是否配置为能够缓存语言协商文档而返回0或1。

8. negotiation_cmds

26381: 定义模块的 command_rec。

9. accept_rec

26399: 定义结构accept_rec。该结构包含从用户浏览器获得关于用户希望得到的内容的信息 。

10. var_rec

26428: 定义结构var_rec。该结构包含模块和服务器需要正确处理语言协商的文档信息。

11. negotiation_state

26484: 结构negotiation_state是为使协商更容易进行而记录的不同客户和模块的信息 。

12. clean_var_rec

26526: clear_var_rec将var_rec重置为缺省状态。

13. set_mime_fields

26556: set_mime_fields例程用accept_rec的字段值设定一部分var_rec的字段。

14. get_entry

26604: get_entry例程获取一MIME类型值，并根据该值来设定accept_rec的各属性值。

15. do_header_line

26732: do_header_line例程的功能是解析一accept_line，并且返回一array_header。

16. do_languages_line

26758: do_languages_line例程从Content_Languages标头记录中取一行，并且返回包含一组语言的一个array_header。

17. parse_accept_headers

26785: parse_accept_headers例程为接受多行信息而解析资源请求头，同时调用多个模块例程，返回一negotiation_state。

18. parse_negotiate_headers

26831: parse_negotiate_headers例程查找一资源请求记录的协商头。

19. maybe_add_default_accepts

26941: 如果浏览器没有给服务器accept信息，则该例程设定缺省的accept_rec。

20. header_state

26985: 定义一枚举数据类型: header_state。

21. get_header_line

26989: 这个例程从一请求头获取一行信息。

22. strip_paren_comments

27073: 例程strip_paren_comments从给定标头获取内容。

23. lcase_header_name_return_body

27105: 例程lcase_header_name_return_body从给定的标头中获取一标头体，并将它返回。

24. read_type_map

27138: 例程read_type_map读取一类型图，并调用其他模块例程解析它，最后返回HTTP

代码。

25. variantsortf

27244: 例程variantsortf是一排序算法函数，例程read_types_multi调用该函数进行var_recs排序。

26. read_types_multi

27270: read_types_multi例程读取一目录过滤类型图，并解析该图，最后返回HTTP代码。

27. mime_match

27431: mime_match例程搜索一符合客户资源请求的MIME类型。

28. level_cmp

27500: level_cmp例程在两个相同性质的变量中选择。

29. find_lang_index

27569: find_lang_index例程从一语言引用的accept_rec数组中返回一索引。

30. find_default_index

27596: find_default_index例程查找并返回特定语言的优先权。

31. set__default_lang_quality

27639: set_default_lang_quality例程设定一negotiation_state对象的default_lang_quality属性值。

32. set_language_quality

27694: set_language_quality例程设定一个var_rec对象的lang_quality属性值。

33. find_content_length

27929: find_content_length例程用于获取发送到客户端的文件的长度。

34. set_accept_quality

27955: set_accept_quality例程设定一个var_rec对象的mime_type_quality属性值。

35. set_charset_quality

28052: set_charset_quality例程设定一个var_rec对象的charset_quality属性值。

36. set_encoding_quality

28159: set_encoding_quality例程设定一个var_rec对象的encoding_quality属性值。

37. algorithm_results

28232: algorithm_results例程枚举了所定义的数据类型。

38. is_variant_better_rvsa

28266: is_variant_better_rvsa例程的功能是使用远程变量选择算法决定哪个变量为较好的变量。

39. is_variant_better

28341: is_variant_better例程的功能是决定哪个变量为较好的变量。

40. best_match

28498: best_match例程确定哪个变量最适合该请求。

41. set_neg_headers

28613: set_neg_headers例程为已协商的文档设定响应标头。

42. make_variant_list

28860: make_variant_list例程以HTML格式产生和返回一组变量。

43. store_variant_list

28940: store_variant_list例程记录make_variant_list例程的输出。

44. setup_choice_response

28961: setup_choice_response例程用于处理多语言变量（该变量选择算法是可行的算法）。

45. do_negotiation

29098: 虽然do_negotiation例程执行许多任务，但其最重要的任务是选择一变量。该例程通过调用其他例程来完成变量选择。

46. handle_map_file

29266: handle_map_file例程通过调用read_type_map例程读取类型图。如果读取失败，则该例程通过执行do_negotiation例程来查找最佳的语言变量。

29274: 调用read_type_map例程。

29278: 调用do_negotiation例程。

47. handle_mutlti

29293: Apache在类型检查阶段调用handle_multi例程。它用于确定是否有多重类型选择和决定采取行动的方法。

48. fix_encoding

29415: fix_encoding例程是一个修正例程。

49. negotiation_handlers

29461: 定义数组negotiation_handlers。

50. negotiation_module

29468: 定义标准模块输出。

29472: 设定create_neg_dir_config为配置创建例程。

29473: 设定merge_neg_dir_configs为配置合并例程。

29477: 设定命令列表negotiation_cmds。

29478: 设定处理列表imap_handlers。

29483: 设定类型检查例程handle_multi.

29484: 设定修正例程fix_encoding.

2.4 mod_rewrite模块

作为模块mod_rewrite的作者，Ralf S. Engelschall是如此适宜地发表了这个模块代码，使它成为“操纵URL的瑞士军刀”。模块mod_rewrite通过使用规则表达式和规则集能够将任一URL模式映射成任意其他的URL模式。本模块提供的指令有：RewriteEngine、RewriteOptions、RewriteLog、RewriteLogLevel、RewriteLock、RewriteMap、RewriteBase、RewriteCond和RewriteRule，各指令具体描述如下：

- RewriteEngine——打开或关闭某一给定的配置容器，改写URL的处理开关。
- RewriteOptions——指定改写引擎的选项。目前，该指令只有一个选项inherit，意思是在缺省状态下，子容器继承父容器的改写引擎配置和规则。

- RewriteLog——指定包含模块mod_rewrite生成的日志文件的文件的路径。
- RewriteLogLevel——用于控制模块mod_rewrite记录日志信息的数量。RewriteLogLevel值设定为0，表示不记录任何日志，而设定为9，表示记录所有的信息。
- RewriteLock——指向一个文件，该文件作为RewriteMap程序的lock文件。
- RewriteMap——指定不同的映射选项，例如：txt是指文本文件映射、rnd是指随机文本文件映射、dbm是指哈希文件映射、int是指内部映射、prg是为外部程序做映射。
- RewriteBase——为目录改写设定URL。
- RewriteCond——定义一个符合RewriteRule指令的条件。
- RewriteRule——匹配一规则表达式模式。

2.4.1 模块结构

模块mod_rewrite的结构是经过精心设计的。当Apache调用模块mod_rewrite时，首先执行模块初始化例程init_module，接着系统调用config_perdir_create例程创建模块缺省配置(29761行)，随后，系统调用config_perdir_merge例程合并conf文件的配置与缺省的配置 (29792行)。系统执行config_server_create例程创建服务器缺省配置，且执行config_server_merge例程合并conf文件的配置与服务器的缺省的配置，模块mod_rewrite命令列表为command_table（在29558行定义）。模块的处理列表为handler_table。将URL翻译为文件名称的例程是hook_uri2file。MIME类型决定例程是hook_mimetype（30843行）。模块修正例程是hook_fixup（30872行）。子进程初始化例程为init_child。模块结构如图2-4所示:

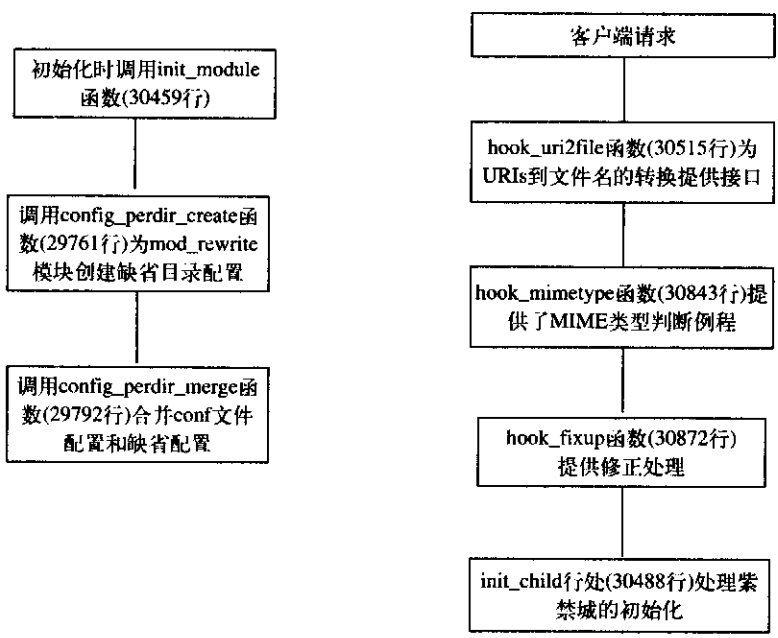

图2-4　mod_rewrite模块使用一组URL重写规则，重定向用户请求

2.4.2 定制

定制模块mod_rewrite是一件不可思议的事情。此模块不仅非常庞大和笨拙，而且已成为

Apache URL改写方面一个事实上的标准。

1. command_table

29558: 定义此模块的command_rec。

2. handler_table

29594: 定义此模块的handler_rec。

3. rewrite_module

29600: 定义此模块的标准输出: rewrite_module。

29602: 设定模块初始化例程为init_module。

29604: 设定目录配置创建例程为config_perdir_create。

29606: 设定目录配置合并例程为config_perdir_merge。

29608: 设定服务器配置创建例程为config_server_create。

29610: 设定服务器配置合并例程为config_server_merger。

29612: 设定命令列表为command_table。

29614: 设定处理列表为handler_table。

29616: 设定将URL转化成文件名称例程hook_uri2file.

29624: 设定MIME类型处理例程hook_mimetype.

29626: 设定模块修正例程hook_fixup.

29632: 设定子进程初始化例程为init_child.

4. config_server_create

29661: config_server_create例程用于创建缺省的服务器配置结构。

5. config_server_merge

29687: config_perdir_merge是合并缺省服务器配置和conf文件配置例程。

6. config_perdir_create

29671: config_perdir_merge是创建目录配置例程。

7. config_perdir_merge

29792: config_perdir_merge是合并缺省目录设置和conf文件配置例程。

8. cmd_rewriteengine

29830: cmd_rewriteengine例程处理conf文件中的RewriteEngine指令。

9. cmd_rewriteoptions

29853: cmd_rewriteoptions例程处理conf文件中的RewriteOptions指令。

10. cmd_rewriteoptions_setoption

29875: cmd_rewriteoptions_setoption例程处理选项设置。它被cmd_rewriteoptions调用(18484行)。

11. cmd_rewritelog

29890: cmd_rewritelog例程处理conf文件中的RewriteLog指令。

12. cmd_rewriteloglevel

29905: cmd_rewriteloglevel例程处理conf文件中的RewriteLogLevel指令。

13. cmd_rewritemap

29919: cmd_rewritemap例程处理conf文件中的RewriteMap指令。

14. cmd_rewritelock

30005: cmd_rewritelock例程处理conf文件中的RewriteLock指令。

15. cmd_rewritebase

30019: cmd_rewritebase例程处理conf文件中的RewriteBase指令。

16. cmd_rewritecond

30039: cmd_rewritecond例程处理conf文件中的RewriteCond指令。

17. cmd_rewritecond_parseflagfield

30123: cmd_rewritecond_parseflagfield例程被例程cmd_rewritecond调用，用于分析标志域。

18. cmd_rewritecond_setflag

30179: cmd_rewritecond_setflag例程被例程cmd_rewritecond_parseflagfield调用，用于设定标志值。

19. cmd_rewriterule

30198: cmd_rewriterule例程处理conf文件中的RewriteRule指令。

20. cmd_rewriterule_parseflagfield

30292: cmd_rewriterule_parseflagfield例程被例程cmd_rewriterule调用，用于分析标志域。

21. cmd_rewriterule_setflag

30348: cmd_rewriterule_setflag例程被例程cmd_rewriterule_parseflagfield调用，用于设定标志值。

22. init_module

30459: init_module例程是模块初始化例程。该例程检查模块的依赖关系，设定lock文件和打开RewriteLogs。

23. init_child

30488: init_child例程为可能的重写准备一子进程。该例程打开一lock文件，创建缓存区。

24. hook_uri2file

30515: hook_uri2file例程用于发现一个匹配请求URI的本地文件。

30536: 输入配置信息。

30544: 如果RewriteEngine设定为off,模块拒绝处理事物。

30625: 实施RewriteRules。

25. hook_mimetype

30843: hook_mimetype例程用于将一文件强制转化为MIME类型。

26. hook_fixup

30872: hook_fixup例程是模块修正例程,当目录配置中有RewriteRule指令时,该例程 被重写引擎触发。

27. handler_redirect

31187: handler_redirect例程用于双重检查重定向请求,如果该请求符合条件,此进程将执行请求。

28. apply_rewrite_list

31216: apply_rewrite_list例程用于处理多个RewriteRule指令。

29. apply_rewrite_rule

31363: apply_rewrite_rule例程用于处理单个RewriteRule指令。

30. apply_rewrite_cond

31857: apply_rewrite_cond例程用于处理单个RewriteCond指令。

31. splitout_queryargs

32062: splitout_queryargs例程从当前的URI获取QUERY_STRING环境变量。

32. reduce_uri

32105: reduce_uri例程用于移开URI中的协议和主机条目。

33. fully_qualify_uri

32191: fully_qualify_uri例程将协议和主机条目加到一个不符合要求的URI中。

34. expand_backref_inbuffer

32242: expand_backref_inbuffer例程处理规则表达式back_references。

35. expand_tildepaths

32290: 当服务器接收到~username格式的URI信息时，模块调用expand_tildepaths例程在UNIX系统上查找和返回一用户的根目录的全路径。Apache的Window版本中没有此例程。

36. expand_map_lookups

32343: expand_map_lookups例程用于从MAP指令中识别多重查找。

37. lookup_map

32462: lookup_map例程用于协调取出重写映射。

38. lookup_map_txtfile

32675: lookup_map_txtfile例程用于从一格式化文件中取出一映射。

39. lookup_map_dbmfile

32723: lookup_map_dbmfile例程用于从一DBM文件中取出一映射。

40. lookup_map_program

32750: lookup_map_program例程用于从一程序中取出一映射。

41. rewrite_mapfunc_toupper

32806: rewrite_mapfunc_toupper例程将一字符串转换成大写。

42. rewrite_mapfunc_tolower

32819: rewrite_mapfunc_tolower例程将一字符串转换成小写。

43. rewrite_mapfunc_escape

32832: rewrite_mapfunc_escape例程用于逃逸一字符串。

44. rewrite_mapfunc_unescape

32841: rewrite_mapfunc_unescape例程用于非逃逸一字符串。

45. rewrite_rand_int

32853: rewrite_rand_int例程用于设定随机数发生器。

46. rewrite_rand

32862: rewrite_rand例程用于生成一伪随机数并将该随机数返回。

47. select_random_value_part

32877: select_random_value_part例程检查许多可用的值，从它们当中挑选一个值，并用rewrite_rand例程将它返回。

48. open_rewritlog

32925: 如果在conf文件中设定RewriteLog，则open_rewritelog例程将打开该文件。

49. rewritelog

32982: rewritelog例程用于在RewriteLog文件中写入一条目。

50. current_logtime

33075: current_logtime例程用于计算和格式化时间。

51. rewritelock_create

33113: rewritelock_create例程为模块创建一lock文件。

52. rewritelock_open

33153: rewritelock_open例程打开一lock文件并得到一文件描述符。

53. rewritelock_remove

33181: rewritelock_remove例程用于移开一lock文件。

54. rewritelock_alloc

33201: rewritelock_alloc例程用于分配一lock文件。

55. rewritelock_free

33214: rewritelock_free例程释放一lock文件。

56. run_rewritemap_programs

33236: run_rewritemap_programs例程通过创建其他进程来处理重写映射。

57. rewritemap_program_child

33293: rewritemap_pragram_child例程用于处理子进程。

58. expand_variables_inbuffer

33362: expand_variables_inbuffer例程将一环境变量放入一缓冲区中。

59. expand_variables

33373: expand_variable例程返回扩充的环境变量。它被expand_variables_inbuffer例程调用。

60. lookup_variable

33412: lookup_variable例程用于决定一Apache HTTP头的值。

61. lookup_header

33687: lookup_header例程从array_header中查找一值。

62. init_cache

33720: init_cache例程用于初始化缓存区。

63. set_cache_string

33731: set_cache_string例程用于在缓存区中存储一字符串。

64. get_cache_string

33744: get_cache_string例程用于在缓存区中取出一字符串。

65. cache_tlb_hash

33767: cache_tlb_hash例程用于在缓存哈希表中创建一索引。

66. cache_tlb_lookup

33780: cache_tlb_lookup例程用于在缓存哈希表中查找一索引。

67. cache_tlb_replace

33798: cache_tlb_replace例程用于在缓存哈希表中替换一索引。

68. store_cache_string

33813: store_cache_string例程在缓存区中存储一字符串，它被set_cache_string例程调用。

69. retrieve_cache_string

33890: retrieve_cache_string例程在缓存区中取出一字符串，它被get_cache_string例程调用。

70. subst_prefix_path

33931: subst_prefix_path例程用另一个路径前缀替换当前的路径前缀。

71. parseargline

33993: parseargline例程是一个命令行解析器。

72. add_env_variable

34072: add_env_variable例程设定一个环境变量。

73. prefix_stat

34099: prefix_stat例程用于对一路径前缀执行一stat系统调用。

74. fd_lock

34132: fd_lock例程用于对一文件描述符进行加锁。

75. fd_unlock

34176: fd_unlock例程用于对一文件描述符进行解锁。

76. compare_lexicography

34216: compare_lexicography例程用于比较两个表示数字的字符串，并决定他们是否有相似之处。

第3章 CGI和MIME模块

本章描述通过访问Apache服务器上的脚本来处理客户端请求并将脚本的结果返回给客户端的四个模块。这些脚本可以被客户端直接指定或由Apache处理例程指定，Apache此处理例程将一客户端请求重定向到一服务器脚本。

本章讨论的四个模块如下：

- mod_action——允许将一脚本定义为适用于MIME文件类型。
- mod_cgi——处理公共网关接口脚本的标准工具，当客户端请求的文件实际上是需要Apache执行的脚本时，Apache将调用mod_cgi模块。
- mod_mime——使用Apache配置文件定义的MIME类型给一请求文件指定文件类型信息。
- mod_mime_magic——使用一个magic数字文件来检查请求文件的内容，并且根据计算的信息给请求文件指定文件类型。

通过使用模块mod_mime和mod_mime_magic的文件分类功能，使Apache能够决定如何使用模块mod_cgi和mod_actions来处理请求。

3.1 mod_actions模块

通过联合使用AddHandler和Action指令，Apache能够使用模块mod_actions将一文件类型和一脚本联系在一起。只要客户端调用这种类型的文件，那么与此类型文件联系在一起的脚本就将被执行。

模块mod_actions在把文件内容发送给客户端之前使用脚本来预处理文件，这是一种很好的方法。例如：如果所有HTML文件都以一种压缩格式或作为文档数据库的一部分来存储，那么使用Action脚本以客户端可读的格式对这些文档进行预处理。（这种处理方法和内容协商方法不一样，因为在后一种处理方法中，Apache把压缩的文件传送给客户端浏览器，然后客户端浏览器对这些文件进行解压缩。）

如果想让模块mod_actions起作用，那么需要用AddHandler指令定义处理类型。接着使用Action或Script指令定义一过滤脚本，使模块mod_actions能够处理该类型的所有文件。Action指令用于定义文件类型；而Script指令用于定义HTTP方法（如：GET或POST）。例如：

```
AddHandler  compressed-html zhtml
Action compressed-html /cgi-bin/uncompress_html.sh
Script GET /cgi-bin/pre-process-html.sh
```

3.1.1 模块结构

模块mod_actions提供处理目录配置和合并目录配置的功能。命令列表定义两条指令为Action和Script。

当每次请求匹配指令Action定义的文件类型或Script指令定义的HTTP方法时，系统调用action_handler例程（11831行）。此例程检查指令中指定的脚本是否存在，接着执行一带有合

法脚本名请求的内部重定向。该处理过程如图3-1所示。

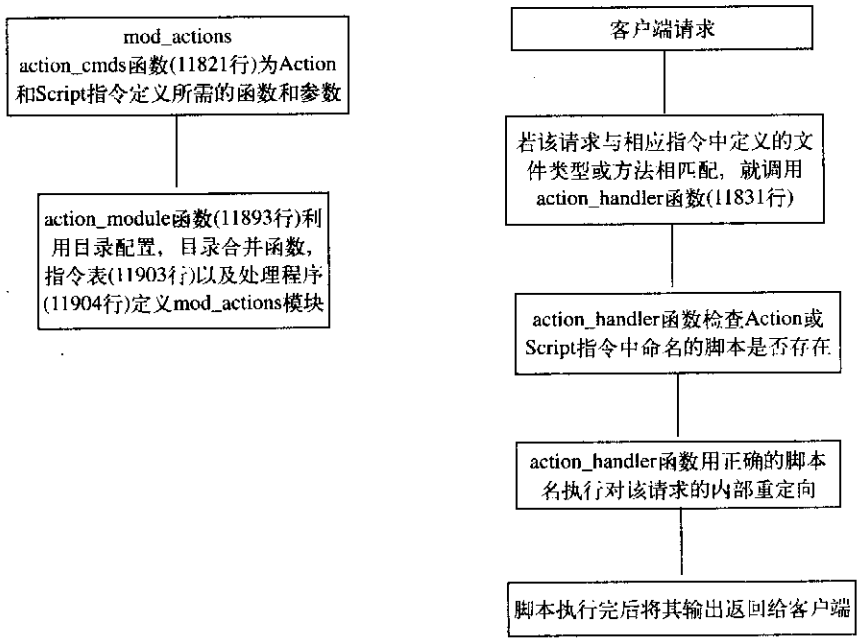

图3-1 当客户端请求为特定的文件类型时，通过mod_actions模块，Apache能够执行一段CGI脚本

3.1.2 定制

模块mod_actions非常简单，其功能是检查脚本是否存在，如果该脚本存在，则系统将控制权交给该脚本。这个模块可以用另外的指令来扩充，这些指令可以对不同类型的动作、脚本、文件进行映射。Action和Script指令能够扩充成包含另外能够指定在请求处理的哪个阶段执行该脚本的参数。以上这些定制必须往模块mod_actions增加处理例程和检查脚本的模块定义。

1. create_action_dir_config

11759: 为模块创建一配置记录，该记录用于一目录上下文中。例程创建一个有四个action_types类型的数组，同时初始化属性scripted。

2. merge_action_dir_configs

11772: 此例程通过联合两个目录的动作列表和脚本方法来合并两个目录配置记录。这两个初始配置记录通过参数basev和addv传送给例程。

3. add_action

11796: 此例程使用指令的参数设定目前配置结构的action_types属性。例程没有检查文件类型是否合法或要执行的脚本是否存在。在11823行为Action指令定义此例程。

4. set_script

11804: 增加Script指令设定的方法，同时将该指令加入scripted域。例程检查通过参数传递进来的方法是否合法。在11823行为Script指令定义此例程。

5. actions_cmds

11821: 定义此模块使用的两个指令需要的例程和参数。

6. action_handler

11831: 处理一个带有Action指令或Script指令的请求。

11847: 处理Script指令，检查方法，并且用 Script指令的脚本名称设定script变量。如果方法为GET，则例程检查args域；如果args域为空值，则例程将script变量设为空。

11860: 检查该请求是否为子请求。这样做是为了避免递归地调用同一请求的脚本。

11865: 通过比较action_types域和调用例程ap_default_type的返回值，决定一个Action指令是否适用于请求该文件类型。例程用模块mod_actions配置记录中的脚本名设定script变量。如果Action指令和Script指令同时适用一请求，则Action指令总是覆盖Script指令。

11869: 如果被请求的文件名没有找到，则例程action_handler将记录一条错误信息并退出执行。

11877: 如果Script指令和Action指令都不匹配该请求，则script为空，例程将返回DECLINED；否则，例程将调用ap_internal_redirect_handler来重定向该请求，例程将脚本名作为ap_internal_redirect_handler的一个参数，并且将args传递给该例程。

7. action_handlers

11887: 设定处理AddHandler指令的例程，这样文件可以使用AddHandler指令来访问模块mod_actions。这里使用“*/*”说明模块匹配所有的MIME类型的文件，因为模块通过action_handler例程检查文件类型（根据Action指令）。

8. action_module

11893: 设定模块的创建目录配置例程、合并目录配置例程、命令列表和处理列表。

3.2 mod_cgi模块

当客户端请求的文件实际上是一脚本时，模块mod_cgi的作用就是执行该脚本并将脚本的输出返回给客户端。

当模块mod_cgi准备执行脚本时，需多次进行安全性检查。如果配置了ScriptLog指令和与其相关联的指令（如：ScriptLogLength和ScriptLogBuffer），则模块在日志文件中将记录该脚本。如果要让Apache使用mod_cgi模块执行一脚本，则必须使用ScriptAlias指令定义包含脚本的一目录，或用带有文件名的AddHandle指令指定文档目录树下的哪些文件为可执行脚本。例如:

```
AddHandler cgi-script pl
AddHandler cgi-script cgi
```

3.2.1 模块结构

记录脚本调用是此模块代码的一大部分，通过在配置记录中设定指令参数来处理脚本记录指令。如果需要，模块将在调用脚本时，记录该脚本的错误信息。使用脚本记录功能是一个很好的Web服务器脚本调试手段。

与AddHandler指令设定的文件类型相匹配的请求，将通过cgi_handler例程来处理（15865行）。此例程多次检查脚本的安全性问题，以决定该脚本能否被执行。如果该脚本能够被执行

并且脚本文件存在，则此例程调用cgi_child创建一子进程（15791行）。此例程调用log_script记录脚本（15685行）。最后，例程cgi_handler将脚本输出信息传送给客户端。模块流程如图3-2所示。

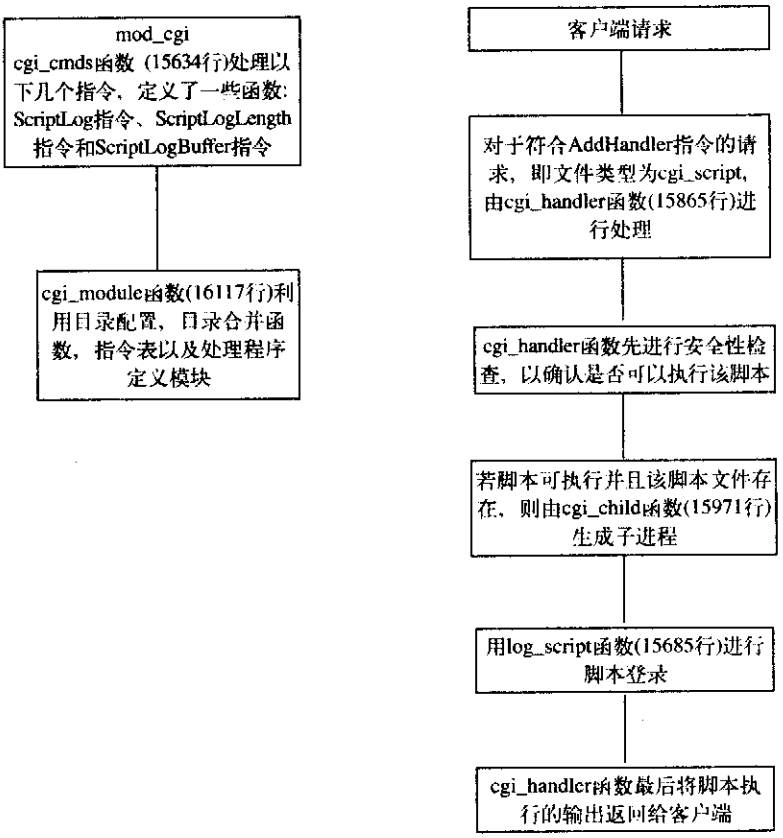

图3-2 通过mod_cgi模块，Apache可以执行CGI脚本

3.2.2 定制

与模块mod_actions相同，模块mod_cgi的目的是检查安全限制和将控制权传递给其他程序。用户可以通过执行脚本的安全性检查来扩充模块。虽然这种定制将使模块很难成为一个通用的系统，但有时某些Web服务器需要通过增加安全性来保证系统安全。

1. is_scriptaliased

15554：例程的功能是决定请求中的脚本名是否由ScriptAlias指令来指定，而不是由AddHandler指令指定。例程检查该请求的备注来决定文件类型是否是cgi-script.

2. create_cgi_config

15572：例程为服务器上下文创建一配置记录，同时初始化该记录的logging域。

3. merge_cgi_config

15586：为模块合并两个配置记录。

4. set_scriptlog

15595：使用ScriptLog指令的参数设定配置记录的logname域。在11823行为ScriptLog指

令定义此例程。

5. set_scriptlog_length

15608: 例程用指令ScriptLogLength的值来设定logbytes域，在命令列表中设定此例程为指令ScriptLogLength处理例程（15638行）。

6. set_scriptlog_buffer

15621: 例程用指令ScriptLogBuffer的值来设定bufbytes域，在命令列表中设定此例程为指令ScriptLogBuffer处理例程（15642行）。

7. cgi_cmds

15634: 定义模块mod_cgi使用的三个指令的处理例程。（三个指令为：ScriptLog、ScriptLogLength和ScriptLogBuffer）

8. log_scripterror

15649: 当模块处理脚本请求时，此例程试图记录一错误信息。15659行开始的这一组测试语句为一日志文件名检查合法性并且试图使用例程ap_pfopen打开该文件（15663行）。如果任一检查条件失败，则例程在15667行返回；否则，例程将一错误信息写入该文件（15672行）。

15672: 例程调用fprintf函数将错误信息输出到脚本日志文件中，错误信息的格式为：（日期、请求命令和错误信息）。

9. log_script

15685: 记录一成功执行的脚本。log_script被例程cgi_handler在16032行调用。

15698: 为当前的服务器上下文测试配置记录提供的日志文件是否合法。如果该日志文件存在，则文件大小不应该比logbytes域的值大，且该文件能够用ap_pfopen打开。

15708: 如果以上的测试不成功，则例程log_script将使用while语句调用ap_bgets来接收脚本的输出，并且例程在15714行返回。

15719: 如果脚本文件合法，则例程使用一组fprintf函数将一日志记录写入该文件。此刻，HTTP方法，请求文本和任何头信息或响应信息都将被记录在此文件中。

15751: 从脚本进程中检查缓冲区并且试图使用fputs调用将该输出写入脚本日志文件中。

15761: 重复上一过程，处理可能的脚本错误输出。

15771: 关闭脚本管道、脚本错误管道和日志文件的文件描述符。

10. cgi-child

15791: 将脚本当作模块mod_cgi的一子进程运行，例程cgi_handler使用sp_bspawn_child调用cgi-child。

15803: 打开一控制台设备作为将执行脚本的输出。设备是与Apache运行的平台相关的。

15819: 增加环境变量，使正在运行的脚本可以获得这些变量。调用ap_add_cgi_vars和ap_create_environment例程创建这些变量。

15823: 如果调试开关打开。则例程将消息写入返回的数据流中，这样以指示CGI脚本调用的进度。虽然大多数的模块没有调试代码，但这几行调试代码对跟踪不同的子进程很有帮助。

15839: 准备执行该脚本，接着调用ap_call_exec例程执行该脚本（用脚本名作为ap_call_exec的参数）（15841行）。这样应当有效的杀死当前例程，如果Apache运

行平台为Windows平台，则例程将在15843行返回。

15857: 如果例程还没有退出，ap_call_exec就不能正常运行，例程将在标准Apache错误日志文件中记录一错误信息并且退出。

11. cgi_handler

15865: 检查被请求脚本的合法性，接着执行该脚本并将结果返回给客户端。如果需要，记录该脚本。

15879: 测试用于该请求的方法是否允许调用，如果不允许，则例程返回DECLINED。

15887: 获得请求的文件名，并且设定argv0变量。

15894: 当设定脚本目录的ScriptAlias指令和ExecCGI选项没有激活时，访问该脚本会产生一错误信息。

15900: 如果被请求的脚本同时使用没有解析的标头和INCLUDED协议，则cgi_handler返回一错误信息并退出。

15913: 如果Apache运行在Windows或OS/2平台上，则例程通过测试脚本的扩展名是否为EXE来决定其是否可执行，接着测试该脚本是否存在。

15926: 在Unix平台上，通过检查finfo.st_mode请求域来确定脚本是否存在。

15931: 检验所请求的脚本不是一目录。

15936: 测试该脚本是否具有可执行权限。

15944: 如果脚本通过所有的测试，则系统创建一个cilent block，例程开始准备创建一子进程来执行脚本。

15948: 创建子进程变量。

15966: 调用ap_bspawn_child开始一子进程。该子进程使用脚本名称和处理输入输出的管道作为参数。

15981: 脚本应当已经运行，调用ap_should_client_block进行结果测试。分配变量来处理返回的数据流。

15989: 开始一超时处理，这里启动超时处理的作用是一旦读取脚本数据时有错误发生，那么Apache可以强行终止此进程。例程在16019行取消超时。

15991: 调用ap_get_client_block读取存储的数据到变量argsbuffer。

16005: 将变量argsbuffer中的数据输出到客户端。

16009: 如果写到客户端的数据的数量没有达到系统要求，则例程cgi_handler继续读取数据直到客户缓冲区为空。例程cgi_handler调用ap_get_client_block从客户缓冲区中读取数据。

16017: 调用ap_bflush清除脚本缓冲区，接着例程调用ap_bclose关闭客户连接。

16030: 检查标头信息，如果头信息正确，则调用log_script例程记录脚本并返回。

16042: 如果标准的动作没有发生，例程将对头信息进行重定向检查。如果例程发现一个Location，则它将读取所有的缓存数据。

16064: 将此重定向请求的方法设定为GET，并且去掉Content-Length头。

16076: 调用ap_internal_redirect_handler来重定向此请求。

16087: 如果脚本没有被重定向，则例程将头发送给客户端。如果它不是一个只有头的请求，则例程调用ap_send_fb发送脚本返回的数据。例程调用脚本ap_bclose关闭脚

本子进程（16091行）；任何保留的缓存信息将被清除，并且最后例程关闭脚本日志文件（16099行）。

16102: 例程检查脚本是否有返回数据，脚本是否使用NPH，如果检查结果为真，则例程调用sp_send_fb返回数据。

12. cgi_handlers

16110: 定义模块的handler_rec。

13. cgi_module

16117: 设定模块的创建目录配置例程、合并目录配置例程、命令列表、处理列表。所有需要模块mod_cgi处理的请求都要通过cgi_handlers的处理例程（16112行）。

3.3 mod_mime模块

模块mod_mime用Apache配置文件中的一组文件类型指令设定请求的各个域（语言，编码和MIME类型等）。Apache其他部分根据模块mod_mime设定的这些域来处理一请求。

3.3.1 模块结构

模块mod_mime定义配置记录，该记录包括语言、处理函数、编码、强制类型等域。（请看23272行的mime_dir_config结构。）模块调用create_mime_dir_config和merge_mime_dir_config例程初始化配置记录。用户可以使用指令如AddEncoding增加配置记录的项目。模块调用init_mime例程读取MIME类型配置文件，同时将该文件的配置信息读到表hash_buckets中。

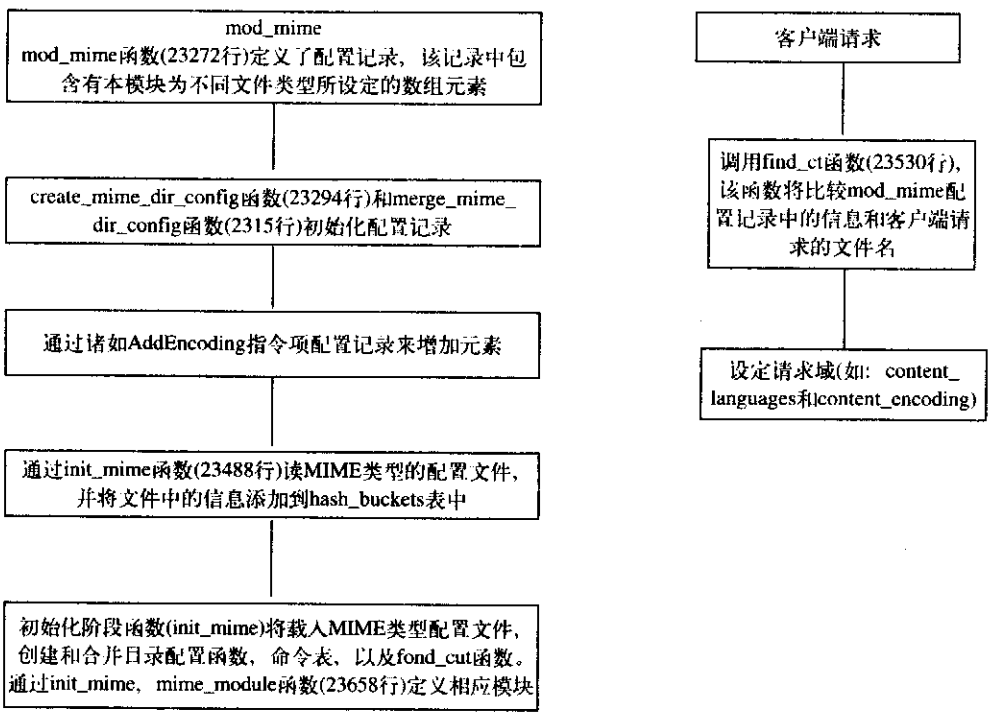

图3-3 通过mod_mime模块可以唯一中文当或文件类型定义相应的处理过程

每次模块处理一请求时，例程find_ct将被执行。该例程将配置记录中的信息和请求项目的文件名进行比较。根据比较的结果，例程设定请求的域（如：content_languages和content_encoding），其他模块可以使用这些域来检查文件的类型信息。模块流程如图3-3所示。

3.3.2 定制

模块mod_mime的功能主要由几个部分组成：1）读取指令参数并把这些参数放进一配置记录中。2）通过一文件名来比较这些参数。用户可以使用已有的mod_mime指令来增加文件类型、编码等。如果用户想增加一新的文件类型（目前模块mod_mime不能处理），则需要增加新的模块来处理此文件类型。利用模块mod_mime_magic能智能地处理许多文件类型（比如根据文件的内容而不是文件的扩展名）。

1. mime_dir_config

23272: 为模块mod_mime定义配置记录结构。该结构包含几个tables，每个tables说明一种文件类型，这些tables可以使用模块配置指令来处理。单一属性、处理者、缺省语言也是模块的属性，但它们是字符串类型，而不是tables。

2. create_mime_dir_config

23294: 为一目录上下文创建新的配置记录。使用上下文中的指令来设定四个空的tables的值，而记录中的type、handler和default_language被初始化为空值。

3. merge_mime_dir_config

23315: 合并两个目录配置记录，例程调用ap_overlay_tables覆盖这两个配置记录中的tables的值。（见23332、23334、23337、23340行）

4. add_type

23352: 例程读取AddType指令的参数来设定记录的forced_types属性。在23433行定义指令AddType的处理例程。

23356: 有些用户在指令参数中包括一句点来指定文件扩展名，如果例程发现有一句点，将丢弃该句点。例程没有执行检查以确定指令AddType指定的MIME类型是否合法;例程find_ct将该类型加入Content-Type请求头中(23561行)。浏览器能够处理服务器传过来的不合法的类型。

5. add_encoding

23364: 例程增加一编码类型到配置记录的encoding_types列表中，将文件扩展名和指令AddEncoding设定的类型进行匹配。在23436行定义指令AddEncoding的处理例程。

6. add_language

23375: 例程将指令AddLanguage设定的语言选项增加到配置记录的language_types列表中。为了使例程find_ct在比较项目时保持一致（23530行），例程将语言选项信息转化为小写。在23440行定义指令AddLanguage的处理例程。

7. add_handler

23386: 例程将指令AddHandler设定的处理者项目增加到配置记录的handlers列表中。此例程是为23444行的指令而定义的。与处理其他带有文件扩展名的指令相同，例程将丢弃文件扩展名包括的句点，并且例程为了以后的比较操作，将处理者名称转换为小写。

8. remove_handler

23403: 定义一个在当前目录上下文中请求的文件扩展名不可能得到的处理者。在23452行定义指令RemoveHandler的处理例程。

9. set_types_config

23423: 例程将MIME类型配置文件的文件名称增加到模块的配置记录中，这样例程init_mime可以访问此文件（23488行）。ap_set_module_config函数用于直接设定此信息，因为该函数只用于配置服务器项目。其他此模块指令提供的例程都用于目录配置。例程在打开该文件之前使用init_mime函数测试文件名是否合法。在23459行定义指令TypesConfig的处理例程。

10. mime_cmds

23431: 定义此模块的配置指令和相应的指令处理例程。此结构中的每条记录都定义模块的配置指令和解析该指令所要调用的例程。注意此模块有三条指令（ForceType、SetHandler和DefaultLanguage）使用相同的例程处理指令参数，实际上这三条指令都调用例程ap_set_string_slot和一个指向配置记录的索引（如：XtOffsetOf(mime_dir_config,type)设定配置对应域的值。每条指令都有相应的注释信息，如果某一指令的参数信息不符合定义格式，则Apache将该指令的注释信息打印出来。

11. init_mime

23488: 此例程读取MIME类型配置文件到数组变量hash_buckets中（23486行）。Apache在init阶段调用此例程。

23493: 读取模块配置信息到变量confname中。配置信息只是MIME类型配置文件的文件名。

23503: 例程调用ap_pcfg_openfile打开MIME类型配置文件。如果文件打开不成功，则init_mime打印一出错信息并退出。

23510: 循环检查数组hash_buckets，将一项目中的空表项赋到MIME_HASHSIZE中(见23482行)。

23513: 使用ap_cfg_getline读取MIME配置文件的每一行，并测试每一行的第一各字符是否是#，如果是，则该行为一注释行；如果不是，则例程调用ap_getword_conf将该行的第一个字符赋给变量ct，同时例程调用ap_getword_conf解析该行的剩余的内容（23512行），最后将该行的所有字符放进数组变量hash_buckets中。

23527: 关闭配置文件。在请求处理过程中，例程find_ct可以得到数组变量hash_buckets。

12. find_ct

23530: 根据此模块指令定义的信息来设定当前的请求域。此例程对本模块的请求执行真实的处理。

23532: 用该请求的文件名设定变量fn，在23552行例程检查变量fn是否为空，如果是，则重新设置该变量。

23538: 将请求结构中的原始处理者域保存在变量orighandler中。

23541: 如果当前请求是目录请求,而这种请求模块mod_mime不支持，例程find_ct返回。

23557: 解析文件扩展名，文件扩展名可以以任何次序组成（例如：.html.it.gz或.it.html.gz）。这一行的while循环语句为发现的每一文件扩展名设定一适当的域。变量ext保存当前正在测试的文件名。

23562: 例程调用ap_table_get检查配置记录中的forced_types和hash_bucket域是否有一个匹配变量ext，如果是，则例程用匹配的域的值来设定请求结构的content_type域。

23572: 例程调用ap_table_get检查配置记录中的language_types数组域中是否有值匹配变量ext。如果是，则例程用匹配的单元值设定请求结构的content_language域(23576行)。

23579: 如果域content_language为空,则例程调用ap_make_array和ap_push_array函数设定域content_languages，这是为了以后的兼容性。

23589: 例程检查配置记录中的encoding_codes域中是否有匹配变量ext的元素。如果有，则例程检查是否编码已存在，如果编码不存在，则例程用最近匹配的元素值设定请求结构的content_encoding域（23592 行）；如果编码已存在,则例程用编码值，逗号和刚才匹配的元素值设定content_encoding域（23594行）。

23603: 例程检查配置记录中的handler域是否有匹配的处理者类型（根据AddHandler指令），如果有，则例程用该域的值设定请求结构的handler域。这只有在请求不是代理请求时才合法（23604行）。

23616: 如果while循环中发现语言、类型、编码中任一项目，则该循环重复。如果没有发现任一项目，则所有的值都将设为NULL。

23631: 如果请求结构中的content_language域为空并且定义了的default_language域，则例程用default_language域的值设定content_language域（23635行）。例程在23637行重复上一步的操作。

23647: 如果用户使用ForceType指令设定一目录，使该目录下的所有文件都为某种类型，则例程重新设置content_type域。同样，如果用户用SetHandler指令覆盖所有的AddHandler指令，则例程用当前配置记录中的处理者域的值设定请求结构中的处理者域。

13. mime_module

23658: 定义模块的init例程，创建和合并目录配置例程，处理模块指令的命令列表和模块处理客户端请求的类型检查例程。

3.4 mod_mime_magic模块

模块mod_mime使用指令提供的值来设定内容类型和请求文件的有关属性。相反，模块mod_mime_magic并不是通过检查Apache有关文件扩展名和数据类型的指令，而是通过检查文件的内容来决定内容类型和有关信息。

模块的文件分类的方法使用magic文件，大多数Unix和Linux系统都拥有该文件。MimeMagicFile指令能够指定文件，该文件包含用于识别文件内容的magic数字串。

3.4.1 模块结构

模块mod_mime_magic在Apache init阶段将MIME magic文件载入请求结构中。当模块处

理一请求时，在type_checker阶段调用magic_find_ct例程，该例程顺序调用magic_process和magic_rsl_to_request来检查文件的内容和比较文件的头几个字节和magic数字串。许多单个的例程用于比较字符串次序、转变数字格式等，因此文件可以是二进制格式、压缩格式或ASCII文本格式。

在模块处理请求期间，将执行两种类型的测试。第一种类型测试是根据文本字符串的比较，例如：一包含字符串“<DOCTYPE”预示该文件的内容类型为text/html。第二种类型测试查找在文件某位置的字节顺序是否与magic文件中描述的信息一样。这种测试方法用于识别二进制文件类型。模块结构如图3-4所示。

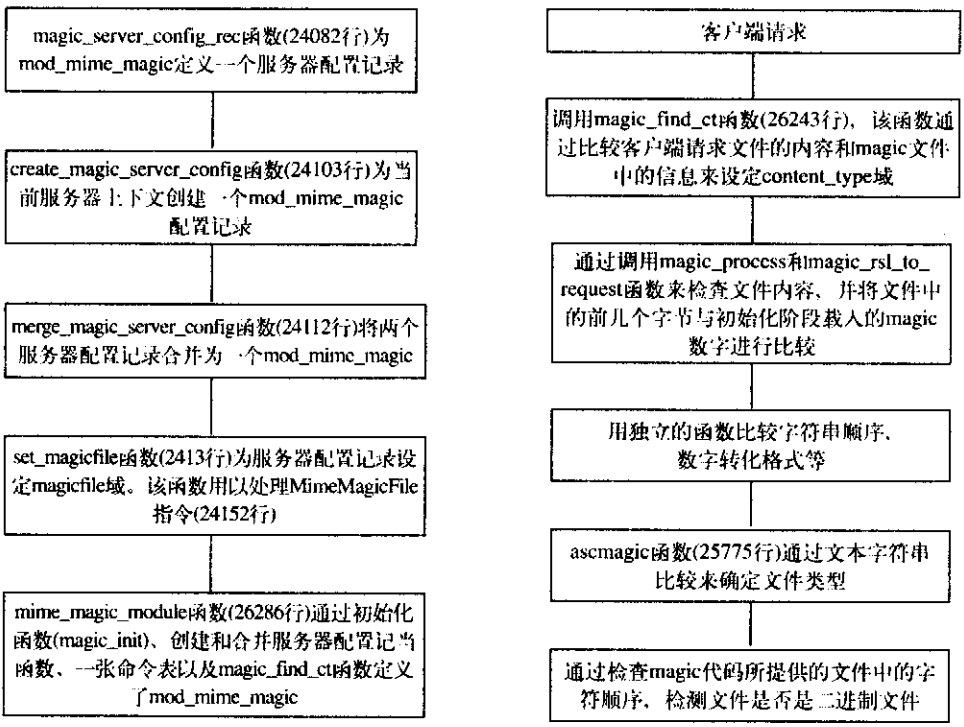

图3-4　mod_mime_magic试图确定一个文件内容的类型，并由其内容判定其相关信息

3.4.2 定制

用户如果需要支持另外的文件类型，则只需在magic文件中增加该文件类型的描述串。

代码的少数几个地方在某些情况下会发生错误（如：多线程环境下）。用户可以定制此模块来修正这些错误，提高系统的性能。

模块mod_mime_magic包括大量调试代码。因为模块mod_mime_magic控制不同例程的数目、字符串的类型和文件，所以当对模块做任何修改时，这些调试代码对用户将很有帮助。关闭调试开关并不影响Apache的大小和编译速度。

mod_mime_magic包含大量调试代码，由于被mod_mine_magic执行的不同函数、字符串类型和文件操作的数目如此之多，当修改模块时，这些代码非常有用。当调试选项打开时，它不会影响Apache服务器的编译速度和大小。

1. Function Prototype Declarations

23818: 开始定义文件功能的原型函数。这些功能函数作为原型定义在这里，因为他们可以相互调用。这些原型函数没有调用其他没有定义过的函数。

2. types

23878: 定义存储content_type域值的结构，模块mod_mime_magic将分配content_type给请求文件。

3. names

23898: 定义带有许多字符串的结构变量，用于测试文本文件的头几个字节，从这些字符串中找出一个来识别该文件。就处理速度和正确地处理而言，这些字符串的次序是很重要的。它们的逻辑关系是比较复杂，例如："the"同时适用于文本文件和HTML文件，但HTML文件是更严格的分类文件，因此，模块将使用结构变量中的字符串来首先检查HTML文件。

4. magic_rsl_s

24070: 定义结构，模块在处理决定文件内容类型期间，使用该结构收集RSL碎片。

5. magic_server_config_rec

24082: 为模块mod_mime_magic的服务器配置记录定义结构。

6. create_magic_server_config

24103: 为目前的服务器上下文创建模块配置记录。例程使用ap_pcalloc分配magic_server_config_rec对象。

7. merge_magic_server_config

24112: 为模块合并两个服务器配置记录。

8. set_magicfile

24131: 设定服务器配置记录的magicfile域。这个文件中包含用于识别文件内容的magic数字串和指令。在24152行定义的例程用于处理MimeMagicFile指令，使用这条指令，用户可以明确地定义magic文件。

9. mime_magic_cmds

24150: 定义处理模块指令MimeMagicFile的例程，该例程放置指令参数到服务器配置记录。

10. magic_set_config

24176: 建立存储当前请求信息的记录。在24182行例程将结构链表的头和尾指针设为空值。例程调用ap_set_module_config将新变量放置到配置记录中，并且该新建的变量返回给调用例程。

11. magic_rsl_add

24192: 例程增加字符串到此请求的结果字符串列表中。

24200: 测试变量reg_dat是否可用于存储传递进来的数据，如果变量合法，或如果magic_set_config例程没有成功返回，则例程magic_rsl_add记录一错误信息并返回。

24212: 调用ap_palloc分配magic_rsl对象。

24216: 用str的值设定rsl变量。

24220: 增加变量rsl值到reg_dat链表中，如果这是链表的第一项，则执行24225行代码。

12. magic_rsl_puts

24233: 调用例程magic_rsl_add，增加一字符串。

13. magic_rsl_printf

24239: 使用Apache函数调用将打印RSL链表变量信息。

24248: 调用函数ap_vsnprintf打印一字符串，接着调用函数magic_rsl_add增加一项目，并将该函数的返回值返回。

14. magic_rsl_putchar

24256: 使用例程magic_rsl_add将一字符插入到rsl_add链表中（24264行），例程事先在该字符后面设定一NULL值。

15. rsl_strdup

24269: 分配且复制一连续的RSL字符串。

24284: 使用ap_palloc为一结果字符串分配空间。

24288: 循环查询RSL的每一部分，直到发现当前碎片。

24296: 循环查询所有的碎片，收集碎片的字符并赋予result变量。

24312: 给一字符串设定NULL结束符，且返回result变量。

16. rsl_states

24324: 为magic_rsl_to_result例程的state-machine算法定义枚举类型，该枚举类型包含算法的各种状态值。

17. magic_rsl_to_request

24331: 处理结果字符串链表并使用该链表设定请求记录中的MIME信息。

24354: 获取服务器配置记录的数据。

24359: 如果req_dat变量不存在或为空，此例程没有要使用的数据，则例程返回DECLINED。

24366: 例程为匹配magic文件中的片段和RSL链表，开始循环搜索。

24373: 循环检查碎片的所有字符，如果发现一空格，则例程将忽略它。

24380: 使用if/else语句决定如何识别state变量的content_type。

24417: 开始处理碎片链表中的无空格单元。

24468: 测试state变量是否设置成合法的RSL子类型或RSL编码。如果是这样,则例程收集类型字符串并用它设定content_type请求域。

24486: 如果可获得合法的信息，填充请求记录的content_type域。

24490: 测试content_type是否已设定,如果没有,则触发一内部出错信息。

18. magic_process

24506: 使用magic文件信息准备处理请求文件。此例程打开该请求文件并读取文件的最前面几个字节，然后开始进行测试。

24519: 例程调用25155行的fsmagic例程测试文件的状态。如果此文件状态为DONE，则例程接着调用magic_rsl_putchar例程。

24523: 如果fsmagic返回OK，则例程接着打开该请求文件。如果fsmagic返回其他值，则一内部出错将发生，因此例程没有打开此文件，并返回。

24530: 例程调用ap_popenf打开请求文件,并用该文件描述符设定变量fd。如果文件不能

被打开，则magic_process记录一出错信息并返回。

24543: 使用读函数读取文件的前几个字节。如果例程在读文件时有错误发生，则magic_process将记录一错误信息并返回内部服务器出错代码。

24550: 检查已读取的文件字节的状态。如果没有任何字节被读取，则例程以MIME_TEXT_UNKNOWN作为参数来调用magic_rsl_puts。否则，调用tryit例程，以决定该文件中包含的信息。

