# 简单的 Web 服务器

实现一个简单的 Web 服务器 myhttpd。服务器程序启动时要读取配置文件/etc/myhttpd.conf，其中需要指定服务器监听的端口号和服务目录，例如：

Port=80

Directory=/var/www

注意，1024 以下的端口号需要超级用户才能开启服务。如果你的系统中已经安装了某种 Web 服务器（例如 Apache），应该为 myhttpd 选择一个不同的端口号。当浏览器向服务器请求文件时，服务器就从服务目录（例如/var/www）中找出这个文件，加上 HTTP 协议头一起发给浏览器。但是，如果浏览器请求的文件是可执行的则称为 CGI 程序，服务器并不是将这个文件发给浏览器，而是在服务器端执行这个程序，将它的标准输出发给浏览器，服务器不发送完整的HTTP 协议头，CGI 程序自己负责输出一部分 HTTP 协议头。

## 基本 HTTP 协议

打开浏览器，输入服务器 IP，例如 http://192.168.0.3，如果端口号不80，例如是 8000，则输入 http://192.168.0.3:8000 。这时浏览器向服务器发送的 HTTP 协议头如下：

GET / HTTP/1.1

Host: 192.168.0.3:8000

User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.6) Gecko/20061201

Firefox/2.0.0.6 (Ubuntu-feisty)

Accept:

text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/

png,\*/\*;q=0.5

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Connection: keep-alive

注意，其中每一行的末尾都是回车加换行（C 语言的"\r\n"），第一行是 GET请求和协议版本，其余几行选项字段我们不讨论，**HTTP 协议头的最后有一个空行，也是回车加换行。**

我们实现的 Web 服务器只要能正确解析第一行就行了，这是一个 GET 请求，请求的是服务目录的根目录/（在本例中实际上是/var/www），Web 服务器应该把该目录下的索引页（默认是 index.html）发给浏览器，也就是把/var/www/index.html 发给浏览器。假如该文件的内容如下（HTML 文件没必要以"\r\n"换行，以"\n"换行就可以了）：

<html>

<head><title>Test Page</title></head>

<body>

<p>Test OK</p>

<img src='mypic.jpg'>

</body>

</html>

显示一行字和一幅图片，图片的相对路径（相对当前的 index.html 文件的路径）是 mypic.jpg，也就是/var/www/mypic.jpg，如果用绝对路径表示应该是：

<img src='/mypic.jpg'>

服务器应按如下格式应答浏览器：

HTTP/1.1 200 OK

Content-Type: text/html

<html>

<head><title>Test Page</title></head>

<body>

<p>Test OK</p>

<img src='mypic.jpg'>

</body>

</html>

服务器应答的 HTTP 头也是每行末尾以回车加换行结束，最后跟一个空行的回车加换行。

HTTP 头的第一行是协议版本和应答码，200 表示成功，后面的消息 OK 其实可以随意写，浏览器是不关心的，主要是为了调试时给开发人员看的。虽然网络协议最终是程序与程序之间的对话，但是在开发过程中却是人与程序之间的对话，一个设计透明的网络协议可以提供很多直观的信息给开发人员，因此，很多应用层网络协议，如 HTTP、FTP、SMTP、POP3 等都是基于文本的协议，为的是透明性（transparency）。

HTTP 头的第二行表示即将发送的文件的类型（称为 MIME 类型），这里是text/html，纯文本文件是 text/plain，图片则是 image/jpg、image/png 等。

然后就发送文件的内容，发送完毕之后主动关闭连接，这样浏览器就知道文件发送完了。这一点比较特殊：通常网络通信都是客户端主动发起连接，主动发起请求，主动关闭连接，服务器只是被动地处理各种情况，而 HTTP 协议规定服务器主动关闭连接（有些 Web 服务器可以配置成 Keep-Alive 的，我们不讨论这种情况）。

浏览器收到 index.html 之后，发现其中有一个图片文件，就会再发一个 GET 请求（HTTP 协议头其余部分略）：

GET /mypic.jpg HTTP/1.1

一个较大的网页中可能有很多图片，浏览器可能在下载网页的同时就开很多线程下载图片，因此，''**服务器即使对同一个客户端也需要提供并行服务的能力**''。服务器收到这个请求应该把图片发过去然后关闭连接：

HTTP/1.1 200 OK

Content-Type: image/jpg

（这里是 mypic.jpg 的二进制数据）

这时浏览器就应该显示出完整的网页了。

如果浏览器请求的文件在服务器上找不到，要应答一个 404 错误页面，例如：

HTTP/1.1 404 Not Found

Content-Type: text/html

<html><body>request file not found</body></html>

## 执行 CGI 程序

如果浏览器请求的是一个可执行文件（不管是什么样的可执行文件，即使是shell 脚本也一样），那么服务器并不把这个文件本身发给浏览器，而是把它的执行结果标准输出发给浏览器。例如一个 shell 脚本/var/www/myscript.sh（注意一定要加可执行权限）：

#!/bin/sh

echo "Content-Type: text/html"

echo

echo "<html><body>Hello world!</body></html>"