24557: 调用ap_lclosef关闭文件，最后执行例程magic_rsl_putchar以知道文件的内容。

19. tryit

24564: 执行几种方法测试文件的内容类型。

24570: 调用zmagic测试该请求文件的压缩编码。此例程可以依次调用tryit来检查非压缩文件的content_type。在处理过程中，zmagic设定请求记录的content_encoding 域。

24576: 调用softmagic例程(25233行)以决定magic文件是否有信息可以帮助判断一非ASCII文件的content_type。

24582: 调用ascmagic例程检查一文件是否是ASCII文件，如果是，则例程测试该文件的前面几个字符是否匹配任何简单测试。

24588: 如果所有其他的测试都不成功，则例程调用magic_rsl_puts函数(用一MIME_BINARY_UNKNOWN作为该函数的参数)。

20. apprentice

24598: 从MIME magic文件中为模块mod_mime_magic载入配置信息。

24610: 为当前上下文读取服务器配置信息。

24615: 例程用magicfile给fname赋值且调用ap_pfopen打开该文件。如果错误发生，则例程apprentice记录一错误信息并退出。

24625: 用NULL值初始化配置记录的magic属性。

24628: 读取该magic文件的每一行，检查每一行并将每一行的内容复制到变量中。

24639: 移去该行前面的所有空格。

24645: 如果当前行为空行或注释行，则例程跳到下一循环。

24660: 调用parse函数处理此文件的一单行。

24664: 读取该文件的所有行，例程调用ap_pfclose关闭文件。

24666: 如果调试开关打开，则例程将检查几个潜在的错误是否发生，如果有，则记录这些出错信息。

21. Signextend

24716: 为变量扩展符号位，用于有符号数的比较。

24720: 用一个switch语句检查变量m->type的类型。根据case所对应的类型，将m->type强制转化为相应的char、short或long类型。

24744: 若所列出的case语句均不满足，表明有错，由signextend记录错误并退出。

24751: 返回强制类型转换后的值v。

22. parse

24758: 从magic文件中分析出一行，并将正确的行赋给magic结构。在将magic文件读入mod_mime_magic时（参见24660行），由apprentice函数调用本函数。

24763: 为上下文检索服务器的配置记录。

24774: 测试链接后的列表状态，该列表包含由magic文件中取出的行。

24782: 初始化magic结构的值。

24787: 在while循环中开始分析magic文件中的一行，查找特殊字符。

24814: 根据magic行中的数据，相应的设定数据类型的值（如long、byte）。若无case语句匹配，转到24824行的default语句，记录错误。

24833: 检查数字前的“+”（正号）和“-”（负号），在24853行的while循环中要用到这些符号。

24875: 参照已分配的类型开始测试数据的类型，根据测试结果设定m的type域，若未找到确定的值，记录错误。

24929: 若找到一个“&”符号，则调用signextend函数，该函数把mask域置为逻辑“与”。

24935: 用一个swtich语句监测magic行中的其他字符。每一个case语句查找一个正确的字符，并把该字符赋给reln域。若无case语句匹配，则由default语句用ap_isspace查找空格符。

24971: 从当前magic行的最后一部分取到该行的描述，然后把它赋给m的desc域(24982行)。

23. getvalue

25003: 根据magic的类型，利用一个指针读取一个数值，该指针在magic指针的value联合中。读完后，更新字符串指针为该指针。

25008: 若所检测元素是一个字符串，则用getstr设定p的值；否则，通过调用strtol来调用signextend（25014行）。

24. getstr

25025: 将包含转义符的字符串转换为非包含转义符格式，遇到第一个空格或“tab”，则停止。

25034: 循环检测字符串中的所有字符，用ap_isspace确定是否有空格。

25043: 用一个switch语句查找转义符。在每一条case语句中，设置p中的字符串为非转义符值。数字将转化为取值。

25102: 多次调用（25104行，25107行和25110行）hextoint函数（参见25137行）来转化字符串中的十六进制数。

25. hextoint

25137: 把一个一位十六进制数转化为十进制值。25139行调用ap_isdigit函数用以检测数字，对于非数字字符，分别由10~15代替a~f（或A~F）。

26. fsmagic

25155: 检测作为参数传入的文件句柄的状态。若该文件已处理，则返回DONE；否则返回OK，指明需要处理该文件。

25157: 根据finfo.st_mode的值，用一个swtich语句为本次请求的文件名分配一个状态值。

25158: 若文件句柄指向一个目录，则返回DONE。不支持文件分类。

25161: 对于S_IFCHR文件，fsmagic用一个未知的文件类型作参数调用magic_rsl_puts函数，并返回DONE（对应于字符设备）。

25170: 对于S_IFBLK文件，fsmagic用一个未知的文件类型作为参数调用magic_rsl_puts

函数，并返回DONE（对应于块设备）。

25182: 对于S_IFIFO文件，fsmagic用一个未知的类型作为参数，调用magic_rsl_puts函数，并返回（对应于FIFO类型文件）。

25190: 对于S_IFLNK（链接的）文件，记录一个错误。

25206: 对于一般的文件，fsmagic跳出到循环尾。

25208: 若无case语句匹配，fsmagic则记录一个内部服务器错误并返回。

25219: 若文件长度为0，调用magic_rsl_puts函数并返回DONE；否则返回OK（文件类型还需要处理）。

27. softmagic

25233: 从apprentice函数载入的magic文件中查找一个文件，调用匹配函数来检查输入的项，若匹配成功，则返回1；否则，返回0。

28. match

25276: 根据magic文件中定义的类型检查所请求的文件，以期查找到相匹配的类型。许多调试检测程序都是本块代码的一部分，但在此不作讨论。一般而言，这些检测程序会把进程的相关信息记录下来，或者登记更细节化的出错信息。

25285: 为当前上下文检索服务器配置记录。

25323: 循环查找magic文件中的各行，通过调用mget函数(25562行)和mcheck(25602行)为所请求文件找出一个匹配。

25341: 若mget或mcheck函数返回一个匹配的值，则删除先前这部分完成的匹配的相关记录。

25379: mget和mcheck函数返回了一个匹配结果，由mprint函数对该结果进行处理。

25388: 主匹配层已经完成，继续检查以下各层的匹配情况。

25411: 通过mget和mcheck函数检查以下各层的匹配情况。若匹配成功，则通过mprint函数将结果打印出来(25427行)。

25449: 对应于magic文件各行的循环结束。

29. mprint

25459: 根据传入参数的数据类型，向结果字符串列表(RSL)中打印一个值。

25465: 用一个switch语句检测变量类型，变量V的值根据不同的类型而设定。

25482: 对于字符串变量，直接调用magic_rsl_printf函数打印该变量。如果没有语句能匹配该变量，则出现内部错误。

25509: 采用符号扩展功能，对v进行符号检查，然后用magic_rsl_puts函数打印出来。

30. mconvert

25517: 转换数据的字节顺序，以供mget和mcheck函数所用。

25522: 检查变量类型。一系列的case语句重新排列数据，如果有必要，则进行循环移位。如果检查成功，则返回1（有些情况下，返回1但没有任何动作）。如果执行了缺省语句，则表明为找到该变量类型而出错，并且记录该错误。

31. mget

25562: match函数将会调用该函数以处理变量p的一个字符。

25569: 如果检查到的偏移大于n个字节，则没有足够的数据可用，mget直接返回。

25579: 检查数据类型，根据p和m的值设置偏移。

29. mcheck

25602: 为一个变量rsl的组件检查数据类型。

25609: 如果正在检查的值是十六进制数或者为空，mcheck记录错误并返回。在调用mcheck前，该值应该已由十六进制数转化为十进制数。

25617: 使用switch语句检查m->type变量的数据类型。根据m->type的类型将v的值设为p->b、p->h或p->l。字符串创建了一个特殊的情况，如果没有任何情况与该数据类型匹配，则记录一个内部错误。

25668: 采用符号扩展函数设置v的符号。

25670: 采用另一个switch语句测试变量的域。这些case语句中的每一个都会为匹配的变量设一个值。如果设置了MIME_MAGIC_DEBUG标志，mcheck记录出错信息，以标记该情况被选中。

33. ascmagic

25775: 比较所请求文件前n个字节值中的明文字符串和类型结构中的值，以期分配一个content_type。

25794: 找一个起始段，该段后有一个关键字符，此字符用以标志一个troff格式的文件。若能找到，则在25801行通过调用函数magic_rsl_puts返回类型值application/x-troff。

25805: 检查Fortran字符，但用magic_rsl_puts函数返回text/plain。

25815: 用少量字节（考虑到性能因素），检查位于name.h文件中的项的匹配。

25829: 循环遍历每个名字。如果有匹配，则通过调用magic_rsl_puts函数（25831行）将其类型置入请求记录中。

25840: 使用一个swtich语句来检查is_tar（26035行）函数的返回值，并判断这是否为一个tar文档文件。适当的内容类型值会使用magic_rsl_puts函数送出。

25856: 没有任何额外的检查是成功的，但该文件仍是ASCII码文件，所以类型会通过magic_rsl_puts函数被设为text/plain。

34. compr

25880: 定义一个用于压缩信息的结构。zmagic函数（25905行）用它来标识压缩文件的类型（编码类型）。

35. zmagic

25905: 判断请求文件的压缩状态。

25912: 循环遍历n个字节，检查存储在压缩结构中（25880行）与各种压缩文件类型相关的值。如有匹配，zmagic函数退出for循环并检查匹配。

25920: 如果到达链表尾，zmagic返回0——该文件没有被压缩。否则，它会调用位于25923行的uncompress函数（开始于25966行）来判断未压缩文件的大小。如果返回一个有效值，调用25925行的tryit函数，它将尽可能的判断未压缩文件的内容类型。

25928: 因为已进行了一个压缩检查，zmagic设置content_encoding域的值为在compr结构中所给出的值。

36. uncompress_child

25939: 为uncompress操作提供函数，这会在uncompress函数中派生一个子进程。

25951: 如果Apache运行在Win32上，必须为子进程ID作一些额外的准备。

25956: 在非Win32平台上，execvp调用被用于引发子进程，如果产生错误，控制就会停留在这个函数；错误会被记录。否则，该函数失去控制，因为子进程通过exec调用进行接管。

37. uncompress

25966: 尽量解压一个文件并判断它的编码类型，该函数的调用在zmagic函数中(25923行)。

25982: 通过ap_make_sub_pool来创建一个子缓冲池，防止zombie进程作为一个子进程被派生来运行解压应用。

25984: 采用ap_bspawh_child调用来启动一个子进程解压文件。该调用包括uncompress_child函数（开始于25939行），如子进程没有启动，uncompress记录一个错误并返回。

25993: 检查子进程是否正常，然后释放上述内存子缓冲池。在25999行通过函数ap_bclose关闭子进程。

26001: 通过ap_bread函数取得待解压文件的文件大小，并存入变量n中。若有错，则解压失败，uncompress记录错误并退出；否则释放保留的内存子缓冲池，并返回待解压文件的文件大小n。

38. is_tar

26035: 根据nbytes中的信息判断所请求的文件是否是一个tar文件。本函数由ascmagic函数在25840行所调用。

26043: 若nbytes的大小不足以进行一个正确的测试，则is_tar直接返回。

26046: 用from_oct函数（26080行）检索文件的checksum，这是判断tar文件的关键。

26050: 对所有数位求和。在26060行的循环中，通过移去checksum字节来调整checksum。

26064: 将from_oct调用的返回值与程序26060行到26062行求得的和进行比较，若不等，则不是一个tar文件；若相等且和TMAGIC头也相等，则是一个标准Unix tar文件；若相等但与TMAGIC头不等，则是一个老版本的tar文件。

39. from_oct

26080: 把一个八进制数转化为十进制数。

26084: 根据ap_isspace函数的返回值进行循环，直到在输入字符串中找到一个非空格符。若未找到可用字符，该函数直接退出。

26089: 在while循环中累加所有的八进制数位，然后返回累加值。

40. revision_suffix

26114: 检查所请求文件的修正后缀（如@1），若存在，则创建一个子请求处理修正的文件名。由于本模块定义了MIME_MAGIC_DEBUG，因此本函数中有一些地方仅在调试打开时才有效。

26128: 去掉所请求文件名的最后一个字符。在程序26129行调用了ap_isdigit函数检测该字符是否为数字，若不是，则不存在后缀，revision_suffix函数直接返回。

26132: 循环检测文件名中的字符，直到找到了一个非数字字符，然后在26135行进行再测试：若后缀的位置索引值（suffix_pos）小于0或第一个非数字字符不是“@”，则revision_suffix函数返回，这是因为无法找到要处理的后缀的类型。

26143: 在sub_filename函数中创建一个修正的文件名，然后在26151行调用ap_sub_req_lookup_file函数生成一个子请求，子请求返回的信息将赋给主请求的某些域。尤其是当子请求有content_type域时，则子请求的content_type值、content_encoding值和content_language值将分别在25156、26165和26168行拷贝到主请求的相应域。

26174: 调用ap_destroy_sub_req函数删除子请求。

41. magic_init

26183: 把MIME magic配置文件载入服务器配置记录中，以初始化mod_mime_magic函数(26289行)。

26193: 为主服务器获取配置记录。26195行的循环语句用来查找是否存在一个更新的可视服务器上下文配制记录，若存在，则以该值设置conf；否则，在26200行用main_conf中的值设置conf。

26202: 如果服务器配置记录中有一个magicfile值，则调用apprentice函数（24598行）载入其值，以供mod_mime_magic函数所用。

26206: 如果apprentice函数成功返回，则模块由此退出；否则，若该模块的调试状态使能，则打印magic文件中的相关信息；如果apprentice函数无效，mod_mime_magic函数则什么也不做。

42. magic_find_ct

26243: 比较所请求文件的上下文和magicfile中的信息，以此判定所请求文件的content_type域。

26249: 若文件系统不支持所请求的文件，magic_find_ct不再判断文件类型。

26253: 若该请求的content_type域已定义（或许是由mod_mime模块定义过了），则不应再用mod_mime_magic模块去创建文件类型。magic_find_ct函数在26255行返回值DECLINED。

26258: 为该模块检索服务器配置信息。若配置不存在，则magic_find_ct函数直接返回。

26265: 用magic_set_config函数(24176行)在服务器配置中建立一条记录，用以处理该请求。若调用该函数失败，或访问服务器配置记录时出错，则返回一个内部服务器错误。

26269: 去除文件名中的所有修正后缀，这些后缀通常是形如“@1”或“@2”的某些东西。这些工作由revision_suffix函数（26114行）进行，该函数或许会用到一个子请求来处理改变的文件名。

26272: 用magic_process函数（24506行）处理所请求文件。该函数将准备一个缓冲区，打开文件并读出一些字节信息，以备判别magic文件的文件类型所用。若一切正常，则magic_find_ct函数开始执行（24331行开始），它通过调用magic_rsl_to_request函数为该请求指定一个文件类型。

43. mime_magic_module

26286: 通过以下几项工作来定义mod_mime_magic：一个初始化函数（magic_init），一些创建和合并服务器配置记录的函数，该模块要用到的单个指令的命令列表，以及magic_find_ct函数（26243行），它用来为所请求文件指定一个content_type。这些工作都在处理请求的type_checker阶段完成。

第4章 核心代码

本章描述Apache源代码的核心部分。他们是控制Apache模块的的文件，包括Apache API调用和其他函数，这些函数用于启动真正处理客户端请求的Apache主程序和子程序。

本章叙述的三种文件是:

http_core——包括处理指令的核心模块，该核心模块不与任何特殊的模块相关联。

http_main——包括Apache初始化、管理多进程、内存分配和Apache记分板的主程序。

http_log——包括将错误信息写入文件及管道日志中的函数。

所有这三种文件包含比其他的大多数核心代码模块更多的代码注释。这些注释对理解代码有很大帮助，可以避免你对编译器定义的为数巨大的对Windows系统的说明产生混乱，使你清楚哪行代码是为你的平台所写的。

4.1 http_core模块

尽管不能像对待其他Apache模块那样看待此模块，但http_core在很多方面与常规模块相似。事实上，在源代码中关于http_core的说明部分把它当作冗余模块。它包括一个模块结构以定义一套在进程不同阶段中的函数调用。此核心模块所完成的主要工作包括处理Apache配置文件中全局适用的指令。全局变量由基于这些指令的函数设置。<VirtualHost>和<Directory>等容器类也可以被处理，基本配置记录也可建立，这些记录由其他模块在处理过程中加入或查询。

http_core文件有很多对源代码很有帮助的说明注释。对core_cmds阵列中指令的研究也是非常有意义的。

1. create_core_dir_config

57: 创建核心或缺省目录配置记录。这是顶级的目录配置，当没有<Directory>结构在配置文件中覆盖它时使用。每个虚拟服务器都有核心目录配置。

61: 分配一个目录配置记录（conf）。

64: 在d域替换目录名，监视特殊情况，例如代理。

77: 设置其他域，包括可选项（opts）和override。

2. merge_core_dir_configs

111: 合并核心（缺省）目录的配置数据。basev 和newv域的变量被组合进conf变量，该变量是此函数的返回值。

3. creat_core_dir_config

251: 创建缺省服务器配置的记录，设置conf变量的域值。每个虚拟服务器都有一个核心服务器配置记录。

4. merge_core_server_configs

272: 合并核心服务器配置记录。basev和newv变量被组合进变量conf，该变量是此函数的返回值。

5. ap_add_per_dir_conf

303: 服务器配置记录中增加一个目录配置。

306: 检索服务器配置到sconf。用ap_push_array将一个阵列元素加到310行，然后dir_config作为一个参数传递给配置记录中新创建的槽而被拷贝下来。

6. ap_add_per_url_conf

315: 服务器配置记录中增加一个per-URL (<Location>)配置。服务器配置被检索，一个新数列元素被分配，per-URL配置作为一个参数传递给服务器配置记录并被拷贝下来。

7. add_file_conf

327: 服务的配置中增加一个per-file配置(由<File>容器类定义)。分配服务器配置记录中空间，然后插入新的per-file配置。

8. reorder_sorter

370: 这个函数被qsort 引用，提供被qsort排列的元素。(配置记录)的顺序由qsort决定。recorder_sorter在382、383、387行使用IS_SPECIAL宏执行一个测试，该测试是针对包括规则表达式的配置。可参见358行的源代码注释。

9. ap_core_recorder_directories

407: 将目录配置排序，以便其以正确的顺序使用。由配置信息建立一个数组，在436行调用qsort来对这个数组排序，这个排序的配置信息将建立一个新的数组。

10. ap_allow_options

460: 这个API调用返回当前目录配置记录的opts域 (目录选项)。这个域定义了此目录上下文中哪个选项可用。

11. ap_allow_overrides

469: 这个API调用返回当前目录配置的overrider域。这个域决定了针对目录的配置文件可重写的项。(配置文件如htaccess文件)

12. ap_auth_type

479: 这个API调用返回配置给当前目录的权限类型。

13. ap_auth_name

489: 这个API调用返回配置给当前所授权目录的名称(ap_auth_name域)。

14. ap_default_type

499: 这个API调用返回从当前目录得到服务的缺省内容类型。

15. ap_document_root

511: 这个API调用返回当前服务器的文档的根目录。这个函数颇受非议(不应该再用了)，但是为了保证NCSA Web服务器的向后兼容性而不得不保留。

16. ap_requires

522: 这个API 调用返回当前目录配置中的ap_requires域，指明必须经授权才能访问此目录的用户。

17. ap_satisfies

533: 这个API调用返回当前目录配置中的satisfy域。

18. ap_response_code_string

550: 这个函数很少用到。它返回当前目录配置中的response_code_strings域(如果定义了)。

19. do_double_reverse

567: 在对客户连接的网络信息中执行双重逆序查找的动作。这一安全措施保证了IP地址和主机名均有效(一一致)。

581: 用 gethostbyname查找客户机的主机名。结果将与586行的请求域(remote_addr)相对比。

20. ap_get_remote_host

596: 这个API调用查找远程客户的主机名，以发出请求。

609: 检查当前目录配置中的hostname_lookups是否允许。

630: 如果hostname_lookups允许，调用gethostbyaddr查找主机名，并将其复制到633行的请求记录中。如果定义了HOSTNAME_LOOKUP_DOUBLE,在639行将调用do_double_reverse。

21. ap_get_remote_logname

684: 这个API调用返回当前请求的远程登录名。此信息在699行通过调用ap_rfc1413，用conneccion和serves域的请求记录来查找。

22. ap_get_server_name

713: 这个API调用返回服务器名，服务器是由ServerName命令所定义的当前的虚拟主机。可参见706行的源代码注释。

23. ap_get_server_port

728: 这个API调用返回一个连接所初始化的服务器端。如果请求记录的server.port域含有此数据，它将被作为返回值。否则通过调用737行的ap_default_port函数来获得缺省端口作为返回值。

24. ap_construct_url

747: 这个API调用返回一个从请求记录部件中来的完整构建的URL。这里会包含服务器、端口server_hostname域和HTTP方法信息的规范名称。

777: 在组装了完整的URL构件后，函数在778和781行返回一串字符串。

25. ap_get_limit_rep_body

785: 这个API调用返回当前目录配置的limit_req_body域。

26. get_interpreter_from_win32_registry

796: 对Windows平台，这个函数要求Windows注册表决定使用何种解释器来处理或启动程序和文件。

27. ap_get_win32_interpreter

900: 对Windows平台，此函数设置执行程序的解释器，并在922行调用函数get_interpreter_from_win32_registry来获得解释器的信息。其中fieldtype域是在985行返回的。

28. ap_check_cmd_context

1013: 这个API调用检查用在Apacher配置文件的命令上下文中。有一连串的if语句来验证可能的位置处的命令的有效性。(可能的位置，比如在<VirtualHost>或<Limit>容器

中)。如果命令使用有误，那么函数将返回一错误字串给调用者。

29. set_access_name

1065: 设置当前服务的配置记录的access_name域给参数变量(arg)。这个参数变量以参数的方式传递。

30. set_gprof_dir

1083: 设置当前服务器配置记录的gprof_dir域给参数变量(arg)。这个参数变量以参数的方式传递。在设置域值之前，在1090行用ap_check_cmd_context来验证设置gprof_dir的命令处于有效的配置上下文中。

31. set_document_root

1101: 设置基于arg值的服务器配置的ap_document_root域。arg值是以参数形式传入的。

1108: 调用ap_check_cmd_context来确定命令(DocumentRoot)在有效的配置上下文中使用。

1114: 通过调用ap_os_canonical_filename从arg值来创建一个规范的文件名。

1126: 把ap_document_root域设置给arg。

32. ap_custom_response

1130: 这个API调用定义了一个用户对基于HTTP响应代码之上的请求如何响应。(这被用于检索response_code_strings结构)

1133: 把目录配置记录读到conf中。

1139: 如果需要的话，在配置记录的response_code_strings数组内分配空间。

1147: 在response_code_strings结构中加入传入的参数。

33. set_error_document

1154: 定义文件返回一错误文档，与HTTP错误代码相对应。

1172: 检索传给此函数的参数中的HTTP响应代码号；响应代码在1173行被转换成整数，这样它就可以作为数组的索引。

1201: 如果在检测传入的响应代码中没有错误发生，则所需的响应被存储到response_code_string数据中，见1208行。

34. set_override

1226: 设置当前目录配置中的override域，此目录配置是基于一个Allow Override路径中的参数使用的。

1238: 用OR_NONE约束(在http_config.h中定义)将override域置为零。

1239: 循环遍历包含在cmd中的所有参数。对每个参数，调用ap_getword_conf来检索cmd变量的下一个文字。有一连串的if/then/else语句来检索可以覆盖的项，对每一项，与之对应的位通过与相应的覆盖选项进行逻辑或(OR)操作，加入到override域中。

35. set_options

1272: 设置选项定义的当前目录配置。

1280: 循环检索传给set_options中cmd命令所列的每一项。

1288: opts域设置为OPT_NONE(无选项设置)。

1292: 检测每个可能包含在一个Options路径中的字串。对每个找到的项，目录配置记录的opt域内，每一个相应的位都会被进行逻辑或(OR)运算以置位。

1330: 检查选项是否用-(减号)或+(加号)定义，如果是，则分别移到opts_remove域或

opts_add域中。

36. satisfy

1348: 设置目录配置记录的satisfy域为all或any。设成哪个，取决于传给此函数的arg变量值。

37. require

1363: 对当前请求，设置requirement和method_mask域，这些设置是基于目录配置记录中的ap_requires域和传入此函数的arg参数值的。

38. ap_limit_section

1378: 定义HTTP方法，使得此方法能够通过解析一个<Limit>容器类的内容而被使用在服务器或目录上下文中。

1382: 用传入参数的arg值，通过调用ap_getword来检索允许的HTTP方法列表。

1387: 调用ap_check_cmd_context来检查所使用的目录上下文。

1399: 执行参数中所列的每个方法的循环（此参数现在包含在limited_methods串中）。

1402: 用方法变量作输入参数，调用ap_method_number_of来给方法串赋以方法号。

1415: 给方法号methnum设置限制变量。

1422: 给limited值赋予cmd中的limited域。

39. endlimit_section

1445: 若配置文件中出现不规则容器类对象，则对其返回一错误信息（该对象中可能没有像</Directory>一样的结束命令）。

40. end_nested_section

1469: 返回end_token值，在结束当前嵌套配置段时应当有此值。若没有发现此值，则报错。

41. unclosed_directive

1494: 当执行一配置文件时，若一个容器类命令缺少结束标志（如>），则返回一个错误信息。

42. dirsection

1501: 在配置文件内处理<Directory>容器类。

1509: 调用ap_create_per_dir_config来创建一个新的目录配置记录，以保存此<Directory>容器类中的配置信息。

1527: 调用ap_getword_conf来检索配置的目录命令。

1534: 如果配置了一个<DirectoryMatch>类容器，在1535行调用ap_pregcomp来编译规则表达式，此规则表达式是用来定义此配置文件应用于哪个目录的。

1545: 调用ap_os_canonical文件名，以确认命令名符合标准。结果置于path域中。

1568: 以目录配置记录new_dir_conf调用ap_add_per_dir_conf。把此目录配置加入到当前服务器配置中。

43. urlsection

1583: 为<Location>容器类定义一项配置。

1595: 调用ap_create_per_dir_config创建一个per-directory配置记录。

1613: 如果指定了一个<LocationMatch>容器类，则在1614行调用ap_pregcomp(在alloc.c

中定义)来编译规则表达式，该规则表达式用于指定此配置应用于哪个位置。

1625: 调用ap_srm_command_loop在容器类中执行命令。

1643: 调用ap_add_per_url_conf加入per-location配置到当前服务器配置中。

44. filesection

1658: 为<Files>容器类定义一项配置。

1671: 创建一个新的配置记录，在<Files>容器类中保存配置路径。

1692: 如果此容器类使用<FileMatch>格式，将编译规则表达式。该规则表达式定义了它应用于哪个文件，该信息将保存在变量r中。

1710: 调用ap_srm_command_loop执行<Files>容器类中所有的命令。

1721: 为此容器类在conf配置记录内设置附加域。

1728: 调用add_file_conf往服务器配置记录c中加入<File>容器类配置。

45. end_ifmod

1748: 检查<IfModule>容器类是否正确地结束，此功能在Apache当前版本中没有实现，所以函数暂为空。

46. start_ifmod

1754: 为<IfModule>容器类进行配置

1773: 调用ap_find_linked_module来判断此<IfModule>容器类配置的模型是否连接到Apache当前的拷贝中，如果找到，则存储结果。如果模型不存在，start_ifmod只返回而不作任何处理。

1779: 调用ap_cfg_getline循环检查每个配置行。每一项置于cmd中的config_file数组中。

47. ap_exists_config_define

1797: 根据ap_server_config_defines数组是否为空（返回一个0）或包含至少有一个元素（返回一个1）来决定此API调用返回值为true或false。

48. end_ifdefine

1812: 标注<IfDefine>配置段的结束。此功能在Apache当前版本中没有实现，所以此函数为空。

49. start_ifdefine

1818: 对<IfDefine>容器类进行配置。

1839: 调用ap_exists_config_define来检查此服务器是否支持<IfDefine>容器类。若不支持，start_ifdefine返回一个空值。

1845: 在每个配置行循环执行（用ap_cfg_getline来检索）。

50. virtualhost_section

1864: 对<VirtualHost>容器类进行配置。

1873: 调用ap_check_cmd_context来查看<VirtualHost>容器类的上下文。此容器类只能是全局的，不包含在任何其他的容器类中。

1897: 调用ap_init_virtual_host集初始化虚拟主机配置。变量为main_server。

1903: 将虚拟主机服务器加入到服务器的链接列表中。

1912: 调用ap_srm_command_loop来执行<VirtualHost>容器类中的所有命令。

1921: 如果<VirtualHost>容器类中包含的命令已经被包含在srm.conf或access.conf文件中，

在1922行和1927行将用适当的参数调用ap_process_resource_config函数。

51. set_server_alias

1937: 设置服务器的别名或改变服务器名后返回给客户。

1944: 循环arg参数，调用ap_getword_conf来接收一个命令选项，然后将其放入服务器配置记录中的wild_names域（1948行）或names域（1953行），取决于它是否包含一个*字符。

52. add_module_command

1960: 用动态共享对象（DSO）给Apache服务器中加入一个模块，由一个AddModule命令所导引。

1963: 检查上下文看AddModule命令在何处。它只能是全局的，不能包含在任何其他的容器类中。

1969: 调用ap_add_named_module给Apache服务器的arg参数中加入模块名。

53. clear_module_list_command

1978: 通过调用1987行的ap_clear_module_list函数，清除Apache副本中动态载入的模块的清单。ClearModuleList指令，只能在配置文件中作为全局信息，不能包含在其他的容器类中。

54. set_server_string_slot

1991: 给arg参数赋一服务器字串值。此函数用来执行一些依赖于单个字串的命令(例如ServerAdmin、ErrorLog和ServerName)。

55. server_type

2010: 设置ap_standalone标志来注明服务器类型是基于ServerType命令的（stand- alone或者inetd)。

56. set_signature_flag

2055: 设置使用在一个目录上下文中的服务器签名。目录配置的server_signature值为on、off或email。

57. set_send_buffer_size

2081: 设置网络发送缓冲区的大小。此函数执行SendBufferSize命令，该命令只能当作全局信息来使用（不包含在任何其他的容器中），在2085行调用ap_check_cmd_context来检查。

58. set_user

2099: 当运行在非窗口系统中时（用户没有在windows中使用指令），设置Unix用户IO，Apache用此ID来建立其权限级别。

2108: 调用ap_check_cmd_context来检查用户指令的上下文。

2114: 给arg参数设置ap_user_name，并把ap_user_id变量赋给相应的用户ID，这通过调用2117行的ap_uname2id来实现。

2120: 如果ap_suexec_enabled标志设置为允许用户指令在<VirtualHost>容器类中，那么server_uid域也将设置给用户ID，此ID对应于arg的用户名。

2130: 若用户ID为0，打印错误信息，说明服务器试图以根用户来运行。

59. set_group

2158: 给组ID设置server_gid域值，以arg参数传递，若需要的话，把名字转换为ID号，

用2169行的函数ap_gname2id来完成。

60. set_server_root

2187: 根据ServerRoot命令中的信息设置服务器根目录。2190行调用的ap_check_cmd_context函数确保此命令不包含在其他容器内。

2197: 调用ap_os_canonical_filename函数来生成与ServerRoot命令标准一起使用的路径和文件名。

2202: 如果服务器根（arg）不是一个目录，返回一个错误，否则ap_server_root串赋给arg值。

61. set_timeout

2207: 根据Timeout命令设置服务器的超时时间值。在2216行，arg参数被转换为整数，并赋给服务器配置的timeout域中。

62. set_keep_alive_timeout

2220: 根据KeepAliveTimeout命令设置保持连接的超时时间，在2230行，arg参数被转换为整数，并赋给服务器配置的keep_alive_timeout域。

63. set_keep_alive

2234: 根据KeepAlive命令设置保持连接标志（不考虑连接是否使用）。在2251行设置服务器配置的keep_alive域。

64. set_keep_alive_max

2256: 设置在一个单独的连接中可以处理的客户请求的最大数目（当保持连接设置为允许时），它是根据MaxKeepAlineRequests命令来设置的。在2265行，服务器配置的keep_alive_max域被设为arg参数的整数值。

65. set_pidfile

2269: 对此Apache服务器配置执行ID文件的位置。PidFile命令只能全局使用，不包含在其他容器类中。2272行的ap_check_cmd_context函数可确保这一点。

2283: 将全局ap_pid_fname变量赋给arg参数值。

66. set_scoreboard

2287: 设置保存Apache记分牌文件的名称。ScoreboardFile命令只能是全局信息，不包含在其他容器类中。2291行调用ap_check_cmd_context函数来确保这一点。

2297: 将全局变量ap_scoreboard_fname赋给arg参数字符。

67. set_lockfile

2301: 设置锁定accept调用的文件名。LockFile命令不能用在其他的容器中。在2310行，ap_lock_fname全局变量赋给arg参数。

68. set_idcheck

2314: 设置目录配置中的do_rfc1413标志，使得identd用户标识查询得以执行。2323行设置此值。

69. set_hostname_lookups

2327: 设置目录配置中的hostname_lookups域为on、off或double（用于完全反向查询验证）。此域的值由HostnameLookups命令决定。

70. set_serverpath

2352: 设置Apache服务器可被访问到的路径名，根据ServerPath命令来设置。在2361行，服务器配置的路径域被赋给arg参数。

71. set_content_md5

2366: 设置content_md5域来决定content_MDS头在每个请求响应中是否返回给客户。根据ContentDigest命令来设置此标志。

72. set_use_canonical_name

2379: 根据UseCanonicalName命令来设置use_canonical_name标志。此标志判断返回主机名给客户端时，是否总是构造标准的服务器和端口名，而不是使用被客户接受的服务器和端口，因为它的名称有可能不是规范的。

73. set_daemons_to_start

2394: 当Apache启动时，设置Apache子进程的数目。此值由StartServers命令决定。

2398: 如果在Windows平台上使用StartServers命令，则打印警告信息。因为它们是基于线程的，此命令无效。

2407: 将全局变量ap_daemons_to_start转化为整数值赋给arg参数。

74. set_min_free_servers

2412: 根据MinSpareServers命令，设置总能运行的Apache子进程数。在2421行将全局变量ap_daemons_min_free赋给arg参数

2422: 检测ap_daemons_min_free值以确保其非负。如果是负数，ap_daemons_min_free置为1。

75. set_max_free_servers

2435: 设置脱离运行的最大Apache子进程数目。这是由MaxSpareServers命令所决定的，在2444行赋值ap_daemons_max_free全局变量。

76. set_server_limit

2448: 设置用以处理过程负载时能够产生的最大Apache子进程数。这由MaxClients命令决定。

2451: MaxClients命令（或者是建议不使用的ServerSafetyLimit指令）只能作为全局指令使用，不能包含在其他的容器类中。调用带GLOBAL_ONLY标志的ap_cheek_cmd_context来验证。

2457: 给参数arg赋以全局变量ap_daemons_limit值。如果arg值大于HARD_SERVER_LIMIT常量（在httpd.h文件中定义），会发出警告，并将ap_daemons_limit重置为HARD_SERVER_LIMIT。

77. set_max_requests

2478: 此函数定义了一个Apache子进程在退出或被替代之前能够服务的最大请求数。这是为了防止过多地占用服务器资源而导致的存储资源不足的危险。在2487行，全局变量ap_max_requests_per_child赋值给arg参数。

78. set_threads

2491: 由ThreadsPerChild指令所指定的每个子进程所允许的最大线程数。

2499: 给arg参数赋予全局变量ap_threads_per_child值。如果该值大于HARD_SERVER-LIMIT，将发生警告并把ap_threads_per_child值置为HARD_SERVER_LIMIT。

79. set_excess_requests

2521: 此函数设定了一个子进程能够处理多于由MaxChillRequests指令所设置的数目的请求数。此数值用在当一个连接向存在的进程进行请求，而此子进程将会被调度结束运行的时候。由ExcessRequestsPerChild指令来控制ap_excess_requests_per_child全局变量的值，它在2530行设置。

80. set_rlimit

2538: 分别由RlimitCPU、RlimitMEM和RlimitNPOC命令来设定时间、存储器和进程数目的资源限制。

81. no_set_limit

2605: 这是针对CPU时间（set_limit_cpu）、存储器（set_limit_mem）及进程数（set_limit_aproc）调用的函数。

2609: 对不支持Rlimit指令的平台，此函数在日志中记录一个错误以注明该情况。

82. set_limit_cpu

2618: 用Rlimit CPU指令调用带有当前目录配置记录的limit_cpu域参数的set_rlimit函数，来设定Apache进程所能占用的CPU时间限制。

83. set_limit_mem

2630: 用RlimitMEM指令调用带有当前目录配置记录的limit_mem域参数的set_rlimit函数，来设定Apache进程所能使用的存储器的限制。

84. set_limit_nproc

2649: 用RlimitNPROC指令调用带有当前目录配置记录的limit_nproc域参数的set_rlimit函数，来设定Apache所能使用的CPU进程数目。

85. set_bind_address

2659: 设置Apache所绑定的IP地址，这是由BindAddress指令定义。在2669行用ap_get_virthost_addr函数的参数返回值来设置全局变量ap_bind_address.s_addr（该函数用arg参数作为查找虚拟主机表的值）。

86. set-listener

2673: 设置Apache应当收听请求的端口（如果能提供，也可以是IP地址和端口）。这由Listen命令设定。

2686: 检测ips参数中是否包含有IP地址。

2697: 给ips参数设置全局ports变量。

87. set_listenbacking

2721: 由ListenBacklog指令确定用于收听的最大后备连接数。此值放置在2737行的ap_listenbacklog中。

88. set_coredumpdir

2741: 用CorePumpDirectory指令来定义在出错退出时内核交换文件应当写入的Apache服务器的目录。在2751行，arg参数被相应地转换成从服务器根开始的路径，然后在2759行赋给变量ap_coredump_dir。

89. iudude_config

2764: 在配置文件中，处理由Include指令所包含的附加配置文件。

2767: 调用ap_server_root_relative，name参数被转换成相对于服务器根的路径。

2769: 调用ap_process_resource_config来执行包含的配置文件。该指令带有用以执行的名字参数，该参数用作路径和文件名。

90. set_loglevel

2775: 通过分析Loglevel指令中所包含的文字并将其与一些可能的值进行比较（比如emerg, error, debug等值），设置当前服务器配置的log_level域。共定义了八级，每一级会增加作为Apache进程请求记入日志的消息数。如果没有这八个字串的任何一串，将返回一个错误信息。

91. ap_psignature

2823: 这个API函数返回服务器的签名。

2836: 此Apache服务器收听的端口赋给sport变量。

2839: 根据服务器配置记录中的server_signature域，选择用电子邮件地址或不用它来组装和返回服务器签名字串。

92. set_authname

2860: 载入授权名字到当前域中，将目录配置中的ap_auth_name域赋给wordl参数值。

93. set_serv_tokens

2894: 根据返回给客户机的server头，处理包含进服务器操作系统平台的请求。这由ServerTokens指令定义。ap_server_tokens全局变量被置为包含操作系统的最小或全部信息，取决于指令所提供的值。

94. set_limit_req_line

2916: 设置HTTP行的长度限制，以避免到来请求的溢出。此值由LimitRequestLine设置。

2926: 设置arg参数的lim变量为一整型值。将检测此值是否为负或大于httpd.h文件中定义的DEFAULT_LIMIT_REQUEST_LINE的值。若有此情况，则返回错误信息；否则，服务器配置记录中的limit_req_line域将赋给lim。

95. set_limit_req_fieldsize

2945: 设定每个带有请求信息的HTTP头域的大小限制，以避免到达请求的溢出问题。由LimitRequestFieldSize指令设定此值的大小。

2956: 将lim变量设为arg参数的整数值。此值在2957行和2964行被检查是否为负或大于httpd.h文件中所定义的DEFAULT_LIMIT_REQUEST_FIELDSIZE值。若是此情况，返回一错误。否则服务器配置记录的req_fieldsize域赋给lim。

96. set_limit_req_fields

2975: 设置一个请求所能包含的HTTP头行的总数以避免到达请求的溢出问题，由LimitRequestFields指令定义此值。

2985: 将lim变量赋给arg参数的整数值。此值在2986行被检查是否为负。如果是负数，将返回一个错误；否则，服务器配置记录的limit_req_fields域，在2993行赋给lim。

97. set_limit_req_body

2997: 设置一个请求体中所能包含的总字节数限制以避免溢出问题，该溢出问题通常是从Apache服务器上的denial_of_service攻击而来。由LimitRequestBody指令来定义此值。

3013: 将配置记录的limit_req_body值赋给arg参数。

98. set_interpreter_source

3019: 当Apache运行在窗口系统中时，此函数执行ScriptInterpreterSource指令来决定代码是如何执行和处理的。

3023: 设置目录配置记录中script_interpreter_source域，用以指明窗口寄存器或包含的脚本路径名（以#1作为开始标志）应当在运行脚本时使用。缺省时假定了指令将提供一个SHEBANG串给命令解释器。

99. core_cmds

3043: 定义了如何运行一个必须是或可能是全局的指令（不是一个像<Directory>或<VirtualHost>等的容器类）。这个结构包含了调用处理指令的函数，指明参数的类型标志(该参数必须和指令一起包含在配置文件中)，以及一行在指令与所接受的语法不符时加入到日志中去作为错误信息的指令。这个结构包含86项；想了解Apache性能的人将从中得到许多有用的信息。

100. core_translate

3378: 在处理一个请求之前，需要执行文件名的转换。此函数定义为核心模块的文件名转换阶段（在3672行）。

3381: 调用ap_get_module_config（在http_config.c中定义）来查询服务器配置记录。

3384: 如果这是一个代理请求，将返回HTTP_FORBIDDEN，因为代理请求在处理过程中不该出现在此阶段。

3387: 检测uri域是否为伪字符，该字符说明是一个错误的请求。

3393: 在路径和文件名域中的字符串（通过3422行）被标准化。

101. mmap_cleanup

3435: 清空由内存映射（memory-mapped）文件所使用的（空闲）内存。在3439行调用munmap函数来完成此动作。如果此函数没有执行成功，将在日志中记录一个错误以指明内存并不空闲。

102. default_handler

3458: 处理没有指明其他句柄类型（比如由AddHandle指令增加句柄）的请求。

3473: 如果包括一个请求体，将调用ap_discard_request_body函数来读取它然后丢弃。请求体是使用PUT或POST方法的请求，但它们并不由此句柄来处理。

3477: 检查请求是否正在使用HTTP方法GET或OPTIONS。如果没有使用，则在3480行记录一个错误到日志。

3486: 如果使用的方法是OPTIONS，调用ap_send_http_options返回请求信息。

3493: 检测请求的finfo.st_mode域查看所要求的文件是否存在。

3518: 用ap_pfopen打开被请求的文件。

3528: 调用ap_update_mtime和ap_set_last_modified函数更新请求的finfo.st_mtime域。

3531: 设置向外发送的头，指明将要发送的字节数。

3546: 如果此服务器使用了存储器映射(memory_mapped)文件，调用mmap来获取被请求的文件。

3563: 如果设置了配置记录的content_md5域，则调用ap_md5digest来设置Content_MD5

头。

3582: 调用ap_send_http_header（在http_protocol.c中定义）来发送此请求的头。

3589: 如果设置了header_only标志，在3649行函数继续执行。否则，准备发送被请求文件。

3590: 循环发送被请求文件的每一部分。在3595行调用fseek，然后在3603行调用ap_send_fd_length向客户机发送文件的一部分。

3609: 如果此服务器中使用了存储器映射（memory_mapped）文件，在3614行为存储器映射文件分配空间，在3617行对该存储注册一个cleanup，在3636行用ap_send_mmap指令来发送存储器映射文件。

102. core_handlers

3653: 定义一个带有核心模块所使用句柄的结构。唯一使用的句柄是default_handler。

103. core_moduls

3659: 定义核心模块与创建、合并目录及服务器配置和处理文件名转换的函数。配置文件通过索引由core_cmds所指向的表来处理（3670行）；缺省的请求句柄是由core_handlers(33671行)结构来索引的。

4.2 http_main模块

此文件包含有用于启动和初始化Apache的函数集。启动和管理子进程以处理客户机请求的程序段也包含在此文件中，此外还有存储空间分配的管理，处理运行Apache副本间的信号，以及记录了Apache每个副本状态的记分牌。

几个http_main中的编译器定义语句把代码划分为非线程和多线程的函数。这个区别可能产生混乱，因为几个函数有相似或相同的名字，或者像是在执行同一任务。

虽然http_main代码对许多服务器信息域进行初始化（定义Apache执行的全局变量），然而还有大多数的信息在http_core中设置，http_core是基于配置文件中的指令的。

httpd.h头文件包含有常量和其他有用信息，在此不作描述。

1. reset_version

4002: 改变version_locked标志，使得Apache版本信息在需要时可以更新。在4006行server_version字串被赋予NULL，以便将来更新。

2. ap_get_server_version

4009: 这个API调用返回Apache版本是在4051行由函数ap_set_version设置的server_ version变量所定义。如果server_version变量没有设定，则返回常量SERVER_BASEVERSION（在httpd.h中定义）。

3. ap_add_version_component

4015: 这个API函数设置了Apache的版本字串。

4018: 如果没有设置version_locked标志，则继续字串配置。

4026: 如果当前的server_version字串为空，将记录一清除操作以便当处理存在时此变量可以被丢弃。server_version字串被赋给component作为参数传递。

4033: 由于server_version不空，则已经在先前对此函数的调用中记录了变量清除，所以component字串只简单的加到server_version字串之后。

4. ap_set_version

4051: 这个API调用设置了Apache的版本字串。以httpd.h中定义的SERVER_BASEVERSION字串为参数调用ap_add_version_component函数（4015行）。如果ap_server_tokens变量的值为SrvTk_MIN，则将PLATFORM加到服务器版本串中。

4064: 如果ap_server_tokens变量没有赋给SrvTk_FULL，则设定version_locked来锁定服务器版本子串。这样阻止任何对此字串的改写，直至服务器重新设置为止。

5. chdir_for_gprof

4079: 由定义的GPROF改变服务器目录的函数。

4081: 检索服务器配置，设置到core_server_config中。

4094: 调用ap_server_root_relative，设置dir到相关的服务器根目录下。

4107: 调用系统函数chdir来改变当前系统的工作目录到dir。

6. clean_child_exit

4116: 这个函数退出一个子进程，释放所有该子进程所占用的空间。

4119: 调用ap_child_exit_modules（在http_config.c中定义）来退出该子进程所使用的所有模块。

4120: 调用ap_destroy_pool（在alloc.c中定义）来释放该子进程所使用的所有存储池空间。

7. expand_lock_fname

4128: 将参数P填入要锁住的文件名。在4132行调用ap_server_root_relative函数，将服务器根包含在完整路径名中。

8. aecept_mutex-init

4145: 对用于多线程环境中的互斥变量（mutex）进行初始化接收过程。（初始化互斥变量的接受过程。）

4153: 调用usconfig来检查互斥变量的状态。如果返回值是在错误信息中（4155, 4161, 4166, 4170或4175行），则用perror记录一个错误。

9. accept_mutex_on

4180: 接受一个变量上的互斥锁。

4182: 开始一个switch代码段来判断对互斥变量进行了什么操作。检测用的值是调用ussetlock函数的返回结果。

10. accept_mutex_off

4195: 接受对一个互斥变量的解锁操作。在4199行调用clean_child_exit来退出当前的子进程。

11. accept_mutex_child_cleanup

4221: 接受互斥变量，子进程退出后清除其存储区。这个函数是为Solaris平台设计的。

4225: 调用pthread_mutex_unlock对互诉变量解锁。

12. accept_mutex_child_init

4229: 在Solaris平台上，通过调用ap_register_cleanup来对互斥变量进行初始化，这样，当变量使用完后，就可以保证释放了。

13. accept_mutex_cleanup

4235: 在4238行，调用munmap函数来清除互斥变量。

14. accept_mutex_init

4245: 初始化一个在Solaris平台上使用的pthread_mutexattr_t互斥变量。

4255: 调用mmap来分配并初始化一个互斥变量。

4264: 调用pthread_mutexattr_init对互斥变量初始化。

4282: 用ap_register_cleanup来登记互斥变量，这样当它不再被使用时，能够妥当的清除。

15. accept_mutex_on

4286: 在4295行准备使用一个互诉变量，这里调用了pthread_mutex_lock函数。

16. accept_mutex_off

4303: 调用pthread_mutex_unlock函数，在4307行，释放一个互斥变量。

17. accept_mutex_cleanup

4362: 在4370行调用semctl函数清除互斥变量所使用的存储区。

18. accept_mutex_init

4375: 初始化一个互斥变量。

4381: 将sem_id值赋给返回值semget。

4406: 用ap_register_cleanup来登记该互斥量所使用的存储空间，以便将来不再需要时清除。

19. accept_mutex_on

4418: 准备使用互斥变量，在4420行调用semop。

20. accept_mutex_off

4426: 在4428行调用semop函数释放一个互斥变量。

21. accept_mutex_init

4446: 用定义的USE_FCNTL_SERIALIZED_ACCEPT来对服务器的互诉变量进行初始化。

22. accept_mutex_on

4482: 在4486行调用fcntl函数将互斥变量准备就绪。

23. accept_mutex_off

4502: 在4506行调用fcntl函数释放一个互斥变量。

24. accept_mutex_cleanup

4525: 在4527行调用unlink函数释放一个互斥变量的文件名（存储在ap_lock_fname中）来释放（清除）一互斥变量。

25. accept_mutex_child_init

4534: 当生成子进程时，为子进程初始化一互斥变量。

4537: 定义lock_fd为ap_popenf返回的文件描述器。

26. accept_mutex_init

4551: 调用unlink来初始化一互斥变量锁，以保证ap_lock_fname值没有被使用。然后在4555行调用ap_popenf来查询互斥变量。

4563: 调用ap_register_cleanup来记录一个新初始化的互斥变量，以便在退出时清除互斥

变量的存储空间。

27. accept_mutex_on

4567：在4571行调用flock函数来设置互斥变量锁。

28. accept_mutex_off

4583：在4585行调用flock函数来清除互斥变量。

29. accept_mutex_deanup

4597：当此服务器自定义了USE_OS2SEM_SERIALIZED_ACCEPT时，请求使用过的互斥变量的存储区。

30. accept_mutex_child_init

4607：为子进程初始化一个互斥锁。此函数是为OS/2序列化信号而用的。在4609行调用DosOpenMatexSem函数来完成初始化工作。

31. accept_mutex_init

4624：在4626行通过调用DosCreateMutexSem来为OS/2信号初始化一个标准互斥量。

4637：登记互斥量存储区以便用完后清除。

32. accept_mutex_on

4641：在4643行调用DosRequestMutexSem函数来为OS/2信号请求一个互斥量。

33. accept_mutex_off

4655：在4657行调用DosReleaseMutexSem函数来清除一个OS/2互斥量。

34. usage

4697：运行Apache时显示命令选项的使用方法信息。

35. timeout

4774：设置一超时计数器。一次只能有一个超时计数存在。在多线程环境中不支持超时计数。

4778：如果alarms_blocked标志被设置，则返回alarm_pending标志；否则，超时请求继续执行。

36. ap_block_alarms

4851：这个API调用通过设置alarms_blocked变量来封闭报警信息，在执行警报过程之前总会检查这个变量。alarms_blocked变量是增量式的而不是通过设置为1来允许对嵌套块的跟踪。

37. ap_unblock_alarms

4856：这个API调用解除对报警的封闭。在4858行alarms_blocked变量减值。如果减值后alarms_blocked值为0，则系统查找exit_after_unblock标志。如果此标志被设置对alarms_blocked增值，exit_after_unblock标志重新设定，4873行调用clean_child_exit清除子进程。通过减少alarms_blocked值和再增加它的值，此函数提供了对最通常情况的最快路径（这种情况是指exit_after_unblock没有设置，因此达不到此段代码）。

4871：如果设置了alarm_pending标志，将重新设置它并产生一个超时计数。

38. alarm_handler

4890：设置并处理作为参数传入的sig信号。

39. ap_set_callback_and_alarm

4899: 调用alaim系统函数或者通过对当前子进程在记分板像中记录时间来设置一个报警的回叫。

40. ap_check_alarm

4948: 仅用于windows系统，这是一个API函数，用以检查一报警超时是否已满了。

41. ap_reset_timeout

4974: 用ap_set_timeout来重新设置先前设置的超时计数。

4980: 调用ap_set_callback_and_alarm来检测超时计数的状态。如果超时计数已满，在4984行重置为0。

42. ap_keepalive_timeout

4992: 此API函数为保持连接设置一超时计数。

4999: 如果连接使用了保持方式，则根据keep_alive_timeout来设置to的值。否则，在5002行设为标准服务器超时值。之后调用ap_set_callback_and_alarm函数，其中含有to超时值。

43. ap_hard_timeout

5007: 这个API函数定义了一个硬超时计数，此超时计数用在取消一个没有在分配时间内完成的等待活动（分配时间基于服务器的缺省超时值）。

44. ap_soft_timeout

5018: 这个API函数定义了一个软超时计数，此超时计数用于取消一个在服务器设置的超时时间已过仍未完成的动作。5023行通过ap_set_callback_and alarm对超时设定的调用对ap_hard_timeout函数中的调用很重要。

45. ap_kill_timeout

5028: 这个API函数将先前设置的超时取消（例如，用ap_hard_timeout）。此函数使用于当一个动作完成，但由于超时而没有自动退出的时候。

46. sock_enable_linger

5072: 对连接设置接口选项以允许延缓关闭，此时连接并没有真正关闭，而是等到最终缓存的数据发送到客户机并得到响应。

5079: 调用setsockopt对延缓关闭设置一合适的选项。

47. lingerout

5096: 这个函数是对使用延缓结束连接中使用超时而设的。

48. linger_timeout

5110: 与lingerout类似，这个函数对使用延缓结束连接提供超时机制。

5114: 用ap_set_callback_and_alarm来设置一个回叫，这样超时就可以通知此函数。

49. lingering_close

5125: 以延缓结束的方式关闭一个连接，允许客户机先接收缓存数据。

5136: 用linger_timeout来设置超时以防止延缓关闭的时间太长。

5141: 调用ap_bflush强制所有剩余的缓存数据写到客户机中。

5143: 调用ap_bclose来关闭连接。

5193: 读取从接收客户端接收到的数据（由Apache发送的响应性数据）。

50. ap_register_other_child

5210: 这个API调用登记其他Apache子进程，在5217行为这些附加的子进程记录分配空间。

51. ap_unregister_other_child

5232: 这个API调用从other_children列表中移去一个Apache子进程记录。

52. probe_writable_fds

5253: 探测可写文件的描述器仍可被写。

53. reap_other_child

5310: 关闭不使用的子进程

54. reinit_scoreboard

5347: 在多线程环境下重新初始化记分牌文件。

5356: 通过对memset的调用为记分牌分配存储区。

55. cleanup_scoreboard

5359: 释放由记分牌映像使用的存储空间。

5361: 验证一个记分牌映像的存在，然后在5362行调用free来消除它。在5363行ap_scoreboard_image被置为NULL，说明记分牌已被清除。

56. ap_sync_scoreboard_image

5366: 这个API调用同步（更新）记分牌映像。它只是占个地方，还没有代码。

57. create_shared_heap

5378: 在OS/2下创建一共享存储空间。

58. get_shared_heap

5397: 在OS/2下使用一共享存储空间。

59. setup_shared_mem

5424: 在OS/2下在非多线程环境中设置共享存储区作为记分牌映像来使用。

60. reopen_scoreboard

5451: 通过在5457行调用get_shared_heap函数分配共享存储区在OS/2平台上重新打开记分牌存储区。在5466行 ap_scoreboard_image变量被存入存储区。

61. cleanup_shared_mem

5505: 调用shm_unlink来清除用作记分牌映像的共享存储区。另见547行开始的源代码注释。

62. setup_shared_mem

5510: 在非多线程环境中准备一个用作记分牌映像的共享存储区。

5516: 调用shm_open为共享存储区分配一块区域，以ap_scoreboard_fname指向新打开的存储块。

5550: 设置ap_scoreboard_fname变量指向新创建的存储区。

63. reopen_scoreboard

5554: 此函数是为非多线程环境准备的，暂无代码。

64. setup_shared_mem

5560: 当定义了MMAP_SCOREBOARD时，也就设置了一个存储映射记分牌。

5581: 调用mmap分配一存储映射区域，设置m指向该区域。

5638: 设置ap_scoreboard_image给由mmap返回的m值。

65. reopen_scoreboard

5642: 当使用存储映射记分牌映像时，这个函数为空函数。

66. setup_shared_mem

5650: 当定义了USE_SHMGET_SCOREBOARD时，也就设置了记分牌文件的共享存储区。

5657: 当shmid设置记分牌的存储区时，将shmid赋给shmget返回的值。

67. reopen_scoreboard

5750: 当定义了USE_SHMGET_SCOREBOARD时，这是一个空函数。

68. force_write

5763: 强制写记分牌，来避免部分写。部分写有可能毁坏记分牌的统计数据。

69. force_read

5779: 强制从记分牌文件中读取。在5784行调用read函数。

70. cheanup_scoreboard_file

5795: 在5797行调用unlink来清除记分牌文件。

71. reopen_scoreboard

5800: 重新打开记分牌文件，更新其存储映像文件。

5803: 如果记分牌文件描述器scoreboard_fd显示记分牌文件是打开的，则调用ap_pclosef来关闭它。

5805: 调用ap_popenf来重新打开记分牌文件。scoreboard_fd变量赋给返回的值。

72. reinit_scoreboard

5816: 重新对记分牌初始化。这个函数由Apache父进程调用，用在子进程由父进程初始发生状态改变的时候。

73. ap_sync_scoreboard_image

5868: 更新记分牌映像。(另见5854行开始的源代码注释。)

5871: 对记录发生更改的子进程搜索记分牌映像。

5872: 调用force_read函数为该子进程读取当前记分牌状态。

74. ap_exists_scoreboard_image

5879: 这个API调用返回true/false值，要根据ap_scoreboard_image变量是否指向一个有效的记分牌映像而定。

75. put_scoreboard_info

5884: 将消息放入记分牌映像中。

5888: 为当前子进程状态的位置搜寻记分牌映像。

5890: 调用force_write将当前子进程状态信息写入记分牌映像中。

76. clean_parent_exit

5898: 在5902行调用ap_destroy_pool为此进程清除存储区池之后，Apache以传入的代码作为参数退出。

77. ap_updata_child_status

5906: 根据子进程的当前状态，更新子进程在记分牌中的状态。

5915: 在以当前子进程状态更新记分牌之前，调用ap_sync_scoreboard_image同步记分牌

映像。

5916: 给当前子进程设置ss到记分牌映像槽中。

5933: 如果当前状态是SERVER_DEAD，记分牌槽的域将被清除（通过5937行）。

5971: 如果当前子进程开始启动，子进程的信息将记录在记分牌槽中。

5975: 如果记分牌文件是用于Apache的副本，则用lseek更正记分牌槽，当前子进程的信息通过5977行调用force_write来写入。

5982: 调用put_scoreboard_info来结束当前子进程记分牌信息的更新。

78. update_scoreboard_global

5987: 更新记分牌的全局（非针对子进程的）信息。

5990: 确定记分牌文件（如果是用在此服务器中的一个文件）的正确位置，在5993行用force_write将全局信息写入记分牌。

79. ap_time_process_request

5999: 计算完成请求的时间，在记分牌映像的当前子进程槽中写入信息。

6010: 将ss写到恰当的记分牌映像槽中。

6019: gettimeofday用于设置激活当前子进程的开始时间。

6030: 当状态变为完成操作时，计算操作的结束时间，在6043行调用put_scoreboard_info来更新记分牌映像。

80. increment_counts

6046: 增加一子进程的计数，发送通知字节和相关数据，在6068行调用put_scoreboard_info在记分牌映像中记录信息。

81. find_child_by_pid

6071: 用传入的进程ID参数来查找一子进程。

6075: 在所有可能的记分牌映像槽中循环。当找到匹配的PID时，返回该槽的索引。

82. reclaim_child_processes

6082: 处理应该退出却未退出的子进程。

6096: 循环反复对子进程进行关闭。

6135: 如果子进程仍在活动，将用kill函数发送一个SIGHUP信号。如果不成功，则在6150行发送SIGTERM信号。最后在6159行发送一个SIGKILL信号。

83. reap_children

6216: 此函数查询记分牌，查找那些被错误的关闭或放弃的子进程，将其正式关闭。

6220: 在尽可能最大的子进程数中循环（max_daemons_limit），通过查询ap_scoreboard_images状态域来检测记分牌文件。6224行将杀掉标志了SERVER_DEAD但仍然表现出是活动的子进程。

84. wait_or_timeout

6294: 一段间隔后（通常是其他子进程结束处理一个请求时），更新记分牌来说明该子进程在等待。对Windows NT和其他平台，此函数用了不同的处理过程。

85. Define signals

6325: 根据NSIG值定义用作NumSIG的信号数。如果不提供这个值，NumSIG在6332行被置为32。

6336: 如果当前平台包括一信号列表，则无需再为其定义什么；否则，在6340行，siglist_init函数被定义为INIT_SIGLIST。

86. ap_sys_siglist

6344: 为当前平台定义信号列表。对此平台定义的每个信号（根据头文件中的信息，以#ifdef检查每一项），定义了ap_sys_siglist数组中的值，其中信号数即是数值索引，信号的文本描述即为该数组元素的值。这个设置数组值的列表持续到6450行。

6451: 在ap_sys_siglist数组中循环。对那些关键字为NULL的信号，将设其值为空串。

87. sig_coredump

6460: 当运行Apache出现严重问题时，处理内核转储请求。

6462: 对目录ap_coredump_dir进行修改，这样内核转储文件能写到正确的地方。

6463: 给当前进程发送一个SIG_DFL信号。

6465: 如果Apache运行在Windows NT平台上，则通过对kill的调用来停止当前进程；否则，在6467行调用raise产生一个给内核转储的信号，这样当函数退出时，信号仍可继续运行。

88. just_die

6484: 发信号给当前子进程，使其立即退出。

6489: 如果是由对ap_block_alarms的调用而使信号暂被封闭，则在6490行设置exit_after_unlock标志，这样在调用ap_unblock_alarms（在报警被封闭的函数中）之后可以执行退出。

6493: 如果报警没有封闭，通过对clean_child_exit的调用来执行退出进程。

89. usrl_handler

6500: 检查usrl_just_die标志状态，如果设置了usrl_just_die，则调用just_die来退出当前进程，并设置deferred_die标志（如果它没有设置的话）。

90. signal_parent

6549: 如果Apacle运行在Windows NT上，这个函数被子进程用来通知父进程关闭或重启。

6564: 如果设置了one_process标志，没有父进程存在，则立即返回signal_parent。

6568: 基于以参数传入的信号类型，signal_name被设置为关闭或重启信号。如果类型不是0或1，6571行函数只返回而不作任何动作。

6574: 用APD2函数记录将要发生的动作。

6577: 用Windows的OpenEvent函数给父进程发signal_name消息。如果OpenEvent返回的值是一错误信息，则在6584行记录下这个错误。

6589: 调用SetEvent给父进程设置信号事件，如果它没有返回0，在6591行记录一个错误。

6594: 在6594或6597行结束事件句柄e，在哪行处理取决于SetEvent是否返回一个错误。

91. ap_start_shutdown

6616: 对任何平台上的Apache关闭进行初始化，而不需要依赖于信号量。

6619: 如果已经设置了shutdown_pending标志，则函数认为它已经被调用，并开始关闭的过程。ap_start_shutdown不作任何动作而返回。如果没有设置shutdown_pending标志，则在6626行将其置为1。

6628: 对非Windows平台，调用signal_parent函数来唤醒父进程查看信号量。

92. ap_start_restart

6634: 试图重启当前Apache服务进程。

6637: 对Windows平台，如果设置了restart_pending标志，函数则认为重启过程已经在进行，不做任何动作返回；否则，restart_pending被置为1，is_graceful变量被置为graceful值，它是作为函数参数传入的。

6645: 对非windows平台，父进程由值为1的signal_parent调用发信号。(graceful参数没有使用)。

93. sig_term

6650: 调用ap_start_shutdown（6616行）来初始化关闭过程。

94. restart

6655: 调用start_restart函数（6634行）来初始化Apache的重启，参数值1代表对UNIX的，信号量SIGUSR1代表对Windows的。

95. set_signals

6664: 设置信号活动以进行Apache进程之间的通信。这些动作和信号存储在sa.sa_handler变量中，它们是当前平台已经定义好的信号量。

6672: 如果设置了one_process标志，则不设置任何信号量，因为只有一个Apache进程可以执行。

6679: 根据对当前平台定义的信号，反复调用sigaction来设置信号动作。如果sigaction返回一负值，则用ap_log_error来记录一个错误。这段延续到6743行为止。

6745: 如果设置了one_process标志，对每个定义的信号调用信号函数。

96. detach

6787: 除非Apache运行在子进程在父进程已消亡（见6794行）的情况下不能继续执行的平台上，当前进程与其父进程分离。

97. set_group_privs

6891: 设置Apache运行的组（仅针对UNIX系统）。

98. init_suexec

6936: 对使用suExec包装器的服务器初始化SetUserID执行模式。

99. new_connection

6957: 为新建立的网络连接设置一个有效的数据结构。

6971: 初始化conn结构的域。作为参数传入的变量——与6976行对ap_update_vhost_given_ip的调用和6982行对inet_nta的调用一样——用来定义conn的域。

100. sock_disable_nagle

6988: 将Nagle算法关闭以防止持久的网络连接，该连接因等待更多数据从而导到性能的下降。在7004行调用setsockopt来禁止算法。

101. make_sock

7017: 根据作为参数传入此函数的端口地址生成一个新端口结构。

7025: 组装端口信号串。

7036: 在封闭信息片刻之后，调用socket来建立新端口。在7041行通过ap_unblock_alarms再取消信息封闭。然后，根据Apache所运行的平台，用几个编译器的define语句来

设置socket选项。这些选项可以防止网络故障或者增强性能。

7150: 将端口绑定到addr域，该域在7025行开始处组装。

7167: 检测socket是否准备好监听连接。

102. copy_listenners

7221: 复制一个端口连接上的侦听

103. find_listener

7247: 在作为参数传入的链接列表中寻找监听者。

104. close_unused_listeners

7262: 关闭链接列表中不使用的监听者。在7270行移去每一项并释放存储器。

105. setup_listeners

7278: 打开一系列端口并更新监听者列表到一个环链接列表中。

7287: 调用find_listener来查找一个唯一的监听者。在7289行调用make_sock来为该监听者准备一个端口。

7305: 将监听者加入到环链接列表中去。

106. find_ready_listener

7330: 查询一个准备好用accept执行从客户机的连接的监听端口。

107. AMCSocketInitialize

7350: 初始化一连接端口。

108. AMCSocketCleanup

7384: 在7387行调用WSACleanup来清除一个端口连接。

109. show_compile_settings

7392: 用printf函数打印当前Apache版本的信息到标准输出设备上。

7394: 打印服务器版本号，建立日期和模块信息。

7401: 打印此Apache版本编译所使用的选项列表。每一个要打印的项都被检测，看其是否在当前配置中定义。每个定义的项都将产生一条printf语句，在配置设置被请求时打印选项。这段代码到7529行结束。

110. common_init

7538: 初始化开始运行Apache所需的一些项目。

7540: 初始化从6325行开始定义的信号。

7547: 如果Apache运行在Windows上，调用AMCSocketInitialize来对端口进行初始化。

7550: 为pconf和ptrans变量分配空间。

7553: 调用ap_util_init和ap_util_uri_init来初始化Apache。

7556: 通过分配一存储区池来设置pcommands。定义一数组来保存ap_server_pre_read_config、ap_server_post_read_config和ap_server_config_defines信息。

111. ap_child_terminate

7579: 这个API函数结束当前Apache子进程。keepalive域设置为0。requests_this_child和ap_max_requests_per_child变量均置为1，标志此子进程的结束。这是为非多线程平台定义的大块代码中的第一个函数（除了Windows NT的所有平台）。对非多线程平台的define语句，在8932行结束，多线程的函数则从该处开始。

112. child_main

7585: 这是对Apache子进程的主请求处理函数（管理子进程产生的初始化进程并不处理浏览器请求）。

7604: 调用ap_block_alarms来封闭信号，这样使初始化得以完成。

7653: 调用ap_child_init_modules来为此子进程初始化模块。

7656: 初始化已完成，可以调用ap_unblock_alarms来允许信号。

7682: 无限循环（直到有信号来杀掉此进程）读取并处理请求。

7756: 调用find_ready_listener来发现请求并处理。

7775: 接受到来的请求。

7952: 用new_connection来建立一个连接。

7964: 循环来读取和处理此链接上到来的请求（如果保持活动被使用）。在7989行用ap_procers_request处理到来的请求。

113. make_child

8044: 生成一个单独的Apache子进程，更新记分牌文件并查询系统的资源限制。

114. startup_children

8139: 开始某个数目的Apache子进程（number_to_start）。记分牌为每个进程进行更新，在8150行用make_child来创建子进程。

115. perform_idle_server_maintenance

8171: 当一个Apache子进程不再用来处理请求时，执行任务。当有太多空闲进程运行时，有可能引发杀死子进程。

8191: 调用ap_sync_scoreboard_image来同步记分牌文件。

8285: 如果从记分牌中算出的空闲服务器数比ap_daemons_max_free允许的数目多，在8264行杀掉一个子进程。

8308: 根据idle_spawn_rate，通过调用make_child来启动一个新Apache子进程来达到运行服务器所需的进程数（尽管这些进程在此刻有可能是空闲的）。

116. process_child_status

8329: 处理子进程状态，相应地更新记分牌。

117. standalone_main

8395: 此函数由REALMAIN调用以处理生成Apache子进程。

8407: 如果设置了one_process变量，则调用detach。不必做其他动作。

8431: 检索服务器配置到server_conf中，并在8433行用setup_listeners来协调监听。

8434: 调用ap_open_logs打开日志文件，登记当前进程的进程ID号。

8443: 调用reinit_scoreboard来对当前进程信息初始化记分牌文件。

8454: 调用set_sigals来设置用于通知Apache子进程重启或关闭的信号名称。

8478: 开始Apache子进程。需要开始的子进程数保存在remaining_children_to_start中。

8484: 如果系统仍努力保持运行，则在子进程以指数增长之前产生一个暂停（先生成两个，然后四个，然后八个，如此进行直到达到所需数目）。

8517: 调用ap_sync_scoreboard_image来重新同步记分牌文件。

8531: 重启一单独子进程来取代那个由于达到了所需最大数目子进程限制和结束的子进程。

8537: 调用reap_other_child来关闭需要被杀掉的子进程。
8567: 所需子进程的数目还未达到，因此调用startup_children来启动它们。
8589: 如果在这期间有一个关闭信号，则调用reclaim_child_process，通知所有的子进程关闭。
8612: 发送一个信号重新启动（SIGHUP和SIGUSR1）。
8615: 如果设置了one_process标志，则调用clean_parent_exit，因为不需要再产生子进程了。
8628: 对所有子进程更新记分牌文件。

118. REALMAIN

8684: 真正的管理子进程开始和启动处理请求的主程序。
8708: 调用common_init来执行普通初始化任务。
8717: 将ap_server_root值赋给HTTPD_ROOT环境变量。
8722: 调用ap_setup_prelinked_modules设置预连接（非DSO）模块。
8725: 循环处理Apache启动时的所有命令行选项。
8732: 开始一个switch代码段来处理所有命令行选项。使用函数，如果找到a?或h参数，打印可提供的选项。
8820: 执行初始化步骤：打开日志文件，设置Apachet版本字符串，并初始化所有模块（包括DSO模块）。
8824: 调用STANDALONE_MAIN，开始处理请求。
8907: 在服务器配置server_conf中设置监听的端口。
8912: 调用new_connection为端口建立一个连接。
8916: 只要ap_read_request表明仍有消息需要监听，则循环监听。
8919: 处理到来的请求。
8927: 如果连接需要结束（没有指明保持连接），则调用ap_bclose函数。

119. joblist

9029: 定义由主线程和工作线程共享的工作列表结构。

120. globals

9039: 定义由线程函数所使用的全局变量，然而它们不是用于父进程的。

121. add_job

9063: 从连接客户机端口的表中增加一个接受的端口。此列表被allowed_globals.jobmutex变量所保护，不允许多个并发的访问。这是对多线程操作设计的大段代码中的第一个函数。另见8935行开始的源代码注释。
9067: 检测jobmutex域的有效性以决定它是否能被当前线程所使用。在9070行引用该互斥变量，然后分配一个新工作（new_jop）并在9071行开始初始化过程。
9078: 将new_job加入到allowed_globals连接的工作列表中。然后在9084行和9085行释放信号量和互斥变量。

122. remove_job

9088: 从连接客户机端口列表中移去一个接受的端口。
9093: 如果给这个服务器定义了拙劣重启，则在9100行调用WaitForMultipleObjects函数来取得互斥量的控制，使它可以被释放。

9102: 通过ap_assert来取得对工作的控制。如果不能取得，则在9106行调用APD1函数，函数返回一个－1错误代码。

9110: 对不用拙劣重启的服务器，调用acquire_semaphore函数来为当前工作检索信号量。

9112: 调用ap_assert来检测工作的互斥变量，然后在9117行访问该互斥量。

9121: 在此行或9129行调用ap_release_mutex来释放工作，根据拙劣重启定义的状态来决定。

9131: 释放工作存储空间并返回端口值（sock）给调用函数。

123. child_sub_main

9159: 这是在多线程环境中处理工作线程的主函数。在等待一个待提交工作后，所有该连接的请求均得到处理，然后调用remove_job来清除它。另见9136行开始的源代码注释。

9197: 调用ap_set_callback_and_alarm初始化此子线程。

9214: 调用remove_job获取一项工作来执行。该函数返回一个可用的端口号。

9228: 在9228行调用getsockname和在9234行调用getpeername来准备端口信息。

9241: 调用带csd端口号的sock_disable_nagle函数来禁止Nagle算法。这样可以防止网络上性能的降低。

9259: 调用带有端口号（现在在ptrans中）和服务器配置的new_connection函数来创建一个连接。

9271: 用ap_read_request循环读取当前连接中所有的请求，分别执行。当不再有请求时，终止连接。

9276: 如果请求的状态域为HTTP_OK，则在9277行调用ap_process_request来处理请求。这个API调用使用所有可提供的模块和给客户机的返回数据。

9282: 当连接由于不是一个（没有设置）保持活动连接或者被通知放弃退出时，则退出当前while循环。

9291: 调用ap_sync_scoreboard_image更新服务器记分牌文件。

9300: 当退出了一个连接的while循环后，调用ap_kill_cleanups_for_socket准备关闭连接。

9310: 如果可能——根据客户机连接的状态，调用lingering_close来避免切断连接，直到客户机完成接收所有的缓存数据并返回一个应答为止。

124. child_main

9324: 此函数只是单独的用来提供中点跳转以找到child_sub_main函数。这样可以避免由于在放弃一个请求时需用longjump而引起的麻烦。

125. cleanup_thread

9338: 清除线程所使用的变量。

9343: 调用free_thread来释放线程变量，然后在所有线程句柄中循环移去指向该释放线程的索引。

126. wait_for_many_objects

9360: 允许Windows NT等待多于64个对象。这个函数辅助或替换了对WaitForMultipleObjects的调用，WaitFor MultipleObjects函数仅允许最多64个对象。

9377: 循环多次，在9379行调用WaitForMultipleObjects。

127. main_control_server

9401: 这是主执行函数，在此定义为一外部函数。保存在hellop.c文件中。

128. setup_signal_names

9421: 对多线程环境中使用的信号初始化信号名。

9423: 使用ap_snprintf组装signal_name_prefix、signal_shutdown_name和signal_restart_name。在9432行调用带有signal_name_prefix的APD2函数。

129. worker_main

9442: 这是对子进程的主控制函数。它管理控制线程，用child_sub_main函数处理每个连接。

9481: 如果设置了one_process标志，调用detach退出此函数。

9504: 调用WaitForMultipleObjects进入一等待循环，等待调度去检测需要处理的连接的进入端口。

9519: 发一退出信号，清除当前存储区池，从此线程的剩余中清除记分牌，在9521行退出。

9526: 在端口上建立一个轮循。调用setup_listeners做好准备以获取下一个到来的连接的控制。

9562: 分配一个线程处理到来的请求。

9623: 调用ap_select监听连接的端口。

9675: 调用accept接受端口进行处理。

9698: 调用add_job把当前工作加入到待完成工作中。

9708: 设置了工作之后，调用ap_release_mutex来释放互斥变量。

9723: 9724行调用wait_for_many_objects函数循环等待多个线程（有可能多于64个）。

9738: 循环杀掉在最后的while循环中（9730行）没有清除的工作线程。对任何剩下的线程，在9739行调用kill_thread，然后在9741行调用free_thread函数。

130. create_event_and_spawn

9781: 创建一个子Apache进程，检测待处理的请求，在收到信号后退出。

9801: 产生一事件，在适当的时刻能给这个子进程发信号结束。

9811: 为此子进程设置参数，将其组装到argv变量中。

9833: 调用spawnv函数生成一个子进程，包含argv参数。

131. cleanup_process

9858: 清除子进程变量。

9864: 对句柄和事件均调用CloseHandle，然后在9869行循环以从数组中移去关闭的句柄和事件。

132. create_process

9880: 创建一个新的子进程。

9888: 调用create_event_and_spawn生成子进程，给child_handle指定句柄，该句柄放在9893行的handles数组中。

133. master_main

9905: 管理所有的Apache进程，在需要时产生新的子进程。

9965: 创建一初始互斥变量来处理请求。

9969: 循环直到不需要创建进程来处理到来的请求（根据此服务器定义的限制）。在9971行调用create_process函数来启动每个新进程。

9990: 通过ap_read_config读取服务器配置来准备在子进程中处理请求，在9992行用ap_open_logs打开日志文件，9993行设置版本号，在9994行用ap_init_modules来为此服务器初始化模块。

10002: 循环直至发生错误或者退出信号打破此循环。

10015: 创建新进程直到10022行的processer_to_create为0为止。

10089: 对一个等待关闭，调用cleanup_process来清除进程变量。

10109: 对一个等待关闭，向所有子进程发信号关闭。

10182: 关闭进程没有很好的进行。调用TerminateProcess来结束进程。

10205: 最初为此进程所获得的互斥变量被消除。

134. send_signal

10216: 这个函数用来发信号给Apache进程，使其运行。

10224: 获得进程ID号。将给该进程发送信号。

10249: 设置发给进程的信号的信号名。

10252: 如果信号关闭，则调用ap_start_shutdown函数。

10254: 如果信号重启，则调用ap_start_restart函数。在此函数中其他信号被忽略。

135. apache_main

10265: 这是对服务器的主控制函数。如果运行在Windows上，函数名称为apache_main，由dllexport输出。如果运行在其他平台上，这个函数叫作REALMAIN（10267行）。

10279: 调用common_init来初始化普通变量。

10311: 调用ap_setup_prelinked_modules来设置预链接（非DSO）模块。

10313: 循环执行Apache启动时包含的每一命令行选项。其中有些选项打印消息并退出。

10316: 开始一个switch语句以处理每个支持的Apache命令行选项。

10404: 读取服务器配置到server_conf中。

10420: 用ap_set_version为Apache设置版本字串。在10421行调用ap_init_modules初始化所有模块。在10425行调用ap_open_logs打开日志文件。

10441: 调用ap_create_mutex获得互斥量以开始处理请求。在10450行调用worker_main作为最高级工作线程，然后在10451行消除互斥量。在10452调用exit_event来退出。

10455: 对非线程环境，调用service_main来处理请求并在需要时生成另外的Apache进程。

136. ap_main

10476: ap_main函数定义为外部的，这样它就可以被非多线程的环境所调用。

137. main

10478: 调ap_main启动所有内容。

10521: 这是独立的、非多线程环境的主程序。

10536: 循环分析命令行，但仅有R选项在此处进行（10556行）。

10604: 设置了共享存储信息后，执行下一个对execve的调用以启动共享核心为可执行的。

4.3 http_log模块

http_log文件包含用于错误日志和其他在Apache工作过程中发生的消息的函数。消息有可能在Apache启动时立即发送给这些函数（例如，见在REALMAIN函数中的8848行）。错误日

志也可以包含系统日志，在Apache指令中指定的日志文件，标准错误（STDERR），或正规的printf语句。http_log中的函数处理所有这些可能的情况。

1. facilities

10634: 定义Apache运行的系统中指向所能提供的不同类型的日志文件的索引的数据结构。它们包括FTP、mail、lpr和标准系统日志等日志文件。

2. priorities

10697: 设置包含有不同日志级别的结构，从emerg（其中几乎什么事也不记录）到debug（其中把任何可能的信息片段都记录下来）。

3. error_log_child

10709: 当配置的错误日志位置以管道标识开始时（|），设置子进程来存储错误日志项。

10719: 调用ap_cleanup_for_exec来准备Apache启动一个子进程。

10722: 发送一个信号以开始产生子进程。不同的平台用不同的命令来启动子进程。它们由编译器的define语句来选择。

4. open_error_log

10741: 在Apache处理过程中，打开错误日志来准备志记错误信息。

10745: 如果错误日志的文件名（error_name）以管道标志（|）开始，则在10748行调用ap_spawn_child开始在管道的另一端处理。

10769: 如果此平台上用了系统日志，并且文件名也指明了是syslog，则使用openlog系统调用。

10784: 如果没有指定系统日志或此系统不提供，则在10786调用ap_pfopen打开日志文件进行追加写。

5. ap_open_logs

10794: 打开Apache配置中所定义的各种错误日志。

10826: 在所有的虚拟服务器配置中循环，在10835行对每个服务器配置中定义的日志文件调用open_error_log函数。

10840: 如果一个虚拟服务器没有指出错误日志，则error_log域为该服务器的配置指向主服务器（缺省）的error_log域。

6. ap_error_log2stderr

10844: 这个API函数记录一个错误到系统的标准错误（STDERR）中，在10847行使用对dup2的调用。

7. log_error_core

10850: 将一个带有日志级别level，以参数形式传入此函数的错误信息记入日志中，此函数由几个高级的日志记录函数调用，这些函数准备了将记入日志的语句。

10870: 在10881行将logf设置到stderr或日志文件的文件名中。

10894: 如果logf有效，则组装记入errstr的字符串。

10906: 检测为此服务器定义的日志记录级别（APLOG_LEVELMASK）是否允许当前级别的信息被记录。

11008: 如果在检测了几项与平台相关的项之后，logf文件扫描器仍然有效，则用fput和fputs（11009行和11010行）或者用一个对syslog的系统调用（11015行）来记录

errstr的错误信息。

8. ap_log_error

11020: 这个API调用记录一个错误。此函数在整个Apache源代码中被调用以记录执行过程中产生的错误。

11027: 调用log_error_core函数将错误信息写到正确的地方。其中日志级别指定此级别的信息需要记录。

9. ap_log_rerror

11031: 这个API调用记录一个请求错误。

11038: 调用log_error_core将错误行记录到正确的位置。

11049: 针对WARNING级别或在priorities结构中（10697行）更高级别的错误，在11051行错误信息将被置入请求记录的notes域中。

10. ap_log_pid

11057: 将当前Apache进程的进程ID号记录到PIDFile中。

11067: 将日志文件名转换成与服务器根相对应的名称。

11068: 为当前进程检索PID。如果没有返回一个有效值，则在11078行记录一个错误。

11087: 打开PIDfile。

11094: 将PID写入PIDfile，然后在11095行关闭文件。

11. ap_log_error_old

11099: 这个API函数立即以合适的参数调用ap_log_error。此函数的参数中并不包括日志级别，所以在给ap_log_error传递消息时包括一个缺省日志级别。

12. ap_log_unixerr

11105: 这个API函数立即在11109行调用ap_log_error函数，用传给ap_log_unixerr的预定日志级别和消息文本作参数。

13. ap_log_printf

11112: 这个API函数直接用log_error_core记录一个信息，而不是调用ap_log_error。在对log_error_core的调用中用了缺省日志级别；对此函数参数中不包含任何信息。

14. ap_log_reason

11123: 这个API函数用一个原因字串（作为参数传入的）调用ap_log_error来记录一个错误到日志中（11126行）。提供了缺省日志级别和完整的文本字串。原因字串也插入到其中，远端主机的名称也在此字串中。

15. ap_log_assert

11134: 这个API调用打印一条消息到标准错误输出中，然后尝试退出(11143行)

16. ap_log_spawn

11156: 通过生成在日志配置指令中定义的子进程来设置一个管道日志。

11160: 启动一个子进程时封闭信号量。

11161: 用fork启动一个新进程，将进程ID号赋给pid。

11174: 调用exec函数之前先做清除工作。合适的信号将传给子进程（11175行和11176行），然后在11177行调用execl，在此带上程序启动的路径（它将通过管道接收日志项）。

11192: 清除对报警的封闭以便能再接收信号。

11194: 调用ap_register_other_child使得Apache了解管道日志的子进程。

17. piped_log_maintenance

11200: 保持用于接收管道日志项的子进程。

11205: 开始一段switch语句，根据传入的原因参数来执行相应动作。

11209: 对一个需要结束的子进程，调用ap_unregister_other_child。

11224: 如果子进程不能够接受日志项（它不能够被写入），或者请求重启，在11276行或11233行以SIGTERM信号调用kill。

18. piped_log_cleanup

11243: 当用于接收管道日志项的子进程结束时，清除该子进程的存储区。如果子进程的PID仍然显示其处于活动状态，则在11248行以SIGTERM信号调用kill函数，然后在11250行调用ap_unregister_other_child清除该进程的存储区。

19. piped_log_cleanup_for_exec

11256: 通过关闭任何存在的文件描述器（11266行和11261行）产生一个子进程来接受管道日志项。

20. ap_open_piped_log

11265: 这个API函数生成一个用来通过管道接收日志项的新的子进程。

11275: 调用pipe为程序参数所定义的子进程打开一个管道，该参数在11272行被赋给pl变量。

11281: 用ap_register_cleanup来记录管道，以便在退出时被清除。

21. ap_close_piped_log

11296: 这个API函数关闭一个用于通过管道接收日志项的新的子进程。

11298: 关闭进程时封闭信号量。11299行调用pid_log_cleanup函数的清除工作是为结束进程作准备，然后在11299行调用ap_kill_cleanup来杀死此进程（用pl变量来标识需清除的进程）。

22. piped_log_child

11305: 当Transferlog指令带有管道日志来使用时，建立一个子进程。

11314: 调用ap_cleanup_for_exec以准备执行一个子进程。子进程以适合于Apache所运行的平台的方法开始（Apache运行的平台在编译器define语句中设置）。

23. ap_open_piped_log

11337: 这个API函数在11343行调用ap_spawn_child函数打开一个管道日志子进程。返回的pl变量用于子进程的管理。

24. ap_close_piped_log

11359: 在11361行调用ap_pfclose函数来关闭一个用作管道日志的文件。

第5章 环境变量模块

本章描述两个模块，mod_env和mod_setenvif模块，它们用于设置和取消环境变量。这些变量要么用在你提供给服务器的通用网关接口（Common Gateway Interface CGI）脚本中，要么用于Apache内部进程中以决定在某个环境中如何响应。

这两个模块均只用于配置文件处理过程中。它们在Apache运行时设置环境变量，但并不能被调用来执行客户机的请求。缺省时这两个模块均包含在Apache中。

5.1 mod_env模块

mod_env模块根据你的服务器所给的配置文件来设置和取消环境变量。这些变量对任何运行于系统上的脚本或程序均是可用的。

变量可以被赋予一个值，或者仅简单地设置，意思是说该值为真或非空；其反面是取消或为空（NULL）。若想设置你要定义的环境变量并使其在一个服务器或目录上下文（容器类）中可用，可用SetEnv指令，带不带值都行。

```
<VirtualHost 192.168.100.13>
...
SetEnv VHOST linguistics
SetEnv INTRANET
</VirtualHost>
```

变量值也可以被取消，意思是说，如果该变量被访问后，它就可以不必再存在或不再有有效的值：

```
UnsetEnv INTRANET
```

当一个请求中含有特定的虚拟主机、地点，或目录等信息时，SetEnv指令定义的变量提供给环境作为它的一部分使用。

PassEnv指令可用于给一个服务器上下文提供任意预先存在的系统环境变量；即，根据Apache所运行的操作系统，设置某种OSTYPE，但是却不能够供你的脚本使用，除非用PassEnv指令来指明可用：

```
PassEnv OSTYPE
```

注意此处并未定义值。值独立于Apache而由系统设置；这一语句只是使其对服务器下级进程，比如脚本可用。

5.1.1 模块结构

由于mod_env仅用于配置文件的分析过程中，因此它只包含不多的几个函数来处理三个它所要处理的指令：SetEnv、UnsetEnv和PassEnv。每个指令都是自包含的。因为它们在任何发现相应指令的地方被调用，所以它们没有特别的顺序。有一个额外的函数，fixup_env_module，用在16867行开始的修正阶段。此函数从下级请求中合并任何活动的环境信息，该下级请求归到有可能使用环境信息的父请求中处理。mod_env的体系结构见图5-1。

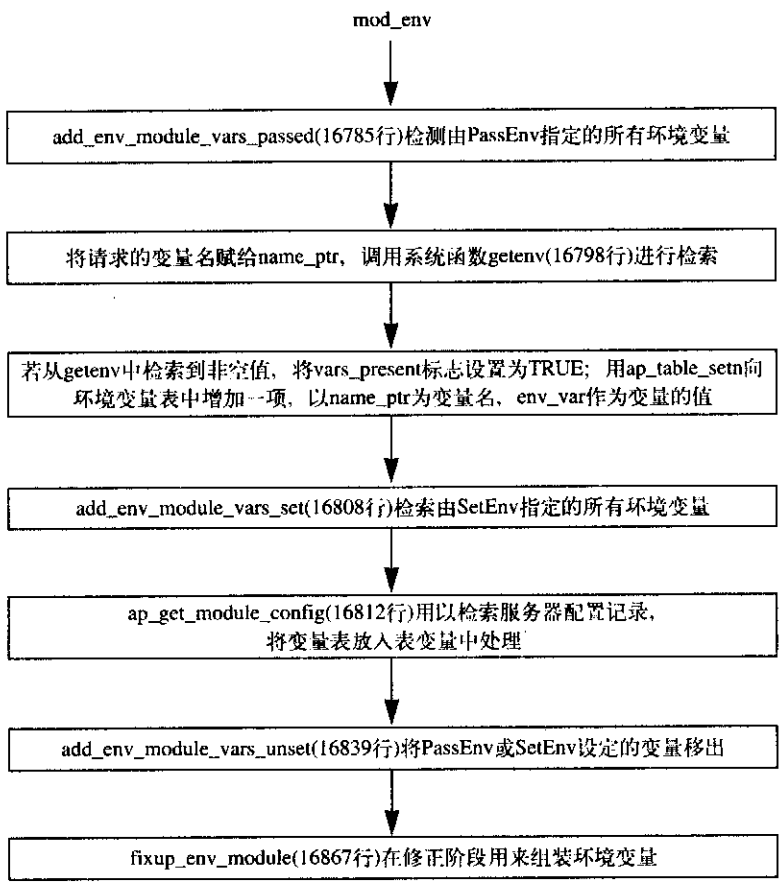

图5-1 通过mod_env，Apache能够将环境变量传递给CGI脚本和服务器端引用

5.1.2 定制

在缺省情况下，mod_env允许最多50个环境变量。如果需要更多的，在16727改变这个值。同样合并配置的缺省动作是最近的一次配置将以前的配置记录覆盖。这点颇有好处，但并不是在所有情况下都适合的。merge_env_server_configs函数（从16733行开始）可以被更新以产生冲突警告或者使用一套你定义的规则来决定哪个变量具有优先权。

很多函数，比如文件名转换和检查用户授权，可以在执行的不同阶段加入到执行过程中，以便mod_env根据其他模块的动作来设置变量，这样信息就对基于此结果的脚本可用了。这个类型的信息可以直接在其他模块中设置，但却不依赖于对mod_env的更新。这样做可以避免将mod_env和其他的日志记录或文件名转换等混淆起来。

1. env_server_config_rec

16713: env_server_config_ree结构包含了可用于Apache脚本的环境变量列表。也包含一个取消变量计数（unsetenv），以及存在于表中的变量的标志。（如果标志位显示没有变量存在，mod_env可以更快地处理一个请求。）

16719: 模块定义是向前索引的。从16985行开始。

2. create_env_server_config

16721: create_env_server-config函数通过创建一个配置记录变量并对其初始化为当前服务器上下文建立服务器配置记录，即vars_present=0（16729行）和unsetenv=""（16728行）。

3. merge_env_server_configs

16733: merge_env_server_configs函数包含两个服务器配置记录。在环境变量中，任何的冲突都通过以新值替代旧值的办法来解决。

16736: 创建变量用来保存基本的、附加的和新的配置记录值。

16751: 这几行代码（到16759行）被注解出来由16761到16775行的代码来代替，16761到16775行代码使用了ap_copy_table API函数。

16761: 通过从基础服务器配置中复制变量来初始化创建的新配置记录。

16766: 每个新（add）配置中的变量由ap_table_setn函数加入。如果在基础配置中含有同名变量，其值将被覆盖掉。

16770: 在16773行用ap_table_unset函数以new_table将add配置中的取消变量列表组合加入到new_table中。

16777: 新创建的表被放入新配置记录，new中。

16779: 通过一个逻辑或（OR，用"11"操作符），在base和add配置记录中的var_present域变量被组装到new配置记录中。

16782: new变量以合并的配置记录返回。

4. add_env_module_vars_passed

16785: 当发现有PassEnv指令时，调用add_env_module_vars_passed函数。此函数接收一个环境变量（Apache从指令参数中传入）并将其加入当前服务器配置记录中，这样变量可以被Apache启动的脚本或程序所访问。

16789: 调用ap_get_module_config检索当前配置记录。

16792: 设置一独立的表保存环境变量。

16796: 所有由PassEnv传递的参数都被解析。对每个参数，每个请求的变量名称被赋给name_ptr并在16798行系统调用getenv来检索它。

16799: 如果从getenv中检索的值不为空（NULL），（它存在于系统中），在16800行将vars_present标志设置为true（因为将要往列表中加入一个变量，至少有一个变量出现）。下一步，用ap_table_setn往环境变量表中加入一项，以name_ptr作为变量名，以env_var作为变量值。

5. add_env_module_vars_set

16808: add_env_module_vars_set函数创建一个由SetEnv指令指定的新环境变量。它们被保存在服务器配置记录中，可供Apache运行的脚本使用。

16812: 由ap_get_module-config检索服务器配置记录。为方便访问，变量表放在表变量中。

16818: 从ap_getword-conf函数提供的args中收集由SetEnv指令定义的变量值和变量名。每个定义的变量都必须有name域；value域可选。如果没有提供值，则仅设置变量（其值为true或非空）。

16827: 变量被放入表之前应检查其有效性。由于可能在使用变量时不带值，在16859行指定RAW_ARGS说明Apache在调用当前函数之前无需完全检验参数的有效性。

16833: 因为一个有效的名字－值对保存在配置记录中，vars_present设为true，表示配置记录中含有至少一个环境变量。用ap_table_setn在16834把环境变量（名字和值）放入配置记录的vars域。

6. add_env_module_vars-unset

16839: add_env_module_vars_unset函数移走一个由PassEnv或SetEnv设置的变量，这样这个变量就不能够再被Apache启动的下级进程所使用了。

16843: 当前服务器配置记录被置入sconf中。

16846: 根据被请求取消的当前消息变量列表和参数（arg），在sconf中的unsetenv被设置一个新值。这个列表可以包含多个变量；这些变量均在此用命令来取消。

7. env_module_cmds

16853: env_module_crds结构定义了当此模块在配置文件中遇到并处理指令时，调用什么函数。在16855行、16858行、16861行开始处定义了三个指令（PassEnv、SetEnv和UnsetEnv）。每个指令名后的第二个参数是处理该指令的函数调用。它们是以前分析过的函数。所有这三个指令都使用了RAW_ARGS来描述能够包含什么样的指令。Apache核心代码并没有真正的检查这些指令行中传递了什么样的参数。add_env_module_vars_set函数检测这些参数以决定值是否与参数包含在一起（见16827行）。add_env_module_vars_passed和add_env_module_vars_unset都将参数存入配置记录中而不作任何检查（因为参数是一个无序的变量名列表，不需要真正的检测）。

8. fixup_env_module

16867: 修正阶段通过fixup_env_module函数用在mod_env中，用来组合下级请求和父请求之间的环境变量。下级请求变量在16869行放于e中；主请求变量在16874行通过服务器配置放入vars中，在16872行访问服务器配置。两套变量在16880行用ap_overlay_tables进行组合。

9. env_module

16885: mod_env的模块定义包括创建和合并服务器配置，与此模块相关指令的命令表和修正阶段等的函数。多数的模块均定义有基本阶段的函数（比如句柄处理、文件名转换、或检查用户ID等阶段），但mod_env并不是这样。

5.2 mod_setenvif模块

mod_setenvif模块设置了可供给由Apache启动的脚本和程序使用的环境变量。除了mod_setenvif使用来自客户请求的信息来决定如何初始化环境变量外，此模块与mod_env相似。

带有客户请求的发送头，以及几种特殊情况例如客户机IP地址（定义为Remote_Addr）等可以被指定为属性，以便与规则表达式进行比较。如果表达式匹配，则设置了一个环境变量。

SetEnvIf指令用于任何配置上下文中以定义如何设置环境变量。例如，

SetEnvIf Remote_Addr 192.168.100.3 USER=DAVID

如果客户机的IP地址与表达式中所列的相符，则将变量USER设置为DAVID值。任何客户机的头均可用作被检测的属性，任何规则表达式（例如带有 * 和位置性检测的规则表达式）均可用作第二个参数。变量可以被赋以一个值，像这个例子中一样，或是简单的设置（值为

true或已定义的，而不是没有定义过的）。

另一个指令，SetEnvIfNoCase，对SetEnvIf指令非常重要，但在操作时无需考虑对属性和检测表达式的比较。其他两个指令，BrowserMatch和BrowserMatchNoCase，是SetenvIf的特殊情况，为向后兼容性的缘故，仍然在使用。二者均检测User_Agent头并允许你根据使用的浏览器设置一定的环境变量。

几个内部环境变量也被定义以便与BrowserMatch一起使用。这将影响到Apache对请求的响应。例如，设置变量force_response_1.0可使对一个请求应用HTTP/1.0。

5.2.1 模块结构

与mod_env相似，mod_setenvif仅用于配置文件分析过程中，它包括一个核心函数来处理指令：add_setenvif_core，它在34304行开始。调用其他函数来分析此模块中使用的指令，但它们反过来也以不同的值调用add_setenvif_core函数。这个模块也使用一个张贴-读-请求函数——34467行的match headers，以优化对当前上下文定义的属性的访问。体系结构见图5-2。

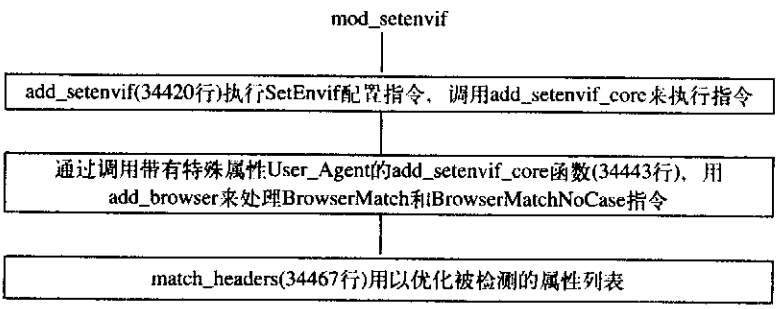

图5-2　mod_setenvif用以控制或设置环境变量

5.2.2 定制

mod_setenvif使用了能用其指令（例如前面的Remote_Addr例子和Request_URL属性）设置的几个内部变量和特殊表达式。这个模块也可以被扩展到允许Apache所运行的脚本和程序中所需要的附加特殊属性以控制环境变量。这些特别定义的属性必须在Apache核心中有对应部分，这样当请求被执行时，其实际值可以与规则表达式相比较；然而，mod_setenvif可根据标准指令来允许规则指令设置这些附加值。下面是一个简单例子，如果定义了一个叫做Time的特殊属性，那么可以用一个标准的SetEnvIf指令来控制一个环境变量使其对脚本可用。如果Time以偶数开始，则设置TIMEVAR变量；Apache运行的脚本可以测试TIMEVAR并做出不同的响应：

```
SetEnvIf Time ^[02468] TIMEVAR
```

这个例子需要核心Apache代码在处理请求时提供关于Time的信息；仅在mod_setenvif中加入Time类型的支持还不够。

1. special

34248：这个计数变量定义了Apache内部定义的特殊属性，它可以为一个请求做检测来判断是否设置了一个环境变量。第一个值，SPECIAL_NOT，说明被指令检测的属

性是一个请求头，而不是此处所列的特殊属性。

2. sei_entry

34256: sei_entry结构用于保存基于mod_setenvif中使用的指令的完整项。它包括测试规则表达式的域，用例感应性（icase）和在被测试项与规则表达式匹配的情况下要设置的变量。

3. sei_cfg_rec

34267: 对mod_setenvif的配置记录在此定义。它是一个保存指令参数原始数据的数列。

34271: 模块的定义在此向前引用。实际的定义从34555行开始。

4. create_setenvif_config

34273: create_setenvif_config函数为配置上下文创建一个服务器配置记录。如果用ap_palloc在34276行定义了一个新sei_cfg_rec变量，那么就用ap_make_array在34279行用一个有20项的数组填入它。

5. merge_setenvif_config

34284: merge_selenvif_config函数在两个服务器上下文必须被合并时合并两个条件数列。base配置和overrides合并，并放入a。在其他模块的大多数合并函数中，合并的配置被称为add或其他类似的名称。overrides的名称是一个提示信息，提示任何在附加配置中的属性测试和将要设置的变量，如果在发生冲突时，将要覆盖base配置。

34302: 一个常量，ICASE_MAGIC，被定义为标志以表明用例敏感性是否对正在被处理的指令信息可用。此处给这个常量赋的值是任意的——一个函数的地址被用到。变量实际上用在setenvif_module_cmds结构中（见34455行和34461行）。

6. add-setenvif_core

34304: add_setenvif_core函数处理mod_setenvif中的任务。它由几个很短的函数调用，这些函数带有指导其动作的参数。

34310: 在此上下文中的服务器配置被查询。之后add_setenvif_core函数向此配置中加入一些项。在34313行定义的sei_entry保存在新定义的项。

34321: 用来测试由这条指令查询参数列表所给出的属性的规则表达式被恢复。如果它不在，则返回一个错误——每条指令必须提供一个规则表达式。

34335: 用一个for循环来查看所有当前应用于此服务器上下文的项。如果存在一个当前指令所指的属性名称项，则属性名称（fname）被设置为列表中业已存在的对应名称，并跳出循环。

34347: i被赋给加入当前上下文的基础数，34348行icase被设置为用例敏感性值（根据当前指令中命令结构所包含的内容）。

34349: 如果指令要求创建的项还不存在（匹配属性和规则表达式），那么它将在34356行继续创建一个新项，由创建一个保存它的变量（new）开始。

34357: 属性名在fname中，用于测试的规则表达式在regex中，用例敏感性设置在icase中。编译过的规则表达式（为了在真正处理一个请求时快速执行）用ap_pregcomp来创建，并被置于new的preg域中。如果规则表达式不能够被编译，它将会变形，意味着它不能够被用来检测。在这种情况下，函数立即退出，并且不把规则表达式加入preg域。在34364行返回一个错误。

34367: 特性（如果规则表达式匹配，要设置的环境变量）由用ap_make_table创建的一个两项表定义。

34369: 下面几行为任何以34248行开始定义的特别属性测试属性名。如果这些属性用在当前指令中，则计数型的值将被置入new的special_type域中；否则该域被置为SPECIAL_NOT值（在34385行），说明在Apache为实际请求测试这些值时，特殊属性不存在。

34388: 如果因为已经存在一个new项而不再需要new项时，new被设为与该项等价。在它被放回配置之前，可以往其中加入任何特性。

34392: 如果性能与在34393行获取的每个剩余参数所给的规则表达式相匹配，在做某动作时，每个包含的特性都被检测。然后用ap_table_setn在34401行将特性加入到new的feature域中。

34411: 如果不提供某特性，将返回一个错误——在设置变量时必须定义一些东西。

7. add_setenvif

34420: add_setenvif函数用来处理SetEnvIf指令。它定义了作为fname检查的属性，然后调用add_setenvif_core来真正处理指令。RAW-ARGS与SetEnvIf一同使用（34453行）。在将控制法传给add_setenvif_core之前，add_setenvif函数检测参数列表中的某一部分是否为有效属性。同一函数也为SetEnvIfNoCase调用。ICASE_MAGIC代码（34455行）在处理过程中由add_setenvif_core查询以决定用例敏感性。