这样浏览器收到的是：

HTTP/1.1 200 OK

Content-Type: text/html

<html><body>Hello world!</body></html>

总结一下服务器的处理步骤：

1. 解析浏览器的请求，在服务目录中查找相应的文件，如果找不到该文件就返回 404 错误页面

2. 如果找到了浏览器请求的文件，用 stat(2)检查它是否可执行

3. 如果该文件可执行：

a. 发送 HTTP/1.1 200 OK 给客户端

b. fork(2)，然后用 dup2(2)重定向子进程的标准输出到客户端socket

c. 在子进程中 exec(3)该 CGI 程序

d. 关闭连接

4. 如果该文件不可执行：

a. 发送 HTTP/1.1 200 OK 给客户端

b. 如果是一个图片文件，根据图片的扩展名发送相应的 Content-Type 给客户端

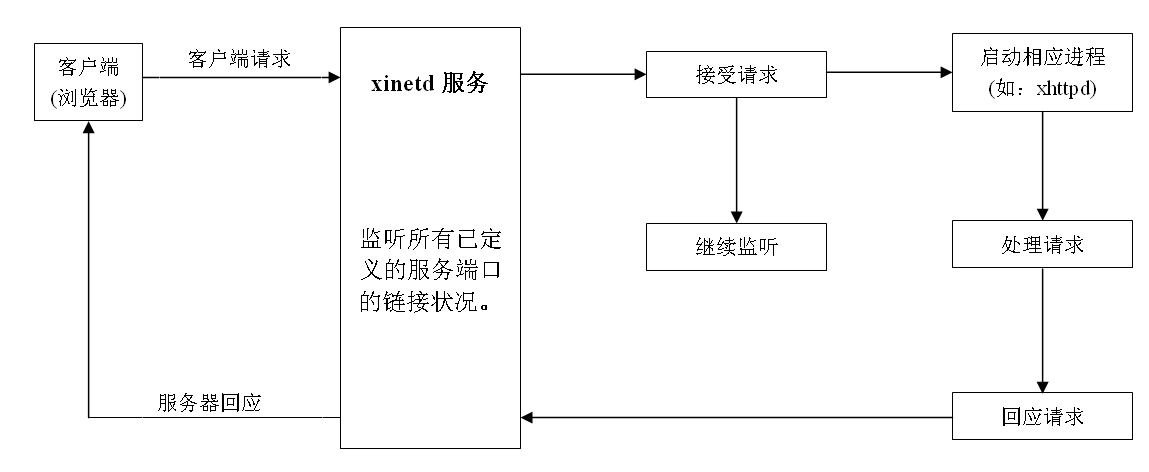
c. 如果不是图片文件，这里我们简化处理，都当作 Content-Type:text/html

d. 简单的 HTTP 协议头有这两行就足够了，再发一个空行表示结束

e. 读取文件的内容发送到客户端

f. 关闭连接

## Xinetd的应用



### 守护进程

Daemon，守护进程是生存期长的一种后台驻留进程。它们独立于控制终端并且周期性的执行某种任务或等待处理某些发生的事件。

常常在系统引导装入时启动，在系统关闭时终止。linux系统启动时会启动多守护进程，大多数服务器都是用守护进程实现的。它们没有控制终端，不能直接和用户交互，不受用户登陆、注销影响，一直运行着。守护进程完成许多系统任务.有些书籍和资料也把守护进程称作：“服务”(service)。通常采用以d结尾的名字。

从守护进程的概念不难看出，对于系统所要通过的每一种服务，都必须运行一个监听某个端口连接所发生的守护进程，这通常意味着资源浪费。为了解决这个问题，Linux引进了"网络守护进程服务程序"的概念。xinetd（extended InterNET daemon）

xinetd同时监听多个指定的端口，接受用户请求时，根据请求端口，启动不同的网络服务进程来处理这些用户请求。可以把xinetd看做一个管理启动服务的管理服务器，它决定把一个客户请求交给哪个程序处理，然后启动相应的守护进程。

xinetd无时不在运行并监听它所管理的所有端口上的服务。 当某个要连接它管理的某项服务的请求到达时，xinetd就会为该服务启动合适的服务器。

### 安装及配置xinetd

1. sudo apt-get install xinetd （sudo aptitude show/install xinetd）

2. sudo vi /etc/xinetd.d/myhttpd （注意三个统一）

service myhttpd

{

socket\_type = stream

protocol = tcp

wait = no

user = nobody

server = /home/itcast/myhttpd

server\_args = /home/itcast/dir

disable = no

flags = IPv4

}

socket\_type: 网络套接字类型,流或者数据包.