8. add_browser

34443: add_browser函数是add_setenvif的特殊例子。它通过调用带有特殊属性User_Agent的add_setenvif_core函数来处理BrowserMatch和BrowserMatchNocase指令。

9. setenvif_module_cmds

34450: setenvif_module_cmds结构定义了用来处理此模块中使用的指令的函数调用。RAW_ARGS用于所有的指令，将处理过程留给被调用函数处理。这样做是因为环境变量有可能有一个指令的值或被单独命名，表明将不指定任何特定的值而设置它们。ICASE_MAGIC代码用作标志来表示用例敏感性。add_setenvif_core被此处定义的所有函数引用；如果正被执行的指令是一个...NoCase指令，那么ICASE_MAGIC对add_setenvif_core是可用的。

10. match_headers

34467: match_headers函数用于优化被检测过的属性列表，它们可能已经在Apache配置中被处理过多次。在获得配置后（34477行），一个for循环（34484行）查看每一项并为内部（非HTTP头的）属性提供特殊值，在34498行开始（看第34507行和34510行，那里来自请求的域被赋予val）。

34534: 根据所用的规则表达式，设置（added）或取消（deleted）数列中的元素。

11. setenvif_module

34555: 模块自己以用于合并和创建配置的函数调用来定义，处理指令，并在张贴－读－请求（用match_headers函数）过程中执行。

第6章 帧头处理模块

本章介绍4个小型模块，它们主要定义了针对不同的用户请求、服务器应答信息的各种帧头。这4个模块是：

- mod_asis：把帧头不加改变地返回给用户。
- mod_cern_meta：把一个已知文件的内容作为应答信息的帧头返回给用户。
- mod_expires：根据请求信息中MIME数据的类型，为Apache服务器的应答信息加一个“到期”帧头。
- mod_headers：根据Apache配置文件中的指令，更新返回给用户的某些特殊的帧头。

以上模块定义的帧头不能由其他模块自动产生。例如，元数据中的文档作用域只能由mod_cern_mate或mod_headers模块将其嵌入服务器应答信息中。

6.1 mod_asis模块

我们可以利用AddHandler指令，为send_as_is处理程序选择任何路径或目录：

```
<Directory/directory name>
AddHandler send-as-is pre
</Directory>
```

有了上述配置信息，当用户访问上述目录中扩展名为.pre的任一文件时，将使用mod_asis发送该文件内容。文件的首部可能包含任何必需的头信息，但Apache不产生任何附加头信息。

6.1.1 模块结构

mod_asis模块的结构很简单，如图6-1所示，没有用到与目录和服务器相关的配置信息。模块中仅定义了一个处理程序表，该表为send-as-is提供一个处理程序。

send_as_is中定义了一个函数（asis_handler，起始行为12344）。该函数检查请求方法的有效性及文件权限，然后把被请求的文件发送给用户。

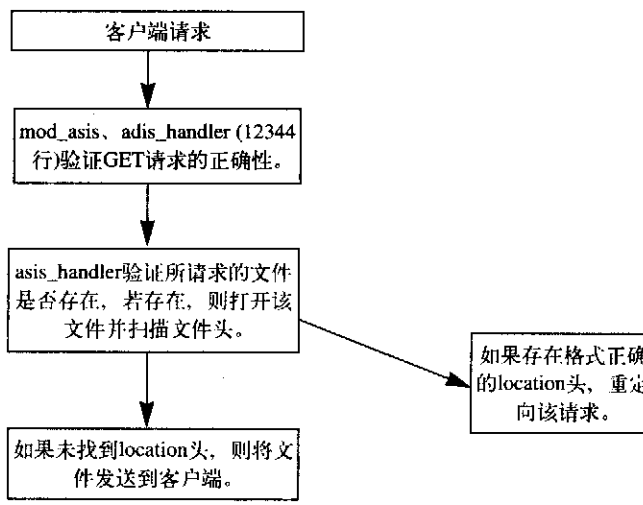

图6-1 mod_asis使Apache能够不加改变地发送文件给用户，不附加任何信息头

6.1.2 定制

send-as-is的最初目的是简化已设置了send-as-is的目录的处理过程。对mod_asis性能上的改进可能会削弱其不复查、不改变或者不附加地发送文件的初衷。

1. asis_handler

12344: 被请求的文件没有附加任何头信息就被发送给客户。(该模块假设webmaster已经把有关信息头作为文件内容包含到文件中)。

12349: 检查请求，判断其是否是GET请求并且GET是允许的。

12352: 检查被请求的文件，确定其存在。

12359: 文件被打开以备发送。如果文件打开失败，错误信息将在12362行记录。出错的可能情况是权限问题。

12368: 检查将要发送给客户的信息头。如果有一个结构正确的路径信息头，则该请求应被重定向。于是程序不发送该文件，而是在12373行检查请求的状态信息，关闭在12359行打开的文件（在12375行），确认使用的是GET方法（12384和12385），最后调用ap_internal_redirect_handler重定向用户请求（12387行）。

12391: 如果文件没有使用路径信息头重定向，一个尽可能小的HTTP信息头将被用来确定由ap_send_http_header发送的信息的格式。

12392: 如果该请求的header_only标志位已被设置，asis_handler转至12395行，关闭文件然后退出（这使得send-as-is响应很快）。否则，调用ap_send_fd发送整个文件（12393行），然后关闭文件（12395行）并退出。

2. asis_handlers

12399: mod_asis模块为其涉及到的处理程序定义了一个结构，处理程序的类型是send-as-is，当一个请求发生在该处理程序的作用区域内时，asis_handler(12344)将被调用。

3. asis_module

12406: mod_asis模块信息在此处定义。定义中仅包含一个到处理程序（12419）的跳转，它是一个asis_handlers结构（12399）。

6.2 mod_cern_meta模块

提供mod_cern_meta模块是为了获得同当前非常流行的CERN Web服务器的兼容性。本模块允许用户定义文件的扩展名，这些文件中包含将返回给用户的元信息（附加信息头）。

例如下面的一段配置指令:

```
<Directory/document_root/products>
MetaFiles on
MctaDir headers
MetaSuffix meta
……
</Directory>
```

当用户请求的文件（例如index.html）位于/document_root/products目录时，mod_cern_meta模块在headers子目录（/docnment_root/products/headers）中查找在该文件名后又追加.meta扩展名的文件（index.html.meta）。这个文件将在被请求的文件之前发送给用户。

实际上，Header指令是提供这类信息的更好的方法，但许多web服务器在初启时都使用CERN元信息文件，因此，mod-cern-meta主要提供向后兼容性。Apache中mod_cern-meta模块的源代码包含有附加说明及例子。

6.2.1 模块结构

mod_cern_meta中仅允许与目录相关的配置。在服务器或目录的配置信息中加入其参数，该指令就会被处理。在fixups阶段，被请求文件发送之前，调用add_cern_meta_data函数 (15417)。该函数检查由配置指令为当前上下文提供的元文件名，打开该文件，然后调用scan_meta_file函数（15353）。该函数读取元文件，检查其中包含头信息的有效性，然后把它们加到将要发送给客户的headers_out数组中。而信息头的发送任务则由Apache的后续过程来处理。模块结构如图6-2所示。

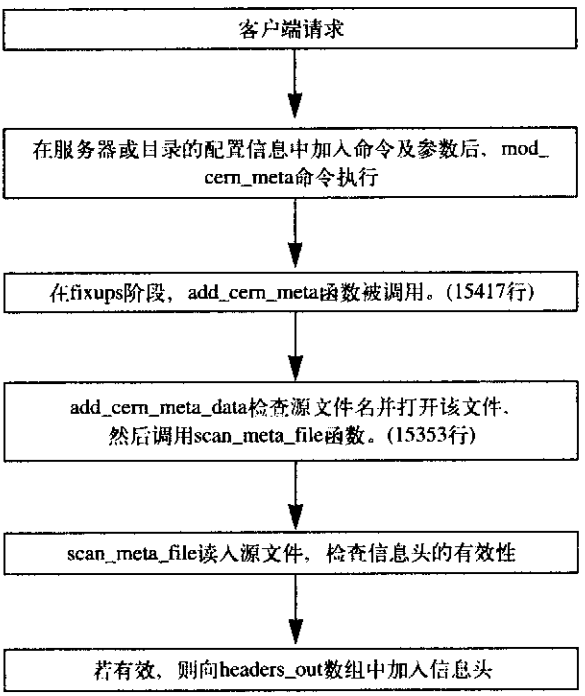

图6-2 mod_cern_meta提供CERN HTTPD所用的附加头

6.2.2 定制

因为mod_cern_meta模块是为了实现向后兼容性，所以要避免对其功能的扩展。当然，例外情况是，如果用户在CERN Web服务器中增加了新特性，那么在mod_cern_meta中就应当有相应的改进。

正常操作时，使用mod_headers而不是mod_cern_meta去扩展带有新特性的mod_headers。

1. 编译器定义语句

15268: 如果MetaFiles被设置为“on”，那么将使用缺省的目录和文件扩展名。但Meta Dir或Meta Suffix指令并不定义怎样定位元文件。

15274: 为mod_cern_meta定义配置记录的结构。这包括在为一个目录上下文定位元文件时使用到的目录和后缀，以及在使用MetaFiles指令设置mod_cern_meta状态（on/off）的标志。

2. create_cern_meta_dir_config

15280: 调用ap_palloc（15284行）来为目录配置分配一个目录配置记录，该记录的所有值都被置为NULL或0。

3. merge_cern_meta_dir_config

15294: 合并两个目录配置记录：base和add，把它们加入到新的配置中。在base和add中都要对元目录和元后缀进行检查，但元文件对应的域要从add变量中删去，而无需检查base变量。

4. set_metadir

15314: 设置在定位元文件的所用到的目录。当遇到一个MetaDir指令时，该函数被调用，正如指令表中定义的那样（15341行）。当前目录配置记录中的metadir被赋予从Apache指令（见15341行的TAKE1参数）传进来的参数值。

5. set_metasuffix

15321: 设置在元文件目录中定位元文件时所用到的扩展名。正如15343行定义的，当遇到MetaSuffix指令时，该函数被调用。当前目录配置中的record被赋予指令提供的值。

6. set_metafiles

15328: 当前目录配置中metafiles域的状态被赋予Metafiles指令提供的值。该值只能为on或off。set_metafiles函数在15338行定义，用来处理MetaFiles指令。

7. cern_meta_cmds

15336: 此处定义了一个指令表，它为每个涉及到mod_cern_meta的指令都指明一个函数，当在配置文件中遇到指令时，就根据该表调用相应函数。

8. scan_meta_file

15353: 扫描作为文件描述器（专门用来描述其他文件的文件）而输入的文件，检查其信息头的有效性，然后把它入加入到当前请求的headers_out域中。该函数由add_cern_meta_data函数（15505行）调用。

15360: 定义一个变量去存放由元文件分析得到的信息。

15361: 以w字符串和f文件描述器作为参数调用fgets来读出整个元文件，并写入到变量w中。

15365: 从字符串w中删去所有换行和回车。因为每个信息头都作为一个数组元素加入到请求的headers_out域中，所以无需使用换行字符。当所有的信息头都作为输出返回给用户时，行格式信息就会被加入。

15380: 如果没有冒号，那么信息头就是非法的。这时ap_log_rerror（15381）就会作一个错误日志，并在15385处返回一个错误，检查是否有冒号是确定信息头是否合法的唯一判断。信息头可以由用户定义，可以包含任何信息，只要信息头名字后加一个冒号。

15392: 在元文件中检查Content-Type信息头。如果找到，则对其进行词法分析并放至请求的Content_type域（15402行）。

15404: 检查元文件中的状态头信息。如果有的话，就把它放入请求的status_line域。

15409: 如果Content_Type和status两种信息头都没有被发现，那么ap_table_set就会把元文件中所有的信息头都放入tmp_headers表中。然后调用ap_overlap_tables(15412行)以请求中所有信息头重置tmp_headers变量。

9. add_cern_meta_data

15417: 根据事先定义的当前目录上下文定位并处理元文件。scan_meta_file函数把信息头从元文件写到当前请求的headers_out域。

15428: 恢复当前目录上下文的配置信息。

15431: 检查元文件标志位。如果该目录没有设置标志位，将返回DECLINED。

15437: 如果请求的文件不存在，DECLINED返回。

15442: 如果当前用户请求的是一个目录，那么该模块不会被调用，直到请求被分解到一个特定的文件为止，这时返回DECLINED。

15448: 通过检查路径分隔符来准备一个有效路径名，以打开一个元文件。

15470: 把变量scrap_book（保存当前被请求文件的初始目录）、当前配置中的metadir域、被请求的文件名以及metasuffix域，四者合并起来形成完整的元文件名（因此，/doc/index.html可能对应于/doc/.web/index.html.meta）。

15486: 创建一个Apache子请求使得元文件名能被自动检查。子请求的状态域在15487行被检查，如果状态无效，模块将返回DECLINED，因为元文件不可得。不论哪种情况（DECLINED或continue），子请求都会被立即取消，因为它已完成了检查文件名有效性的任务。

15493: 打开元文件，将文件描述器存入f。如果打开文件过程中出错，该错将被存入日志，返回DECLINED或FORBIDDEN。

15505: 因为元文件被成功打开，scan_meta_file函数读取其内容，检查后将内容放入当前请求的headers_out域。

15506: 关闭元文件，使模块完全退出。

10. cern_meta_module

15511: 此处定义mod_cern_meta。create_cern_meta_dir_config创建与目录相关的配置信息，并合并目录配置信息。该模块使用的指令在15521行处由指令表定义。元文件的处理工作由add_cern_meta_data函数在请求处理的fixups阶段完成（15528行）。

6.3 mod_expires模块

mod_expires模块根据文件的MIME数据的类型来定义文件的到期日期。到期信息包含在信息头中一起返回，使得代理服务器和浏览器能够决定何时从服务器重复取某一文档，而不是从本地缓冲区取其备份。有两种信息头，Cache_Control和Expires，由mod_expires计算并加入请求中。

为了在一个目录上下文中使用mod_expires，ExpiresActive指令首先被设为“on”。Expires By Type指令根据文件的类型定义它们的终止日期；ExpiresDefaut指令能为所有文件设置终止日期，只要文件还没有明确地被ExpiresByType指令定义。

终止日期（如上所述，由ExpiresByType或ExpiresDefault定义）的格式有两种，即访问时间或修改时间，然后都再加上附加时间段。例如：

```
ExpiresByType text/html A604800
```

表明当前上下文中所有HTML文件在被访问后（被客户端浏览器读取）的1个星期内（604 800s）到期。此处还可使用更友好的格式。下例同上例有相同效果。

```
ExpiresByType text/html"access plus/week"
```

第一个域可能值为access，modification或now（与access相同）。字段plus是可选的。有关数值和类型能够重负组合使用，如下例：

```
ExpiresByType text/html"access plus1month 15 days 2 hours"
```

由于其中有空隔符，所以引号是必须的，ExpiresDefault格式除了在终止日期之前没有指明MIME类型外，其余同ExpiresByType是一致的。

6.3.1 模块结构

在使用create_dir_expires_config和merge_expires_dir_configs指令创建或合并目录配置变

量后，能够影响mod_expires模块的指令按照指令表（17098行）指定要求执行。函数set_expires by type（17066行）和set_expiresdefault（17083行）把相应的指令参数放入目录配置中。它们一直等到某单独请求被处理时才会被分析处理。

函数add_expires（17139行）在往即将发送的应答信息中加“终止”头信息的fixups阶段被调用。add_expires调用check_code函数（16959行）来分析目录配置结构的各域值，为实际的信息头内容作准备。模块结构如图6-3所示。

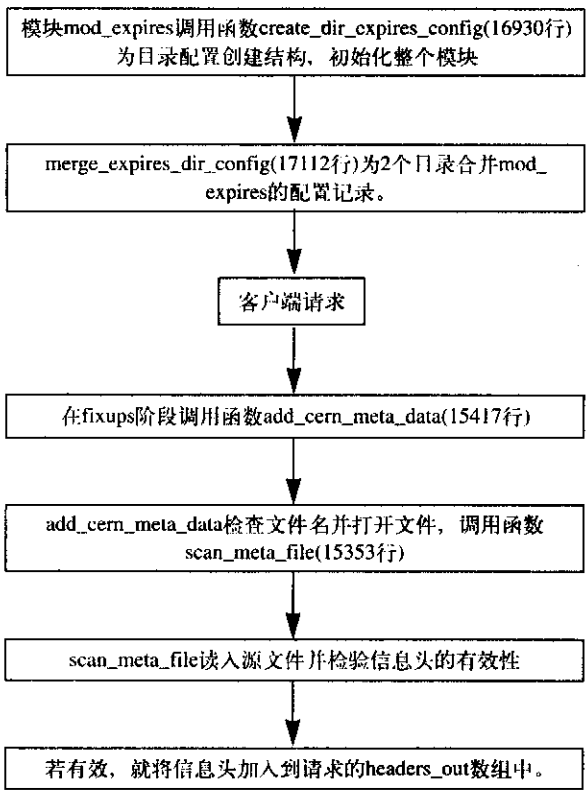

图6-3　mod_expires根据文件的MIMC类型为其定义一个超时日期

6.3.2 定制

在ExpiresByType和ExpiresDefault指令的实现中可以添加代码或选择。这些变化可能基于系统信息或其他信息（如数据库）的访问。

1. 编译器定义语句:

16914: 为mod_expires模块定义一个配置记录。包括一个on/off标志位，一个缺省的到期代码，以及一个包含各种到期数据的表（数组），该表由ExpiresByType指令根据MIME类型来确定。

2. create_dir_expires_config

16930: 为当前目录配置创建一个结构并初始化该结构。在16938行首先定义了4个数组元素来保存基于类型的到期数据。

3. set_expiresactive

16942: 在当前目录配置中设置激活标志使得mod_expires提供的信息头的状态得以确定，即根据指令提供的arg参数确定它们可用还是不可用。当存在Expires Active指令时，该函数被调用。

4. check_code

16959: 根据ExpiresByType或ExpiresDefault指令提供的信息产生一个到期代码，然后把它存入配置记录供当前目录上下文使用。

16970: 如果到期代码的首字母是A或M，则使用秒计数的简单格式。在这种情况下，变量real_code被赋予指令的参数值；并且无需进一步的处理。real_code是函数的返回值，所以返回值为NULL时表示成功。

16975: 如果到期代码的第一个单词同三个关键字（now、access、modification）之一匹配上，那么终止时段就作为一系列单词和数字来处理（例如“now plus 1 week”）。

base域（即首项元素）在16980处有说明。如果第一个单词是now或access，那么base被赋予A（16983行），如果是modification, base被赋予M（16986行），如果两者皆不是，那么到期问题不能被处理，所以将返回一个错误（16989）。

16996: 如果遇到plus关键字，那么到期代码的下一部分将在16998行被获取。单词plus只是为了增加可读性——此处不提供minus选项。

17003: 此处是处理成对<NUM><TYPE>的循环语句。其中的数字应当已经在word变量里，请把本段代码同mod_usertrack中相类似的代码段（从37067行开始）相比较。

17006: 整个数字应当由阿拉伯数字组成。如果是这样，该数字被转换成整数（17007行）；如果不是，将在17010行返回一个错误信息。

17017: 获取到期代码的下一部分（它们应当是weeks、months或days），在17018行检查word内容，如果为空，则在17022行返回一个错误信息。否则继续执行17026行的语句。

17027: 确定了指明的类型，根据类型，为秒数定义factor变量。如果没有找到类型信息，在17049行将返回错误信息。

17054: 把数字同类型的积赋予modifier（例如1天的总秒数*5）。每次while循环（17003行开始），modifier都会更新。到期代码的下一部分在17058处获得，循环继续往下进行。当到期代码为无时在17061行退出。

17061: 向调用程序运用base和modifier的值。base和modifier同按秒计数的A或M的简单格式是一致的。

5. set_expiresbytype

17066: 按照ExpiresByType指令的要求为某MIME类型的文件定义一个到期数值。当遇到该指令时，Apache就会调用该函数。

17072: 如果check_code函数能处理指令提供的到期代码，它就会被存放至当前目录配置中的expiresbytype结构中（real-code保存处理过的到期代码）。否则向ExpiresByType指令（17078行）返回一个语法表格式的错误。

6. set_expiresdefault

17083: 在当前目录上下文中，为那些MIME类型没有被设置ExpiresByType指令的文件定义一个过期值。当遇到这样的指令的，Apache调用该函数。

17088: 如果函数check_code能处理指令提供的过期指令，它就被存入当前目录配置的expiresdefault结构中（变量real_code包含处理过的过期代码）。否则为ExpiresDefaut指令返回一个语法表格式的错误（17093行）。函数set_expiresbytype和set_expiresdefault的区别在于函数头中的mime参数。Apache根据指令表格式发送该数据（TAKE1与TAKE2之间的不同见17104行和17107行）。

7. expires_cmds

17098: 定义由mod_expires所处理的指令，这些指令在函数中调用。指令表结构中3个元素中的每一个都定义了Apache需要的指令参数类型。它们按照事先定义的顺序传给某个具体函数。

8. merge_expires_dir_configs

17112: 为两个目录合并mod_expires的配置记录。base和add对应的expiresbytype数组也

被合并为一个新的配置记录（17133行）。

9. add_expires

17139: 根据本模块指令创建的数值为请求产生到期信息头。

17148: 如果当前请求有一个错误状态，那么到期信息头就不会加进去。同时，检查main以确定它是不是子请求（17152行），如果是，add_expires对请求不加处理即退出。

17155: 为当前目录上下文获得配置记录。如果不存在，则返回一个内部错误。很少有模块会在获得下一配置记录后有此项检查。（该错误表明create_dir_expires_cofig的工作失效）。

17165: 检查当前配置中的激活标志。如果ExpiresActive指令把该目录的Expires信息头设为活跃状态，程序继续执行，否则返回DECLINED。

17182: 确定即将返回的文件的MIME类型，以使ExpiresByType可以在需要的时候被找到。如果没有定义content_type域，code以NULL返回，这时ExpiresDefault将会被使用；否则，将会使用ap_table_get来定位请求的MIME类型。该请求位于当前配置的expiresbytype域里（17185行）。

17188: 如果code为NULL（原因可能是设有定义请求的MIME类型），那么expiresdefault将会在17191行被使用。

17193: 如果是因为没有定义expiresdefault导致code为空，返回OK。此时当前请求就没有相应的到期信息。

17200: 开始执行switch语句来确定被请求文件的访问或修改时间，把M或A作为code变量的首字母。变量base被赋予请求中finfo.st-mtime或request_time域的值。code变量的其余部分（首字母以后的那部分）被转换或整数（17209行或17217行）。

17219: 如果code的首字母既不是M也不是A，错误将被记入日志（17224行），不知何故，一个错误的到期代码会使函数完成运行过程。

17231: 有了base和additional，变量expires可以被赋予当前请求的实际到期数值。

17232: 使用ap_snprintf把年龄信息头的内容放入变量age中。该值是通过截取expires的request_time域来获得的（17233行）。

17234: 把变量age的值放入Cache_Control信息头中。

17240: 把expires变量的值放入Expires信息头中，调用ap_gm_timestr_822，把它转化为更可读的数据。

10. expires_module

17245: 通过create和merge这两个目录配置函数来定义mod_expires（17249和17250行）；一个用来处理Expires Active、ExpiresByType和ExpiresDefault指令的指令表（17255行）；以及add_expires函数（17263行），该函数在fixups阶段为正在处理的请求加入Expires和Cache_Control信息头。

6.4 mod_headers模块

mod_headers模块能够为请求定义、添加或删除任何信息头。该模块要比其他模块更灵活（例如mod_cern_meta）。它定义了一个通用的接口，其中的信息头名字和取值可覆盖任何目录

或服务器上下文。

Header指令可以在服务器或目录上下文中定义或修改信息头，其可能的取值为：

- add：用信息头名字定义一个附加的头，但并不覆盖其他同名信息头。
- append：为一个已知名字的信息头添加一个值，但不另起一行。
- set：为一个已知上下文中的请求定义信息头，并覆盖所有同名信息头。
- unset：从请求响应信息中删除已知信息头。

如果使用unset，仅需已知信息头名字：Header unset Cookie对于set、append或add，需要增加一项：

```
Header set Author "Joseph Young"
```

6.4.1 模块结构

Header指令按照从配置文件中分析出来的要求被执行，然后被传递给header_cmd函数。（17375行）在fixups处理阶段，fixup_headers函数被调用（17453行），然后依次调用do_headers_fixup，首先以服务器配置记录为参数，然后以目录配置记录为参数。模块结构如图6-4所示。

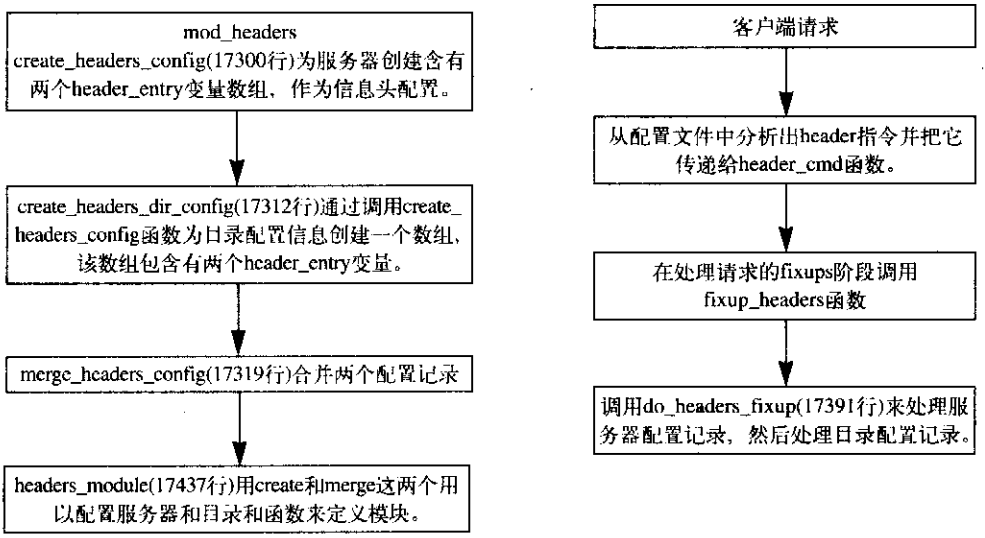

图6-4 mod_headers使得Apache能够在HTTP响应中增加、附加以及删除信息头

6.4.2 定制

mod_headers模块能够通过扩展来允许系统环境的变量被包含到信息头变量中。请注意在执行过程中什么样的环境变量对服务器进程是可得的。对于其他的系统信息，mod_headers能够基于Header指令参数值的特殊代码部分对其直接访问。mod_headers也能够根据事先定义的或动态确定的条件来产生信息头。这些条件都与Header指令的参数合并在一起。

1. hdr_actions

17273：为mod_headers模块在添加信息头时所可能使用的方法定义一个枚举类型。

2. header_entry

17284: 定义一个包含动作类型的简单信息头入口（基于17273行处的枚举类型），一个信息头以及要使用到的数值。在17294行利用一个元素为header_entry的数组为mod_headers模块定义一个配置记录（包括服务器和目录配置）。

3. create_headers_config

17300: 创建一个包含两个header_entry变量的数组作为服务器的信息头配置。

4. create_headers_dir_config

17312: 调用create_headers_config为目录配置创建一个包含两个header_entry变量的数组。在为服务器和目录同时创建配置信息时也调用该函数。它使得问题变得更清楚，因为两种配置被区分开了。

5. merge_headers_config

17319: 把over rides变量中定义的头信息和base中的头信息加在一起，实现两种配置记录的合并。

6. header_cmd

17335: 按照17386行指明的方法处理所有Header指令。

17343: 获取服务器配置记录。

17347: 检查cmd，确定该指令置于何处。如果是目录配置信息，那么信息头入口在目录配置中（17348行），否则在服务器配置中（17353行）。

17356: 检查Header指令定义的动作，根据指令为动作域赋一个数值。如果四个字符串中有一个检查不到，则返回错误。

17368: 如果指定的动作为unset，则检查指令提供的参数。在任何情况下，new的header和value域都被赋予传入的参数值（hdr和value）。

7. headers_cmds

17384: 定义本模块使用到的指令（Header）以及在配置文件（header_cmd）中出现该指令时应该调用的函数。

8. do_headers_fixup

17391: 处理配置记录提供的一系列信息头（在17431和17432处由fixup_headers调用），根据需要把它们加入当前请求的headers_out域。

17396: 对当前配置记录中的每一个信息头作一次循环。对每一个信息头，用switch语句检查请示动作并作相应处理。

17400: 如果动作为add，那么ap_table_addn就为该请求的headers_out加入header和value的域值。（这有可能导致两个信息头同名。例如可能有两个“Author”信息头。）

17404: 如果动作是append，调用ap_table_mergen就把value的值和已有的信息头的value的值合并起来。（但信息头并不另起一行。）

17408: 如果动作为set，调用ap_table_setn把header和value的值加到请求的headers_out中，同时替换所有同名的已有信息头。这就使得客户永远不会收到两个具有相同名字的信息头。

17412: 如果动作为unset，调用ap_table_unset从请求的headers_out中删除以header域命名的信息头。

9. fixup_headers

17420: 以配置记录为参数，反复调用do_headers_fixup把headers指令定义的信息头包括到当前服务器和目录上下文中。

17423: 把服务器配置写到serverconf中。

17427: 把目录配置写到dirconf中。

17431: 根据当前服务器上下文中Headers指令的要求，以服务器配置信息为参数，调用do_headers_fixups行以在当前请求的headers_out域中加入适当的信息头。

17432: 对当前目录上下文做同样处理。

10. headers_module

17437: 用配置函数create和merge为服务器和目录定义该模块。注意针对服务器和目录的create函数是不同的，但目录配置函数create_headers_dir_config仅简单调用服务器配置函数create_headers_config。相反的，同样的函数merge_headers_config被服务器和目录配置直接调用。17446行提供一个header_cmds表来定义怎样处理Header指令（该模块仅包含此指令），最后，函数fixup_headers在17453行定义，用来处理fixups阶段的请求。

第7章 目录索引模块

本章描述两个模块（mod_dir和mod_autoirdex），它们用于查找目录索引文件并在浏览器申请目录（而非确定的文件）时产生索引。这两个模块都包含在标准Apache配置中，由于这些模块相当简单，如果你只是想了解模块如何组织到一起，它们提供了很好的开端。

当浏览器请求一个目录而不是确定文件时，标准应答是从那个目录返回一个名为index.html、index.htm、welcom.htm或者default.htm的文件，当我们请求一个URL（例如www.yahoo.com）时，这种情况很常见，最后的目标是指向那个站点的文档树的根目录，不会显式的给出文件名。

如果找到了index.html文件（或者由mod_dir提供的命令之一Directory Index确定的另一文件），则该文件返回给浏览器。如果找不到index.html文件（或者由Directory Index命指定的文件），Apache将试图产生目录内容列表并将它作为html文档返回给浏览器，该列表的创建受控制的请求地址的目录包容器（directory container）中的Options Indexes命令的约束。

如果系统安全允许目录列表，那么该列表由mod_autoindex模块产生一系列命令控制该列表的外观，包括Index Options和各种AddIcon命令。mod_dir和mod_autoindex以前是一个模块，在Apache 1.3.4中它们被分成两个模块以便当没有Web站点目录允许产生索引列表的情况下，自动索引端口能从Apache中被移走。

7.1 mod_dir模块

DirectoryIndex命令定义了当Apache接收到未指定文件名的目录请求时Apache搜索的一个或多个文件名，该文件的缺省设置是:

```
DiretoryIndex index.html
```

该命令能加进任意数量的索引文件名，文件之间用空格分开，例如:

```
DiretoryIndex.index.html index.htm
```

模块从左到右按顺序扫描这些文件，看它们是否在目录中，当找到第一个文件时，模块创建一个浏览器对索引文件名的请求访问的内部重定向。例如：当浏览器请求www.abccorp.com/products/而模块在该目录下找到index.html文件时，请求就被重定向到www.abccorp.com/products/index.html，以进行进一步处理（返回该文件）。

如果在被请求目录下找不到DirectoryIndex命令列出的任一文件，mod_dir模块就返回DECLINED值并把处理递交给另一模块（通常是mod_auto index,它试图创建一个目录文件列表）。

7.1.1 模块结构

该操作的代码很简单。其中定义了一个数组（开始于16514行）用来保存要搜寻的目录索引文件（从DirectoryIndex命令取得）的列表。该数组由add_index函数（16520行开始）填写，当DirectoryIndex命令被处理时（见16533行dir_cmds），调用该函数。

如图7-1所示handle_dir函数（16564行）以for循环（16613行）遍历可能的目录索引文件列表中的每一个元素，完成所有的工作，每一轮循环创建一个子请求（16615行）来检查索引文件是否存在。如果文件被找到（由子请求中的HTTP_OK状态显示），那么主请求被重定向到该文件（16628行），同时模块以OK返回（16629行）。

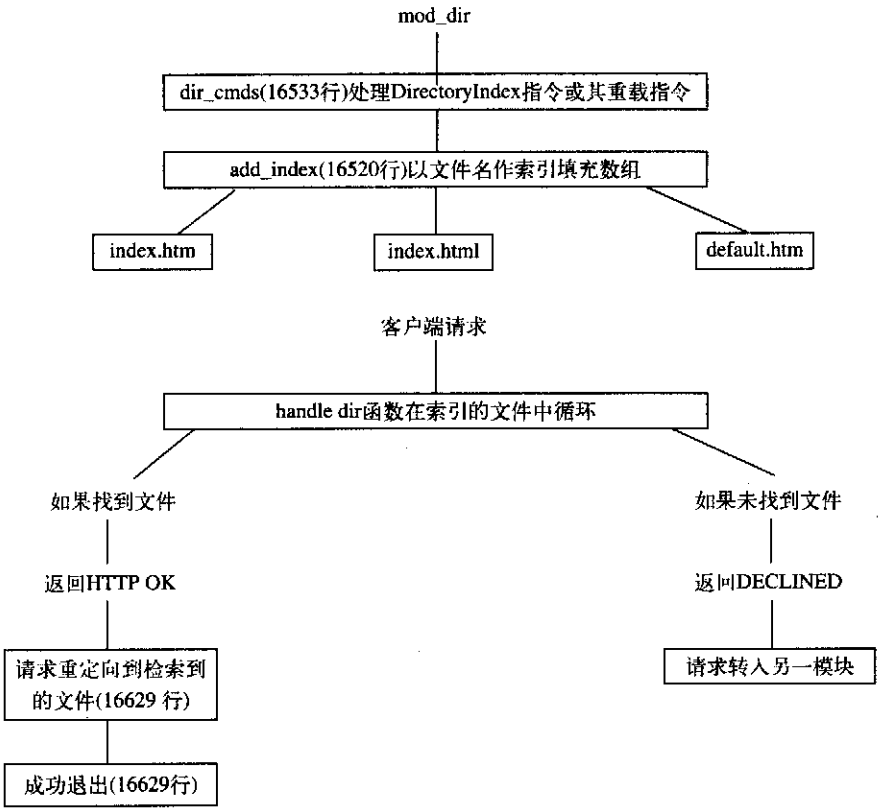

图7-1 当未使用文件名而请求访问一个目录时，利用mod_dir，Apache能够查找索引文件

如果没找到索引文件，则为被请求的索引文件的重定向和其他错误条件作检查。根据检查结果模块以状态码退出（16649、16670、16673行）。

如果for循环没有找到任何索引文件，也没有找到其他任何条件，循环就终止，模块返回DECLINED值（16677行），让下一模块mod_autoindex(如果它被包含在当前的Apache中）进行处理。

7.1.2 定制

关于mod_dir有几个问题值得改进。例如潜在的目录索引文件列表可以用其他方法计算非DirectoryIndex命令。能在add_index函数中加进代码以从数据库查询环境变量，或其他输入来源填充列表。如果没有任何索引文件，则可以在16677行之前定义插入其他行为。然而如果行为在该处发生，用专门处理该任务的独立模块处理该行为将更好。

1. 模块定义

16512: 或者模块定义或者所有函数必须能被提前引用以在编译时它们能相互找到对方，

显然模块定义更容易被提前引用，实际定义开始于16687行。

16514: 作为指向潜在索引文件名数组的指针定义dir_config_rec结构。

16518: 定义DIR_CMD_PARMS为值OR_INDEXES。DIR_CMD_PARMS在16536行被用于指明Directory Index模块的命令能够通过使用Indexes选项或AllowOverride命令而重载，因此等价于OR_INDEXES。

2. add_Index

16520: add_Index函数将来自DirectoryIndex命令的（做为指针arg传递给函数）的参数放进目录配置记录（作为指针dummy传递），以让handle_dir函数使用。

16523: 创建变量等于dummy的变量d，这里保存来自DirectoryIndex命令的所有索引文件参数的地方。

16525: 如果dir_config_rec结构(d)的index_names域指向NULL，ap_make_array调用为两个索引名元素分配空间。

3. dir_cmds

16533: dir_cmds函数为Apache定义了该模块使用哪个配置命令和如何使用他们。当Apache分析一组命令时访问该定义（和其他模块中的定义）。

这里域指示:

- DirectoryIndex——模块需要了解或处理的命令。
- add_index——用来处理命令给出的数据的函数的名称。
- DIR_CMd_PERMS——如果AllowOverride Indexes命令被包括在内，那么能在子目录（使用.htaccess文件）中被重载（该变量等于16518行的OR_INDEXS）。
- NULL——没有附加数据用于该命令。
- ITERATE——该命令带一个重复一次或多次的参数（要搜寻的文件名），命令中也可以包括进多个文件名，每一个文件名都作为同种类型的参数解释——要搜索的文件名。
- A list of file names——如果发生错误，本描述写出关于如何使用本命令的指令。

4. create_dir_config

16541: 按照16691行的模块定义初始化模块，为dir_config_rec记录分配内存，add_index使用dir_config_rec记录保持来自DirectoryIndex命令的文件名，调用标准的ap_palloc从(p)提供的内存池分配内存。

16547: 新dir_config_rec变量的index_name列表被初始化指向NULL，因为当模块初始化时，列表中不包含任何东西。

16548: 返回指向函数中创建的新dir_config-rec的指针，以便在模块中使用，主要用于add_index和handle_dir函数。

5. merge_dir_configs

16551: 从具有重要命令名的多个目录中合并配置。

16554: 创建一个新的dir_config_rec变量以保存合并的目录数据。ap_palloc调用为该变量分配空间。

16556: 分别给base和add赋值到来的base和additive目录配置记录。

16559: 合并base和add目录配置中的索引名字，放入变量new中，最后new作为一个合并后的dir_config_rec被返回。

6. handle_dir

16564: 该模块的主要函数是handle_dir。一个请求将以变量r的形式传给模块。根据配该请求相联的dir_config_rec配置信息（由16568的ap_get_module_config获得），函数handle_dir检查每一个可能的目录索引文件，把当前请求转换成返回第一个被发现的目录索引文件。

Apache服务器根据当前上下文来提供ap_get_module_config信息，该上下文在模块初启时通过调用create_dir_config函数来定义。

16570: 为循环处理一系列可能的目录索引文件名，定义一些指针变量和计数器。但此处只有error_notfound被初始化。

16575: 如果请求的路径部分（即请求r的uri域，内含标准资源标识符，通常是一个网页地址）长度为0（即首字母即为字符串尾\0）或没有以分隔符（/）结束，则handle_dir认为uri中的网页地址已被删除。在加入分隔符（/）后从模块返回，返回代码为HTTP_ MOVED_PERMANENTLY（16589行）。Handle_dir也检查各参数，即最初请求中问号（？）后面的部分。如果有参数，则它们被加到新地址中；否则往当前URI中加一个分隔符（/）。两种情况下ap_pstrcat都会为新的URI连接字符串。

16587: 在该请求中加入Location信息头以指明一个重定向的文档（位于新的路径）。handle_dir然后调用ap_table_setn加入一个名字/值对，以Location作为名字，以网页地址作为值（由ap_construct_url使用ifile字符串产生的，ifile是由ap_pstrcat和当前请求r创建起来的）。最后返回HTTP_MOVED_PERMANENTLY。

16598: 通常情况下（即URI没有被删除），在请求的文件名域的末尾加上分隔符（/），否则调用ap_pstrcat字符串concatenation函数。这样就允许潜在的索文件名被加到URI并在16613的for循环中被检查；否则，如果一请求的URL的最后一部分是目录名，它会被当作文件名。

16603: 初始化循环处理一系列潜在的目录索引文件名所用到的变量。如果目录配置中的index_names域非空，则namer_ptr被赋给index_names的第一个元素，同时num_names被赋给索引名的数目（d -> index-names -> nelts），它将在16613行处初始化循环计数器。

16608: 如果index.names中没有数据，则把names_ptr赋给dummy_ptr, dummy_ptr是一个单元素数组，包含缺省索引文件名：index.html。同时num_names被置为1，因为仅有一个索引名字被检查。

16613: 循环处理names.ptr中的每个元素，直到num_names为0。每次循环中，URI的有效性都会受到检查，该值是通过在基本URI中追加index_names的下一个元素创建的。如果URI有效，则请求被重定向到该URI。

16615: 基于name_ptr指向的字符串和当前主请求(r)来创建一个子请求，以此来检测当前潜在目录索引文件名的有效性。

16617: 如果子请求的状态是HTTP_OK，并且子请求中针对文件名的finfo.st_mode域没有指明一个不存在的文件（非0），那么该索引文件就是将要使用的。

16618: 如果子请求成功，将产生一个新的URI，其中包含names_ptr中被检测为有效的文

件名。如果有必要，任何参数都加到其中（这些参数来自初始请求，16623行或子请求16620行）。

16627: 子请求已完成任务，被删除。然后主请求(r)被重定向到新URI。该URI在16621行或16624行处产生。最后返回OK。

16635: 如果当前子请求的状态表明该URI已被重定向，或者HTTP请求不能被接受，并且这是最后一次循环（num_names为1，无其他文件名可试），那么子请求信息（notes、 headers_out以及err_headers_out域）被ap_overlay_tables函数合并加入主请求中。然后返回error_notfound消息，表明文件名出了问题（被重定向或导致一个错误HTTP请求）。

16652: 程序执行到这里表明文件名的状态既不是重定向，也不是不能被接受的HTTP请求，同时它也不是试图使用的最后一个索引文件名（num_names!=1）。检查子请求的状态，如果非0但不是NOT_FOUND或OK，那么状态值将存到error_notfound里。如果在后继的循环里没有发现一个有效索引文件名。那么在16660行返回error_notfound。源程序中的注释的说法是返回一个可能不安全的目录索引。这是Apache 1.3.4以前版本的解释，说的是当mod_dir和mod_autoindex模块被合并为一个模块时（该模块返回目录链表），没有发现任何索引文件名（如index.html）。在现有版本中，该解释不是非常中肯。

16664: 在for循环结尾处，子请求被用来检查当前潜在索引文件名是否已被删除。

16667: 在for循环对所有可能索引文件名处理后，开始检查error_notfound。如果error_notfound非0，在16662行它被置为子请求的状态值，并返回该值。如果error_not found被置为0，表明未出现异常情况，因此handle_dir转至16670行，返回DECLINED。

16670: 如果请求的方法不是GET，则返回DECLINED，另有模块会试图对该请求作出响应。

16675: 如果请求的方法不是GET，并且没有找到索引文件，则返回DECLINED，另有模块处理该请求。注意在16671行和16675行两处的返回是完全相同的。在16670处的检查为回答为什么请求被DECLINED提供了附加的信息。该信息当然只能在跟踪调试时才能发现。

7. dir_handlers

16679: dir_handlers把符合DIR_MAGIC_TYPE的目录和函数handle_dir关联起来；这样，handle_dir将为所有符合条件的目录提供索引。

8. dir_module

16685: 定义模块信息，使得Apache核心代码能够在初始化模块时找到所需要的函数。目录配置函数create和merge为Web站点的每个目录设定配置信息。Apache为每个应用目录调用这些函数，并且保存函数设定的配置信息使得模块以后能访问（如16568行）。

函数dir_cmds（始于16533行）定义了该模块要使用哪些命令。无论何时Apache遇到命令，都会调用dir_cmds来分析该命令。最后，函数dir_handlers（始于16679行）把mod_dir的处理程序同它的主函数（handle_dir）关联起来。每次Apache使用mod_dir来处理请求时，handle_dir都会被调用。

7.2 mod_autoindex模块

mod_autoindex模块被用来产生一系列目录的内容，存入HTML文件中，返回给发出请求的浏览器。该模块主要包含各种字符串操作以产生一个漂亮的HTML页面，其主要内容来源于目录中的文件。由于该模块使用了大量命令来控制目录列表的外观，因此需要很多函数来产生所有可能的HTML字符串。例如，一个旧版本中的命令FancyIndexing已被Index Options更新，但FancyIndexing仍被支持。

mod_autoindex的使用是被Options Indexes命令控制的，见16533行。其他在autoindex_cmd中列出的命令（始于14083行）控制目录列表的外观。请记住mod_autoindex和mod_dir过去是一个模块；它可以解释诸如Index Options命令（14109行）的外在表现（那时该命令仅在mod_dir模块中被使用）。

7.2.1 模块结构

handle_autoindex函数被用来创建HTML目录列表。它首先访问ap_allow_options函数以确定当前请求的目录是否允许显示目录列表。如果允许，就取出该目录的配置记录。然后调用index_directory函数以产生索引列表并把它作为模块响应返回。

函数index_directory（始于15187行）调用另外的函数来产生目录和文件入口，加入图标和描述信息、排序等，最后形成HTML的索引页面。HTML文本被函数ap_rvputs和ap_rputs返回给客户端。

这两个函数以及其他几个函数，例如emit_preamble(15187行)，位于index_directory函数中。模块结构见图7-2。

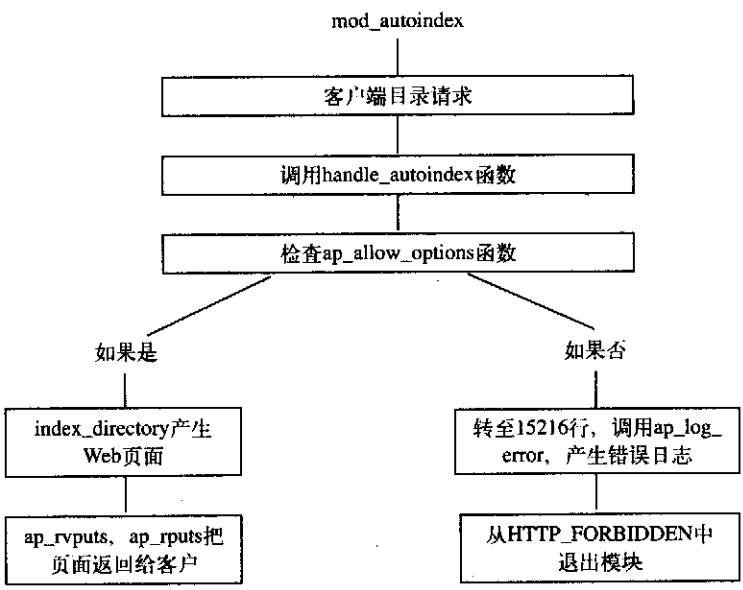

图7-2 mod_autoindex产生一个目录列表页面并将其返回给浏览器

7.2.2 定制

大多数Web站点不允许目录索引，所以不使用mod_autoindex模块。在这种情况下，如果

想为Web服务器节省内存，可以去掉该模块重新编译Apache。

但有些站点可能依赖服务器产生的目录列表来索引其他目录，尤其是那些目录内容经常变动的站点，该模块就显得更重要。利用自动产生的索引列表，网管可以不重做网页就能引导用户浏览目录内容。记住检查诸如IndexIgnore和Options Indexes命令以确定哪些文件和目录允许浏览器访问。任何有HTML知识的人都能够通过修改mod_autoindex模块来产生更漂亮的目录列表。可以从以下几个角度来改善目录列表：

- 在目录列表的顶部或底部添加自己的图片。
- 创建背景图案，以提供一个logo或在屏幕左下方产生一个色带。
- 检查产生的路径列表，在被请求的Web站点区域使用图案，背景颜色以及其他风格样式。
- 查看请求中的cookies，在目录列表中加入个性化信息（可以是欢迎信息或定制的格式，都是基于cookie或其他的客户端信息）。

许多这样的选项都能在一个文件内执行，该文件被当作文档信息头来使用（由HeaderName命令定义）。但有些命令不能在一个静态的HTML文件中完成处理过程。对于这种情况，函数emit_preamble(13730行)被用来检查请求的某些域（如r->URI或r->filename）。然后调用ap_pvputs输出该HTML文件。注意，如果头文件满足HeaderName命令所定义的要求，emit_preamble就不全被调用。

在autoindex_cmds函数中（14083行）可以申明新的命令，为特殊的目录或文件类型定义可能的值。例如，要增加一个含有颜色值的命令Index Background，首先在autoindex_config_rec(15187行)中增加一个域，然后在emit_preamble函数中检查该值以便在HTML文档的<HEAD>之间定义合适的HTML标记。

1. 模块定义

13615: 此处提供模块的预定义，这样，模块中的其他函数都能找到它。模块定义开始于15232行。

13622: 定义该模块用到的各种值，这些值在创建目录列表时将作为各种标记来指明相应的设置，当然，这是以当前活跃的命令为前提的。这些标记的意义包括：水平规则(line）的使用，排序方法以及目录列表中应当包括的域。这些定义开始于15232行。

目录列表中各元素图标的大小以及元素的缺省长度能够通过变动#define定义的各种值轻松改变。

2. struct item

15234: item结构包含目录列表中的元素。每个元素都有类型，例如readme文件或常规文件以及与其相关的附加数据。当由这些元素组成的数组被赋值后，该数组就被输出以形成实际的目录列表。

3. autoindex_config_rec

15241: 模块mod_autoindex的配置记录在此处定义。该结构包含目录（该参数从命令那里取得）中用到的各选项和各种变量，如图标宽度和缺省排序方法。此处也说明了几个数组用来存放命令指定的图标和描述信息。在调用autoindex_cmds(15232行)中定义的函数来处理命令时，这些信息将被初始化。

4. is_parent

15271: 函数is_parent检查传入的字符串是否为“. .”，以此来判断是否为上层目录入

口。该函数由output_directories函数调用（14873行）。

5. emit_preamble

15294: 向新创建的HTML索引文档发送一个前导码。ap_rvputs将把信息返回给客户。因为该函数发送的是字符串序列，各元素可以不进行任何预处理直接作为参数加到序列中。由于该函数检查诸如请求路径、客户/浏览器，以及根据处理结束决定向客户返回的信息等重要属性，所以它是改进目录列表的好位置。

6. push_item

15303: 函数push_item把一些数据写入指定的数组中（即参数表中的arr）。它将由专门处理始于13770行程序的命令使用。函数的主要工作在15307行开始，有一个新元素被加到数组中。

15316: 作为数组元素传入的数据将被ap_pstrdup写到新元素中。

7. add_*

15334: 所有这些add_函数（如add_alt、add_icon和add_desc）都用来处理autoindex_cmds函数中申明的命令。Apache调用这些add_函数把从命令中取得的数据写到列表中。列表将在处理请求时返回给模块。因为不同类型的信息要同文件或目录的各种属性相匹配。By_PATH、BY_TYPE和BY_ENCODING的代码显示在这些函数中以帮助决定在信息列表中应贮存什么。每个列表由匹配项(例如文件名或文件类型)和匹配找到时使用的数据(例如描述或图标URL)组成。

15347: 对每个以add_开头的函数，push_item函数的作用是把信息写入到相应的列表中，在13824行的add_desc函数给出了对push_item函数的最简单的示例。带有命令的文本被push_item放到列表中，Apache决定如何调用函数来处理在autoindex_cmds域中的命令信息。这此命令可以是ITERATE或TAKE2等等。

8. fancy_indexing

13866: FancyIndexing是一个受到非议的命令，但系统仍然完全支持它，除非它与IndexOptions命令的选项相冲突，否则FancyIndexing一般会被加到opts域中。

FancyIndexing命令提供图标，带有列表头的多重信息域以及额外的链接。

9. add_opts

13886: add_opts函数处理在Index Options命令中的信息。该命令有许多的可选值，它们一起组成了配置信息的opts域。

13898: 这个命令需要注意的一个要点是可选项是可以附加的；也就是说一个子目录可以向那些已经生效的可选项增加新的可选项，而不是重新定义一组可选项。附加是通过在可选项前面放加法符号（+）来实现的，在打算使用可选项时必须注意不要对一个目录重复定义可选项。

13910: 它是一组检验，用于检查每一个可能的选项字符串，在找到合法的字符串时，option变量的对应选项的标志位就会置位，如果有很多选项，option的值就是与在14010行上的opts域作“与”运算。使用“+”号或“-”号的附加选项被分别放到opts_add或opts_remove域中。

13963: 有一些由Index Options中定义的可选项并不是作为opts域中的某些位来存放的，它们被放在配置信息的其他域中，例如图标的尺寸和名称宽度在检验的有效性和

冲突后，被放在分离的域中。注意：当发生冲突后，在冲突或有无效值时，系统使用缺省的选项。

10. set_default_order

14034: set_default_order函数设置变量来控制以IndexOrderDefault命令的值为基准的排序。排序的方向和关键字参数由Apache来提供，而Apache在解析IndexOrderDefault命令时直接调用这个函数（14112行）。

14042: 通过调用ap_cpystrn为temp变量赋缺省值k=d（意为降序排序），然后以本指令的参数与两个可能的值进行比较，这两个值是提供给IndexOrderDefault、Ascending和Descending指令的，最后根据比较结果将temp的第三个属性置为相应关键字（在13654行和13655行中定义）。

14054: 对于key参数，操作是相同的，即把temp的第一个属性置为关键字，该关键字属于可能的关键字串，例如姓名、日期或大小。如果对于排序方式和排序关键字都无相应的可用值，函数返回错误信息，指明应如何正确的使用该命令。

14073: 如果使用类似N=D这样的字符串（在14043行中通过指令所提供的值把缺省值k=d覆盖了），这些行将为配置记录中的default_order域提供一个4属性序列，然后通过调用ap_cpystrn函数把temp变量拷贝到default_order域中。

11. autoindex_cmds

14083: autoindex_cmds函数把14条用于控制mod_autoindex的指令，及在Apache解析执行这些指令时要调用的函数关联起来。对于每一条指令，需提供如下的信息项：

- 出现在Apache配置文件中的命令名。
- 本模块解析该指令的数据时所需调用的函数。
- 该指令递交给解析函数的额外数据（对于这些命令，像BY_PATH这样的数据片在13695行开始定义）。
- 该指令如何在每个指令特有的配置文件中（.htaccess）进行重载，这些指令中的DIR_CMD_PERMS在14081行被设为OR_INDEXES，指明若要下一级命令能够重载该命令，则必须设置AllowOverrideIndexes值。
- 所需数据的格式。例如，ITERATE指明一块数据或其多个拷贝应包含在该命令中，这里所包含的值决定了参数是如何由Apache传到列出的控制函数中的。
- 若提供给该命令的数据格式不正确，则还需返回相应的描述信息，这些信息也可作为使用这些命令的简短说明。

12. create_autoindex_config

14138: create_autoindex_config函数为本模块分配一个新的配置记录，并将其中各项赋予缺省值。Apache一旦要对指令进行语法分析，就从autoindex_cmds中调用其他函数，然后对其进行赋值。这样，在handle_autoindex函数发出请求时，本模块就得到一个合适的配置记录返回值。

13. merge_autoindex_config

14169: 当覆盖某些选项时，为了处理从一个目录到底层子目录的继承问题，本函数联结两个配置记录（它们分别由函数标题中的basev和addv给出），创建一个称为new的新配置记录，并检查两个源配置记录的每一个域，要么用addv里的值更新它们，

要么调用ap_append_arrays联结它们。

14200: 在设置目录的选项时要特别小心，因为它们有可能是可附加的（也就是说，可能某个特性加入到已经生效的选项中去了，也可能替换了已经生效的选项集）。

14218: 如果新配置记录中没有选项（add=>opts==0），那么新增加或者减少的选项将直接从base和add配置中获取。将新联结配置记录的options域设置成基本选项（14231行）。

14240: 如果同时提供普通配置与完全选项（无增量），这些选项将代替以前由基本配置所提供的任何选项而成为这里的新选项。

14247: 这两行将去掉在已生效的选项和附加/减少选项之间的重叠部分。例如，去掉incremented_opts中所有已经在opts域中存在的字节。

14258: 如果name_width域的值仍然是K_UNSET(在13642行设置)，将其设置成基本值；反之，将其设置成附加值。

14. struct_ent

14278: 这个结构将保留目录表中所含文件或目录的全部信息，例如名字、图标URL和图标/文件类型的候选说明。为每个入口收集这种信息，供output_directories函数使用，以创建实际的HTML代码。

15. find_Item

14290: find-item函数全面搜查由指令定义的项表(List of items)（例如图标和说明），以此来寻找能够应用于请求所传来文件名的项。这些请求实际上是在make_autoindex模块入口创建的子请求。在对指令进行的语法分析时，将像add_desc和add_icon这样的函数填写在项表中。使文件匹配正确信息的方法取决于指令所使用的额外数据，例如BY_PATH或BY_TYPE。正是依赖于这种设置，才能用关于文件的不同信息检查它们与表中的某一项的匹配情况。

14293: 像保存文件名一样保存这次请求的内容和编码信息。建立一个for循环（14303行）来搜索表中的每一项。表是图标URL类型还是说明类型，取决于被传来用于搜索的表的类型。for循环在表中用关键字进行匹配，然后返回表中被匹配的那一部分值。

14307: 由于目录中只能含有一些关于路径的信息（不能有类型和编码信息），在这里处理目录的这种特殊情况。如果apply_path（表中这一项的应用所在路径）与传进来用于测试的路径匹配，那么就返回从该项所得到的数据。

14312: 如果比较类型为BY_TYPE，同类型的比较完成。如果能找到的话，返回该表项的数据。

14319: 某些调用可能设置了path_only标记，这就需要基于类型或编码的检查（如果BY_PATH为false，立即进行检查）。在两种类型（类型或编码）的任何一种情况下，都要将content_type或content_encoding与表中当前项比较。如果匹配，返回相关数据项。

14340: 如果表中没有apply_to项与所请求的文件类型匹配，返回NULL值。

14343: 用find_item的特例在特定表中进行搜索。# define语句使得搜索看上去很特别，它在决定搜索标准时使用了与本语句匹配的参数名（d,p），这就形成了一种可靠性。然而，这种可靠性在对模块进行修改时将出现问题。

16. find_default_icon

14349: 这个函数实际上使用了find_item函数来查找信息。与它不同的是，find_icon操作需要传入一个不同的名字参数，所以它使用一个独立的函数来创建一个以bogus_name为名字的请求，重置content_encoding，并在图标表中调用find_item查看。

17. ignore_entry

14366: ignore_entry函数决定作为参数传入的路径是否包括在配置记录的ign_list中。如果是，返回1；否则，返回0（不忽略这个入口）。make_autoindex_entry调用这个函数来决定一个文件名是否是目录列表中的一部分。

14370: 为了便于对表中ignore项进行访问，创建一些变量（该项由13830行的add_ignore函数产生），并验证被检查的项中有没有斜线。如果出现斜线，必须在比较操作中忽略斜线。用strrchr函数来决定斜线的位置，并且只比较斜线前面的字符。

14381: 使用一个for循环来检查忽略表中的每一项。对每一项来说（在找到了斜线的位置后，把它当作比较中划分字符串的分界点），使用ap_strcmp_match还是使用ap_strcasecmp_ match进行字符串匹配比较取决于Apache所运行的操作系统。CASE_BLIND_FILESYSTEM编译器# define将在服务器被编译之前对配置进行设置。如果比较成功，在14395行和14407行用值1退出。如果比较不成功，在完成检查每一项的for循环之后，在14411行用值0（不忽略本入口）退出。

18. insert_readme

14423: insert_readme函数往客户端输出一个首部或脚注文件（通常是README或者HEADER）。首先验证该文件是否存在并且可访问。在14435行得到当前目录的配置，并着手准备在新的子请求中所使用的路径，这种子请求将测试Apache读文件的能力。

14442: 从传入到函数的readme_fname参数开始，收集和测试文件名。如果设置了hrule标记，将水平线标记写到客户端（14453行）。

14463: 创建一个子请求来查找和fn一样的文件名，如果该查询请求的状态不是HTTP_OK（14464行），删除子请求，函数不输出文件而退出。

14467: 如果文件可读，由于子请求的目的已经完成，删除子请求。用fn的文件名字符串调用ap_pfopen打开文件（3014行）。

14471: 如果这是一个首部文件（出现在文档前端返回给客户端的文件）并且SUPPRESS_PREAMBLE选项不活跃，则先用emit_preamble函数传送序文。

14475: 如果发送的不是无格式的文本文件，为了使文件中出现的所有方括号和&号正确显示在浏览器中，需要做一些额外处理。这里先发送一个<PRE>标记到客户端，然后循环读取文件中的每一个字节(见14482行)，将整个文件写到变量buf中。

14478: 如果文件是明文，要先进行处理以使文件中的尖括号和&号在浏览器中显示正确。这可以通过发送一个<PRE>标记到客户端达到，在14482行循环文件的每一字节，把整个文件读入buf变量。

14493: 循环遍历整个文件缓冲区buf，使用ap_rputs调用每次输出一个字符到客户端，同时检查是否有<，>或&号。这样就输出了这些字符的HTML代码。

14516: 文件被送出之后，关闭文件描述符。如果是无格式的文本文件，<PRE>标记出现在14518行（<PRE>标记不使用于非文本文件）。

19. find_title

14524: find_title函数在一个HTML文档中定位<TITLE>标记，以便能用它来描述索引表中的文件名。

14526: 这次用于定位的字符串是<TITLE>——HTML中的标题标记。

14533: 被检查的文件必须是HTML类型的文件或者是不出现<TITLE>标记的文件。如果通过了这种检测，则打开该文件（14538行）并将其读入titlebuf缓冲中（14542行）。

14549: 这里用for循环对所查找字符串首次匹配的字符（所查找字符串为<TITLE>）进行定位。一旦找到，从头至尾扫描标题（位于两个<TITLE>标记之间的文本）。为便于在做字符串拷贝时返回（14570行），用变量x记录标题尾的位置。

14559: 解析行失效并将该行转换成空格。行失效一般不出现在<TITLE>标记中，但是在<TITLE>文本中，索引表的描述域可能出现行失效。

20. make_autoindex_entry

14583: make_autoindex_entry函数使用了从文件系统（见14584行的fread函数所做的询问）返回的入口，将其用ent结构格式化，这种结构含有output_directories函数所需信息。

14591: 如果入口名是单周期类型，它表示当前目录。由fread函数返回这个入口，但不把它包含在目录表中。返回NULL值。

14595: 如果入口名是ignore_entry表的一部分，返回NULL值。

14600: 对有效入口，用ap_palloc为其分配空间，然后初始化每一个域的值。name是关键域，用ap_pstrdup将其拷贝到入口(p)处。

14612: 如果FancyIndexing可用，需要更多的步骤来创建表所需附加的域和图标。这需要提出一个子请求，用来检验文件的有效性以及确认文件类型。

14616: 如果子请求状态非零，继续处理。14618行说明这是一种目录。如果没有其他图标返回，这个图标被记录成一种目录，该图标（act域）的可选文本被记录为DIR(14625行)。目录名用斜线结束，可以用ap_pstrcat函数来添加目录名。

14632: 对非目录（也就是文件），图标和可选文本由相关函数决定（这些函数只不过是对find_item函数的各种调用）。文件大小是子请求的一部分，可自动生成，存放在size域中。

14638: 描述从find_desc函数的结果开始，这个函数也是find_Item的化身。如果描述返回的是空串，而且使用HTML标题的标识活跃。这里则试着使用作为描述域文档的HTML标题来调用find_title。

14645: 使用子请求查找完信息后，删除它。

14652: 如果是索引关键字，检查最后所修改日期的有效性。该日期小于零时，将它设置为零(14654行)。

21. terminate_description

14660: 如果SUPPRESS_LAST_MOD或SUPPRESS_SIZE选项被目录表中某个目录激活，terminate_description函数删除关于文件的描述信息。如果这个目录中没有域，则允许有长描述域。

14667: 通过检查为SUPPRESS_LAST_MOD和SUPPRESS_SIZE选项重置最大长度（字符串长度）。

14674: 如果字符串中没有HTML标记，删除这些域。这里的for循环是用来确保没有标记计数（或者出现半个标记，使得浏览器看上去丑陋无比）。另外，在发现一个特殊的HTML字符时，使用14685行的循环将几个字符转换为一个字符（例如&amp转换为字符&）。

14700: 变量x保留了描述记录最末有效字符的字符串引用。这个字符串由一个右方括号和一个字符串终止符（\0）结束。

22. emit_link

14714: 当域是索引关键字时，emit_link函数为锚标记（<A>）将HTML文本发送到客户端。如果域不是索引关键字，仅仅发送不含链接的标记（12405行）。然而，如果域不是索引关键字，受nosort参数的制约，附加处理必须通过点击链接提供反转索引的能力。

14721: 如果当前锚域是索引关键字，则要在指向相同文档的链接处组织锚标识，但同时要翻转这个域的索引顺序。当前索引顺序（curdirection）决定了链接所含的索引方向（在14728行设置）。用ap_rvputs调用输出完全的锚标识（14729行）。例如，位于大小域上的链接所含HPEF值可能是?S=A，这说明允许用户翻转当前的索引顺序。

23. output_directorios

14775: 在完成大部分工作之后，output_directories函数将大部分目录表数据发送给客户端。

14796: 根据K_ADJUST的值来建立表中名字的宽度。如果这样设置了，就把name_width变量扩展到一个文件的最大长度，这便于将表中每一个文件或目录都用最长名字水平排列起来。为了在表中的名字域后面提供空白键，在14805行添加一个附加空间。

14806: 分配一个大小与刚决定的名字长度相同的随机内存空间。

14811: 如果打开了FancyIndexing，输出以图标为标题的<PRE>标记和<IMG>标记。正确的图标由find_default_icon函数决定（在14813上使用该函数）。如果可能的话，还要包括图标的高度和宽度。在14287行关闭<IMG>标记。

14829: 使用emit_link函数为Name域输出标题。14833行是填写在列标题里的空白集合。在14842行加入一个空格是为了确保域间有空格。

14843: 是否为Last Modified Date、Size和Doscription域跟踪同一进程，取决于Index Options是否允许有这些域。为了使这些域与它们从filesystem调用返回的域的宽度匹配，将14842、14849和14852行的空白填写在列标题中。标题通过发送水平线标记结束。

14862: 如果没有选中FancyIndexing，不输出首部。一个有关目录内容的中心表从这里开始。

14865: 用循环扫描目录中的每一个元素（直到总数达到传入的参数n）。

14873: 如果这里处理的入口是一个父目录，那么更新为了到达这个入口所使用的文件名和链接。将14880行的项（tz）名修改成ParentDirectory。如果选用了此项（t），那么它所链接的URL由ap_getparents决定（14875行）。

14893: 先测试FancyIndexing的使用，将正处理的入口发送往客户端。如果设置了FancyIndexing，再检测ICONS_ARE_LINKS选项。如果把图标当作链接使用，发送锚标记到客户端。FancyIndexing块接下来发送<IMG>标记和所有由这个数组入口域提供的信息（14899行），例如图标URL和图标的可选文本。在14911行关闭

<IMG>标志。

14917: 由于文件或目录本身在很多情况下就是一个链接，所以在14917行开始有一个<A>标记，后面伴随一个标准宽度的链（文件或目录名）和一个锚标记。这样就允许用户点击链接进入表中所含的命名文件。

14929: 由于名字的宽度不同，所以采用空白空间来将入口名调整成等长，widthify函数使用这些空白空间来创建名字。

14936: 下面的域宽度相同，在检查了是否应该使用Last Modified Date之后，用strftime标准函数将入口时间准备成字符串。先发送这个字符串（14942行），再发送填充空间（14947行）。

14951: ap_send_size调用把表示大小的数字和一些填补空间一起组织成字符串发往客户端。

14954: 使用teminated_description函数发送描述，如果不包括Last Modified和Size域，这个函数允许一个更长的描述字符串。

14963: 如果FancyIndexing无效，将表项<LI>和t2（文件名）一起作为链接发送。受是否使用FancyIndexing的影响，要结束一个准确形式的HTML页面，必须先关闭<PRE>标记和<UL>标记。

24. dsortf

14983: dsortf是一个比较函数，它由qsort函数调用，用于对15157行的索引入口数组进行排序。如果第一个元素像首项一样排序，返回1；如果第二个元素像首项一样排序，返回-1。

14994: 在排序表中父目录应该一直是首项。用is_parent函数对这些首项进行检测（13703行）。

15003: 根据使用的排序顺序将传入的参数分别分派给c1和c2。

15011: 根据索引关键字，将被索引元素的一个不同域在c1和c2之间进行比较。根据上述比较中的较大值，函数退出时返回排序中较高项的值。

25. index_directory

15043: index_directory函数创建实际的HTML目录索引，并将其返回给客户端。要在handle_autoindex函数中检查了许可权之后才能调用它。

15046: 本函数为目录处理用到的一些变量申请空间。这些变量中有的已初始化为目录信息，例如目录名和目录选项。

15062: 试图打开在本请求中命名的目录（name在15049行定义的请求）。如果打开目录失败，使用ap_log_rerror函数将错误消息写入错误日志文件，返回HTTP_FORBIDDEN供Apache送往客户端。

15071: 如果能打开目录，将HTML索引发往客户端。这里首先将这次回答的首部发送出去。

15073: 如果只请求首部（HEAD请求与GET请求相对应），关闭在15062行打开的目录，并成功返回0。

15077: 在开始处理目录表之前，设置timeout的值。这样，如果目录表返回的不是服务器上设置的timeout值（请求结束），就要取消请求并将字符串send_directory记录到出错日志文件中去。

15081: 为创建索引准备目录名。为了能将它用于后面的字串符函数，使用while循环来

正确终止该字符串（15083行）。

15088: 作为返回到客户端的HTML的第一部分，这一行搜索HeaderName指令定义的首部。find_header函数实际上并不存在，它只是14347行的find_item使用配置记录hdr_list域作为参数的一次调用实例。定位了首部后，用insert_readme函数将其输出给客户端。如果上面两个操作都失败了，这说明首部未定义或不可得。在这种情况下，跳转到15092行调用emit_preamble函数来发送HTML的通用标题。

15093: 发送完首部或序文之后，开始发送表名——使用目录名的第一层标题(title_name)。这里调用ap_rvputs将表名发往客户端。

15107: 将随请求发送的参数设置成qstring。用该参数来决定目录表的排序顺序。如果取消了列排序选项（通过检查autoindex_opts中是否含有SUPPRESS_COLSORT）或没有参数，将qstring设置成缺省的排序顺序值（15110行）。这个域将在IndexOrderDefautt的指令中配置。

15117: 检查qstring值是否有效。如果为Null，将关键字域设置成文件名（K_NAME）和目录（D_ASCENDING）。否则，将排序关键字设置成与qstring相等（15122行），并读“=”号之后的目录。如果没有给定目录，使用缺省的D_ASCENDING。

15138: 这里的while循环从15062行所打开的目录读入口信息。make_autoindex_entry函数将入口解析成一种标准格式。每个入口包含有本模块定义的全部指令，每个入口都属于ent类型（在14278行定义）。如果make_autoindex_entry返回的信息是非Null，while循环继续指向表中的下一项，并反复读取目录（15137行）。

15147: 如果目录入口表非空（num_ent比零大），为3784行的数组分配内存，而且在15152行中的while循环将链接表中的每一个元素读入到数组中去。这样做是为了允许对入口进行快排序（15157行）。

15162: 根据排序过的目录内容，使用目录信息数组（ar）调用output_directories函数，以此来察看其他目录，例如那些被删除的列。

15165: 这里并没有从目录中读取更多的数据，所以用ap_pclosedir函数关闭它。

15167: 输出所有表。find_readme检查注脚信息（任何README文件）。如果find_readme（实际上是find_item (p, d->rdme_list,0)）找到了这样的文件，使用insert_readme函数把它们输出到客户端，接着查找旧FancyIndexing指令的设置，用它们来发射水平线标记（<HR>）。如果使用了ServerSignature指令，在必要时还要发送服务器签名。

15176: 不管是否发送了README脚注，HTML的输出在这里用关闭<BODY>和<HTML>章节的标记结束。

15178: 完成超时设置的任务之后，取消超时并返回值0，这表示本模块成功地处理了这次请求。

26. handle_autoindex

15184: 本模块的主要处理函数是handle_autoindex。它从创建配置记录(d)入手，在该配置记录里通过调用ap_get_module_config填写本目录的应用配置信息，获得本请求所允许的选项（15187行），并将保存在allow_opts中；保存在allow_opts中的选项位图。本模式必须包含OPT_INDEXES，否则索引无法进行。

15193: 如果请求没有GET方法，不返回目录索引，返回DECLINED，退出模块（15195行）。

15201: 如果本目录的allow_opts值包含OPT_INDEXES（通过带逻辑“与”操作的值测试），继续处理；否则，转到15216行，这一行使用ap_log_rerror函数往日志文件发送日志消息，以HTTP_FORBIDDEN从模块中退出（HTTP_FORBIDDEN返回给客户端）。注意，出错日志消息包含被请求的目录（15220行的r-> filename）。

15209: 如果15201行的测试允许索引，必要时要在请求文件名后加入反斜杠（/），将其准备好，因为对目录的请求看起来像是对文件的请求。被请求路径的最后部分保存在请求的名字域中。

15214: 返回index_directory函数给定的值，如果成功创建目录表，这个值为0。index_directory函数真正创建了HTML并把它发送给客户端浏览器。

27. autoindex_handlers

15226: 定义调用本模块的主函数：handle_autoindex。

28. autoindex_module

15232: 为Apache内核定义本模块的异常分支。本模块只包括create和merge的配置、命令表格以及处理器。

第8章 登录模块

本章描述四个用于用户请求登录的模块：mod_log_agent、mod_log_referer、mod_log_config和mod_usertrack。如果定义了给定服务器的上下文相关指令（后面章节将要介绍），前三个登录模块为每个请求生成登录入口。mod_log_referer和mod_log_agent（登录相关页面和客户端的浏览器）的使用均与mod_log_config（可用Common Log Format [CLF]格式或其他自定义格式登录一般请求信息）独立。用户自己可以定制有很多域的复杂登录格式。

Apache使用mod_usertrack为任何访问某种上下文的请求生成cookie，这种上下文必须支持cookie。如果请求已有一个cookie，mod_usertrack将该cookie与请求重新联结起来。创立在cookie和请求之间的联结允许同一个客户端通过Apache发送用户脚本来跟踪本服务器上的前一个动作。

8.1 mod_log_agent模块

本模块与其他登录模块相互独立。如果在服务器上下文中使用了AgentLog指令，mod_log_agent为这些指令中说明的文件发出请求，然后登录到客户端使用的浏览器上去。

8.1.1 模块结构

在mod_log_agent的初始化中，为服务器上下文准备一个保存登录文件名的变量。在请求处理的登录阶段调用mod_log_agent。如果为当前上下文定义了一个有效的登录文件名，而且检查文件描述符时发现该文件已打开，那么从请求首部就开始询问agent并将该询问写入到登录文件。这种结构如图8-1所示。

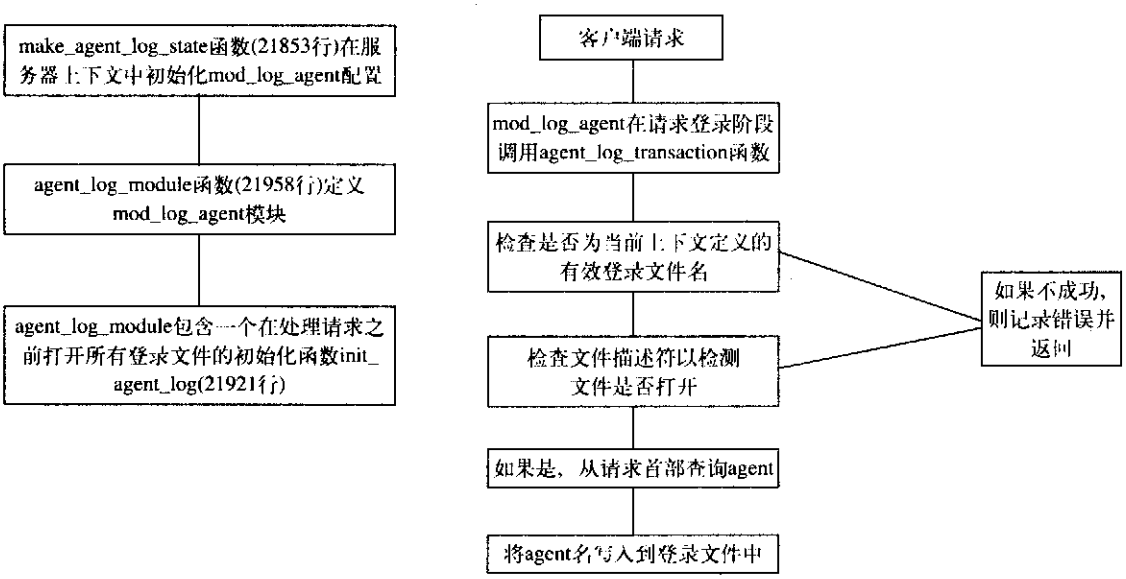

图8-1 mod_log_agent通过登录到浏览器来提出请求

8.1.2 定制

我们可以在任何基本的登录任务中使用mod_log_agent，由客户端浏览器提供的首部信息均可通过修改mod_log_agent被记录下来。这要比扩展mod_log_config模块来处理LogFormat指令中的附加参数简单得多。

如果agent做出了与你的web服务器附加行动相联系的请求，可以扩展mod_log_agent模块，将其用于登录关于请求的附加信息，或将其用于检测其他agent的行动（例如发送email或自动减少静态计数器）。

1. agent_log_stute

21848: agent_log_state结构是mod_log_agent模块的配置记录。它包括两个域：fname和agent_fd。其中fname是登录文件的文件名，agent_fd是文件描述符（在为执行写操作打开文件时创建）。

2. make_agent_log_state

21853: make_agent_log_state函数通过定义变量（agent_log_state类型）来对一个服务器上下文中的mod_log_agent配置进行初始化，并初始化该变量的两个值。

3. set_agent_log

21866: 当分析AgentLog指令时调用set_agent_log函数（21879行）。在21870行获得当前服务器配置。在该上下文中agent应该登录的文件名（从arg变量中取得）保存在配置记录cls中（21873行）。

4. agent_log_cmds

21877: agent_log_cmds结构定义了mod_log_agent使用的简单指令_AgentLog。set_agent_log函数的定义是用来处理在配置文件中发现的AgentLog指令。

5. open_agent_log

21885: open_agent_log函数打开了在一条AgentLog指令中定义的全部登录文件。为了在处理请求时使登录写得更快，这一步在服务器的初始化阶段完成。init_agent_log函数反复调用了open_agent_log函数，它要为每个可能有单独定义AgentLog的服务器调用一次。

21888: 使用ap_get_module_config获得当前服务器的配置。

21891: 根据cls ->fname的值打开，使用ap_server_root_relative来设置它的文件名；结果文件名保存在fname中。

21894: 如果本登录文件（agent_fd域是cls）的文件描述域大于零，那么自从登录文件初始化后（open_agent_log假设文件已打开），不采取进一步行动而返回。

21899: 如果登录贮藏量文件的名字以一个管道记号（|）开始，使用21902行的ap_open_piped_log命令将当前日志文件与通过管道传输的主服务器日志文件联结起来。如果运行命令（存贮在pl中）结果为空，则出错。记录了错误信息后，open_log_agent仍然返回值1（返回给init_agent_log）。

21908: 如果成功打开管道，使用ap_piped_log_write_fd调用将文件描述符存贮在agent_fd域中，这样agent_transaction(11927行)即可以用它来写日志文件。

21910: 如果文件名不以管道记号开始（也不是空字符串），使用ap_popenf来打开日志文

件（21911行）。这个命令也立即把文件描述符放到agent_fd域中。21838行开始处定义了xfer_flags和xfer_mode。

21913: 如果打开文件失败，记录错误信息，open_log_agent返回。

6. init_agent_log

21921: init_agent_log函数使用for循环来处理每一个它能访问的服务器配置，重复调用open_agent_log，试图为每一个服务器打开AgentLog文件。

7. agent_log_transaction

21927: agent_log_transaction函数为每个事务（请求）登录agent。可以在请求处理的登录阶段调用它（29974行）。

21929: 为了使所写文件描述符为函数所得，使用ap_get_module_config为服务器获得配置。

21937: 测试文件描述符的有效性。如果无效，函数返回值OK，并在21942行检测文件名，如果是空字符串，函数返回值DECLINED。

21940: 检查这次登录请求是否是一列子请求中的最后一个。

21948: 使用ap_table_get为user_agent首部询问首部表。在agent中设置该首部。

21950: 如果agent非空，选用agent和新的一行填充str变量，再将该str写到文本描述符agent_fd定义的文件中去，这样就登录了请求。

8. agent_log_module

21958: 定义mod_log_agent模块。这里包括一个初始化函数——21921行定义的init_agent_log，它在处理请求之前打开了所有的登录文件。还包括标准配置和函数的指令类型。本模块的核心代码——21974行的agent_log_transaction供登录阶段调用。

8.2 mod_log_referer模块

本模块与其他几个登录模块相互独立。如果为某个服务器上下文提供了RefererLog指令，mod_log_referer将在客户端细化的访问页面登录到指令中指明的文件中去。当选中当前请求时，登录的信息是浏览器所浏览页面的URL（Web地址）。如果某个用户从书签表中选择了URL或熟练地输入了URL，从客户端来的首部无访问页面。

在大多数情况下，用户不希望登录所有访问页面。比如说，如果一个用户从页面上你的站点移过，访问的就是你的站点所含的URL。如果你不需要跟踪用户移过你站点时的踪迹（一个艰巨的任务），这种信息不会包括在你的访问日志中。使用了RefererIgnore指令的mod_log_referer忽略了匹配指令参数的参数域。这种情况下不登录任何信息。

8.2.1 模块结构

在mod_log_referer的初始化中，准备一个为服务器上下文而存贮登录文件名的变量，并初始化这些变量的域。如图8-2所示，在请求处理的登录阶段，调用mod_log_referer。如果为当前上下文定义了一个有效的登录文件名，并且通过检查文件描述符发现该文件已经打开，从请求的首部查询访问信息。如果访问信息不在为上下文而定义的忽略表(ignore list)中，将其写入登录文件。

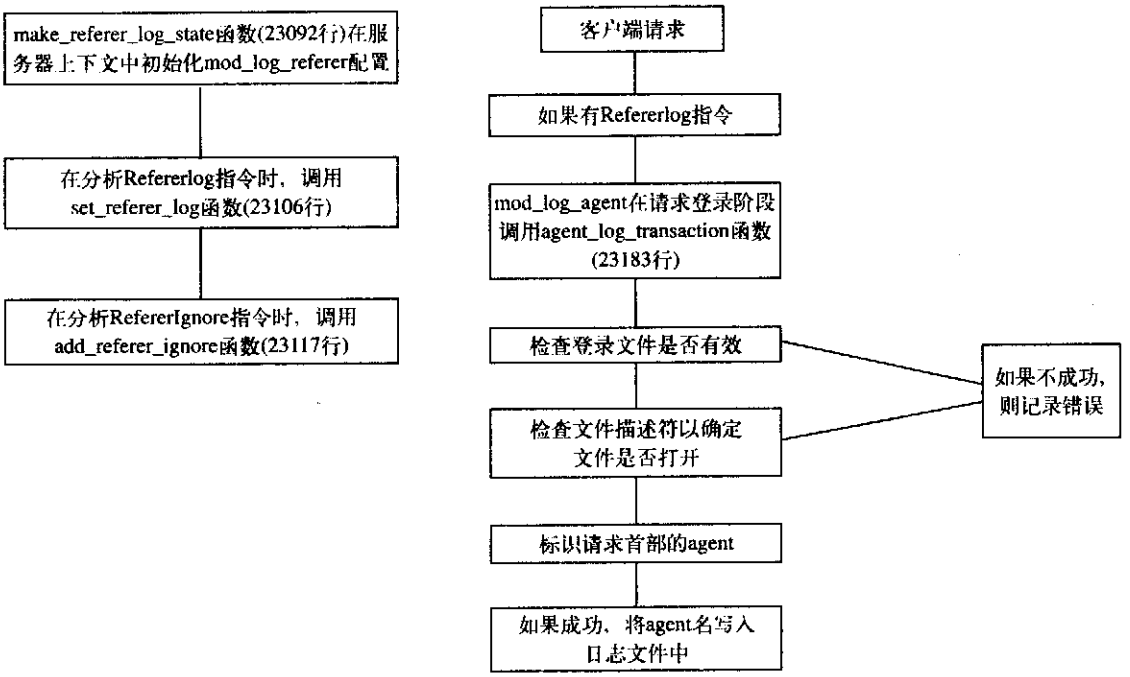

图8-2 在生成请求时，mod_log_referer登录浏览器所浏览页面的URL

8.2.2 定制

像mod_log_agent一样，mod_log_referer是一个能被任何基本登录任务扩展和重用的简单模块。可以通过修改mod_log_referer来记录客户浏览器提供的任何首部信息。相比较而言，这比扩展mod_log_config模块来处理LogFormat指令中的附加参数要简单得多。

如果想要Web服务器执行基于访问Web页面URL的具体任务（例如返回消息而不是返回所请求的页面），可以通过修改mod_log_referer来做到这一点。

忽略表的自动生成要基于各种对本模块和其他程序所定义的简单规则的探索，也可以完成更复杂的修改。这将使访问信息登录对跟踪系统和初始化你的Web站点变得更有价值。

1. 常数和结构定义

23076: 定义为打开文件操作所使用的标记（23168行）。

2. referer_log_state

23086: referer_log_state结构是mod_log_referer模块的配置记录。它包括三个域：登录文件的域名fname，文件描述符（在文件为写打开时创建）referer_fd和从RefererIgnore指令中取来的忽略访问数组。

3. make_referer_log_state

23092: make_referer_log_state函数创建一个referer_log_state类型的变量并初始化它的三个值（从23099行到23102行），这样它就在服务器上下文中初始化了mod_log_referer的配置。

4. set_referer_log

23106: 在分析指令时，发现一个RefererLog指令后，调用set_referer_log函数。在23110行获得当前的服务器配置。在上下文中该登录访问信息的文件名（从arg变量中获得）被存贮在名为cls的配置记录中。

5. add_referer_ignore

23117: 发现RefererIgnore指令时调用add_referer_ignore函数。它取来传入的参数，并将它们加入到mod_log_referer模块的当前服务器配置记录的referer_ignore_list域中去，开始于23213行的referer_log_transaction将查询这个域。

6. referer_log_cmds

23131: referer_log_cmds结构定义了mod_log_referer使用的两条指令：RefererLog和RefererIgnore。当在配置文件中发现这两条指令时，对应于这两条指令调用的操作分别是：set_referer_log和add_referer_ignore。

7. open_referer_log

23142: open_referer_log函数打开在RefererLog指令中定义的全部登录文件。在处理请求的过程中，服务器在为使登录写操作进行更快的初始化阶段调用本函数。init_referer_log函数重复调用open_referer_log函数，每次都是为了某个定义了RefererLog的服务器而调用。

23145: 使用ap_get_module_config获得当前服务器的配置。

23148: 为使用了ap_server_root_relative结构的fname变量设置登录文件名。cls变量的fname域是作为基本文件名使用的。

23151: 如果本登录文件（cls的referer_fd域）的文件描述符域大于零，登录文件初始化后，假设文件已经打开，open_referer_log不做进一步处理返回。

23155: 如果登录存贮文件名以管道记号开始，可以用23158行的ap_open_piped_log命令通过管道的方式将当前日志文件与主服务器日志文件联结起来。如果可利用命令（存贮在pl中）的值为NULL，则报错并记录出错信息，但函数仍然返回值1（返回给init_referer_log）。

23165: 如果成功的打开了管道，为提供给referer_log_transaction以用来写日志文件，使用ap_piped_log_write_fd调用将文件描述符存贮在referer_fd域中。

23167: 如果文件名不以管道标记（|）开头，使用23168行的ap_popenf调用打开日志文件。这个命令以同样的命令行方式将文件描述符存入referer_fd域。xfer_flags和xfer_mode在23076行的开始就有定义。

23170: 如果打开文件不成功，记录出错信息，open_refoerer_log返回。

8. init_referer_log

23177: init_referer_log函数使用for循环来处理它能访问的服务器配置，重复调用open_referer_log，试图打开为每个服务器上下文而定义的RefererLog文件。

9. referer_log_transaction

23183: referer_log_transaction函数为每个事务（请求）记录访问Web页面。我们在请求处理的登录阶段调用它。

23187: 为了所写文件描述符为函数所得，使用ap_get_module_config为服务器获得配置。

23195: 测试文件描述符的有效性。如果无效，函数返回值OK，并在23200行检测文件名，

如果是空字符串，函数返回值DECLINED。

23198: 检查这次登录请求是否是一列子请求中的最后一个。

23203: 使用ap_table_get为Referer首部询问首部表。在referer中设置该首部。

23204: 如果referer非NULL，在为请求登录做准备时，将访问写入referertest，在忽略访问表（它由任何一条referer ignore指令定义）中检测它。

23213: 初始化在启动检测忽略访问信息的循环时所用的指针。

23222: 使用for循环检查在忽略表中是否有一项访问与客户端所提供的相对应。在23227行进行检查，一旦在循环中出现匹配，不记录任何信息返回。

23232: 因为for循环不退出模块，所以访问信息不在忽略表中。用一项访问（当前页的URI）和新的一行填充变量str。最后，将str写到文件描述符referer_fd定义的文件中。

10. referer-log_module

23240: 定义mod_log_referer模块。这里包括一个初始化函数——23243行定义的init_referer_log，它在处理请求之前打开了所有的登录文件，还包括标准配置和函数的指令类型。本模块的核心代码——23256行的referer_log_transaction供登录阶段调用。

8.3 mod_log_config模块

Apache中更好的登录工具是mod_log_config。

这个模块通过使用CustomLog、LogFormat、CookieLog和Transferlog四条指令，让用户自由定义登录格式，并让Apache登录任何根据这些格式提交的事务。通过mod_log_referer和mod_log_agent登录的项是一些特殊情况下的项，它们都能用mod_log_config来登录。然而，mod_log_config的缺省登录配置并不包含referer和agent域。

使用mod_log_config，可以确定TransferLog的具体位置。如果希望用更复杂或非标准的格式，可以用LogFormat指令定义一种新格式，再用CustomLog指令来实现它。也就是说，用LogFormat可以定义自定义的格式字符串，而在CustomLog指令中可以使用这种字符串或包容这种字符串。

mode_log_config允许根据已定义好的不同格式多次登录单一事务。可以在服务器或目录上下文中定义格式；如果上下文无具体格式，mod_log_config把它设置成缺省的主服务器登录格式。在整个模块中，如果不提供用户自定义的格式字符串，CLF将被认定为是缺省格式。众多的Web登录分析工具均采用这种缺省格式。

8.3.1 模块结构

mod_log_config模块的大量工作涉及到对配置文件中出现的指令进行语法分析。这主要是由22786行的log_format函数和22817行的add_custom_log函数来实现。在定义完格式后，使用multi_log_transactnon函数（22695行）来登录每一个事务。这个函数先检查对等服务器的配置，或先设置成主服务器配置，然后调用config_log_transaction函数来为当前上下文已定义好的每一种格式准备登录入口。Process_item函数（22654行调用）使用从指令分析中得来的格式字符串来准备登录入口，相对来说这更容易实现。本模块的结构如图8-3所示。

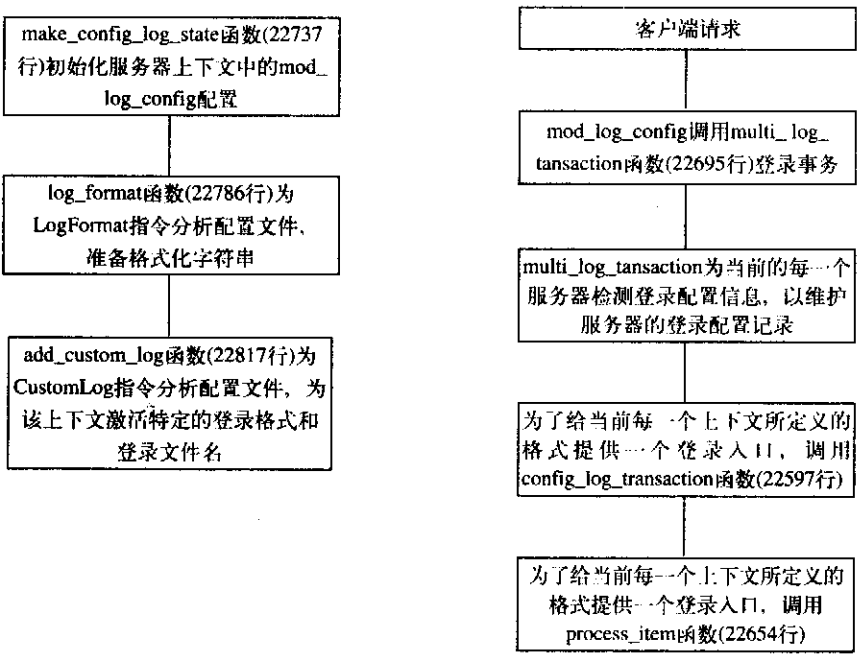

图8-3 用mod_log_config产生Apache登录文件

8.3.2 定制

你可以使用两种方法来扩展mod_log_config模块。第一种是扩展CustomLog选择的信息格式化参数，使其包括各种各样的关于系统、请求或环境等附加信息。这样的参数必须在使用了log_item_list结构的指令分析函数和登录处理函数process_item中(22552行)有定义。必须为支持的每一种数据提供附加的小函数（这与22194行的log_header_out函数所举例子相同）。

mod_log_config支持登录域的条件包含，这种条件包含只是在登录域参数前加上一个“!”操作符而已。我们既可通过分析格式字符串中的附加运算符（用户自己定义的）来扩展功能，也可以通过编写具体例程来测试函数体内的代码并返回一部分信息来扩展功能。

定制mod_log_config的第二个主要方法是修改写登录的位置。这在config_log_tansaction中很容易实现，在这个函数里，完整的登录入口写到了22689行上的一个文件里。你可以将登录入口记载到数据库中去，也可以通过email发出去，还可以为了测试而丢弃它。在改变这样入口的处理函数时，请注意登录将给服务器带来很大负载，所以避免使用对CPU和硬盘进行独占的操作。

另外需要指明的是，本模块的源码中有非常多的注释。当由于某些具体情况需要改进模块时，阅读这些注释将使你受益非浅。

1. 常量和结构定义

21990: 为写操作打开登录文件定义一些标识。

2. multi_log_state

22306: 为mod_log_config所需配置记录定义了multi_log_state结构。它包括一个缺省的格式字符串和为写登录信息所提供的配置文件。这是一个在22054行开始定义的config_log_state数组变量。

3. format_integer

22083: 格式化一个整数将其返回成一个字符串。pfmt函数（22088行）使用本函数来返回负数。

4. constant_item

22098: 返回作为参数传入的常量。尽量它显然有点多余，但它为登录字符串的所有可能部件提供了一个常量接口。

5. log_remote_host

22104: 用信息的一部分返回字符串，这些字符串将被完全组织成登录字符串。本区域的函数为CustomLog或LogFormat指令中定义的项提供返回字符串。这些函数中的大部分从请求变量r或标准Apache API调用中查询信息。这些函数包括：log_remote_host(22104行)、log_remote_address（22112行）、log_remote_logname（22118行）、log_remote_user（22124行）、log_request_line（22138行）、log_request_file（22158行）、log_request_uri（22163行）、log_status（22168行）、log_bytes_sent（22174行）、log_header_in（22188行）、log_header_out（22194行）、log_note（22208行）、log_env_var（22212行）、log_request_time（22217行）、log_request_duration（22248行）、log_virtual_host（22259行）、log_server_port（22265行）、log_server_name（22278行）、log_child_pid（22284行）。你可以通过修改上述函数来改变Apache为自定义登录文件准备信息的方法。

6. log_item_list

22297: 格式化字符串选项（例如用a%表示远端地址）可用于配置登录格式，这里定义的结构就是用来保存格式化字符串选项的。在登录获得格式化字符串所指部分信息的事务时，这种结构包含有被调函数的名称。例如，22307行上的a表示log_remote_address函数（22112行）。

7. find_log_func

22365: 返回在log_item_list（22297行）中能匹配函数所传入参数的项。

8. parse_log_misc_string

22377: 为使字符串逐字传入登录文件，分析登录格式化定义中的一个名叫misc的字符串。在22404行开始检测各种遗漏字符。在22433行，参数sa被设置成为函数准备好的s值。

9. parse_log_item

22437: 为了后面能用某种格式登录事务，准备该格式字符串（由CustomLog或LogFormat指令生成）的内部表示。为了在登录中设置条件域，parse_log_item查找具体字符（诸如! ）。变量it（登录项）保存了为本函数准备的信息。

22442: 如果参数不以百分号(%)开头，认为它是misc字符串，使用parse_log_misc_string对它进行语法分析。

22510: 如果在语法分析中不能辨认格式化字符串，或者当前项需要更多信息时，语法错误引起字符串结束，返回错误信息。

22517: 返回变量sa中的处理字符串。

10. parse_log_string

22526: 每次使用parse_log_item处理格式化字符串中的一项。为便于在登录时使用，用一个数组寄存语法分析后的格式化项。变量s保存了将要进行语法分析的格式化字符串。在while循环中（22534行）和函数末尾（清理最后项，22542行）调用parse_log_item函数。

10. process_item

22552: 为要登录的某个实际事务处理格式化字符串中的单一项。为了查看该项是否真正被包含在输出的登录字符串中，本函数要测试该项的条件属性（在格式化字符中用“!”设置）。

22567: 循环遍历当前登录格式的每一个元素。如果找到登录项，将in_list设置成1。

22574: 如果条件表中存在该项，根据当前condition_sense的值检查它是否应该包含在条件表中。

22582: 如果condition_sense有效或该项无条件（总是包含它），使用log_item_list结构（22297行）中为这种项定义的函数返回该项。例如，如果登录的项为“Bytes Sent”，item_func将指向log_bytes_sent函数（22334行）。

11. flush_log

22588: 使用变量cls（表示当前登录状态）提供的登录文件描述符log_fd来将登录缓冲区写入硬盘。在编译这个函数时是有条件的：平台上是否正使用这种缓冲登录。

12. config_log_transaction

22597: 使用特定格式和登录文件登录单一事务（请求）。multi_log_transaction函数重复调用本函数来为当前请求做所有必需的登录工作。

22611: 检查这种登录工作的文件名是否存在，若不存在，拒绝处理。

22620: 检查当前格式化字符串中的条件项。根据结果，进一步检查环境变量（subprocess_env域）。

22636: 定义当前登录事件的格式。如果已经为该事件定义了格式化字符串，就使用该字符串（cls>format）。如果没有定义这样的字符串，使用缺省格式default_format。然后为这种基于在格式中使用的元素而重组的字符串分配空间。用ap_palloc来将strs和strl设置成重组登录字符串的工作区。

22653: 循环遍历格式表中的每一个元素。用process_item函数来获取每一项的实际数据。

22657: 为了下一步工作重组整个登录字符串，在strs数组中计算process_item返回的每一项的行长。

22661: 如果为本平台定义了缓冲登录，检查缓冲区大小，看空间大小是否允许缓冲登录项。如果大小不够，在22671行上将信息写入硬盘；如果大小足够，使用22674行的循环，一个元素一个元素的缓冲登录字符串。

22681: 对于无缓冲平台，用strs和strl数组中的信息重组变量str。

22689: 将重组后的登录字符串写入到文件描述符log_fd。

13. multi_log_transaction

22695: 使用为当前上下文定义的所有登录方式登录一次事务（请求）。如果没有这样的定义，主服务器使用登录定义来登录请求。这是Apache为一次请求调用的主登录函数，它在23060行的模块定义中设置。

22707: 如果为当前上下文定义了一种具体上下文（virtual_host_type）登录格式（变量mls -> config_logs中的一个值），那么本函数使用22710的循环来遍历为这个上下文定义的所有格式，每次调用一次config_log_transactron函数。

22717: 如果没有定义具体的上下文登录格式，主服务器登录将server_config_logs中的配置信息作为本次事务的格式。本函数使用22720行的循环来遍历所有的主服务器登录配置，为每个登录配置调用一次config_log_transaction函数。

14. make_config_log_state

22737: 通过创建保存登录信息和文件名信息的multi_state_log变量来为登录初始化服务器的上下文。这里使用mod_log_config支持的指令来为单个上下文保存多次登录指令。

15. merge_config_log_state

22763: 调用ap_overlay_tables API为两个服务器上下文联结multi_state_log记录。必要时，将缺省格式添加到返回结构中去。

16. log_format

22786: 从一条LogFormat指令提供的信息中准备一个分析好的登录格式字符串。Apache根据config_log_cmds的指示对这些域进行语法分析，并将它们做为参数传送到这个函数。用LogFormat指令可以定义格式，但不能使其活跃。也就是说，如果没有一条TransferLog或CustomLog指令提供登录文件名，就不会登录。

22791: 为了登录结构mls能够用于存贮语法分析后的格式化信息，获取当前上下文的配置。

22802: 如果提供了这种格式的别名，使用parse_log_string进行语法分析，并将其放置在当前上下文的数组mls中。

22808: 如果无别名（该名字为NULL），使用parse_log_string对其进行语法分析，并将其放置在mls的default_format域中。

17. add_custom_log

22817: 为使一个具体登录格式和登录文件名为上下文可用，处理一条CustomLog指令。在单一上下文中可以使用多条CustomLog指令。

22822: 获得当前上下文的配置。

22831: 检查格式化字符串中的条件项。

22843: 将变量fname和formate_string设置成函数头部提供的值。

22849: 如果format_string非空，parse_log_string对其进行语法分析。

22852: 因为登录文件并没有为写操作打开，设置本次登录文件描述符值为－1。

18. set_transfer_log

22857: 使用TransferLog指令创建事务登录。Apache在22877行调用本函数。本函数调用add_custom_log来为当前上下文形成一个缺省格式。TransferLog指令被当作是为CustomLog指令的特殊情况而提供。

19. set_cookie_log

22863: 根据cookielog指令为cookie建立登录。Apache在22884行调用本函数。本函数通过使用事务所含cookie格式调用add_custom_log来定义一次登录。

20. config_log_cmds

22870: 定义本模块使用的指令。位于配置文件中的每条指令后都有它所调用的函数，每条指令所含参数格式和个数与具体指令有关，Apache利用这些参数调用结构中第五个域所定义的命名函数（例如，TAKE23或TAKE1）。在本模块中，处理各条指令的函数紧密相关。例如，set_transfer_log和set_cookie_log是add_custom_log的特殊情况，它们内部均调用了add_custom_log函数。

21. open_config_log

22890: 打开已定义的登录文件。

22895: 如果文件描述符大于0，说明这次配置使用共享主服务器登录，不做任何事。

22900: 如果文件名不存在，不做任何事。

22905: 如果具体文件名以管道标记开头，使用ap_open_piped调用为写登录入口准备管道。使用ap_piped_log_write_fd调用将文件描述符分配给log_fcl。

22915: 对非管道文件名，获得与服务器相关的根路径，为某个具体登录使用ap_popenf调用打开文件。如果打开文件操作失败，记录出错信息。否则，log_fd包含将来写这次登录所用的文件描述符。

22. open_multi_logs

22932: 使用open_config_log函数为特定的服务器上下文打开每一个登录文件。

22936: 为当前服务器上下文获得配置。

22943: 如果这个msl结构的缺省格式化字符串并不存在，这里试图用以下两种方法来创建它，一是使用LogFormat指令提供的default_format_string，二是使用CLF的常量定义DEFALILT_LOG_FORMAT。两种情况下，均使用parse_log_string来准备字符串并把它保存在default_format域中。

22957: 如果已定义了特定上下文服务器的登录信息，在这里使用它。然后它再通过调用parse_log_string建立所有的格式化字符串。接着就是通过调用open_config_log打开这些字符串。

22976: 如果没有为本上下文定义特定的登录配置信息，使用存贮在server_config_logs中的缺省服务器登录启动。接着检查是否存在有效的格式化字符串，必要时调用parse_ log_string对其进行格式化。最后，使用open_config_log打开登录文件。

23. init_config_log

23000: 初始化为服务器定义的所有登录文件。首先，使用主服务器记录s，调用Open_multi_logs。其次，通过循环遍历s数组为每一个虚拟服务器上下文打开所有登录文件，并为它们调用open_init_logs函数。最后，Apache在初始化阶段调用init_config_log函数。

24. flush_all_logs

23017: 如果为本平台定义了缓冲登录，本函数将保留在缓冲中的所有登录入口复制到适当的文件描述符中去。

23024: 循环遍历每一种服务器的上下文，为每一种上下文获得配置信息，并为每一种服务器的每一种登录配置调用flush_log函数。

25. config_log_modnle

23044: 使用函数为初始化登录文件（23047行的init_config_log）定义模块，创建登录数据结构（make_config_log_state和merge_config_log_state），创建为本模块(config_log_cmds)所定义指令的命令表格和自己的登录阶段(nulti_log_transaction)，以及创建支持本模块的平台和child_exit阶段的写缓冲流。