protocol: IP协议,通常时TCP或者UDP

wait: 取值yes/no

user: 运行进程的用户ID

server: 执行的完整路径

server\_args: 传递给server的值

disable: 用于在默认的{}中禁止服务

flags: 所使用的互联网协议

3. sudo vi /etc/services 向其中加入端口号，如：2222

4. 一定要 **重启xinetd服务器** sudo service xinetd restart

5. ps aux | grep xinetd 验证xinetd服务器启动是否成功。

【注意】：/etc/xinetd.d/myhttpd 配置文件中的 选项，“=”左右需各自一个空格。如需对齐，使用Tab键。

## http协议基础

超文本传输协议。

以下是浏览器发送给服务器的http**协议头**内容举例, 注意：9行的空行(\r\n)也是协议头的一部分：

**1. GET /hello.c HTTP/1.1**

2. Host: localhost:2222

3. User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:24.0) Gecko/201001 01 Firefox/24.0

4. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

5. Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3

6. Accept-Encoding: gzip, deflate

7. Connection: keep-alive

8. If-Modified-Since: Fri, 18 Jul 2014 08:36:36 GMT

**9.**

以下是经服务器按照http协议，写回给浏览器的内容举例，1~9行是**协议头**部分。注意：9行\r\n的空行不可忽略

**1. HTTP/1.1 200 Ok**

2. Server: xhttpd

3. Date: Fri, 18 Jul 2014 14:34:26 GMT

**4. Content-Type: text/plain; charset=iso-8859-1 (必选项)**

5. Content-Length: 32 （要么不写， 要写一定得精确）

6. Content-Language: zh-CN

7. Last-Modified: Fri, 18 Jul 2014 08:36:36 GMT

8. Connection: close

**9.**

10. #include <stdio.h>

11.

12. int main(void)

13. {

14. printf(“ Welcome to itcast ... \n");

15.

16. return 0;

17. }

### 常见网络文件类型：

普通文件: text/plain; charset=iso-8859-1

\*.html : text/html; charset=iso-8859-1

\*.jpg : image/jpeg

\*.gif : image/gif

\*.png: image/png

\*.wav: audio/wav

\*.avi : video/x-msvideo

\*.mov : video/quicktime

\*.mp3 : audio/mpeg

charset=iso-8859-1 西欧的编码，说明网站采用英文编码

charset=gb2312 说明网站采用的编码是简体中文

charset=utf-8 代表世界通用的语言编码；可以用到中文、韩文、日文等世界上所有语言编码

charset=euc-kr 说明网站采用的编码是韩文；

charset=big5 说明网站采用的编码是繁体中文；

Content-Length: 指定发送文件大小,可以不指定该项,如若指定,务必精准！

## 超文本标记语言html

超文本标记语言（Hyper Text Mark-up Language HTML）是构成网页文档的主要语言。可以说明文字、图形、动画、声音、表格、链接等。

html文件结构包括头(head)和主体(body)两大部分，头部描述的是浏览器，所需信息，而主题则包含所要说明的具体内容。可在Linux下用浏览器打开一个网页，使用Ctrl-u查看该网页的html文件形式。Ctrl-w恢复。

简单的错误网页形式，入：

<html>

<head><title>404 Not Found</title></head>

<body bgcolor="#cc99cc">

<h4>404 Not Found</h4>

File not found.

<hr>

<address>

<a href="<http://www.itcast.cn/>">sina</a>

</address>

</body>

</html>

## 参考框架

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define N 4096

void send\_headers(char \*type)

{

HTTP/1.1 200 Ok

Content-Type: text/plain; charset=iso-8859-1

Connection: close

\r\n

}

void send\_err(错误号, 错误名称, 错误描述)

{

<html>

<head><title>错误号 错误名称</title> </head>

<body>

错误描述

</body>

</html>

}

int main(int argc, char \*argv[])

{

char line[N];

char method[N], path[N], protocol[N];

char \*file;

struct stat sbuf;

FILE \*fp;

int ich;

if (argc != 2)

//send\_err();

if (chdir(argv[1]) == -1)

//send\_err();

if (fgets(line, N, stdin) == NULL) //GET /hello.c HTTP/1.1

//send\_err();

if (sscanf(line, "%[^ ] %[^ ] %[^ ]", method, path, protocol) != 3)

//send\_err();

while (fgets(line, N, stdin) != NULL)

if (strcmp(line, "\r\n"))

break;

if (strcmp(method, "GET") != 0)

//send\_err();

if (path[0] != '/')

//send\_err();

file = path+1;

if (stat(file, &sbuf) < 0)

//send\_err();

fp = fopen(file, "r");

if (fp == NULL)

//send\_err();

//send\_headers();

while ((ich = getc(fp)) != EOF)

putchar(ich);

fflush(stdout);

fclose(fp);

return 0;

}

### 注意事项：

1. /etc/xinetd.d/myhttpd 配置文件中的 选项，“=”左右需各自一个空格。如需对齐，使用Tab键。
2. xinetd 在启动Web服务器的时候会将/etc/xinetd.d/xhttpd中的server\_args的值，当做xhttpd的arg[1]传进来
3. 先fgets从浏览器读一行，并保证读到的这一行必须是GET /xxx HTTP/1.1 格式
4. 写http头的时候，不需要使用<html><head><titile>这些。
5. shell输入“unicode 大” 能查看汉字“大”的 utf-8 (可变长度) 表示方法----能与ASCII兼容

utf-16(固定2字节) 表示方法----不兼容ASCII Decimal(十进制) 表示方法

1. 使用命令：echo 大 | iconv -f utf8 -t gb2312 | hexdump -C 能查看“大”字的gb2312表示方法。