8.4 mod_usertrack模块

与本章其余三个模块相比，mod_usertrack并不是同种意义下的登录模块。然而，可以用它来保存用户操作轨迹。当使用了CookieEnable指令定位了你的Web站点以后，一旦对这个位置的请求到达，mod_usertrack就被激活。为了让Apache当做标准环境信息发送的脚本或程序得到请求首部，mod_usertrack在请求首部内部控制cookies。

如果所到请求有Apache cookie，mode_usertrack将其放置在为了请求（例如登录）而跟踪的信息中。如果所到请求应该有却没有cookie首部，mod_usertrack分配一个唯一的cookie，并让它能为脚本和程序所得到。

8.4.1 模块结构

当Apache对请求做出反应、发送了脚本或程序以后，为使它们得到一个cookie首部，mod_usertrack在fixups阶段处理请求。spot_cookie函数在客户端请求首部查找一个cookie。如果找到一个，将其放入请求的notes域；如果没有找到，调用make_cookie函数产生一个新的cookie并将其放入请求的notes域（如图8-4所示）。本模块仅在设置了CookieTracking指令的位置使用。

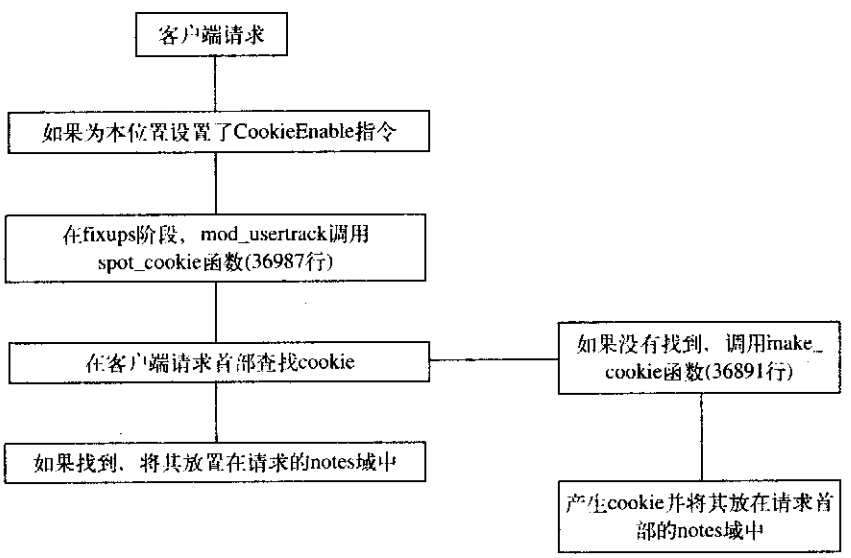

图8-4 通过cookie方法，用mod_usertrack来跟踪用户轨迹

8.4.2 定制

mod_usertrack为由客户端请求而引起Apache启动的脚本或程序提供了一种用户跟踪机制。

为了直接控制这些cookie，你可以扩展mod_usertrack来执行特定的任务，诸如将它们发往某个数据库。为了保存每个cookie的访问频率之类的统计信息，必须与客户端其他的首部信息联系起来（例如访问页面、时间或者主机名）。

这些控制cookie的特色必须与使用cookie的行为相分离。例如，基于cookie数据库，为客户端准备文档的函数并不在mod_usertrack模块中，必须使用另外的与用户跟踪无关的模块来单独保存这些内容函数。

可以不用Apache＝，而用别的不同的cookie名来定制本模块，从单Web服务器到单客户端允许发送和维护多个cookie，这对虚拟主机环境很有帮助。

1. cookie_log_state

36874: 定义一种叫做cookie_log_state的结构，将其做为mod_usertrack的配置记录。为了保留在服务器上下文中为cookie定义的终止时间，本结构包含有终止时间域。

36883: 为了使make_cookie函数产生的cookie在2000年前后一致，这里定义了一个标记。

36889: 为所有与本模块相联系的cookie定义名字。用spot_cookie函数查看这个名字，看本服务器的某个客户是否返回了一个cookie。

2. make_cookie

36891: 当客户端请求并没有cookie时（但根据CookieTracking指令应该包含一条），返回一个新创建的cookie。

36894: 为本次上下文获得服务器配置。

36896: 准备一个在产生唯一cookie时所使用的保留时间信息的数据结构。某些不同操作系统下的时间系统需要使用本节的DEFINE语句。

36904: 定义一个变量来保留正被定义的cookie。最大长度不超过1024字节。

36906: 使用ap_get_remote_host调用获得远端客户机。在试图创建新cookie时将其用做为cookie的一部分。

36917: 如果当前使用的操作系统不提供时间-日期函数，使用times函数得到时间值。

36919: 使用远端主机名、时间值和进程ID重组新cookie，并且ap_snprintf将新cookie重组入变量cookiebuf。

36932: 不使用任何时间函数，用ap_snprintf将新cookie重组入变量cookiebuf，Get Tickcount函数用于产生当前时间的整数表示。

36938: 用gettimeofday获得日期——时间，接着在36940行将其重组到新cookie中去。

36946: 如果cookie的expires域设置在这里，在变量when中记录基于请示时间和终止时间的时间长度。

36959: 如果变量when太大，而且设置了MILLENIAL标记，在2000年前改变when的值，这样可以避免与其他（客户）程序的“翻滚”问题。

36966: 添加到cookie字符串中去，将新字符串放置在new_cookie中。我们事先用ap_snprintf和从服务器配置中的expires域取来的终止时间准备好cookie名（Apache＝）和cookiebuf信息，以组成新字符串。

36976: 如果这个位置没有指明终止时间，用cookie名（Apache＝）、cookiebuf和路径语句重组cookie首部。

36979: 把刚创建的、返回给客户端的cookie加入。

36981: 为了允许登录者记录每个新cookie（新用户）访问站点的时间，将cookie加入到notes中。如果mod_log_config使用的LogFormat指令包含有恰当的域标识符，仅仅登录cookie。

3. spot_cookie

36987: 为某个cookie检查到来的请求首部，如果它们应该包含而没有包含cookie，使用make_cookie函数为其创建一个。

36990: 使用ap_get_module_config获得服务器配置。

36995: 如果服务器配置表明服务器上这个位置的cookie不能使用，返回DECLINED。

36998: 设置了enable，那么某个cookie应该与这次请求相联系，spot_cookie这时使用ap_table_get在到来请求首部查找cookie。如果不返回cookie，spot_cookie转到37016行准备一个新cookie。

37000: 如果返回的cookie不是Apache cookie（检查COOKIE_NAME），spot_cookie还是跳转到37016行为这个客户端建立一个新的Apache cookie。

37003: 发现一个有效cookie后，使用字符串粘贴来隔离cookie本身。为了能登录cookie，使用37011行的ap_table_setn将其加入到这次请求的notes中去。

4. make_cookie_log_state

37020: 使用ap_palloc创建一个cookie_log_state变量并初始化该变量的域，这样，我们就为一个服务器上下文配置了新的记录。

5. make_cookie_dir

37032: 通过返回一整数变量，为目录位置配置一个新记录。

6. set_cookie_enable

37037: 为当前服务器和目录配置将参数值设置成调用本函数的CookieTracking指令所给出的值。

7. set_cookie_exp

37044: 在本次服务器上下文中为cookie设置终止时期和时间。在发现CookieExpires指令时调用本函数。

37054: 如果CookieExpires指令提供的参数为一数字，将ASCII文本参数转换成数字，这个数字保存在配置记录cls的expires域中。

37067: 将CookieExpires提供的第一块信息分析成word。它不是简单的数字。如果它以plus开头，立即获取第二块信息，准备处理指令。

37073: 循环遍历所有参数，为终止搜索正确的日期和时间。

37075: 因为已经分析了plus，下一部分必须是数字。如果不是（用ap_isdigit检查），返回下列错误信息：不能处理这种终止时间。

37082: 获取下一部分的指令参数，它应该是一个<TYPE>字符串（例如weeks或months）。

37087: 根据word字符串内容，为使终止日期与累加值（根据天数、周数、月数等累加）匹配，设置时间要素。如果没有匹配标准字符串，最后返回下列错误信息：不能处理终止值。

37104: 根据项的号数（num）和每一个if语句测试项的要素（天数、周数、月数等），更

新变量modifier。

37107: 将指令数据中的下一个参数读入到word中去。如果还剩有别的参数，while循环将再一次重复更新终止值。否则，跳出循环，将modifier的值分配到配置记录cls的expires域中去。

8. cookie_log_cmds

37115: 创建一种保存mod_usertrack所使用指令的结构。这些指令包括CookieExpires（一旦发现这种指令，调用set_cookie_exp函数）和CookieTracking（一旦发现这种指令，调用set_cookie_enable函数）。

9. usertrack_module

37125: 用下列信息定义模块本身：目录和服务器配置函数、寄存应用指令的命令表以及在fixups处理阶段调用的spot_cookie函数，也正是在这个阶段检查了cookie或生成了必需的cookie。

第9章 其他模块

本章讨论的是Apache中经常使用，但很难归于某一类的七个模块，它们是：

- mod_perl——允许Apache预装入Perl脚本，并在Apache中执行，而不用另外启动一个Perl解释器。这些脚本还可以用于配置Apache。
- mod_example——一个用于帮助开发者学习Apache模块构件结构的示例模块。这一模块输出每个请求的状态信息，以便跟踪发生在模块中的事件。
- mod_mmap-static——定义Apache启动时将加载到内存的文件。当收到请求时，这些文件将直接从内存发往客户方，因此提高了性能。然而这些在内存中映射的文件不能改动，否则必须重启服务器。这一模式仍处在测试阶段。
- mod_userdir——将Web服务器中用户主目录内的请求转换成Apache能基于UserDir指令取到的文件名形式，如/~jsmith变成/home/jsmith/public_html。
- mod_so——在1.3版及以后的版本中，用来加载动态共享对象。这个模块主要用来加载另外的模块，而无需重新编译Apache，尽管它也能用来加载任何Apache代码用到的其他共享对象。
- mod_spelling——当某个被请求的文件在Apache服务器上没有找到时，这一模块可以查出某些细小的拼写错误，如顺序颠倒、遗漏字母和大小写错误等。如果找到一个类似名称的文件，这个文件名将作为目标。
- mod_unique_id——为每一个请求创建唯一的标识符。这个标识符很像mod_usertrack创建的cookie，但对于mod_unique_id，标识符是放在环境变量中，服务器的脚本必须定义是否以及如何使用该标识符来跟踪请求。

9.1 mod_perl模块

mod-perl是所有Apache模块中最复杂、功能也最强大的一个模块，通常mod_perl作为一个独立的软件包下载和编译，而不是包括在缺省安装中。它包含一个Perl模块和一个Apache模块，为Apache提供一个内置的Perl解释器，利用这个解释器，可以使运行CGI的脚本在Apache中得以执行。使用mod_perl可使Apache的性能显著提高，因为每次执行脚本时不用重新启动Perl解释器。Apache上加载mod_perl（重新编译或使用动态共享对象）之后，做以下配置，就可以运行脚本了：

```
<Location/perl/>
SetHandler perl-script
PerlHandler Apache::Registry
Options ExecCGI
</Location>
```

一旦URL请求/perl/中的某个文件，mod_perl中的Perl解释器就执行该文件。在Apache服务器启动时预载入Perl脚本可以进一步提高性能，这和利用mod_mmap_static（本章后面将做进一步介绍）预载入文件兼容。在虚拟主机用户中使用下面示例的命令，可以预载入Perl模块

或脚本:

```
PerlModule Apache::SSI  MyMoudle::handler
PerlScript /www/cgi-bin/process_form.pl
```

一旦Perl代码被载入，客户端请求或服务器端调用包括SSI命令就可以访问到它。例如，一个将命令或程序执行结果嵌入到HTML页面的标准方法就是使用SSI exec命令来执行程序。然而，如果用户自己编写了与该程序执行同样任务并用exec调用的Perl函数时，mod_perl通常执行用户自己编写的那个函数而获得比SSI exec命令更好的功能。用户自编的带输出的Perl函数（这里叫做过程）语法如下:

```
<!--#Perl sub="Apache::Test::procedure"-->
```

使用mod_perl时，可以将Perl代码嵌入到Apache配置中，这些Perl代码可以解释Perl并配置Apache Server。给出下面示例说明:

```
<perl>
# Insert a test for server performance
if($test=0){
  $MinSpareServers=5;
  $MaxSpareServers=5;
  $MaxClients=50;
}
else  {
  $MinSpareServers=10;
  $MaxSpareServers=25;
  $MaxClients=250;
}
1;
</Perl>
```

只有对Perl和mod_perl文档仔细学习和研究才能深入理解和熟练使用mod_perl的功能，而这里给出的信息只是指导用户去探索mod_perl代码。

模块结构

由mod_perl创建的Apache和Perl解释器之间的互操作比较复杂，为了实现对Apache的交互设置和在Apache子进程中运行脚本，就需要很多其他模块中不必要的控制机制。为了更深入地学习mod_perl是如何工作的，有必要先介绍一下mod_perl.c不包含的一些相关函数。

Perl解释器的使用贯穿于整个mod_perl模块，验证了方法之后将脚本传送到解释器，并对脚本执行一系列函数处理操作。其中一些脚本对Apache进行配置（包含<Perl>的），其他的则作为通常的CGI脚本处理，除非解释器已经被载入到Apache中了。mod_perl的结构如图9-1所示:

1. perl_cmds

38813: 定义了一些标志mod_perl进程的配置文件，该列表包含了Apache找到的与每个调用相匹配的目录的函数，这部分通常不与其他模块兼容，因此<Perl>和</Perl>目录调用函数一般是读取大部分配置文件并作为一perl脚本来执行两标志点之间的语句（命令行）。除大量的目录外，该表还包括在用Perl写成的Apache模块中用于模块结构中定义的其他处理过程的入口（hook）（从38875行到38918行）。

2. perl_handlers

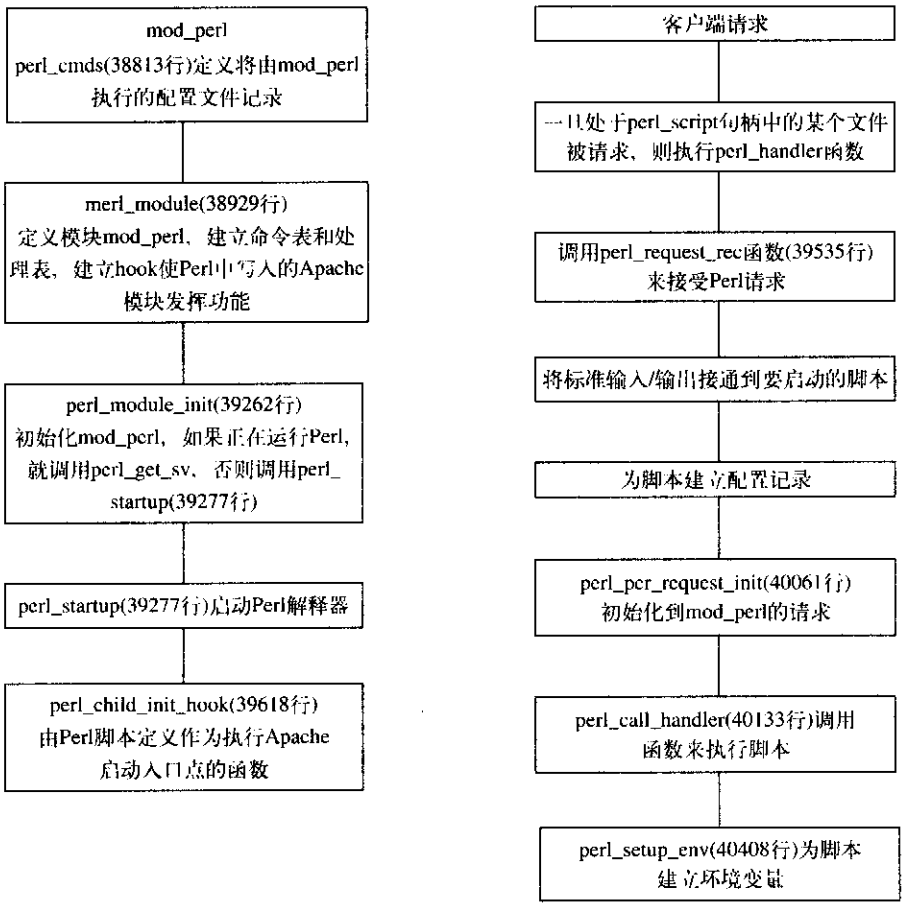

图9-1 强大通用的mod_perl模块将脚本预载入并在Apache执行

38923: 该函数定义了一个用于响应对分配了perl_script句柄请求的调用，它利用标准SetHandler指令来建立。perl_handler函数从39535行开始。

3. per_module

38929: 为初始化和创建配置结构，mod_perl用标准函数定义。命令表和处理表也被定义。其他阶段在Apache请求处理时在这些点使用hook(hook允许用perl写的Apache模块)到函数中。

4. seqno_cheek_max

38974: 检验请求句柄的最大数目，如果达到了规定的最大数目，函数终止执行。

38987: 重新获得MaxModPerlRequestsPerChild的值并与第38992行的当前segno值相比较，如果达到了最大值，则调用第38993行的child_terminate函数。

5. perl_shutdown

39002: 并闭Perl。

39020: 执行所有必须运行的结尾块。

39025: 调用perl_util_cleanup函数清除所用余留项目。

39043: 调用perl_destruct，然后调用perl_free函数关闭Perl解释器。

39050: 将perl_id_running的值设为0，表示解释器已经被并闭。

6. mp_fake_request_rec

39054: 建立一个请求记录的副本以备Perl脚本使用。

39057: 分配一个新的请求记录（r）。

39059: 根据传递到该函数的参数将r赋值，返回一个“写入”的请求记录。

7. perl_restart

39067: 建立一个“回叫信号”来定义一个句柄，以备必要时重启Perl使用。

8. perl_restart

39077: 调用perl_get_sv和gv_stashpv函数为重启Perl做准备。

39015: 如果还存留有损坏的句柄，则在重启前调用hv_clear来清除它们。

39110: 调用Perl_reload_inc函数重新启动Perl。

9. mod_perl_set_cwd

39120: 为Perl建立当前的工作目录。

39124: 利用getenv函数从PWD（显示工作目录）环境变量中获得当前工作目录名。

39130: 调用perl_eval_pv创建工作目录。

10. scriptname_val

39137: 获得Perl脚本名称。

39146: 调用perl_eval_pv 来比较文件名，再利用sv_setsv建立该文件名。

11. mod_perl_tie_scriptname

39154: 利用sv_unmagic和sv_magic调用连接Perl脚本名及其相关的附加信息。

12. mp_dso_unload

39173: 如果平台中使用了动态共享对象，则利用该函数卸载mod_perl代码。

39180: 测试模块名mod_perl.c，如果有效，则在modp结构中执行循环操作，并将dynamic_load_handle赋值为NULL，以表示该模块不再可用。

13. mp_server_notstarting

39194: 调用Apache_ServerStarting，如果返回值为FALSE，则将Server状态标识为非启动。

14. mp_check_version

39209: 测试已启动Perl的版本。

39217: 使用perl_get_sv进行版本比较，如果不匹配，则记录错误，mp_check_version给出返回值。

15. set_sigpipe

39253: 如果定义了HAS_MMN_136，则set_sigpipe函数调用perl_require_module，在Apachet和Perl处理器之间建立信号管道。

16. perl_module_init

39262: 初始化mod_perl，如同第38931行定义的。

39622: 如果已经运行Perl，则利用Apache::Server::AddPerlVersion方法调用perl_get_sv，然后调用ap_add_version_component来更新由Apache记录的Perl版本。

39274: 如果没有运行Perl，则调用第39277行的per_startup。

17. perl_startup

39277: 启动Perl解释器。

39313: 如果已经运行了Perl并执行了启动进程，则调用第39209行的mp_check_version来验证已记录的正确Perl版本信息。

39336: 如果需要重启Perl，则先调用Perl_restart_handler，然后调用39339行的Perl_restart。

39348: 准备Perl的启动参数，使之形如来自命令行。这些参数包括-T和-w参数。

39360: 当事件发生时通过MP_TRACE_g发送信息。

39367: 调用PERL_SYS_INIT，然后调用perl_init_i18nl0n来启动Perl解释器。

39386: 调用perl_parse建立刚启动的Perl解释器的状态。

39493: 循环加载PerlModule指令中列出的所有Perl模块，以便服务器初始化后运行的脚本能够使用。

18. perl_sent_header

39522: 如果MP_SENTHDR_on值允许，则调用MP_SENDHDR，使用mod_perl 向客户方发送头信息。

19. perl_handler

39535: 处理为perl_script句柄定义的请求（使用SetHandler指令）。

39550: 调用MP_SENDHDR_on，使Perl脚本处理的请求能发送头信息。如果定义了MP_SENDHDR，头信息将在39554行MP_SENTHDR_off处置为off。

39556: 调用perl_request_rec（开始于40388行），取得Perl请求，然后，它将调用mp_request_rec。

39581: 连接perl_stdout2client和perl_stdin2client启动的Perl脚本输入输出管理。

39584: 如果没有得到配置记录，将调用perl_create_request_config，然后是set_module_config来建立请求所需的配置记录，并将其置为脚本可访问。

39601: 处理后，在39543行获得独占锁的变量将被释放。

20. perl_child_exit_cleanup

39612: 如果为mod_perl定义了一个CHILD_INIT过程，那么这将作为拥有Apache服务器入口参数的结构。同时，一个Apache用来清理child_init阶段Perl脚本所使用的内存的cleanup函数也将被定义。

21. PERL_CHILD_INIT_HOOK

39618: 定义一个Perl脚本用做入口点的函数，它将在Apache启动时的child_init阶段执行。

39630: 调用mp_fake_request来设置这一函数所使用的伪装请求记录。

39633: 这一入口由mod_perl_init_ids初始化，然后使用一个Perl回调操作进入Apache运行阶段。

22. PERL_CHILD_EXIT_HOOK

39638: 如果PERL_CHILD_EXIT定义为在Apache处理阶段使用Perl脚本，这个函数将使用mp_fake_request_rec设置一个伪装的Perl请求，并注册一个callback以便适当时候正确的Perl脚本能得到执行。

39648: 当Apache服务器退出时，Perl通过调用perl_shutdown退出。

23. PERL_POST_READ_REQUEST_HOOK

39652: 如果定义了PERL_POST_READ_REQUEST，39664行将通过PERL_CALLBACK设置一个回叫，以允许Perl脚本能在post_read_request阶段运行。

24. PERL_TRANS_HOOK

39673: 如果定义了PERL_TRANS，在此将创建一个hook，以使Perl脚本能在Apache请求处理的filename_translation阶段运行。

25. PERL_HEADER_PARSER_HOOK

39684: 如果定义了PERL_HEADER_PARSER，在39690行将设置一个回叫，以使Apache能在每次请求到达头信息处理阶段运行Perl。

26. PERL_AUTHEN_HOOK

39699: 如果定义了PERL_AUTHEN，PERL_AUTHEN_HOOK函数将建立一个回叫，以使Perl脚本能用于请求的权限验证。

27. PERL_TYPE_HOOK

39732: 如果编译期间定义了PERL_TYPE，这个函数将为Perl建立一个回叫，使得Apache在每次请求到达MIME文件分类阶段时能运行Perl脚本。

28. PERL_FIXUP_HOOK

39743: 如果定义了PERL_FIXUP，这个函数用来定义一个能在fixup阶段调用Perl脚本的hook，使得请求能在处理完以前做最后的修改。

29. PERL_LOG_HOOK

39754: 定义一个在Apache请求处理阶段使用Perl记帐函数的入口函数。这一函数仅在PERL_LOG定义之后才能包含在内。

39765: 基于PERL_STACKED_HANDLERS值定义这一模块的请求句柄。

30. per_request_cleanup

39774: 检查当前配置中的每条记录，如果存在，则调用hv_clear清除这些记录，然后调用SvREFCNT_dec。

31. mod_perl_end_cleanup

39788: 设置一个回叫，使得mod_perl退出时可以使用一个清理句柄。per_request_cleanup和perl_clear_env也将被调用来清除请求和为Perl脚本建立的环境信息。

32. mod_perl_cleanup_handler

39852: 循环执行39862行，调用perl_call_handler清除所有Perl进程。

39868: 调用av_clear来完成清理。

39872: 开始清理（39859行）释放所有获得的独占锁。

33. perl_handler_ismethod

39876: 处理mod_perl指令指定的Perl方法。

39887: 用gv_fetchmethod取得要使用的方法。

39892: 将strnEQ的返回值设为is_method。

34. mod_perl_register_cleanup

39903: 注册结束Perl进程所用的清理函数，以mod_perl_cleanup_handler函数的名义在39909行调用register_cleanup。

35. mod_perl_push_handlers

39922: 如果服务器使用了压栈的Perl句柄，则指向栈本身的句柄也应压栈。
39929: 如果传给此函数的句柄有效，hv_fetch将用来取得句柄栈的信息。
39939: 将句柄值设为newAV所给出的值，然后调用SvPV设置栈中的句柄。
36. perl_run_stacked_handlers
39975: 在一个Perl脚本中运行多个句柄。
39984: 以stacked_handlers为参数调用hv_exists，检查栈内是否存在多个句柄。如果存在，则将hv_fetch所要的句柄取出并存于svp中。
40015: 如果所要的句柄不在栈中，则输出一条错误信息，然后用sv_dump退出。
40024: 循环检查AvFILL返回的句柄，调用40034行的SvTRUE测试其合法性，然后调用40041行的perl_call_handler使用该句柄。
37. perl_per_request_init
40061: 初始化mod_perl的请求。
40069: 将子进程头设为On，以包含PERL_SEND_HEADER。
40086: 如果没有设好配置信息，则用perl_create_request_config创建。
40092: 调用40096行的perl_setup_env和mod_perl_dir_env设置请求环境。
40102: 用mod_perl_pass_env向请求传送环境信息。
40104: 用mod_perl_tie_scriptname处理运行的脚本名。
38. perl_call_handler
40133: 调用句柄来处理Perl脚本。
40146: 用get_module_config取得模块配置信息。
40151: 调用perl_get_cv决定分发句柄。
40165: 如果请求是基于每个目录的配置，则调用perl_per_request_init来初始化。
39. perl_request_rec
40388: 调用mp_request_rec获得Perl脚本的请求记录。返回所请求的记录信息。
40. perl_bless_request_rec
40398: 调用 sv_setref_pv检查请求记录。同时调用了MP_TRACE_g记录这一事件。
41. perl_setup_env
40408: 设置Perl脚本所用的环境变量。
40411: 调用perl_cgi_env_init将arr设为环境信息。
40414: 循环检查arr中所有元素，调用mp_setenv创建每个脚本环境。

9.2 mod_example模块

mod_example是一个样本模块，用来帮助开发者理解如何构造模块，以及所有的Apache模块使用了哪些构件、hook和控制流。作为样例，尽管这一模块除了打印出Apache内部和请求处理的各个操作的信息以外什么都不做，其源码中的注解比其他模块都要多。

这个模块只能在学习模块时才编译到Apache中，在其他任何时候使用这一模块，即使没有任何请求用到它，都会浪费内存和处理时间。

为了使用mod_example，必须在某个目录上下文中加入一个句柄。

```
< Directory />
```

```
AddHandler example_handler html
</Directory>
```

当任何指向该目录上下文的请求到达时，mod_example中的代码都将获得使用，并显示怎样使用的模块函数。

9.2.1 模块结构

mod_example代码包含了所有可能出现的Apache处理阶段的函数，另外还有一个句柄函数（见37665行example_handler）来处理请求。trace_add函数可为每个阶段的函数添加请求中的注释。对于样例句柄，当请求到达时，这些注释将和其他状态信息一起输出。

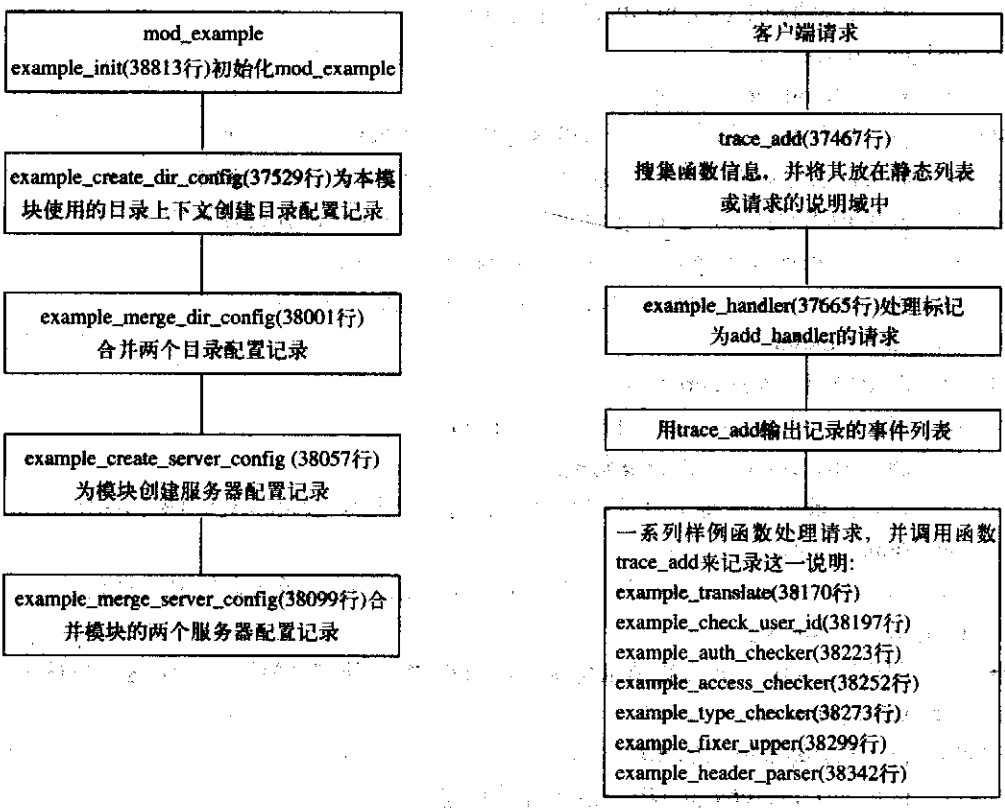

图9-2　mod_example是一个描述模块如何构造的测试模块

记住：应该先阅读源代码中的注解，加上下面给出的信息，然后再去发出一个新模块。

9.2.2 定制

当研究设计中的新模块应完成什么功能时，学会配置mod_example是很有用的。在mod_example的适当函数中添加代码——这要看所加功能适合在什么阶段工作——使开发者能清楚这段代码是如何与其他模块或过程交互的。

1. excfg

37187：为mod_example配置记录定义数据结构。这一结果并不寻常，因为它用一个决定谁是活跃的标志（cmode）在一个结构中既显示了目录又显示了服务器配置。

2. Additional Comments

37250: 定义了如何用类似FLAG和TAKE1这样的代码定义指令参数的形式。这些代码使得Apache可以分解指令参数，并将其传入模块中处理这些参数数据的函数中。这些注释一直到37373行。

3. our_dconfig

37387: 用ap_get_module_config API调用返回当前目录上下文的目录配置记录。

4. our_sconfig

37399: 用ap_get_module_config API调用返回当前服务器上下文的服务器配置记录。

5. our_rconfig

39410: 用ap_get_module_config API调用返回单个请求的配置记录。

6. setup_module_cells

37422: 在37429行ap_make_sub_pool函数定义的内存池中设置样例表。16个记录的表的生成在37437行。

7. trace_add

37467: 从所有阶段的mod_example描述函数中搜集信息。跟踪信息在静态表中，或者在当前请求的notes域中。

37482: 调用setup_module_cells来分配一个静态空间存放没有附接到任何请求（notes域中）的跟踪信息。

37487: 如果请求正在处理，它的内存池将用来把请求notes中的mod_example跟踪信息拉到trace_copy域中。

37508: 如果请求不在处理中，则创建新的子内存池，把trace的值指向它。如果子内存池已经存在，则在37518行释放它，以防止内存泄漏。

37521: 将trace拷到trace_copy。

37541: 如果请求不在处理中（r == NULL），37544行的代码将决定正在传入的说明是否已经是不属于某个请求的跟踪信息的静态表的一部分。

37544: 定义key为受检的说明，然后在static_calls_made表中用ap_table_get查找它。如果传入trace_add的说明已在表中，则trace_add立即返回。

37552: 没有在静态表中找到notes，所以ap_table_get（37558行）将关键字（key）放在static_calls_made表中。

37562: 定义addon为用来显示模块内的跟踪说明和位置的HTML文本。这种文本将在example_handler中使用。

37571: 决定请求是否再处理一次，如果是，则addon中的跟踪文本将放在请求的notes域中。

8. cmd_example

37621: 显示怎样处理指令。这个函数是为38371行的样例指令定义的。

37631: 使用trace_add函数记录事件（指令的处理）。

9. example_handler

37665: 处理被AddHandler标记的请求，此时正被example_handler处理，它是mod_example的样例句柄。这个函数将输出trace_add记录的事件信息给用户。

37671: 记录调用trace_add所处理的事件，并以函数名作为字符串。因此，事件列表将包含这个句柄。

37688: 在向客户发送头信息的准备工作中，将content-type设为text/html。

37689: 开始计时，以使传输过程中发生问题时，该进程能被Apache父进程用ap_soft_timeout终止。

37690: 用ap_send_http_header向客户方发送头信息。

37695: 如果HTTP请求仅仅是头信息，计时终止，example_handler返回，否则，这一过程将继续到37704行。

37740: 对每一个HTML标签开始用ap_rputs调用向客户方一段段地发送HTML文档。

37785: 用ap_kill_timeout取消计时，因为数据已成功地发到客户方。

10. example_init

37846: 初始化mod_example。这个函数在这一阶段的定义在38430行（模块定义中）。调用init阶段时配置记录还不可用，因此37856行的setup_module_cells调用将建立一个静态表。

37863: 用ap_pstrcat组合一个字符串，并加到跟踪表中，在说明中存放结果。

37865: 调用trace_add记录setup_module_cells设置的静态表中的说明字符串。

11. example_child_init

37883: 记录对子服务器进程初始化阶段的调用。

37893: 调用setup_module_cells准备容纳跟踪说明的静态表。

37900: 将字符串编入note。字符串中包含有函数名。

37902: 用trace_add将函数的note加到事件的静态表中。

12. example_create_dir_config

37952: 为模块所使用的一个目录上下文创建目录的配置记录（上下文由<Directory>容器定义，其中使用了某一个模块指令）。

37963: 用ap_pcalloc为配置变量分配空间。

37969: 为配置记录中的每个可用域设置缺省值。

37976: 用带函数名的trace_add记录这个函数被调用。

13. example_merge_dir_config

38001: 合并mod_example的两个目录配置记录。

38006: 为新的配置记录变量分配内存，以容纳合并后的记录信息。

38017: 从配置记录中拷贝一些域到合并后的记录。其他域是复合的（即参与合并的两个记录配置进行“或”运算）。

38045: 调用trace_add记录对本函数的调用。函数名被包含在跟踪记录的字符串中。

14. example_create_server_config

38057: 为mod_example创建服务器配置记录。

38068: 用ap_pcalloc分配记录的空间，然后对记录中各域进行初始化。

38077: 用trace_add记录本函数的调用，函数名作为字符串保留。

15. example_merge_server_config

38099: 合并两个mod_example的服务器配置记录（缺省服务器和一个虚拟服务器）。

38104: 为两个合并的数据集合的结果配置记录分配内存。数据合并，example_merge_server_config决定了缺省服务器中哪些应复合，哪些项应被虚拟服务器的项覆盖。

38130: trace_add记录下对example_merge_server_config调用。

16. example_post_read_request

38144: 在读入一个请求之后，但在调用其他模块之前处理该请求。除了用trace_add记录了到达该函数的事件以外，该函数对这个模块没起任何作用。通过返回DECLINED，其他在此阶段有函数定义的模块会陆续被调用，直至其中一个返回OK。

17. exmaple_translate_handler

38170: 处理作为请求的一部分收到的URI，将它转化成能用来从服务器上获取数据的文件名。在本模块中，这个函数仅用trace_add记录该函数名达到了。返回值是DECLINED，允许Apache调用其他模块中的转换阶段的函数来准备URI到文件名的转换。

18. example_check_user_id

38197: 检查请求中带有的授权信息，以判断发出请求的用户是否有权取得所请求的文件。本函数实际上只调用trace_add来说明它被调用了。当在这个阶段具有这种函数的模块返回OK时，就不再调用其他模块。假定用户都是授权的。

19. example_auth_checker

38223: 检查被请求的文件是否需要授权，函数仅用trace_add记录该函数是否被调用。当本阶段具有该函数的模块返回OK时，就不再调用其他的。如果所有模块中的本函数都返回DECLINED，请求将导致返回一个错误给客户方。

20. example_access_checker

38252: 检查与模块相关的请求对象的限制。mod_example中的本函数用trace_add登记了自己的函数名，然后返回DECLINED，以允许本阶段的其他函数有机会处理请求。

21. example_type_checker

38273: 处理文档类型和与请求的文件相关的信息，如编码和语言，并将其放在请求的头信息中。本函数登记自身，然后返回DECLINED，因此本阶段的其他函数能处理该请求。（mod_mime是完成本动作的函数例子。）

22. example_fixer_upper

38299: 在调用适当的处理者处理请求，向客户方返回一个文件以前，处理任何剩下的头信息。这个函数仅仅用trace_add登记一下，以表明执行了该函数。

23. example_logger

38322: 登记处理的请求。所有加载了的模块中的记帐函数都要执行一遍，返回OK也不中断记帐函数链的执行。这个函数仅用trace_add记录了它被访问。

24. example_header_parser

38342: 分析当前请求的头信息，在头信息送往客户方以前（大多数情况下），执行任何附加的必要操作。本函数在此仅用trace_add记录它被调用了。

25. example_cmds

38367: 定义影响本模块的指令。对每个指令，example_cmds结构中都提供了函数名。当相应的指令位于配置文件中时，命名的函数则被调用。example_cmds也定义了

（用类似NO_ARGS代码）如何格式化指令的参数，以允许Apache准备这些参数，并将其传给处理这个指令的命名函数。

38370: mod_example有一条指令（仅用于描述）：Example。它由cmd_example处理。

26. example_handlers

38403: 定义结构来显示本模块支持哪些数据类型（正如AddHandler和SetHandler指令所用的），以便在需要那个句柄的文件被请求时可以调用该函数。example_handler（37665行）是为example_handler类型定义的。

27. example_module

38427: 在结构中定义本模块，该结构显示了在Apache初始化和请求处理的所有可能的阶段需要调用的所有函数。在mod_example中，每个阶段都有用，而其他模块仅用其中几项，其余项为NULL。

9.3 mod_mmap_static模块

mod_mmap_static是一个试验性的模块，它将Web服务器上的某个文件在服务器启动和配置时映射到服务器的一段内存区中。mmap Unix调用可以将一个磁盘文件映射到内存，这样以后对该文件的操作效率的提高就成为了可能。并不是所有文件都可以用来做内存映射。如果映射到内存的文件发生了改变，Apache必须重启来加载该文件。

mod_mmap_static仅有一条指令（mmapfile），其中包含一个文件列表，列表中的文件将在服务器初始化时映射到内存。因为本模块还有待检验，所以它没有编译到缺省的Apache服务器中。

9.3.1 模块结构

在服务器初始化时，mod_mmap_static按照mmapfile指令所描述的，将每个文件都加载到内存。为了快速访问，这些文件排了序。如图9-3所示，mmap_static_handler函数处理映射文件的请求，检查可用文件组，并用Apache API调用ap_send_mmap将文件从系统内存传到客户方。

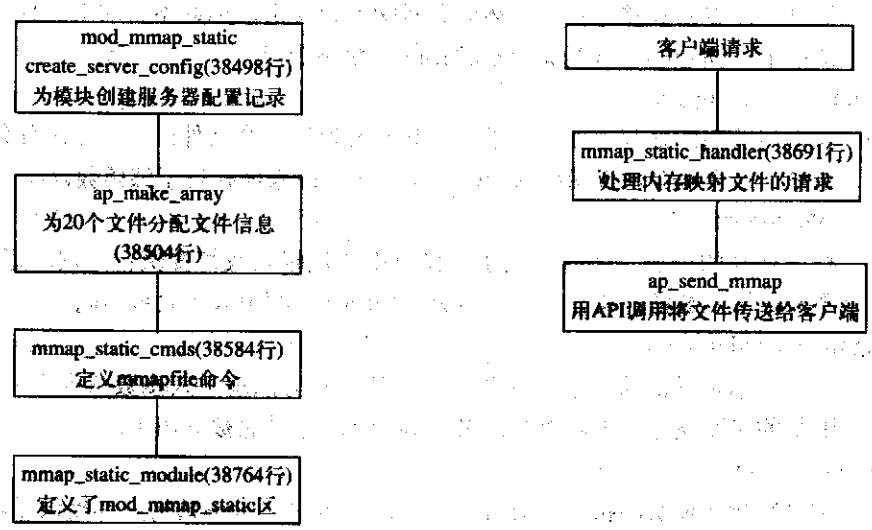

图9-3 mod_mmap_static可用来将Web服务器上的文件映射到内存

9.3.2 定制

因为一旦映射的文件发生改变，Apache就必须重启，所以有必要开发一个功能向模块发信号，指示它重新加载改变的文件。这是一个模块内的信号，因此没必要重启整个服务器。

还有一些允许内存中的文件按照服务器的特性改变自身的功能有可能需要开发。这种情况下，内存映射文件的性能提高程度就需要计算一下了。

1. create_server_config

38498: 为mod_mmap_static创建服务器配置文件。create_server_config用38504行的ap_make_array为20个文件分配了文件信息数组，然后将inode_sorted初始化成NULL。

2. cleanup_mmap

38509: 为mod_mmap_static清除内存使用。这个函数将传给Apache清理函数，使它能决定在服务器关闭时如何清理被这个模块占用的内存。

38517: 循环处理mod_mmap_static使用的所有文件。对每个文件都调用munmap将文件从系统内存中清除掉（mmap的相反调用，见38555行）。

3. mmapfile

38524: 处理38575行定义的mmapfile指令。因为参数使用的是ITERATE格式（38587行），函数要在参数表中对每一项调用一次。因此代码可以假定单个文件名是作为参数传递的。

38533: 检查作为mmapfile的参数的文件状态。如果stat调用返回错误，这个错误将被记录，mmapfile退出，文件不能访问。

38539: 校验作为内存映射请求的文件是规则文件（否则无法使用）。如果不是规则文件，mmapfile记录一个错误信息，然后退出。

38547: 尝试用open系统调用打开一个文件。如果调用不成功，mmapfile记录一条错误信息，然后退出。

38555: 用mmap系统调用将刚才打开的文件映射到内存。

38557: 如果mmap系统调用失败，mmapfile关闭该文件，记录错误，然后退出。

38568: 如果mmap调用成功，文件关闭，文件名放在服务器配置记录的files数组中。

38575: 如果是第一次将文件加入内存映射列表中，cleanup_mmap内存清理函数将用ap_register_cleanup在Apache中注册。

4. mmap_static_cmds

38584: 定义mod_mmap_static用的指令。mmapfile指令由38524行的mmapfile函数处理。

5. file_compare

38596: 比较两个文件，返回一个值，说明哪个文件应排序更高。在mmap_init阶段函数名传给了qsort，准备内存映射文件列表。通过对文件排序，bsearch系统调用能更快地定位所需的文件。然而，一般用inode在mmap_static_handler中找到正确的文件。

6. inode_compare

38604: 比较两个索引结点，并返回一个值，说明哪个应排序更高。在mmap_init阶段，函数名被传入qrort以准备与内存映射文件相应的索引结点列表。通过对索引结点

排序，能用bsearch快速定位所要找的文件。

7. mmap_init

38617: 在Apache服务器启动时初始化mod_mmap_static。

38629: 为需要做内存映射的文件名列表设置变量。

38631: 对文件列表排序以方便以后访问。将file_compare函数传入qsort作为文件数组元素的比较函数。

38635: 创建内存映射文件的inode索引以加速访问。

38637: 循环处理文件列表中的所有项，将所有文件的索引结点放在inodes数组中。

38640: 用qsort为内存映射文件的inodes数组排序，inode_compare作为排序的函数传给qsort作为参数。

38644: 对所有使用一个Apache副本的服务器定义（虚拟主机）采用相同的配置。

8. mmap_static_xlat

38655: 转换文件名以访问内存映射文件。这一函数定义在38755行的filename_translation阶段。

38622: 提取模块配置记录。

38667: 如果内存映射文件列表是空的，则本模块不返回任何文件，因此文件名不转换，mmap_static_xlat立即返回。

38672: 在其他模块中使用translate_handler函数，在将文件转换为内存映射文件以前将其准备好。

38673: 如果其他模块不能转换文件名，或者文件目前不存在，mmap_static_xlat 处理直接返回。

38677: 用bsearch调用在mod_mmap_static配置的files数组中匹配被请求的文件名。

38680: 如果匹配不成功，mmap_static_xlat返回DECLINED，说明该文件没有映射到内存。

38686: 将请求中的finfo域设为匹配到的内存映射文件的finfo域值，以保留几个系统调用留待将来使用。

9. mmap_static_handler

38691: 处理和返回由mmapfile标记为内存映射的文件。这个函数在38760行的句柄表中定义为处理阶段函数。

38701: 测试当前请求方法。如果不是GET，mmap_static_handler立即返回。

38704: 如果当前请求的finfo.st_mode域表明请求文件不存在，mmap_static_handler立即返回。这个域很可能是从map_static_xlat的模块函数匹配到的项中复制来的。这个测试证实文件已经存在，可以继续处理。

38706: 取出服务器配置。

38711: 用bsearch在内存映射文件inode表中查找请求的文件。这比查找文件名要快。索引结点是请求记录可用的文件信息的一部分。

38715: 如果匹配失败，mmap_static_handler直接返回。

38727: 调用ap_discard_request_body存下客户请求中任何多余信息。如果出错则立即退出。

38730: 更新即将返回的文件的最后修改时间。

38740: 将请求信息发往客户方。

38742: 如果请求不只要求头信息，mmap_static_handler则检查rangestatus。如果合法，则用带mm、finfo.st_size域的ap_send_mmap将文件从内存发往客户。

38747: 如果rangestatus无效，mmap_static_handler将在38749行循环调用ap_each_byterange，使ap_send_mmap把文件分成更小的块发往客户。

10. mmap_static_handlers

38758: 定义标记为内存映射的文件的处理函数。

11. mmap_static_module

38764: 定义mod_mmap_static的区。一个初始化函数（mmap_init）准备好内存映射要用的文件列表，同时也准备好各服务器的配置。提供一个命令和句柄表来指定处理mmapfile指令内存映射句柄类型的函数。最后，定义mmap_static_xlat_函数将对内存映射文件的请求转换为服务器初始阶段内存中的信息的匹配项。

9.4 mod_userdir模块

mod_userdir用于将指定的URL格式转换为Apache服务器的有效路径和文件名，以便返回给客户方。URL格式中用“~”访问各用户的Web目录，例如，如果UserDir指令如下所写：

```
UserDir public_html
```

则URL www .server.com/~jsmith/resume.html将映射为服务器上的/home/jsmith/public_html/resume.html，这由mod_userdir完成。

因为UserDir的文件映射将访问Web文档根目录以外的文件，所以使用时要非常谨慎。UserDir指令可包括全路径，也可以仅包含每个用户主目录下的目录名。

UserDir可带Enabled 参数，后跟一个用户名列表；也可相应地带Disabled参数，后跟一个用户名列表；或者什么也不带。这些参数使客户可以或不可以请求某个用户的Web目录。

9.4.1 模块结构

mod_userdir含有各目录的配置记录（由create_userdir_config创建），其中包括了目录名和enabled/disabled状态。UserDir指令由set_user_dir函数（始于36606行）处理。请求由translate_userdir函数单独处理（始于36681行），见图9-4。

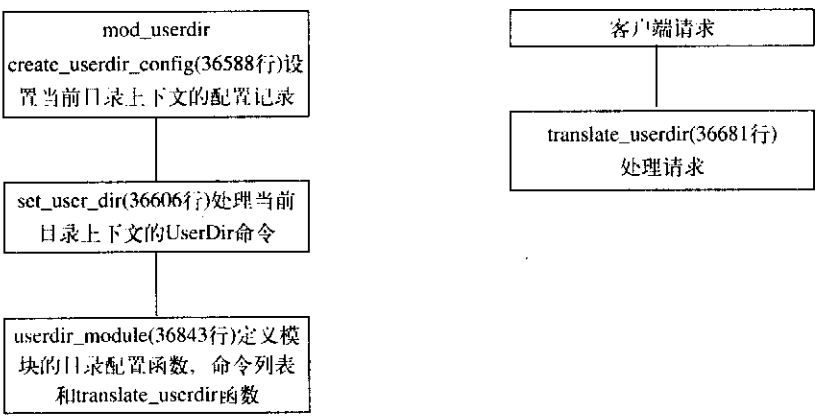

图9-4 mod_userdir使Apache能处理各个用户的Web目录

9.4.2 定制

设好UserDir的Enable、Disable参数，以便检查可选的用户列表，比如通过数据库查询或者某种类型的授权请求。

1. userdir_config

36573: 定义mod_userdir的配置结构，包括转换请求的各用户目录名和允许或不允许使用该特性的用户名数组。

2. create_userdir_config

36588: 分配userdir_config变量并初始化，然后创建当前目录上下文的配置记录。一个四个元素的表初始化为enabled_users和disabled_users数组。

3. set_user_dir

36606: 处理当前用户上下文的UserDir指令。

36610: 取得当前上下文的配置记录。

36618: 取出指令参数的第一个词。因为UserDir指令有几种形式（如目录名或使能表），RAW_ARGS的值用于定义它。因此，Apache将整个参数表作为一个参数传进去。

36624: 如果第一个词是disabled，set_user_dir开始查找用户名。如果没找到，它将globally_disabled（对所有用户有效）设为1。否则，disabled_users域设为36635行的usertable值。

36637: 如果指令的第一个词是enabled，则必须包括一个用户名列表。如果没有，set_user_dir将在36646行返回错误。否则，enabled_users数组将设为36649行的usertable的值。

36657: 如果第一个词既不是enabled，也不是disabled，则假定它是一个目录名。这一名称不做检查即复制到userdir域。

36665: 决定了用户列表应放入哪个表中，一个while循环用ap_getword_conf取得每个用户，并用ap_table_setn将它放在合适的表中。

4. userdir_cmds

36672: 定义 mod_userdir使用的指令：UserDir。set_user_dir函数被定义来处理这个指令。

5. translate_userdir

36681: 将请求的文件名按照当前目录上下文中的UserDir指令信息转换为映射到某个用户主目录下的文件名。

36684: 取出当前目录上下文。

36698: 检查请求是否可被模块mod_userdir处理。在配置记录中的userdir域必须定义，而且所请求的表项必须是以“/~”开头的。如果以上两条有一条不满足，则函数translate_userdir立刻结束模块的运行。

36707: 将所请求文件的下一个部分返回到域w中。

36726: 检查所请求的表项是否为当前目录或父目录(一或两个点表示)。如果是其中一个的话，则函数translate_userdir立刻返回。

36734: 比较用户请求中的用户名是否与数组disabled_users中某项相同。如果在数组中查找到该用户名，则函数translate_userdir立刻返回。

36741: 检查是否设置了全局变量globally_disabled。如果是，则函数translate_userdir检查请求中的用户名是否在数组enabled_users中。(不属于被全局禁止命令的设置。)如果用户不能通过全局禁止命令(global disable command)的检查，则函数translate_userdir立刻返回。

36753: 循环检测结构userdir中变量，以建立一修改的文件名及路径。该文件名和路径中包含了命令UserDir中所提供的目录名。

36764: 如果编译标志指示系统平台支持驱动器名的使用，则程序执行一些额外的检查。

36779: 拷贝新装配的文件路径名，并将其作为数组headers_out(在Location中)的一部分，提供给用户使用。然后函数translate_userdir返回，返回值为REDIRECT。

36785: 根据命令UserDir中目录名的设置检查其他各种情况，包括asterisk(*)。在新装配的路径用于重定向用户请求之前，系统将文件名设置为另一不同的值。

36801: 如果系统运行在Window系统上，则函数translate_userdir不再进行下一步的处理。因为在Windows系统平台上，模块mod_userdir不处理根目录。

36827: 使用系统调用stat检测新生成的文件名。

36835: 若stat工作正常(该文件名有效)，则将请求的finfo域内容设置为stat系统调用所返回的结果。

6. userdir_module:

36843: 定义模块mod_userdir，包括目录配置函数(create_userdir_config)模块命令列表，以及函数translate_userdir。在进行文件名转换(filename_translate)阶段，该函数负责将用户目录下的请求转化为一合适的文件名。

9.5 mod_so模块

该模块用来加载一些额处的模块及其他目标代码，而Apache Web服务器不需进行重新编译。在该模块有效之前，在系统中加上新的模块需要更新系统配置文件并且重新编译。

使用mod_so命令，可在处理用户请求之前将共享目录代码格式的模块加载到Apache系统中，同时解析系统配置文件。命令LoadModule用来加载一目标模块，其使用方法如下:

```
<VirtualHost www.wyz.com>
LoadModule mod_speling mod_speling.so
.....
</VirtualHost>
```

同样的，也可将一非模块形式的目标文件加载到Apache的地址空间。此时使用命令LoadFile，其功能是提供一可访问库函数的模块。该模块不属于模块，甚至也不是Apache的一部分。命令Load File的使用方法如下:

```
<VirtualHost www.xyz.com>
LoadFile libtoolkit.so
.....
</VirtualHost>
```

命令LoadModule用来在系统运行时加载一些动态共享目标模块(DSO Dynamic shared Object)。它与Apache源代码配置文件中的命令AddModule不同。命令AddModule的功能是在编译时加载一些附加模块。该命令不被模块mod_so使用。

9.5.1 模块结构

模块mod_so通过调用Apache API函数来加载模块或者其他共享文件。它自身的结构很简洁。函数so_sconf_create用来创建一个服务器配置记录(从第34601行开始)。函数laod_module和函数load_file分别用来处理命令LoadModule和命令LoadFile以及解析配置文件。此外，不再包含任何其他函数，因为模块本身在配置文件处理完毕之后不再做任何事情。

模块中提供了两个不同版本的load_module和load_file函数。当然在编译时只能加载其中的一套，这取决于编译期间有关系统操作平台的设置。实际上辅助版本的函数的功能就是在当前系统平台不支持动态共享目标代码时(DSO)将错误记录入日志。

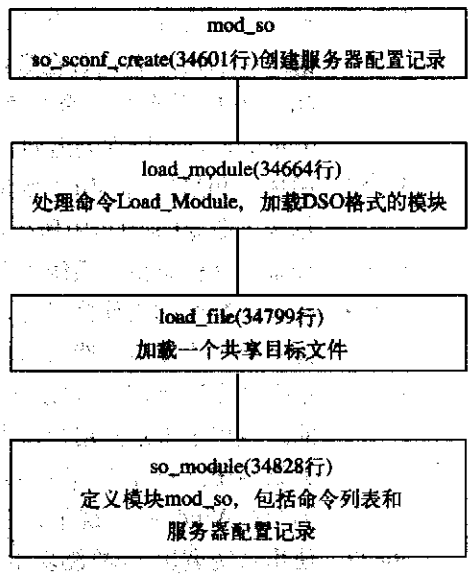

图9-5 模块mod_so在Apache系统启动时加载附加模块或者其他目标代码

9.5.2 定制

在使用命令LoadFile将特殊类型的共享目标加载到模块时，函数load_ file试图在结果代码的头信息中处理和报告处理结果。

在动态加载共享目标时，函数load_module及load_file所调用的Apache API调用可针对该共享目标进行优化或者定制。

1. moduleinfo

34592: 在服务器配置记录中定义模块mod_so所用的数据结构。这个数据结构保持一份已加载模块的名字列表。在执行命令LoadModule时，程序对比命令参数和已加载的模块名列表来检查已经是否加载了该模块。

2. so_server_conf

34597: 定义服务器记录数据结构。

3. so_sconf_create

34601: 为模块mod_so创建一服务器配置记录。使用变量DYNAMIC_MODULE_LIMIT作为系统支持的最多模块数来初始化表结构loaded_modules。

34611: 如果程序未定义参数NO_DLOPEN，则函数so_sconf_create在处理配置文件时，通过调用函数ap_os_dso_init来为加载动态共享目标(DSO)做好准备(使用函数load_module，34697行)。如果该标志未定义的话，则函数load_module仅将错误记录入日志。因此，程序将不再执行函数ap_os_dso_init。

4. unload_module

34625: 从Apache系统中卸载一共享目标。这个函数不是在模块mod_so中进行调用，而是通过使用Apache API函数进行调用。

34629: 检查一确定模块是否已被加载至系统。如果未被加载，则函数unload_module不需要任何操作而直接返回。

34634: 调用函数ap_remove_loaded_module从Apache核心数据结构中删除指向该模块的指针。

34637: 调用ap_os_dso_unload函数卸载目标模块代码。

34641: 清除数据结构modi中引用被卸载模块的变量。因为模块已被卸载，不能再继续使用。

5. unload_file

34653: 将使用命令LoadFile加载的动态共享目标文件卸载。此函数不在模块mod_so中被调用，而是在系统关闭时期，从Apache的核心代码中进行调用。该模块通过调用函数ap_os_dso_unload将DSO文件从Apache的地址空间上删除。此后，关于该文件的任何信息都不再可用。在卸载之后若发生错误，则系统不知道如何记录，也不再做任何检查工作。

6. load_module

34664: 将服务器配置文件中命令LoadModule指定的DSO格式模块加载进系统。本函数用来处理命令LoadModule，它在34819行定义。

34681: 为当前服务器上下文返回配置记录。

34685: 将命令LoadModule所要载的模块名同当前已被加载的模块名列表进行比较。如果所要加载的模块已被加载进系统，则函数load_module不再进行任何操作，在34689行直接返回。

34691: 调用函数ap_push_array，为新加载的模块准备存储空间，然后将其中的name字段设置为命令中参数所提供的模块名。

34697: 调用函数ap_os_dso_load，尝试将模块代码文件加载进系统。若在加载时发生错误，则函数load_module将其错误记入日志后返回。

34705: 记录成功加载的模块。尽管加载成功，该信息仍然被记录到错误日志中，因为将其记录在访问日志中会造成检查该日志的统计工具的混乱。

34715: 通过调用函数ap_os_dso_sym返回模块的结构信息，以便对模块进行初始化工作。如果在此期间发生了错误，则函数load_module记录错误并返回。此时，模块代码虽然已被加载进内存，但仍为不可用状态。

34730: 检验函数ap_os_dso_sym返回的结构信息，该信息表明加载的代码已成为Apache的一个模块。若不是的话，则函数load_module将错误记录入日志，然后返回。

34742: 检查完毕之后，函数load_module通过调用函数ap_add_loaded_module返回模块句柄，以便Apache将新加载的模块名追加到已加载模块名列表中去。

34750: 注册清除程序，以便系统关闭时使用。该函数在unload_module中定义，并调用了函数ap_register_cleanup（从34625行开始）。这个函数是在Apache服务器关闭的时候被调用，而不是在模块mod_so中的任何地方。

34758: 调用函数ap_single_module_configure来初始化和配置新加载的模块，同时还要使用模块自己定义的配置函数。

7. load_file

34770: 将一共享目标文件加载到Apache服务器地址空间。

34776: 准备要加载的目标文件的文件名。

34778: 调用函数ap_os_dso_load，来决定是否可以加载该共享目标文件。如果不能被加载，则函数load_file将错误记录错误日志后返回。

34787: 将已被加载的共享目标文件名记录进日志。

34791: 注册清除函数unload_file。该函数从34653行开始，它是在Apache服务器关闭时调用的。

34799: 若Apache的系统操作平台不支持所要加载的共享目标(根据NO_DLOPEN标志)，则使用本版本的load_file函数。它向STDERR发送一错误信息，表明在当前平台上，系统不支持LoadFile命令。

8. load_module

34807: 定义函数Load_module辅助版本。该版本的Load_module在系统不支持共享目标代码时使用。在运行时，这个函数向STDERR发送一错误信息，指明在当前平台上，系统不支持LoadModule命令。

9. so_cmds

34818: 定义模块mod_so所支持的命令(命令LoadFile和命令LoadModule)。同时又定义了一个函数，该函数用来处理在扫描配置文件时发现的命令。

10. so_module

34828: 定义模块mod_so，包括所支持的命令列表(命令处理的函数，34818行)，以及模块mod_so所使用变量的配置记录。模块mod_so中函数在处理用户请求时不再使用，因为所有模块mod_so的工作均在扫描配置文件期间完成。

9.6 mod_speling模块

模块mod_speling用来自动更正用户请求中文件名的拼写错误。若在相同目录下发现有与被请求文件名同名的文件，除去有一些小的差异，比如大小写区别，次序颠倒以及字符丢失等，则模块 mod_speling将请求重定向到正确的文件名位置。如果存在几种可能的话，则使用HTML文档格式将所有有效的文件名清单返回给客户以供其选择。

模块mod_speling不属于Apache系统的缺省配置，但可在编译期间加载该模块，也可在运行时通过命令LoadModule动态加载该模块。使用模块mod_speling会极大地增加Web服务器的处理负担，尤其是在Web服务器的文档目录中包含许多文件的情况下。

9.6.1 模块结构

如图9-6所示，模块mod_speling使用一简单命令(CheckSpelling)来激活服务器上下文中的

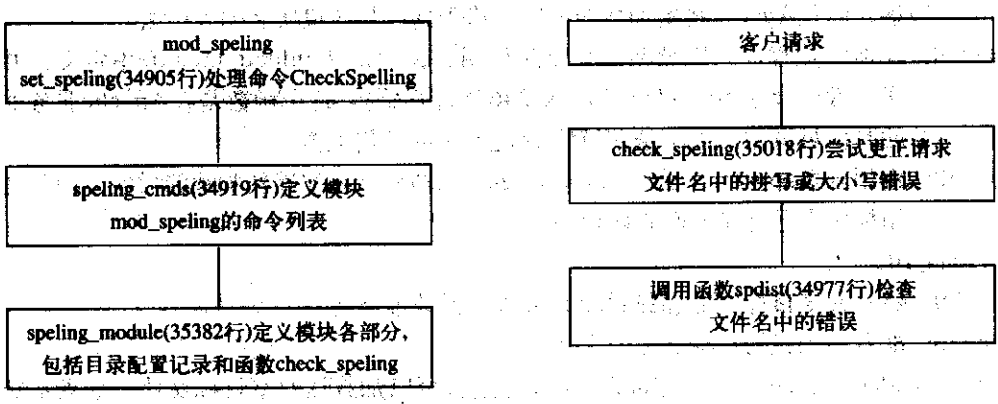

图9-6 模块mod_speling更正URL中的拼写错误

拼写检查。在激活状态下，使用一个简单函数(check_speling，从35018行开始)来尝试更正请求文件名出现的简单拼写错误。函数spdist(从34977行开始)用来检查文件名中的一些特殊拼写错误，它由函数check_speling 调用使用。

9.6.2 定制

由于模块mod_speling会加重Web服务器的处理负担，因此对于管理员和客户来说，可以对模块mod_speling进行改进以增强其功能，以下是一些建议。

- 另外定义一个命令来允许对一些特殊的拼写错误进行转化，比如很常见的Web服务器本身的版本拼写错误，以重定向错误的服务器名称。该命令的功能与Word处理器中的自动更正列表功能相类似。
- 可以是一个附加命令也可作为函数spdist的一部分，根据字典来跟踪和更正一些较为复杂的拼写错误(和字处理器中的功能相同)。如果设定了可能更正次数的阈值，则可向用户返回更正清单信息，以避免含义不明确。
- 更正文件扩展名错误功能。在模块mod_speling中当前版本不包括这种功能，用户也可以将其加到拼写检查中。在查找MIME类型匹配时，该功能可作为一特殊原则来使用。比如用户请求了filename.gif文件，但仅有filename.jpeg文件可用，此时可以将请求重定向到filename.jpeg文件。还可以在用户请求filename.html文件时，将其定重向到filename.txt文件。这种类型的更正也可以通过命令来驱动。在未发现被请求文件时，它可以选择重定向为另外一种或一组文件扩展名。

1. mkconfig

34874: 建立一配置记录并将其设置缺省值“0”。服务器以及目录配置函数均通过调用此函数来初始化配置变量。该变量用于指示在服务器上下文中是否打开了拼写检查功能。

2. create_mconfig_for_server

34886: 调用函数mkconfig，为当前服务器上下文创建配置信息。

3. create_mconfig_for_directory

34896: 调用函数mkconfig，为当前目录上下文创建配置信息。

4. set_speling

34905: 处理CheckSpelling命令，根据命令CheckSpelling中的值为当前服务器上下文设置配置记录标志。本函数用来处理命令CheckSpelling，它在程序第34912行定义。

5. speling_cmds

34919: 定义模块mod_speling的命令列表，包括命令(Check Spelling)及相关的处理函数。当发现配置文件中的相应命令时，调用该函数进行处理。

6. sp_reason

34928: 定义一枚举类型，主要由函数spdist使用。它指示在当前请求中发现的拼写错误属于哪一种错误。程序从34938行开始说明为何选择这些类型。该原因用于将拼写更正记录在错误日志中去。

7. spdist

34977: 检测客户请求的文件名。检查它包含了何种类型的错误(可以识别的话)，函数

check_speling在程序第35137行调用本函数。被请求的文件名以及其他在相同目录下的文件作为函数spdist的参数“s”和“t”被传递给函数spdist。

34979: 循环检测“s”和“t”中所有字符，在将其全部变为小写字符之后再进行比较。若它们中存在匹配，则文件名除了大小写拼写错误之外其他完全相同。函数spdist立即返回，返回值为SP_MISCAPITALIZATION。

34986: 使用函数ap_tolower对s和t中字符继续进行比较，若两个字符串除了一个字符的变化之外其余相同，则该文件有一字符错误。函数spdist立即返回，返回值为SP_TRANSPOSITION。

34994: 使用函数strcasecmp继续比较s和t中字符。如除一字符外其余相同，则函数spdist返回，返回值为SP_SIMPLETYPO。

34999: 比较s和t中字符，检查是否其中的一个包含有额外的字符。如果是的话，则函数spdist返回，返回值为SP_EXTRACHAR。

35004: 和34999行类似，继续比较s和t中字符。但此次检查是否其中一个丢失了某个字符，若是，则函数spdist返回，返回值为SP_MISSINGCHAR。

35007: 若以上检查均不能辨别发生了何种拼写错误，则函数spdist返回值为SP_VERYDIFFERENT，表明函数不能识别其中的简单拼写错误。

8. sort_by_quality

35011: 为函数qsort提供一个值，以确定在重定向一个错误拼写请求时，最有可能成为正确文件名的文件名。该函数在第35222行调用qsort函数。函数返回的结果决定了函数 qsort应将哪些项级别设得更高。

9. check_speling

35018: 检查客户请求中文件名的拼写，并且试图解决其中发现的错误输入及大小拼写错误。

35027: 获取模块配置记录。如果配置指出：当前上下文中拼写检查没有激活，则check_speling立即退出。

35035: 检查当前请求的方法。如果不是GET，checkspeling立即返回。

35041: 检测当前请求是否为代理请求(这种情况下，文件不必须储存在当前目录中)或者已被找到(在finfo.st_mode域中使用非零值表示)。其他情况下函数check_speling立即返回。

35046: 检查当前请求是否为子请求。若是，则模块不需尝试更正文件名。

35060: 分离开客户请求中的文件名及其路径名，将第一部分放在filoc中。如果filoc不在操作后定义或者对uri域检查时不显示斜杠“/”，则函数check_speling立即返回。此时被请求的文件名错误不是模块mod_speling所能更正的。

35070: 将请求的路径名的第一部分放在good中，其第二部分，包含被错误拼写的文件名，放在域“bad”中。错误拼写的文件名及在此其后的一切数据比如参数都被放在postgood中。

35081: 检查是否正确使用计算好的字符串参数urlen和pglen更正了拼写错误的文件名。如果不是的话，则在解析用户URL时发生了某种错误，此时函数check_speling立即返回。

35090: 使用ap_popendir函数打开包含被请求文件的目录。如果目录打开失败，则发生了

模块mod_speling不能修复的错误，此时函数check_speling立即返回。

35096: 创建一candidates数组来保持当前目录中的所有目录入口，其中可能包含有当前被请求文件的入口。

35104: 使用函数readdir循环检测所有的目录入口。

35113: 若目录入口与所请求的文件名相匹配，则发生了其他某种错误，比如不能附加符号链接等。此时，模块mod_speling不能解决这类问题，函数立即返回。

35122: 比较当前目录与被请求的文件名(bad)。如果两者相匹配的话，则函数check_ speling将目录名(dir_entry>d_name)放在数组candidates中，并设置quality域指明发生了大小写拼写错误。

35137: 当前目录入口调用函数spdist。检测当前请求的文件名是否包含颠倒错误或字符丢失错误。如果函数spdist返回且返回值指示发生了这类错误，则函数check_speling将目录入口放进candidate数组，同时设置quality域指明函数spdist所发现的错误。

35171: 检测是否设置了WANT_BASENAME_MATCH编译标志。该标志使得模块mod_speling可以更正某些文件扩展名的错误。(比如使用htm代替.html。)

35188: 检查文件名中句点的位置。若第35194行的比较语句可以正常工作，则将文件名放在数组candidates中，并将quality域标记为SP_VERYDIFFERENT。其结果是把程序在这部分发现的匹配放在已经发现的匹配之下。

35207: 在循环检测完所有目录入口之后，调用ap_pclosedir函数关闭目录。

35209: 如果canditlates数组不为空，可以发现至少一个更正。

35218: 将发送到客户端信息中的Referer 值返回，以便以后使用。

35220: 根据quality域的设置对candidates数组中各项进行排序。

35232: 若最可能的选择未标记为SP_VERYDIFFERENT(此时已经排序)，则为重定向用户请求做好准备。只有最有可能的选择是可用的，因为数组中前两项并不具有相同的优先级。

35236: 根据模块mod_speling更正过的文件名，准备一个新的URL。

35243: 为新URL设置Location头信息，同时在错误日志中记录模块mod_speling修复的拼写错误，然后返回一重定向代码。

35258: 必须将存在的多种选择返回给客户端。

35275: 创建一存储器缓冲区来收集所需要的信息。

35283: 通过将文本元素在数组t中，创建将发送到客户端的HTML文档。文本元素包括相应的解释和可能的选择列表。这些由程序第35295行开始的循环语句进行装配。与代码中注释相类似，它的格式不是很直观，但每一行都包括一指向特定文件的连接。用户可以简单地通过单击此连接获得所需的文件。

35331: 继续输出由于文件扩展名错误而被标记为SP_VERYDIFFERENT的可能选项列表。

35346: 若此页中Referer域可用，则函数check_speling包括一条有关的注释。这种情况下，并没有发生客户拼写错误，而是Web主页的编码发生了错误。

35360: 将HTML文本格式的数组传递到客户请求的notes域，这样可在模块结束时(35366行)安全地删除为创建它而分配的内存缓冲区。

35368: 在错误日志中记录模块mod_speling的操作，注明有多个可能的匹配被返回给了客户。

10. speling_module

35382: 定义模块各部分，包括创建目录和创建服务器配置记录的函数，模块mod_speling所支持的命令列表和函数check_speling。函数check_speling在更正文件名错误期间被调用，并且它实际上检查客户所请求的文件名，并有可能更正或者重定向该请求。

9.7 mod_unique_id模块

模块mod_unique_id用来为客户请求设置一个唯一的标识名。该模块所创建的标识符在所有Apache实例、请求和Web服务中是唯一的，该标识主要由处理用户请求的脚本所使用。它与cookie类似，但并不在服务器和客户机之间进行传递，而且不作为客户请求头信息的一部分进行管理。

这个唯一的标识符可被作为某种类型的cookie使用。它不属于客户请求头信息中的一部分而且不存储在客户端的cookie文件中。该标识符还可作为URL的一部分返回给客户，或者放在一隐式域中以标识或者跟踪用户对Web服务器的请求。由该模块所产生的标识符可以通过环境变量UNIQUE_ID来进行访问。

9.7.1 模块结构

与其他模块不同，模块mod_unique_id中不含任何目录或服务器配置函数。根据服务器调用的不同，可分别通过函数unique_id_global_init或者unique_id_child_init进行初始化工作。请求的标识符在post_read_request由函数gen_unique_id生成(见图9-7)。

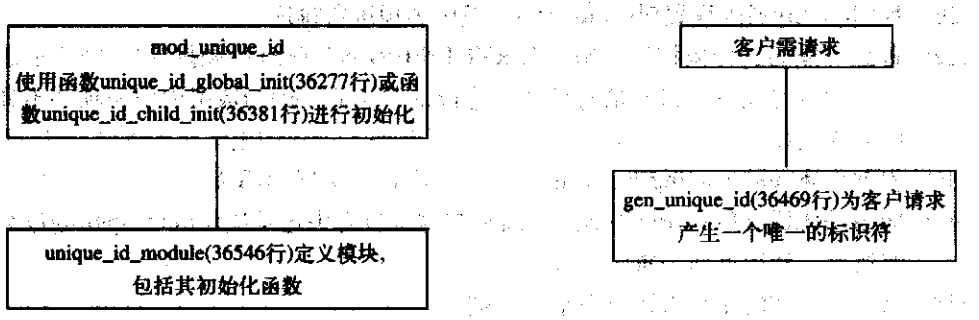

图9-7 mod_unique_id为每个用户请求分配一个唯一的标识符

9.7.2 定制

标识符的生成可以取决于服务器的硬件地址，客户请求的时间，以及处理该请求的Apache服务器的进程号等等。实际上，这些参数可能会提供唯一的标识符，但在某些环境下必须对标识符做更加细致的选择。

1. unique_id_rec

36172: 定义一数据结构，在最终ID标识符生成之前用以存放该唯一标识符的各部分。

36271: 创建一unique_id_rec_offset的简单静态实例，该实例用来存储有关当前平台的信息。函数gen_unique_id将在创建唯一标识符的过程中使用这个信息。

2. unique_id_global_init

36277: 启动主服务器时，初始化模块mod_unique_id所使用的信息。

36292: 计算变量cur_unique_id中各字段的大小及偏移量。结果被存储在数组unique_id_rec_offset中。

36316: 使用可阅读字符形式的标识符时计算标识符大小。

36325: 返回服务器的主机名，若主机名不可用，则函数unique_id_global_init将错误写入日志后退出。另外，还使用函数gethostbyname返回服务器的有关信息。当该函数不能正常返回时，函数将错误记录入错误日志。

36342: 将变量global_in_addr设置为服务器的硬件地址，该信息被用于产生唯一的标识符。若硬件地址不可用，则函数记录错误信息后返回。

36370: 返回当前系统平台跟踪使用的时间、日期信息。返回结果存储在结构tv中。

3. unique_id_child_init

36381: 初始化子服务进程。由于子进程实际对客户请求进行处理，因此子进程的进程标识符(Process ID，PID)也是函数gen_unique_id所生成的唯一标识符的一部分。在该函数中标记了PID以便以后使用。

36399: 使用函数getpid返回PID，并将其存在结构cur_unique_id.pid中。

36409: 检查分配操作。它使用了32位的PID值，若不能正常分配，则函数unique_id_child_init将错误写入日志后退出。

36416: 为物理服务器（global_in_addr）将结构cur_unique_id中字段in_addr值设置为全局值。

36424: 若当前系统平台上的时间、日期工具可用，则函数unique_id_child_init调用函数gettimeofday为cur_unique_id的counter字段设定值。若当前系统的时间、日期工具不可用，则将counter的字段设置为0。

36445: 调用函数htonl和htons函数来更正结构cur_unique_id.pid在字节顺序中所出现的任何差异。其原因是标识符可能在不同的系统结构上使用。

4. uuencoder

36456: 定义一个代码数组，该数组用于将标识符转化为可阅读的格式。本函数同UNIX的函数uuencode相类似，但代码不同。

5. gen_unique_id

36469: 在用户请求的post_read_request时期为用户请求产生唯一的标识符。

36489: 检查当前请求是否已被重定向。若是的话，则该请求应已被分配了一个唯一的标识符。此时，使用函数ap_table_get将标识符返回到结构e中，然后再使用函数ap_table_setn为当前子请求重新分配该ID值。

36495: 调用htonl函数将结构cur_unique_id的stamp域中值设置为与客户请求中域request_time的值相同。

36504: 使用for循环语句将唯一的标识符记录unique_id_rec_offset从当前缓冲区中拷贝到临时缓冲区中。这个操作避免了执行uuencode操作时可能出现的问题。结构x和y用于存储结果。

36515: 给标识符填充补丁。

36522: 使用函数uuencode，循环检查字符串y，将其索引到数组uuencoder中，并将结果存储在字符串str中。

36533: 使用“NULL”标志来标识字符串str的结束。

36536: 设置环境变量UNIQUE_ID为字符串str的值。该操作使得它可以被由其他请求启动的Apache服务进程所调用(比如脚本)。

36539: 递增计数器counter的值，以便为下一个请求生成唯一标识符。

36542: 函数返回，返回值为DECLINED。这样任何处于post_read_request阶段的其他函数仍可被调用。

6. unique_id_module

36546: 定义本模块，包括主服务的初始化函数(unique_id_global_init)，子服务的初始化函数(unique_id_child_init)，以及在post_read_request期间产生唯一标识符的函数。在该模块中，没有用到任何基于服务器或者基于目录的配置记录，也没有定义任何命令来影响本模块的操作结果。

第10章 服务器信息和状态模块

本章介绍以下两个模块：mod_info和mod_status模块。可以使用这两个模块通过浏览器向客户报告服务器的状态信息。由于浏览器本身提供了浏览服务器实时信息的很方便的方法，因此必须严格控制客户对这两个模块所提供信息的访问，以防止由于错误的管理信息而引起的安全问题。

这两个模块都不属于Apcache服务器的默认配置，如果你使用编译方法或者动态共享对象(Dynamic Shared Objects，DSO)方法将其加载到系统，你就必须自己控制对模块所提供信息的访问。对于模块mod_status来说，将其加载入系统需在httpd.conf文件中包含如下的信息：

```
<Location /server-status>
SetHandler server-status
</Location>
```

对模块mod_info来说，将其加载入系统所需的信息如下所示：

```
<Location /server-info>
SetHandler server-info
</Location>
```

对于上述所指定的文件名，可以随意设定，也可以换成其他的指向服务器状态信息的文件名，但是必须设定一处理者以对用户从浏览器中输入的URL作出响应。除此之外，还需在上述的配置语句中加入如下指令来保护对这两个模块所提供信息的访问。

```
order deny, allow
allow from your-IP-address
deny from all
```

10.1 mod_info模块

模块mod_info提供了Apache服务器系统运行时的实时配置信息列表。这些信息来自于系统的配置文件以及模块自身的配置信息。使用模块mod_info提供的信息与直接使用系统配置中的信息相比，用户可以清楚地知道配置文件中信息是如何转化的以及如何使用的。

虽然使用mod_info模块本身不需要任何附加命令，但系统仍提供了AddModuleInfo命令以提供对一些附加的信息的支持。这些信息可以在模块mod_info的输出中找到。

对模块mod_info进行询问时有几个可以选择的选项参数，这些选项参数作为标准询问参数被传递到由用户定义的模块mod_info的处理函数(如前例所示)。这些选项参数分别是：

- /server_info?server——只返回服务器的配置信息。
- /server_info?module——返回服务器中已命名模块的配置信息。
- /server_info?list——返回服务器中模块的列表信息。

10.1.1 模块结构

针对每台服务器，模块mod_info设置服务器的配置记录，该配置记录中包括用户在配置

文件中的AddModuleInfo行所输入的信息。当模块被激活时，它调用display_info函数(21458行)。该函数为服务器打开配置文件并将配置详细信息返回给客户(见图10-1)。

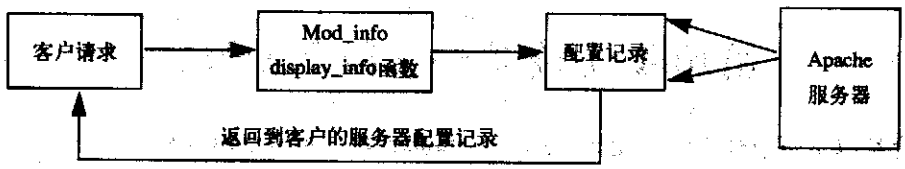

图10-1 模块mod_info 根据客户的请求将服务器配置信息返回到客户端

模块mod_info所提供的信息不包括服务器的性能参数，如果需要这些参数的话，可以查询mod_status模块。

服务器中配置的详细信息包括<VirtualHost>、<Directory>和<Location>等。除此之外，模块mod_info还发送有关系统内所安装的模块的配置细节。这些信息是通过函数mod_info_module_cmds得到的，目的是供函数 display_info使用。该函数从21294行处开始，并在display _info函数代码中的第21729进行调用。

10.1.2 定制

因为模块mod_info仅提供服务器的静态信息，所以用户可以自行改进或者增强它的功能，包括增加不在服务器配置文件中的环境信息以及围绕格式化输出作出某些改进。对模块mod_info输出的格式化可通过使用表驱动格式来改进，以便于阅读。表驱动格式又可分为根据模块分组及使用颜色编码两种方式。其他有关服务器的环境信息还可以包括操作系统配置、文件系统格式以及与系统相关的使用统计信息。(虽然它们放在模块mod_status中可能会更好一些)。用户可以在函数display_info中执行这些类型的修改。

显然，模块mod_info中所提供的信息越多，并且同时对系统的配置文件没有采取足够的重视，则客户通过浏览器访问这些信息时可能造成的危害就越大。

1. info_entry

21168: info_entry的数据结构提供了由命令AddModuleInfo设置的许多额外信息，包括它们的名称和值。

2. info_svr_conf

21173: 该数据结构是此模块中每个服务器的配置记录。因为不存在其他能够影响模块mod_info输出的命令，所以它只保持由命令AddModuleInfo提供的信息。

3. info_cfg_lines

21177: 这个数据结构主要用来缓冲Apache配置文件中的记录，直到该记录被发送到客户。

21183: 在这里向前引用了模块定义，这样其他的函数可以对其访问。事实上的模块定义从21089行开始。

21184: 这里引用了top_module变量，它是个外部变量，该变量允许模块mod_info查询在服务器中的所有活动模块信息。

4. create_info_config

21186: 函数create_info_config首先使用ap_palloc函数为服务器分配一配置记录空间，然

后建立服务器的配置记录，创建一入口数组(这些可以在以后使用AddModuleInfo命令对数组进行填充)，最后将结果返回给Apache内核。

5. merge_info_config

21197: 当程序必须同时使用基本配置(使用basev传递)和虚拟主机服务器配置(作为overridesv传递)时，函数merge_info_config用来将两个服务器配置记录结合在一起，同时创建一个新的配置记录，其中含有basev和overridesv记录信息。

6. mod_info_html_cmd_string

21221: 函数mod_info_modules_cmds大量地使用mod_info_html_cmd_string函数来返回HTML行信息。这些行是使用ap_rputs函数(如21332行所示)来发送给客户端的。本函数的目的就是将配置记录中的尖括号替代成相应的HTML代码，以便它们能在用户浏览器中正确显示。相应的HTML代码在程序的21224行和21228行。

7. mod_info_load_config

21250: 本函数返回配置记录中非注释行的数组。函数主要用来加载配置文件，这样对于某些独立模块来说，可以使用函数mod_info_module_cmds将其信息抽出。函数mod_info_load_config的调用方式如21490行所示。

21259: 该函数使用ap_pcfg_openfile函数来打开相应文件。如果文件不能正常打开(21260行进行测试)，则调用函数ap_log_rerror将错误记录到日志，然后函数返回，返回值为空“NULL”。

21268: 使用while循环语句读取配置文件中的每一行信息。读取工作是通过ap_cfg_getline API 函数来实现的，当遇到注释时(以#号开头)时，则跳过该行。21269行测试是否为注释行。如果不是注释行，则在第21272行使用ap_palloc函数为其在内存分配空间，然后再设置指向下一行的指针。

21283: 本行通过调用ap_dstrdup函数将配置记录中的某一行拷贝到变量new中。

21290: 在读完配置文件中信息之后，使用ap_cfg_closefile关闭该文件，并且返回指向配置信息起始位置的变量“ret”。

8. mod_info_module_cmds

21294: 函数mod_info_module_cmds用来输出包容器中的一些特殊模块的配置信息，比如<Directory>、<Location>和<File>中的信息，如程序中21328行和21373行所示。这些配置信息在使用HTML语言定义格式化完毕之后(使用<DL>、<DT>以及<DD>等标志)通过函数ap_rputs发送给客户。

9. find_more_info

21436: display_info函数使用find_more_info函数将一些由AddModuleInfo命令提供的有关模块的信息发送给客户。函数头中提供了模块名参数，然后在服务器配置记录之中查找与该模块同名的模块信息。查询结果在21451行被返回给调用函数。

21440: 在21441行使用函数ap_get_module_config对服务器配置进行查询，返回结果信息存储在conf变量中。函数执行所需的信息放在entry数据结构中。另外，在第21449行的for 循环语句也可以使用这些数据。

21449: for循环语句逐个检测由命令AddModuleInfo创建的配置信息列表。如果列表中的某项与作为查询参数的模块名同名，则配置列表中该模块信息被返回给调用函数。

10. display_info

21458: 这个函数是模块mod_info的核心。它将所有有关当前服务器及其安装模块的配置信息装配起来并发送给客户。

21471: 检验命令Allow的设置。即当前上下文是否允许“GET”方式，这种方式正是模块mod_info所要响应的。如果正在使用的方式(r->method_number)不是GET，则请求被拒绝，函数在21473行处返回。

21475: 将模块mod_info要产生的信息转化为text/html格式。在21476行处所有的头信息被发送出去。如果客户只请求信息(检测header_only域)，则请求处理结束，函数返回，返回值为0。

21480: 在开始处理请求之前，函数设置一个时钟。这样，若函数在时钟周期内不能正常完成，则Apache服务器会自动中止该处理程序的运行。

21482: 使用ap_rputs函数将所请求的信息页的头信息发送出去。包括<H1>标题（21484行）。

21486: 如果用户请求的URL中包含参数，则函数创建Apache服务器中所包含模块的简明列表。函数ap_server_root_relative可用来获得当前服务器的有关信息。然后在21250行，服务器信息被传递给函数mod_load_config，该函数将根据相应的模块名加载mod_info_ cfg_httpd变量，然后是变量cfname(服务器的配置名称)和mod_info_cfg_srm(服务器所需资源)。程序在21497行处还使用了mod_info_load_config函数将资源的相对独立信息加载至变量mod_info_cfg_access中。

21499: 如果客户请求不包含任何参数(比如表或者模块名)，则函数开始将服务器中每个模块的配置信息都返回给客户。首先是一个标题，然后在21502行处使用for循环语句，扫描每个模块，输出模块名以及指向其配置信息的指针。

21513: 在检测完客户请求的参数之后，程序在21515行处开始一系列的ap_rprinf调用以将服务器配置信息返回给请求的客户。函数ap_rprintf语句格式和标准printf语句相同，但其参数则由Apache 的函数提供，主要是服务器的配置细节。例如：在21518行，通过函数ap_get_server_version返回当前服务器的版本号。对服务器配置信息的输出在程序的21568行处结束，这里也是条件语句(if语句)的结束。该函数输出的选项和配置文件中的服务器选项相类似(如httpd.conf)。这些信息在系统启动的时候被加载进系统，而且在21515行和21567行可以使用相应的Apache API调用来将其返回。

21570: 在发送完规则以及HTML定义列表标志<DL>之后，21570行的for循环语句开始扫描模块，为每个模块产生其配置信息。这些配置信息与程序从21518行到21568行所产生的信息是相同的。但这些信息是从数据结构modp及其连接中得到的，而不是通过API系统调用。从21570行到21726行程序是将每个模块的信息输出。这些信息是基于全局结构modp的，包括模块在Apache中注册的处理客户请求的不同阶段的处理函数列表(如鉴别、日志和文件名转化)。程序在21656行使用一条if语句检查当前模块描述符中是否包含一个进行文件名转化的函数。如果包含的话，则在模块所提供的服务描述中增加Translate Pate特征(21660行)。除此之外，还使

用模块或描述符中cmd域列出模块支持的命令列表。

21727: 可以从Apache标准配置文件中找到一些有关模块的附加信息。在程序第21729行、第21732行和第21735行，程序调用了函数mod_info_module_cmds从文件httpd .conf、srm.conf和access.conf等中查询附加的信息。

21742: 与之相类似，程序在第21742行调用find_more_info函数以输出属于被查询模块的信息。

21762: 如果请求的参数并不需要模块的全部细节信息，而仅是简单的列表信息，则程序在这里执行for循环语句，该循环仅打印输出当前系统中的活动模块列表(21764行)。

21771: 如果为服务器定义了一数字签名，则首先调用函数ap_psignature将签名发送给客户，然后再删除这个模块的时钟。完毕之后函数返回，并给客户发送一请求成功的应答。

11. add_module_info

21779: 这个函数将命令AddModuleInfo中的项添加到模块mod_info的信息列表中去。在21785行通过调用函数ap_get_module_config返回新的项目，然后再在第21587行使用ap_push_array函数将其添加到的组中去。

12. info_cmds

21794: 这个结构定义了模块的命令以及处理这些命令所需调用的函数。因为模块mod_info除使用命令AddModuleInfo之外，不再使用其他的任何标准命令。任何时候，程序只要发现这条嵌入式命令就调用函数add_module_info。

13. info_handlers

21803: 这个结构定义了函数display_info，该函数负责处理客户对服务器信息的请求。

14. info_module:

21809: 在这部分定义模块。包括创建、组合服务器配置函数、命令列表(info_cmds)以及对客户请求作出响应函数(info_handlers)。

10.2 mod_status模块

模块mod_status的用法同模块mod_info相类似。用户必须在配置文件中设置处理的函数，然后才能通过浏览器中的URL对其进行查询。两个模块之间最大的区别就是：模块mod_info用来报告系统的静态配置信息，而模块mod_status则用来报告系统的瞬间状态，比如有多少服务在运行，Apache系统所占用的CPU时间(如果可以确定的话)等等。

在询问服务器状态时，程序同样提供了几个可选参数。

- sercver_status? refresh=5——每隔5s更新一次状态信息。如果不进行选择的话，则系统默认每隔1s进行一次更新。
- server_status? notable——不使用HTML表而直接返回信息。
- server_status? auto——将状态信息以另一种比较容易处理的格式返回客户端以便于使用脚本处理。

参数refresh 可以和另外两个参数之一结合使用，此时，需在它们之间加入一分隔符“,”(例: server_status? refresh=5, auto)

模块mod_status所返回的信息包括:

- Apache系统的平台及版本号。
- 系统的启动时间和正常运行时间。
- CPU的使用信息。
- 记分牌文件。该文件跟踪系统内运行的服务以及它们的状态。

10.2.1 模块结构

模块mod_status的结构十分简单。在检测到用户请求是个有效的GET请求后，该模块运行一系列的函数，收集有关系统的信息，然后再将信息格式化之后发送给客户，如图10-2所示。

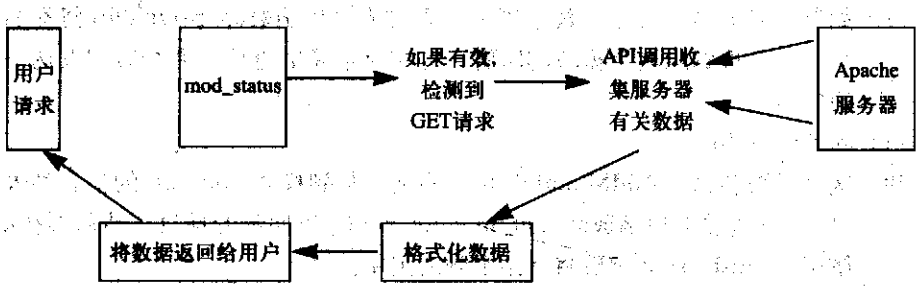

图10-2 模块mod_status收集系统信息，格式化，然后将结果返回到用户

模块mod_status之所以复杂，主要是因为需要对数据信息进行格式化。它主要包括以下几个选项:

- Auto——创建一文本格式报告，而不是HTML格式报告。
- Notable——创建一显著的HTML格式报告。
- ExendedStatus——实质上是个命令，而不是格式化的选择，该选项与提供给客户的服务器信息的多少相关。

因为客户可以使用不同的方式将这些选项结合在一起。模块mod_status的代码不像想象的那样直观。但是模块的核心仍然是基本信息的输出。

除了将状态信息返回给客户之外，模块mod_status还使用了其他一些函数来对数据进行格式化，以及输出和标记信息的时间。

10.2.2 定制

Apache源代码可在不同的OS平台运行。因此对于模块mod_status来说，如何能精确地对信息进行统计和记时是一种挑战。在源代码中，我们可以看到无数的#ifdef编译命令，它们并不影响系统的二进制代码的性能。

如果你不担心你所创建的新版Apache不能运行在其他OS平台上，有很多方法可用来增强模块mod_status的功能，主要是增加模块所提供的统计信息。以下是一些建议。

- 在状态信息输出中加上关于系统平台的信息。这样就可以知道，系统是运行在什么平台之上的。举例来说：如果基于Linux系统，则可对文件系统(/proc)进行查询以报告Apache系统使用的资源，或者报告Web服务器的一般状态信息。但是在使用自动处理功能来报告系统的详细信息时必须注意安全问题。

- 将模块mod_status的函数集成到Apache系统的核心代码中。这样系统就可在一定时间间隔内将系统状态信息写到文件中去。用户可以自定义一个命令来设置时间间隔。对于Apache系统的性能数据可使用脚本进行处理。
- 升级收集状态信息的代码，将状态信息写到数据库文件中。这样系统的统计信息就可记录在数据库中，同时也使得跟踪及处理状态信息变得更容易。
- 重新编写status_handler函数，使其支持对某些选项的处理。例如，可以使其具有产生HTML表版本的功能，甚至可以是HTML版本(增强notable和auto选项的功能)。这些改动将改变模块mod_status的大小，但对其性能没有什么影响。
- 如果感兴趣的话，你可以给状态页加上背景图像，给表单元加上阴影，使用图表显示每个主要状态下的服务函数(消亡状态、读状态和等待状态)。这样对于服务器来说可能会更容易理解。如果你在HTML页中加上了很多细节信息(比如图表和图像)而同时又担心可能会对系统性能产生影响，可以将默认的刷新速度设置为大于1的某个值或者取消快速刷新选项。

1. 编译器的DEFINE语句

35430: STATUS_MAXLINE 设置模块mod_status所提供的每条信息的最大长度。

35432: 定义了KBYTE，MBYTE和GBYTE，它们被函数format_bytes_out和format_kbytes_out作为约数使用。

35437: 定义了DEFAULT_TIME_FOTMAT，即程序在第35686行和第35689行通过API函数所输出的时间格式。但函数show_time(35500行)不使用这种格式的时间变量。当然用户可以直接对函数进行修改来选择时间格式。Apache API函数的默认输出时间格式为：Tuesday, 11-MAY-1999 17:35:08。

2. status_module

35441: 本行向前引用了模块定义。这样其他函数可以直接使用它。实际的模块定义在程序第36139行。

3. set_extended_status

35449: 当在Apache的服务器配置文件中发现命令ExtendedStatus时，调用函数set_extended_status(35468行)。该函数将把全局变量ap_extended_status设置为“on”(35458行和35461行)。

4. status_module_cmds

35446: 这个结构定义了模块mod_status使用的命令——ExtendedStatus。该命令值为“On”或者“Off”。当发现命令ExtendedStatus时，调用函数set_extended_status进行处理。

5. format_bytes_out

35476: 函数format_bytes_out使用一长整数参数(字节单位)，并且根据其大小将其转化为格式化数据，其单位可为KB、MB或GB，然后再将其返回给客户。相应的常量从35432行开始定义。

6. format_kbyte_out

函数format_kbyte_out接收一整数参数(K字节单位)，然后根据其大小，转变为格式化数据，其单位可以是KB、MB或者是GB。函数基本上同format_bytes_out函数相同，根据API 调用所

返回的值(必须返回给客户端)的单位的不同，程序可选择使用format_kbyte_out或者format_byte_out。除此之外，两个函数基本上是相同的函数。

7. show_time

35500: 函数show_time通过API函数ap_rprintf将一时间值返回给客户。变量tsecs包括一以秒为单位的计时器，用来修正所要输出的时间值。在35437行定义的DEFAULT_TIME_FORMAT决定了时间的格式。

8. stat_opt

35534: 这个结构为存储头信息及格式化数据定义了存储空间，在35529行开始定义的常数将在函数status_handler中使用。

9. status_options

35540: status_options数据结构定义了模块的选项参数，包括refresh、notable和auto，这些参数可被附加在调用模块mod_status的URL上。

10. status_handler

模块mod_status的核心。它通过调用一系列的Apache API调用来收集服务器的状态信息。收集的信息在格式完毕之后，将被返回给请求的客户(具体输出信息由模块选择参数比如refresh 或nofable确定)。另外，在函数status_handler中定义了许多变量，尤其在程序第35507行，调用了sysconf函数来设置函数系统时钟的值(tick)，在程序第35579行和第 35580行，根据HARD_SERVER_LIMIT的值定义缓冲区大小(Apache最大拷贝个数)。

35586: 使用函数ap_exists_scoreboard_image 来检查记分牌文件是否存在。如果不存在，则假设服务器运行在“inetd”模式下(而不是standalone模式)，然后程序调用ap_log_rerror函数将错误记录进错误日志，模块mod_status返回。Apache服务器很少运行在“inetd”模式下，在程序第35589行所记录的错误日志也不能反映由于其他原因而造成的配置文件丢失。

35592: 本行检查是否允许客户的“GET”请求(取决于位置及鉴别检查)，如果方式值不是“GET”，则用户请求被拒绝。因为模块mod_status只能处理“GET”请求。

35596: 将请求文档的内容格式设置为text/html类型。因为status_handler函数的剩余部分将产生一HTML格式的文档。

35603: 本行检查在用户请求URL中包含的参数。如果带有参数，则进入从35604行开始的代码。35605行的while循环语句，检查是否在URL中的每个参数都得到了处理。若未处理完毕，35606行使用if语句将参数放在status_options变量中,然后从第35609行开始使用switch语句对其进行处理。

35610: 对于refresh选项，首先检查参数中是否含有“=”号。如果有的话，则其后的数值被储在变量status_options的hdr_out_str字段中，否则的话，将系统状态的更新时间间隔设置为默认时间1秒(35618行)。

35620: 对于“notable”选项，将参数no_table_reports值设置为1，然后switch语句退出。

35623: 对于“Auto”选项，将变量content_type 值更改为text/plain(35596行设置text/plain为默认值)，short_report值设置为1，然后switch语句退出。

35633: 在所有的头信息都正确设置之后(取决于刚处理的参数)，将其发送到客户端。这些信息包括服务器状态页的更新信息。如果客户端仅请求头信息的话(第35635行

进行测试)，则函数mod_status 退出。

35638: 本行通过调用函数ap_sync_scoreboard_image使Apache记分牌文件同步，做好向客户端发送状态信息的准备。

35639: 使用“for”循环语句来收集所有正在运行的服务信息。指针ap_scoreboard_image用来将收集到的信息放在局部变量中去，包括35643行的stat_buffer数组以及35644行的pid_buffer。另外，程序还保持了一计数器，用于记录处于“ready”或者“busy”状态的服务器的个数(35645行~35648行)。

35649: 如果检测到ExtendedStatus命令，则为当前服务记录下访问次数和字节数，它们是在第35660行和第35661行实现的。

35654: 程序的第35655行到第35658行用于记录Apache所占用的CPU时间，如果本行的NO_TIMES选择项被设置为1的话，则表明运行Apache系统的平台不提供这些信息。

35670: 根据函数ap_restart_time提供的当前时间和值计算出系统的正常运行时间。

35672: 设置一时钟，以便Apache在模块出现错误情况下，能中断模块的运行。程序在第36112行处写操作完成后，删除此时钟。

35674: 如果用户请求一完全的信息(short_report=0)，则调用ap_rputs将HTML文档发送到客户端，首先是<TITLE>和<H1>信息。另外，Apache 还提供了标准API函数以建立时间和版本号信息。

35685: 将服务器的当前时间和启动时间发送到客户端，然后再将程序第35670行计算出的正常运行时间发送到客户端。

35698: 检查是否通过命令设置了“扩充的状态报告”(extended status report)，来决定是否为客户端提供全部有关访问和CPU的数据。

35699: 检查是否带有“请求简单报告”参数。若是的话，则将全部访问次数(35700行)和CPU所占用的资源信息(第35713行、第35715行、第35719行和第35723行)发送给客户端。

35726: 如果用户请求一个通常的报告，则在此行之后将较为详细的信息发送给客户端。

35767: 收集所有正在运行的服务信息，对其进行处理，然后使用正常或简化的格式将有关正在运行的服务信息发送给客户(35767和35771行)。

35776: 对“请求简单报告”再进行一次测试，然后发送一HTML<PRE>标记或者一个记分牌的标号。

35781: 使用for循环语句给每个正在运行(潜在运行)的服务输出一个字符。这样，就构成了一个状态标志行(最大值STATUS_MAXLINE定义)。在发现带refresh参数的请求时，通过检查此状态行信息，可以很快发现、更新系统内所发生的变化，以及处于idle状态的服务等等。

35792: 向客户输出几行信息以解释状态行中的信息。这些信息是静态的，而且其功能也就是解释程序在35781行使用for循环所输出的标志信息。

35812: 检查是否使用了命令“ExtendedStatus”激活了扩展状态，如果是(通过ap_extended_status调用)，则在35814行输出标题PID key,然后再使用for循环语句(从35816行开始)输出每个正在运行的服务的状态信息。对于服务器中每个进程标识(ProcessID)，该循环语句都为其输出一行信息，其中包含了对应的Apache服

务的状态。

35824: 如果未激活扩展状态，则使用第35825行的else语句将信息返回给客户。

35833: 本行开始检查命令ExtendedStatus,报告参数以及选项参数以决定怎样在发送给客户的信息中表示状态信息。

35838: 在Apache编译配置阶段根据#ifdef编译命令进行设置。这部分代码包含两个方面的内容：一种是指当Apache运行在不支持CPU信息统计的平台上，另外一种则是标准的UNIX系统，支持对CPU信息的统计。

35852: 循环检查Apache的每项服务(最大值HARD_SERVER_LIMIT)，记录记分牌信息(同样，使用了#ifdef编译命令来处理不同的平台)。在35898行，信息通过ap_rprinf函数调用被返回给客户。switch语句(第35909行到第35940行)的功能是将每个服务的当前状态输出至客户端。对于状态信息的编码，在notable方式下是以全字（full word）为单位，而在表格式下（table）仅使用一字符来表示。

35941: 计算输出时间信息。这里使用了DEFINE编译命令来选择针对于不同平台编写的代码。

35958: 这部分使用format_byte_out函数将一系列字节计数值发送到客户端。

35970: 如果用户请求的是表报告输出(选项notable未被始用)，则程序在35972行为正在运行的服务创建一信息行(注意从35973行开始的标志<TR>)。对于每个正在运行的服务，第35983行的switch语句为其在相应的表格内填充一代表其状态的字符。根据Apache所运行的操作平台的不同，程序在36017行或者36019行的相应状态表中填加了一些附加信息。

36054: 从本行开始将HTML表头发送给客户端。HTML表中存储的是服务器的当前状态信息。

36112: 在所有状态信息都被返回到客户端后，时钟监视器通过调用函数ap_kill_timeout删除时钟。

11. status_init

36117: 函数status_init 的功能是初始化函数status_handler所使用的变量，这些变量主要用来向客户端发送计分牌文件信息——对每个正在运行的服务根据函数所设置的status_flags 数组，使用一字符来代表该服务状态。对数据结构status_flags的索引关键字在文件scoreboard.h中定义。

12. status_handlers

36132: 定义模块mod_status用来处理所有客户请求的函数status_handler，而不管其所在文档树中的位置。换句话说，如果请求的URL同STATUS_MAGIC_TYPE相匹配(36134行)，则模块调用函数status_handler进行处理。STATUS_MAGIC_TYPE的定义可见本章的第一部分。

13. status_module

36139: 模块的定义从这里开始。本模块仅定义一初始化程序、一命令列表及相应处理函数列表。和其他模块一样，本模块不提供基于目录或者基于服务器的配置链接。命令表在第36148行定义，其中仅包含一个简单命令。在第316132行开始的status_handlers函数定义了对应处理函数。

第11章 服务器端include模块

本章描述模块mod_include。该模块提供通过Apache服务器进行服务器解析的HTML文件服务。服务器解析文件允许客户方在发出请求时在标准HTML文件中嵌入某些命令，在服务器端将执行这些命令，Apache服务器根据这些嵌入命令的执行结果把更新信息返回给客户机。这样的嵌入式命令很多，有条件式命令，执行外部程序命令(将它们的输入插到解析HTML文件中)等等，这些命令主要由模块mod_include 提供支持。

使用这个模块将大大加重Apache服务器的负担。因为服务器必须逐字检查客户所发送的服务器解析文件。在发现embed语句时，还必须将其他信息插入到文件去。另一方面，该模块还增加了Apache系统的不安全性，特别是对于即些通过Apache服务器执行的外部程序而言。

正因为这种不安全特性，Apache系统设置了两个Allow选项,以在同一目录上下文中选择使用不同类型的服务器端模块。如果选择使用了服务器端include模块，则必须和下面选项一起使用:

```
Allow Includes
```

当然，如果服务器管理人员觉得这个选项可能造成的潜在访问太多，可以使用下面这条语句对其限制。

```
Allow  IncludesNoExec
```

命令IncludesNoExec将使服务器忽略客户在服务器解析文件中嵌入的exec命令。下面就是一嵌入命令的例子，模块的源代码将其作为一条指令。

```
<!-- # echo var=DOCUMENT_URI-->
```

如果上面的字符串包含在服务器解析HTML文件中的话，则客户每次请求当前文件时都需将该文件的URL插入到文件中去。

11.1 mod_include模块

Apache服务器的服务器端-include功能是通过一处理含特定扩展名文件的过程来控制的。比如: 类似于下面的指令将使Apache服务器将所有扩展名为shtml的文件看成为服务器解析HTML文件。这样使用命令AddHandler后，程序将调用函数 send_shtml_file来处理扩展名为shtml 的文件(21088行):

```
AddHandler server-paresd.shtml
```

在Apache的某些老版本中，服务器端include对于某个特定目录下的文件始终是打开的，这样更加重服务器的负担，因为大多数的文件并不需要服务器进行解析。

对于处理服务器解析文件的AddHandler命令，Apache扩充了XBitHack命令。它使Apache服务器将那些拥有执行权力的请求都做为服务器解析文件处理(XBit)，这种方法是基于UNIX文件许可机制的，因此它不能在所有运行Apache的平台上正常工作。在Apache的源代码中有相应的#define语句，用来卸载相关的代码。

命令XBitHack的使用形式如下所示:

```
<Directory/web/docs/dynamic_files/>
XBitHack on
...
</Directory>
```

命令XBitHack的设置除开关状态之外，还可以为满（full）状态。这种情况下，Apache服务器检测被解析文件的组执行权限(group execute permission)。如果该位已被设置，则Apache向客户发送头LastModified，其中附加了当前的时间和日期信息。这样文件就可被缓冲在代理服务器或者浏览器中(因此，当服务器解析文件中数据经常变动时候不要将XbitHack位设置为full状态)。

11.1.1 模块结构

模块mod_include的源代码中包含许多宏定义，以及许多关于模块更新和重写部分的注释。根据命令表和进程表，函数send_parsed_file(20981行)被定义为处理服务器解析文件的主进入点。对于这个模块，仅定义了一条命令：AddHander。另外为配置记录还定义了一个简单枚举变量。

基于对程序结构和执行速度的考虑，模块的长度可根据待处理嵌入式命令的多少以及所需实现的一些基本输入/输出函数来确定。

函数send_parsed_file首先检查是否能使用服务器解析功能，若允许，则调用函数send_parsed_content(20771行)为客户请求做好准备，这个函数再通过调用函数find_string和get_directive两个函数来处理客户机的整个请求。当检查发现嵌入命令时(和Apache配置文件中命令相区别)，则调用另一相关函数返回该嵌入式命令所需数据。

模块结构如图11-1所示。

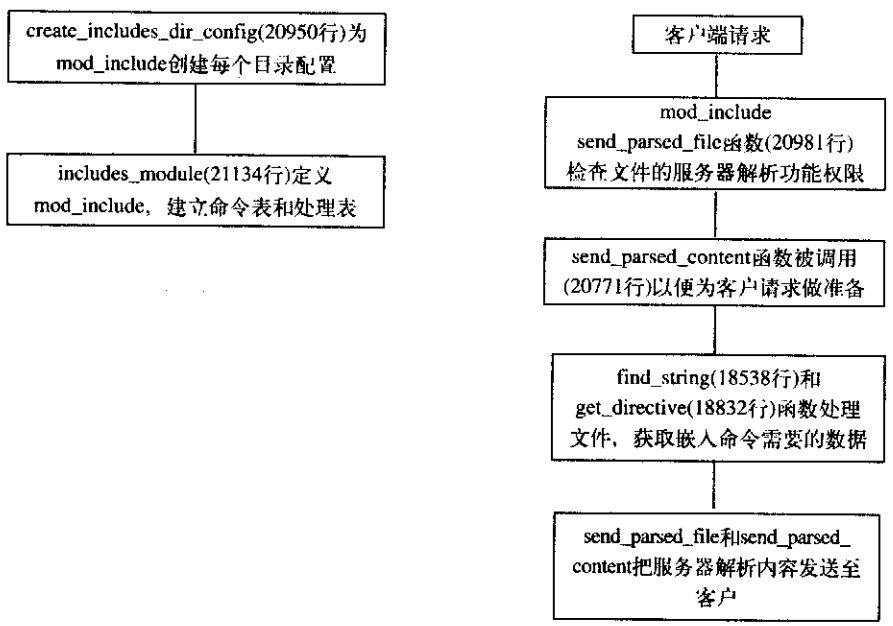

图11-1　模块mod_include使得Apache服务器可以动态解析文件和根据嵌入式命令进行内容更新

11.1.2 定制

尽管源代码中标记了这个模块需要重新编制，但由于模块本身的安全隐患，仍有必要对模块mod_include的功能进行详细介绍。实际上，可以通过好几种办法来扩充系统的功能或者提高系统的性能，比如：

- 可以定义扩充嵌入式命令(在源代码中调用命令)，该命令由模块mod_include提供支持。它们可以用来提供系统或环境信息(像命令echo一样)，或对其功能进一步细化，使之避免运行外部脚本。以及减少其对应的资源开销。
- 可以减少模块mod_include所支持的指令个数，简化解析文件请求的处理过程以提高吞吐量，同时还不会增加硬件负担。此时，源代码仅支持那些必要的命令。
- 可以采用其他方法来提高处理客户解析文件请求的速度，或者对现有源代码进行改造，或者重新编制该模块。

1. add_includes_vars

18471: 设置环境信息，这样任何变量在需要环境信息的时候都可以得到该信息。本函数在函数send_parsed_file第21056行调用。

2. PUT_CHAR

18530: 在编译期间，PUT_CHAR、GET_CHAR和FLUSH_BUF等语句都被程序中#define语句所定义的代码替代，这样使用这些宏定义，比如18546行，可以将ap_rewrite这样的函数加载至函数find_string中(18591行)。

3. find_string

18578: 在扩展名为in的文件中查找str字符，通过宏GET_CHAR(18588行)逐个读取字符，然后再逐一将其写回（使用PUT_CHAR宏，18593和18600行），直到条件满足为止，此时即找到了需求的字符。

4. decodehtml

18647: 对不同类型的HTML文件进行解码，将其转化为对应的简单类型以便于处理。

5. get_tag

18743: 展开标志以继续解析嵌入的命令，宏GET_CHAR用来从输入文件中逐个读取字符，直到标志装配好且处理完毕。这种基本的字符串操作，需多次循环才能读取相应字符以及对一些特殊字符比如引号“”进行检查。最终的标记在第18829行被返回给调用函数。

6. get_directive

18832: 和函数get_tag相同，当发现嵌入式命令时调用该函数，该函数从输入文件中逐个读取字符直到读完嵌入命令为止，然后再将该嵌入命令返回给调用者(函数头的dest字段)。

7. parse_string

18866: 对字符串进行操作(in参数)，使得嵌入式命令的各部分(比如环境变量名或者执行的命令名)对于调用函数来说都可用。该函数在函数handle_include之中被调用(19073行)。

8. include_cgi

18975: 处理运行CGI脚本(通用网关接口)的嵌入式命令。程序首先建立一了请求来处理

待运行脚本的文件名，然后程序在第18980行检查文件名的状态，在18986行检查路径。

19010: 当所有检查均正确返回时，函数include_cgi执行一子请求来运行脚本语言。如果脚本的状态是“重定向”的话，则在19013行设置Location标志。

19019: 待子请求完成任务后，清除该子请求。

9. is_only_below

19033: 测试作为参数给出的路径在目录结构中是否高于当前所请求的文档，这是通过检查路径名中的“..”(父目录指示来实现的)。

10. handle_include

19056: 处理那些请求文件的嵌入式命令并将其嵌入到当前文档中。

19068: 检查嵌入式命令的有效性(file或者virtual)。如果命令是有效的，而且所请求文件的路径是安全的(在19079行使用函数is_only_below)，那么文件中的子请求继续进行查找(19085行)。

19085: 使用子请求查找包含在HTML页中的文件，并为其使用做好准备。检查子请求的状态，如果当前目录上下文中的子请求或者行为拒绝当前文档中的嵌入式命令的使用，则将可能发生的错误信息返回(19049行开始)。

11. include_cmd_ child

19200: 在19256行处，本函数执行一子进程，通常情况下，该函数并不返回，因为在成功的情况下，程序通过函数ap_call_exec在19256行处接管了控制权。

19227: 在第19230行处使用类似于ap_sub_req_lookup的函数检查子请求的相关环境。此后，其他进程即可访问其环境信息。

19254: 使用函数ap_cleanup_for_exec为子进程的运行做好准备，然后在第19256行调用函数ap_call_exec启动子进程的执行。

12. include_cmd

19283: 处理一些必须由函数ap_bspawn_child处理的文件，该函数通过调用include_cmd_child(第19200行)来启动新的子进程。

13. handle_exec

19035: 处理“exec”嵌入式命令，该命令在Apache 服务器上启动相应的处理函数，并且将函数的结果输出插入到当前文档中。

19318: 查找字符串“cmd”，它指示所要执行的命令。如果查找正常的话，程序在19321行处调用include_cmd函数，然后再调用函数include_cmd_child。以上两步导致了嵌入式命令的执行。

14. haudle_echo

19366: 处理“echo”命令，返回当前文档中的环境变量值。

19377: 查找“var”参数，在字符串“var”之后，命令提供了它在执行时所需要的变量名(val)。程序在19379行通过使用函数ap_table_get得到变量的值，然后在19382行立即使用函数ap_rputs将变量的值返回到客户端。

15. handle_perl

19403: 如果嵌入了perl命令，则程序首先检查perl命令的有效性，然后再调用执行相应的

Perl命令(程序第19437行到19439行)。

16. handle_config

19447: 处理“config”命令，该命令用来设置此后服务器如何处理嵌入式命令。

19461: 使用函数parse_string查找每个可能的配置选项。查找成功后，函数ap_table_setn使用嵌入的命令中的配置信息来更新系统配置，并准备进行下步处理。

17. find_file

19504: 使用子请求检查嵌式式命令中所包含的文件名的有效性，函数find_file首先检查“file”情况，然后是“virtual”情况。如果子请求检查发现文件状态为无效的话，系统将其记录到错误日志，然后函数返回，返回值为“unsuccessful”,返回代码为“-1”。

18. handle_fsize

19580: 在当前文档中插入其长度值。在初始化检查完毕之后，程序在19602行调用函数ap_send_size将文档长度值插入到当前文档中。

19. handle_flashmod

19626: 在当前文档中插入其最近更新时间。在初始化检查完毕之后，程序在19647行通过调用函数ap_rputs和函数ap_ht_time将当前文档的最近更新时间返回到客户端。

20. re_check

19654: 编译一表达式以便使用嵌入式命令，程序在第19660行通过调用函数ap_pregcomp来处理嵌入式命令中的字符串。

21. get_ptoken

19690: 从当前文档的字符串中获取下一字符(token)，这些字符可以用来装配和测试嵌入式命令，该函数使用了一系列的case语句来测试可能的字符以及将该字符变量设置为相应枚举变量值。相应的枚举变量从程序的第19675行开始定义。

22. parse_expr

19856: 使用函数get_ptoken对由多个字符组成的表达式进行解析（19690行）。为此，程序使用了一系列的case语句来检查可能的字符并根据这些字符建立一解析树。

20182: 函数的最后部分对由第一部分创建的解析树进行评价(求值)。有几个地方调用了parse_string函数来处理字符串。

23. handle_if

20509: 处理嵌入式条件语句，根据命令提供的表达式值设置条件变量(conditional_status)(在20534行使用函数parse_expr函数检查、求值)。

24. handle_elif

20560: 继续处理“elif”(else if)嵌入式命令，该函数从检测条件变量conditional_status开始，然后在20596行解析由elif嵌入式命令提供的表达式值。

25. handle_else

20623: 继续处理嵌入式条件语句的else命令。因为该命令同elif命令不同，条件状态变量(condifonal_status)通常被设置为1(20640行)。

26. handle_endif

20656: 完成对嵌入式条件命令的处理（if或者if/else命令）。在第20663行调用的函数

get_tag用来检查下一标志位，然后在第20672行将变量printing设置为1，这意味着打开当前文档的输出(当前文档被“打印”到客户端)。

27. handle_set

20687: 根据嵌入式命令所提供的参数，设置一环境变量。第20717行的函数parse_string用来获取必需的参数，然后在20719行通过函数ap_table_setn将子进程的环境变量设置为该返回值。

28. handle_printenv

20733: 为当前请求打印环境信息。

20748: 循环检测数据结构 subprocess_env中的每一项(20739行)，同时使用ap_rputs将每一项发送到客户端。

29. send_parsed_content

20771: 将一服务器解析文件发送到客户端。这是模块mod_include的主要控制函数，它是在函数send_parsed_file中调用的。

20788: 将条件指示设置为“1”，以便将文档发送给客户端。

20794: 如果用户请求中附带参数的话，则这些参数将成为子进程环境的一部分。这样，这些参数对于那些可被嵌入到HTML页中的嵌入式命令来说，也是可用的。

20807: 在20808行使用函数find_string从输入文件中不断读取字符，以确定是否包含嵌入式命令。如果发现嵌入式命令的话，则程序在20810行调用get_directive函数来获取这条命令。当函数get_directive成功返回时，程序再在20820行调用send_parsed_content函数，从而开始对这条嵌入式指令的处理。

20825: 处理嵌入式条件命令（if或else/if）之后的任一可能的嵌入式指令。根据get_directive返回的结果，程序将根据嵌入式命令的不同，分别调用handle_if(208215行)、handle_exee(20875行)、handle_echo(20889行)等函数来进行处理。

30. create_includes_dir_config

20950: 为模块产生一配置信息(针对每个目录)，该配置信息中包含状态XBitHack的简单信息。

31. set_xbithack

20959: 为当前目录设置配置记录中的XBitHack状态信息，该函数在程序的第21120行定义。

32. send_parsed_file

20981: 发送一服务器解析文件到请求的客户端。程序中有多处地方通过调用这个函数来访问模块mod_include (比如命令XBitHack以及函数send_shtml_file)。

20990: 对当前目录上下文中允许进行服务器文件解析的选项进行验证。

20993: 检验当前请求的方式是否为“GET”。

20997: 检验被请求的文件在解析之前是否存在。

21008: 尝试打开用户请求的文件。如果打开失败，则程序在21009行将错误写入日志，并且返回HTTP_FORBIDDEN错误代码，该代码将导致模块mod_include 通过调用函数send_parsed_file来结束自身的运行。如果打开文件正常的话，则程序在

21015行开始进一步处理。

21015: 如果状态域被设置为xbithack_full，则表明命令XbitHack中含有参数“full”，程序在第21021行和第21022行对被请求文件的修改时间进行更新。

21028: 为当前请求返回头信息。如果客户端只请求头信息(21030行)，则程序在第21008行关闭所打开的文件后，函数成功返回。

21047: 为执行解析功能的子进程设置环境变量。它使用函数ap_add_common_vars和ap_add_cgi_vars为该子进程提供可能的变量(第21054行和第21055行)。

21065: 程序很自然地设定了一个时钟以防止发生重大的错误而导致请求处理进程被中止，而且和服务器捆绑在一起。如果程序正常执行，则程序在第21084行删除这个时钟。

21074: 使用函数send_parsed_content解析以及发送文件的内容(从20771行开始)。

33. send_shtml_file

21088: 将参数content_type设置为HTML,然后使用标准send_parsed_file函数将请求的文件返回给客户。这个函数是在使用命令AddHandler将模块mod_include设置为HTML文件类型时调用的。

34. xbithack_handler

21094: 为命令XbitHack检查配置记录中的状态位。若配置记录中状态位已被设置，则在第21114行调用标准send_parsed_file函数来解析和返回文件到客户。当服务器的配置上下文中包含命令XbitHack以指示使用服务器解析文件时，调用此函数。

35. include_cmds

21118: 定义模块中所需使用的命令，即XBitHack命令。函数set_xbithack则是在发现配置文件中的XBitHack命令时，用来对该命令进行处理。

36. includes_handlers

21125: 定义当发现模块所支持的处理类型时调用的函数。在以下四种情况下，将调用此模块：两个服务器端包含的特殊类型、服务器解析类型(使用AddHandler命令)以及命令XbitHack，它适用于所有html文件。所有这些情况都使用Xbit位来进行检测，如果XBit被设置的话，则激活mod_include模块。以上四种情况下的处理都是在几个项目（比如当前目录的配置记录中XbitHack状态）的检查后，以调用函数send_parsed_file结束。

37. includes_module

21134: 定义模块mod_include。通过调用函数create_include_dir_config为每个目录创建相应的配置信息(对于这个模块来说，仅有一条包含在配置记录中)。在程序第21144和21145行分别定义了模块使用的命令列表(仅有一条命令XBitHack)和处理函数列表。

第12章 代理服务器模块

本章介绍模块mod_proxy，这个模块可被加到一标准的Apache服务器配置之中，并使其作为局域网的代理服务器工作。Apache 服务器代理模块mod_proxy可以进行灵活的配置，比如：阻塞域访问控制，类似于Intranet的特点以及转发访问请求。模块mod_proxy 的一个标准功能部分就是缓冲客户端发出的代理请求。

Apache代理服务器的功能可通过一主处理函数来访问，在某种程度上和脚本以及模块mod_info和模块mod_status的输出类似。

模块mod_proxy被编译加载进Apache标准配置之后，用户仅需一条命令即可使其工作在代理方式下：

```
ProxyRequests on
```

除此之外，模块mod_proxy还定义了19条命令用来配置Apache服务器的代理及缓冲特点，这些命令在源程序的第41354行开始定义。

模块mod_proxy的源代码中包含多个功能调用，它们都和Apache的标准API调用相类似，但是并不属于Apache标准API接口。例如：源程序的第40832行调用ap_proxy_http_handler函数。类似于ap_*的函数调用都不属于标准的Apache API集合，它们的源代码在proxy_http.c、proxy_connect.c以及其他文件中定义。

12.1 mod_proxy 模块

使用ProxyRequests指令可将代理模块句柄选择放置在任意目录/位置下：

```
<Directory/>
ProxyRequests on
</Directory>
```

对于客户端来说，必须将其配置在代理方式下，即将其Web请求的Proxy：URL发送到Apache服务器，由代理服务器进一步转发该Web请求。

模块mod_proxy所支持的命令将为服务器设置一数据列表，该数据列表在对客户请求进行处理时使用。另外还会设置某些标志用来指示可能的操作。

12.1.1 模块结构

对模块mod_proxy的指令是根据函数proxy_cmds中的表进行处理的(从41532行开始)。有一个独立的函数负责将命令的参数写进当前服务器上下文的配置记录中。

主函数proxy_handler(40699行)从客户端接收访问请求，然后根据服务器的配置信息进行处理，最后将请求结果返回给客户端。该函数通过调用ap_proxy_cache_check和ap_proxy_http_handler等类似于标准API的函数来处理客户端的访问请求。这些调用既不属于模块mod_proxy的源码，也不属于API的标准API函数。也正因如此，模块mod_proxy的结构十分直观，其示意图如图12-1所示。

如果用户想了解Apache API函数的具体工作流程,即API函数是如何接收下载文件的，可以浏览Apache服务器代理模块部分的附加文件(这些文件在源代码中通过名字进行引用)。

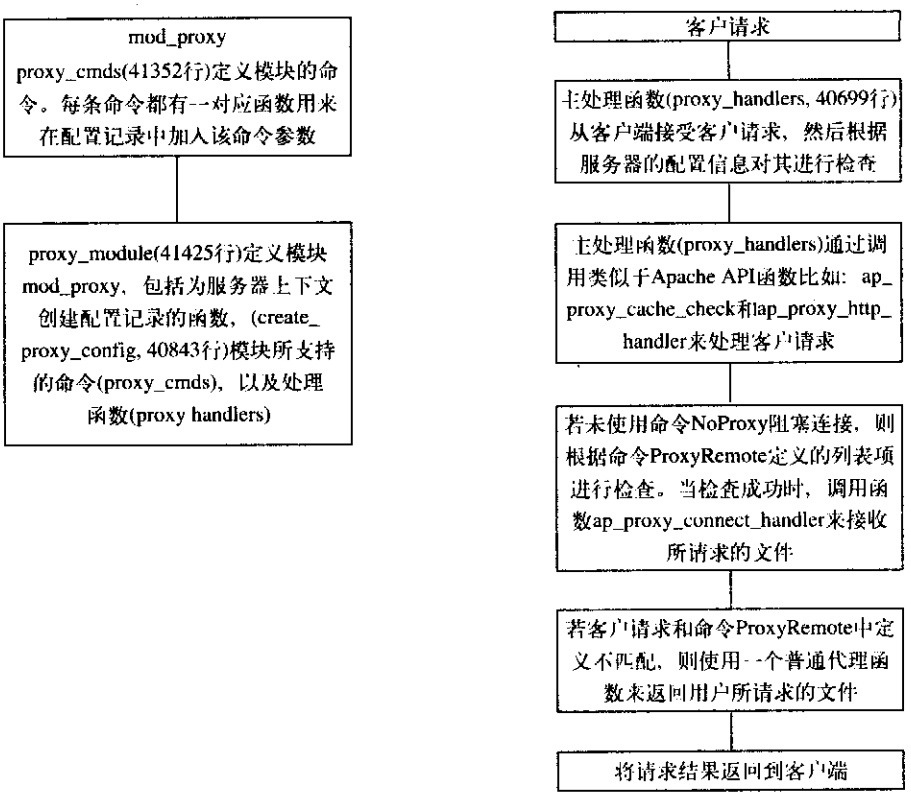

图12-1 模块mod_proxy使Apache服务器作为LAN的代理服务器工作

12.1.2 定制

对Apache服务器的代理功能的访问是通过模块mod_proxy来实现的。但它们并不作为Apache标准API的一部分在核心代码中实现，而是在另外一些和代理模块相关的源文件中实现。这些文件可以用来扩展系统的功能，加强访问控制，加强安全保密特点，定制访问跟踪等等。

1. alias_match

40468: 对URI和别名字符串进行匹配，通常由函数proxy_trans对其调用(40588行)。

40475: 根据fakename内容循环检查alias中每一项(在40471行和40473行作为参数被传递给函数，然后更新，以进行字符串比较操作)。

40476: 检查路径分隔符。

40489: 检查是否存在其他字符（非路径隔符）。若 alias和fakename中存在不同字符，则函数alias_match立即返回，返回值为“0”，即alias和fakename匹配不成功。

40499: 检查alias(别名)的结尾，如果不匹配的话，则返回值“0”。

44509: 如果所有测试均能正常结束的话，则返回URI值。

2. proxy_detect

40526: 在处理请求的预读阶段，若检测到代理请求，则函数设置一正确请求域和一修改后的URI值，以便于处理。相关函数在程序第41449行定义，主要处理模块mod_proxy在预读阶段的工作。

40543: 程序执行一个较为复杂的测试(第40534行到40542行)来检测是否当前请求是代理请求。如果是：则参数proxyreq被设置为1，另外程序在第40545行将代理的名称附加到文件名上，然后在第40547行将该请求设置为使用proxy_server进行处理。

40552: 如果请求的方式是“CONNECT”，则使用另外一种不同的测试方式，同时为代理请求设置相同的信息(程序第40555到第40559行)。

40561: 任何情况下，程序均返回“DECLINED”，但是请求域内容可能会因为代理请求而被更新。

3. proxy_trans

40564: 对待处理请求的文件名进行转化，以便能像代理请求一样进行处理。本函数在程序第41439行定义，主要进行文件名的转化。

40574: 如果标志“proxyreq”已被设置的话，则程序不需再做任何工作而立即返回。

40587: 使用函数alias_match(第40468行)对待处理请求的URI和当前服务器配置中的别名进行比较。如果匹配成功的话(第40590行进行测试)，则代理的名称在40591行被加到文件名的开始处，请求被设置成使用proxy-server进行处理，同时设置标志proxyreq，然后函数成功返回。

4. proxy_fixup

40608: 对代理请求中的文件名执行字符串操作，将其标准化之后再调用代理函数进行处理。第40612行执行了对代理请求的验证。在第40620行处程序调用了ap_proxy_http_canon函数(在文件proxy_http.c定义)。

40625: 如果URI不包括一个冒号，请求变化太大，难以修正，错误返回。

5. proxy_init

40633: 使用函数ap_proxy_garbage_init对代理服务器的缓冲区回收功能进行初始化。相关函数在文件proxy_cache.c中定义。

6. proxy_needsdomain

40651: 客户请求在以基于代理方式发送到目的HTTP服务器之前，为其准备一个合适的域名。该域名是根据命令ProxyDomain所定义的消息进行构造的。

40658: 对请求的方式进行检查，仅“GET”请求得到进一步处理，其他方式的请求则被拒绝。

40664: 检查目的主机是否为本地主机。若是，不需再进行任何远程访问操作。若目的主机名中含有符号“.”时，则还需调用其他函数进行处理。在其他情况下，程序不再进行进一步的处理而直接返回。

40670: 返回“Referer”头。该参数的值被用来记录因域重定向而引起的错误日志(程序40691行)。

40676: 使用代理请求中的URI对域名进行重新装配。域的值将会被传递给这个函数。

40683: 设置“Location”头以引用被装配成nuri形式的新域名。

40684: 这种情况将作为错误而被记录到日志中，因为域名不完整，或者说与原先的不相

匹配。用户不能想当然的去更正它。

40693: 返回一个重定向指示：HTTP_MOVED_PERMAENENTLY，这个参数使得正在访问的函数直接返回(第40752行到第40754行)。

7. proxy_handler

40669: 处理代理服务请求，该函数首先返回一配置记录。

40714: 如果客户请求不包含代理指示的话，则不能被模块mod_proxy处理(客户方没有配置Apache代理服务器)。

40718: 检查允许转发的层数(通过MAX_Forwards头)以决定是否可以响应代理请求。若允许，则头Max_Forwards值被递减(40713行)，否则的话，则不对该请求进行处理，函数直接返回。

40736: 准备从客户端获得数据。

40740: 从客户端请求中解析文件名。

40745: 使用函数ap_proxy_cache_check检查是否被请求文件已在Apache服务器的缓冲区内(该函数在文件proxy_cache.c中定义)，如果该函数返回值不为“DECLINED”，则表明在资源rc中发现用户请求的文件，程序在40747行将该文件返回。

40751: 若从客户端发来的代理请求中不包含完整的域名，则函数(40651行) proxy_needsdomain给其附加上域名。若函数返回指示需要重定向时，则在第40754行处返回此指示值。

40770: 根据数据结构“dirconn”中列表对待处理的请求进行检查。若存在匹配的话，则需要直接连接(代理功能则被阻塞)。程序在40789行处开始进行测试，此后一直到第40828行处都是尝试建立一直接链接。

40790: 使用NoProxy指令时并不中断服务器现有链接，此时，函数根据命令ProxyRemote的数据结构所定义的有关表项进行检查。如果匹配成功，则使用函数ap_proxy_connect_handler直接将文档返回给客户端。这种情况下，URL、主机名以及端口号都在函数中进行细化，在文件proxyconnect.c中实现。

40808: 如果存在客户请求与命令ProxyRemote中有关表项不匹配，可使用普通代理功能来处理用户的请求。函数ap_proxy_http_handler在文件proxy_http.c中定义，其功能就是指示从客户端到另一HTTP服务器的连接。

40817: 检查资源rc中是否存在最后几个操作的结果，若有，则返回该结果。

40827: 当代理请求失败时，程序使用更少信息的连接进行下次尝试(使用参数0和NULL)，包括以下几个函数：ap_proxy_connect_handler、ap_proxy_http_handler和ap_proxy_ftp_handler。如果这些函数全部失败的话，则程序在第40836行返回，返回值为“HTTP_FORBIDDEN”。

8. create_proxy_config

40843: 程序在40845行处为当前服务器的上下文分配一proxy_server_conf类型的新变量，以记录代理服务器的配置记录。这个数组包含一个默认的10个元素的空数组，默认常量在头文件mod_proxy.h中定义，注意到参数dirlevels和参数dirlength并未事先设定，这两个参数的乘积必须小于CACHEFLLE_LEN的值(在文件mod_proxy.h中定义)。

9. add_proxy

40884: 使用命令ProxyRemote设置一远程代理。命令的语法检查从第40896行处开始。如果语法正确，则远程代理的参数(如协议、主机名、端口号等)被记录在配置记录中(40923行)。相关函数在41358行定义。

10. add_pass

40932: 使用ProxyPass命令为从URL到Apache Server的内部路由设置一目录别名。使得其他类似于自动镜像的结点可以充分使用Apache特点。该命令的结果存储在配置记录的aliases字段中。相关函数在程序第41361行定义。

11. add_pass_reverse

40949: 使用ProxyPassReverse命令为从URL到Apache服务器的内部路由建立别名。这个命令定义了如何使用代理请求返回的头Location信息，这样客户端就不会绕过由命令ProxyPass设置的内部路由别名。函数在第41363行处定义，命令的参数信息保存在配置记录中的raliases字段中。

12. set_proxy_exclude

40966: 根据ProxyBlock命令设置受限制主机和IP地址列表，限制其对代理服务器函数进行访问。相关函数在41367行处定义。

40981: 根据当前配置记录检查受限制的结点，标记其中相匹配的项。

40987: 如果不存在相同的入口，则相应参数被存储在配置记录的noproxies字段中，同时对地址进行变换以利于测试函数对其访问。另一方面，还要注意到程序代码本身并不能处理由指令ProxyBlock所提供的多个IP地址。

13. set_allowed_ports

41008: 根据命令AllowCONNECT的参数，设置代理服务器可以建立连接的端口号。相应的函数在第41383行定。该函数的处理结果就是将由命令AllowCONNECT提供的全部端口列表不加任何测试地存储在配置记录中的allowed_connects_ports字段中。程序在第41021行处使用atoi函数将这些端口号转化成对应整数值。

14. set_proxy_dirconn

41030: 定义或者追加域名(主机名、IP地址)列表，对这些地址服务器将使用直接连接，而不是提供代理服务。这个函数是由命令NoProxy控制的，在程序第41375行定义。

41044: 从检查配置记录中需直接连接的列表项内容开始，函数检查是否有同命令NoProxy的参数相同的项。

41049: 若未发现有相同的记录，则程序继续检查命令NoProxy中的参数类型比如IP地址、域名、主机名等(分别在第41054行、第41062行和第41068行)。这种情况下，域名或者主机名(已在41050行被放置在配置记录中)被转变为低层代码以便于比较。当发生故障时，记录在错误日志中的调试行信息取决于命令NoProxy中所包含的信息类型。

15. set_proxy_domain

41085: 设置代理的默认域为Intranet。这个域被附加在不含完整域名的代理请求上。命令ProxyDomain用来设置这个参数，相关函数在第41379行定义。

16. set_proxy_req

41101: 根据命令ProxyRequest为服务器上下文打开或关闭代理服务器功能。这个标志在41108行设置，在存在代理请求服务时必须是打开的。相关函数本身在第41354行定义。

17. set_cache_size

41114: 根据CacheSize指令中参数设置缓冲区Cache的字节数。相关函数在41388行定义。

18. set_cache_root

41130: 使用CacheRoot命令设置缓存文件的路径。这个函数在41386行定义。

19. set_cache_factor

41143: 设置一个参数。该参数用来计算缓冲区内文档的默认时间。它由命令CacheLastModifiedFactor提供。相关处理函数在程序第41396行定义。命令所提供的浮点值（比如0.25）乘上当前文档上次修改到现在的时间长度（比如48小时），就得到了当前文档的默认时间（12小时）。当命令CacheMaxExpire和命令CacheDefaultExpire用来为服务器上下文提供参数时，不考虑命令CacheLastModifiedFactor提供的参数值。

20. set_cache_maxex

41160: 设置一文件在缓冲区内的最大允许缓冲时间（以小时为单位）。该参数由命令CacheMaxExpire确定。相应处理函数在第41390行定义。

21. set_cache_defex

41176: 设置一文件在缓冲区内的默认缓冲时间（以小时为单位）。该参数由命令CacheDefaultExpire确定。相应处理函数在第41393行定义。

22. set_cache_gcint

41192: 设置缓冲区的回收时间间隔（以小时为单位）。该参数由命令CacheGcInterval确定。函数在第41202行处对配置记录进行更新。

23. set_cache_dirlevels

41208: 根据命令CacheDirLevels的值设定在缓冲区允许的子目录数。对应函数在程序第41404行处定义。命令CacheDirLevels设置的值必须大于1，并且参数Length和time的乘积不大于20（可参考41228行的set_cache_dirlength函数）。程序在第41217和41220行对它们进行测试。如果需要长路径的话，改变文件mod_proxy.h中的全局参数CACHEFILE_LEN。

24. set_cache_dirlength

41228: 根据命令CacheDirLength的值设置缓冲区内目录名的长度。对应函数在第41408行处定义。同样，命令中定义的值必须大于1。如果需要长路径的话，改变文件mod_proxy.h 左中定义的全局参数CACHEFILE_LEN值。

25. set_cache_exclude

41248: 在配置记录中加入主机名、域名或者IP地址列表。这些项在客户通过代理发送请求时并不被缓冲，也就是说它们总是从远程服务器上获得。相关处理函数在第41411行定义。

41263: 循环检测配置记录中nocaches字段中各项，如果存在和当前命令中参数相同的项，

则对该项进行标记，以避免重复。

41269: 如果不存在与其相匹配的项，则在第41270行创建一个新入口，并将该命令参数缓冲进nocaches字段中。注意代码中有关进行多个IP地址处理时的注释。

26. set_recv_buffer_size

41287: 根据命令ProxyReceiveBufferSize设置接收缓冲区大小。命令参数为0时，使用系统默认字节的缓冲区，否则参数必须大于512。程序在第41294行处对此进行检查。如果参数值有效，则将其记录在psf配置记录中的recv_butter_size字段。相关处理函数在程序第41371行定义。

27. set_cache_completion

41304: 根据命令“CacheForceCompletion”将cache_completion字段的值设置为百分制形式。如果正处理的文件达到预先设定的值，则操作结束，并且结果被缓冲在缓冲区内，即使此时客户端取消了该请求。程序在第41311行检查命令所设定的值是否在0到100范围内。然后在41318行将其写进psf配置记录中。对应处理函数在第41415行处定义。这个字段主要由文件proxy_cache中代码如第40715行使用。

28. set_via_opt

41323: 命令ProxyVia的配置参数可选择使用开、关、阻塞和满状态。这个函数在发现指令ProxyVia时调用（第41419行）。

29. proxy_handlers

41346: 对客户请求的处理函数进行定义。函数从40699行处开始。

30. proxy_cmds

定义于模块相关的操作，包括有20多条命令，每个均使用对应函数将其参数写进配置记录中去。(这些函数从40884行开始，首先是add_proxy。但是并未按在proxy_cmds中出现次序排列)。对每条命令都简短的声明其应该包含什么。如果命令参数不正确的话，Apache服务器将把这个信息返回到用户。

31. proxy_module

41425: 定义模块本身，包括为服务器上下文建立配置记录的函数(第40483行，create_proxy_config函数)、各种命令列表(proxy_cmds)、模块的处理函数列表(proxy_handlers)以及如何处理在翻译和更正阶段的问题。从客户端发出的代理请求主要由函数proxy_handler进行处理。对应函数在第41366行通过proxy_handlers数据结构定义。

附录A 联机参考信息

作为一个不断发展中的协同性工程项目，Internet是查找有关Apache服务器和独立模块的信息的最好地方。在线的开发人员和程序的最终用户一起查找系统中存在的bug,提出改进的建议，编写程序模块以及在充实个人网页内容的同时为Apache官方服务器的全球网页提供信息。

每个人都可以通过报告软件中存在的bug或者通过成为Apache 的用户来加入Apache工程。下面，我们列出了一些网址，从中我们可以发现很多有用的信息，包括如何使用Apache服务器，如何运行Apache服务器以及如何管理Apache服务器(www.apache.org/httpd.html)。

Apache 服务器工程网址

任何Apache的开发人员都应该清楚知道Apache 的官方服务器网址在(www.apache.org/httpd.html)所提供的信息的重要性。在这个站点上我们可以发现一些有关Apache的最新发布版本的详细信息。有兴趣的人可以通过将自己加入到Apache经常性报告的电子邮件列表中来获得Apache 的经常性报告。在其主页上，点击下载连接即可得到Apache 的最新版本。

这里还有其他许多有关Apache Web服务器的Internet站点的信息，或者你会很有兴趣:

- Apache 开发人员资源(dev.apache.org)——Apache 的最新发展、进展将被记录在这个站点上，其中的一些区域比如负责查找、报告bug的组需要验证。无论如何，如果你正在使用Apache 的源代码并且希望能提供补丁程序，可以通过Internet 站点dev.apache.org/patches.html主页来实现。
- Apache 开发人员向导（C语言设计）(http://dev.apache.org/styleguide.html)——Apache官方网站的组成部分。该站点为Apache 开发人员提供了许多重要信息。包括如何编排源代码，如何声明函数等等。
- Apache bug报告主页(www.apache.org/bug_reports.html)——如果你在使用Apache 过程中，遇到了系统中存在的bug，可以通过访问www.apache.org/bug_reports.html主页来将该bug报告给Apache的工作组，他们将会跟踪并且修复这个bug。
- Apache 问题数据库 (bugs.apache.org)——在这个站点上你可以浏览所有用户曾经提交的有关系统的bug以及系统问题的报告。即使你自己没有遇到过这些问题，对它进行查询也有助于解决你所遇到的与数据库中问题相类似的问题，比如，如何实现一条命令，如何配置系统等等。
- Apache 公开版本的非官方补丁(www.apache.org/dist/contrib/patches)——即使某些补丁代码没有被加进Apache的核心和标准模块中，我们也可以通过www.apache.org/dist/contrib/patches主页来访问它。
- Apache HTTP服务器信息(www.apache.org/info.html)——这个主页(www.apache.org/info.html)提供了使用Apache系统建立的广大服务器列表。用户可以通过www.apache.org/ info/apache_users.html 主页将自己的站点加入到站点列表中去。

对于那些为Apache系统编写代码的人，以及那些希望了解Apache的核心和模块结构的人来说，以下的站点将会为他们提供一些有用的信息。

Shambhala API Notes

这个站点(dev.apache.org/API.html)对于那些希望了解和使用Apache 应用程序接口的人来说是个很好的地方，在这里，你可以了解如何建立Apache的句柄和请求，以及如何编译模块。

Apache模块注册

如果你打算和其他开发人员一样充分使用Apache的应用程序接口，编制自己的模块，你可以通过这个站点(modules.apache.org)注册该模块。同时也可以自己加入到声明新模块的电子邮件列表中去。

Apache/Perl集成

这个站点（perl.apache.org）致力于通过编制模块mod_perl和其他Apache/Perl模块将Apache和Perl 集成在一起。

Apache Week

这个站点（www.apacheweek.com）每周发表一些关于Apache 的不同观点的文章，此外还有一些关于新发布的服务器声明。另外，这个站点上每周还有一封C2Net发布的电子邮件。C2Net公司发布了一个使用SSL 的强壮的商业Apache版本。如果用户对Apache的某些方面不太了解的话，可以到这个站点寻找。

Ralf S. Engelschall 的个人站点

Ralf S. Engelschall 是Apache 的工作组成员之一，他还是模块 mod_ssl 和mod_rewrite的作者。他的个人站点（www.engelschall.com）中包含了许多关于这两个模块的背景信息。在这里你还可以发现一些介绍Apache Web 服务器模块的用户手册，（www.engelschall.com/pw/apache/rewriteguide），该用户手册详细介绍了模块mod_rewrite的工作原理。另一有关模块mod_ssl的用户手册可在www.engelschall.com/sw/mod_ssl/下找到。

标准C参考

这个站点 (www.valpatken.com/RandRs/std_c/index) 是由 P.J. Plunger 建立的。该站点为阅读和编写C语言程序提供了很好的参考信息。

Usenet 新闻组

Comp.infosystems.www.servers.unix和comp.infosystems.www.servers.ms-windows两个站点为Apache 提供了丰富的资源和信息。Unix 新闻组产生了关于Apache 的最活跃讨论。而windows新闻组则为在windows 环境下探讨使用Apache 时所出现的问题提供了场所。

HotWired

这个站点(www.hotwired.com)是使用Apache系统建立起来的。从1994年开始已陆续发表了多篇与Apache相关的文章。其中Webmonkey部分主要讨论与Web开发者相关的话题，如：1998年9月23日的文章就是在探讨如何在Apache下使用mod_perl 模块。

在www.hotwired.com/webmonkey/backend/backend_more.html站点下，你还可以发现许多与HTTP cookies，CGI及管理Web服务器相关的以前的文章的存档。

附录B GNU通用公共许可证

GNU通用公共许可证适用于本书中所介绍的软件，但GNU通用公共许可证不适用于本书中的正文内容。

前言

大多数软件许可证的目的是限制你共享和修改软件的自由。相反，GNU通用公共许可证则旨在保护你共享和修改软件的自由——即保证软件对所有用户都是自由的。通用公共许可证（GPL）适用于大多数自由软件基金会的软件，以及由使用这些软件而承担义务的作者所开发的软件。（有些自由软件基金会的软件受到GNU库通用公共许可证的保护。）你也可以将它用到你的程序中。

当我们谈到自由软件时，我们所指的是自由而不是价格。我们的通用公共许可证的目的是保证你有分发自由软件副本的自由（如果愿意的话，你可以对这个服务收费）；保证你能得到源程序代码或者在你需要的时候能得到它，保证你能够修改软件或者将其用到新的自由软件中，保证你知道你能做这些事情。

为保护你的权利，我们规定禁止任何人不承认你的权利或者要求你放弃这些权利。如果你分发了软件的副本或者修改了软件，这些规定就转化为你的责任。

比如说，你分发了这样一个程序的副本，不管是收费的还是免费的，你必须将你具有的一切权利给予这些接受者，必须保证他们收到或能得到源程序，而且必须让他们知道自己所拥有的权利。

我们采取两项措施来保护你的权利：1）给软件以版权保护；2）给你提供许可证。它给你复制、分发和修改软件提供法律许可。

另外，为了保护每个作者和我们自己，我们需要清楚地让每个人明白，自由软件不承担担保。如果由于其他人修改了软件并继续传播，我们需要它的接受者明白：他们所得到的并不是原来的自由软件。由其他人引入的任何问题，不应损害原作者的声誉。

最后，任何自由软件经常受到软件专利的威胁。我们希望避免这样一种风险：自由软件的再发布者以个人名义获得专利许可证，从而将软件变为私有。为防止这一点，我们必须明确：任何专利必须以允许每个人自由使用为前提，否则就不准拥有专利。

下面是有关复制、发布和修改自由软件的确切条款和条件。

关于复制、分发和修改的条款和条件

这个许可证适用于任何包含版权所有者声明的程序和其他作品，版权所有者在声明中明

确说明程序和作品可以在GPL条款的约束下发布。下面提到的“程序”指的是任何这样的程序或作品。“基于程序的作品”指的是程序或者任何受版权约束的衍生作品。衍生作品是指包含程序或程序一部分的作品，其包含形式可以是原封不动的，也可以是经过修改的/翻译成其他语言的。（下文中，翻译包含在没有附加限制的“修改”条款中。）每个领到许可证的人将用“你”来称呼。

许可证条款不适用复制、发布和修改以外的活动，他们超过了许可证条款的范围。运行程序的活动不受条款的限制。仅当程序的输出构成基于程序作品的内容时，这一条款才适用（如果只运行程序就无关）。是否普遍适用取决于程序具体用来作什么。

1) 只要你在每一副本上明显和恰当地表达版权声明和不承担保证的声明，保持此许可证的声明和没有保证的声明完整无损，并和程序一起给每个其他的程序接受者一份本许可证的副本，你就可以用任何媒体复制和发布你收到的原始的程序的源代码。

你可以为转让副本的实际行动收取一定费用。你也有权选择提供保证以换取一定的费用。

2) 你可以修改程序的一个或几个副本或程序的任何部分，以此形成基于程序的作品。只要你同时满足下面的所有条件，你就可以按前面第一款的要求复制和发布这一经过修改的程序或作品。

a) 你必须在修改的文件中附有明确的说明：你修改了这一文件及具体的修改日期。

b) 你必须使你发布或出版的作品（它包含程序的全部或一部分，或包含由程序的全部或部分衍生的作品）允许第三方作为整体按许可证条款免费使用。

c) 如果修改的程序在运行时以交互方式读取命令，你必须使它在开始进入常规的交互使用方式时打印或显示声明：包括适当的版权声明和没有保证的声明（或者你提供保证的声明）；用户可以按照此许可证条款重新发布程序的说明；并告诉用户如何看到这一许可证的副本。（例外的情况：如果原始程序以交互方式工作，它并不打印这样的声明，你的基于程序的作品也就不用打印声明）。

这些要求使用于修改了的作品的整体。如果能够确定作品的一部分并非程序的衍生产品，可以合理地认为这部分是独立的，是不同的作品。当你将它作为独立作品发布时，它不受此许可证和它的条款的约束。但是当你将这部分作为基于程序的作品的一部分发布时，作为整体它将受到许可证条款约束。准予其他许可证持有人的使用范围扩大到整个产品，也就是每个部分，不管它是谁写的。

因此，本条款的意图不在于要求或争夺对全部由你写成的作品的权利；而是履行权利来控制基于程序的集体作品或衍生作品的发布。

此外，仅将另一个不是基于程序的作品与程序（或与基于程序的作品）一起放在存储体或发布媒体的同一卷上，并不导致将那个作品置于此许可证的约束范围之内。

3) 你可以以目标码或可执行形式复制或发布程序（或符合第2款的基于程序的作品），只要你遵守前面的第1、2款，并同时满足下列3条中的一条。

a) 在通常用作软件交换的媒体上，随带其相应的、机器可读的、完整的源代码。这些源代码的发布应符合上面第1、2款的要求。或者

b) 在通常用作软件交换的媒体上，随带一份为第三方提供响应的机器可读的源代码的书面报价。其有效期不少于3年，费用不超过实际完成源程序发布的实际成本。源代码的发布这些源代码的发布应符合上面第1、2款的要求。或者

c) 随带你收到的发布源代码的报价信息。(这一条款只使用于非商业性发布，而且你只收到程序的目标码或可执行代码和按 2) 款要求提供的报价)。

作品的源代码指的是对作品进行修改时最优先择取的形式。对可执行的作品来说，完整的源代码包括：所有模块的所有源程序，加上有关的接口定义文件，控制可执行作品的安装和编译的脚本。作为特殊例外，发布的源代码不必包含任何常规发布的供可执行代码在上面运行的操作系统的主要组成部分（如编译程序、内核等），除非这些组成部分和可执行作品结合在一起。

4) 除非你明确按照许可证提出的要求去做，否则你不能复制、修改、转发许可证和发布程序。任何试图用其他方式复制、修改、转发许可证和发布程序是无效的。而且将自动结束许可证赋予你的权利。然而，对那些从你那里按许可证条款得到副本和权利的人们，只要他们继续全面履行条款，许可证赋予他们的权利依然有效。

5) 你还没有在本许可证上签字，因而你没有必要一定要接受它。然而，没有任何其他东西赋予你修改和发布程序及其衍生作品的权利。如果你不接受本许可证，这些行为是法律禁止的。因此，如果你修改或发布程序（或任何基于程序的作品），你就表明你接受这一许可证以及它的所有有关复制、发布和修改程序或基于程序的作品的条款和条件。

6) 每当你重新发布程序（或任何基于程序的作品）时，接受者自动从原始许可证颁发者那里接到受这些条款和条件支配的复制、发布或修改程序的许可证。你不可以对接受者履行这里赋予他们的权利强加其他限制。你也没有强求第三方履行许可证条款的义务。

7) 如果作为法院的判决或专利的侵权争议或任何其他原因（不限于专利问题）的结果，对你强加了一些与本许可证的条件有冲突的条件，它们也不能开脱本许可证条款对你的约束。如果你不能同时满足本许可证规定的义务和其他相关的义务，那么你可以根本不发布程序。例如：如果某一专利许可证不允许所有那些直接或间接从你那里接受副本的人们在不付费的情况下重新发布程序，唯一能同时满足两方面要求的办法是停止发布程序。

如果本条款的任何部分在特定的环境下无效或无法实施，就使用条款的剩余部分，并将条款作为整体用于其他环境。

本条款的目的不在于引诱你侵犯专利或其他财产权的要求，或争论这种要求的有效性。本条款的主要目的在于保护自由软件发布系统的完整性。它是通过通用公共许可证的应用来实现的。许多人坚持应用这一系统，已经为通过这一系统发布大量自由软件作出慷慨的贡献。作者／捐献者有权决定他／她是否通过任何其他系统发布软件。许可证持有人不能强制这种选择。

本节的目的在于明确说明许可证其余部分可能产生的结果。

8) 如果由于专利或者由于有版权的接口问题使程序在某些国家的发布和使用受到限制，将此程序置于许可证约束下的原始版权拥有者可以增加限制发布地区的条款，将这些国家明确排除在外，而在这些国家以外的地区发布程序。在这种情况下，许可证包含的限制条款和许可证正文一样有效。

9) 自由软件基金会可能通过随时发布通用公共许可证的修改版或新版。新版和当前的版本在原则上保持一致，但在提到新问题时或有关事项时，在细节上可能出现差别。

每一版本都有不同的版本号。如果程序指定使用于它的许可证版本号以及“任何更新的版本”，你有权选择指定的版本或自由软件基金会以后出版的新版本；如果程序未指定许可证

版本，你可选择自由软件基金会已经出版的任何版本。

10) 如果你愿意将程序的一部分结合到其他自由程序中，而它们的发布条件不同，写信给作者，要求准予使用。如果是自由软件基金会加以版权保护的软件，写信给自由软件基金会，我们有时会作为例外的情况处理。我们的决定受两个主要目标的指导，这两个主要目标是：我们的自由软件的衍生作品继续保持自由状态；以及从整体上促进软件的共享和重复利用。

不保证

11) 由于程序准予免费使用，在适用法准许的范围内，对程序没有保证。除非另有书面说明，版权所有者和／或其他提供程序的人们“一样”不提供任何类型的保证，不论是明确的，还是隐含的，包括但不限于隐含的适销和使用特定用途的保证。全部的风险，如程序的质量和性能问题都由你来承担。如果程序出现缺陷，你承担所有必要的服务、修复和改正的费用。

12) 除非受适用法的要求或以书面形式达成协议，在任何情况下，任何版权所有者或任何按许可证条款修改和发布程序的人们都不对你的损失负有责任。包括由于使用或不能使用程序引起的任何一般的、特殊的、偶然发生的或重大的损失（包括但不限于数据的损失，或者数据变的不精确，或者你或第三方的持续的损失，或者程序不能与其他程序协调运行等），即使就这种损失的可能性咨询过版权所有者和其他人也不例外。

如何将条款使用于你的新程序

如果你开发了一个新程序，并且希望它得到公众最大限度的利用，达到这一点的最好方法是将它变为自由软件，每个人都能在遵守条款的基础上对它进行修改和重新发布。

为了作到这一点，给程序附上下面的版权声明。最安全的方式是将其放在每个源程序的开头，以便最有效地传递专有权保证信息。每个文件至少应有“版权所有”行以及指出版权说明放在何处。

<用一行空间给出程序的名称和它的用处的简单声明>

版权所有 © 19XX<作者姓名>

这一程序是自由软件，你可以遵照自由软件基金会发布的GNU通用公共许可证条款 来修改和重新发布这一程序，即可参照该许可证的第二版，也可参照(根据你的选择)任何更新的版本进行。

发布这一程序的目的是希望它有用，但不提供任何保证，甚至没有适销或适合特定目的的隐含的保证。更详细的情况请参阅GNU通用公共许可证。

你应该已经与程序一起收到一份GNU通用公共许可证的副本，如果还没有，写信给The Free Software Foundation,Inc., 59 Temple place, Suite330, Boston, MA02111-1309USA。

还应加上如何用电子邮件和书面邮件与你联系的信息。

如果程序以交互方式进行工作，当它开始进入交互方式工作时，使它输出类似下面的简短声明：

Gnomovision 第69版，版权所有 （C）19XX 作者姓名

Gnomovision 绝对没有保证。要知道详细情况，请输入show w

这是自由软件，欢迎你遵守一定的条件重新发布它，要知道详细请况，请输入show c。

假设的命令“show c”和“show w”应显示通用公共许可证的相应条款。当然，你使用

的命令名称可以不同于“show c”和“show w”。根据你的程序的具体情况，也可以用菜单或鼠标选项来显示这些条款。

如果需要，你应该取得你的雇主（如果你是程序员）或你的学校签署放弃程序版权的声明。下面是一个例子，你应该改变相应的名称:

Yoyodyne 公司在此放弃James Harker所写的Gnomovision程序的全部版权利益。

<Ty Coon 的签名>, 1989年4月1日Ty Coon，副总裁

这一许可证不允许你将程序并入专用程序。如果你的程序是一个子程序库，可以考虑把库链入专用应用程序中。如果这是你想做的事，使用GNU Library通用公共许可证代替本许可证。