httpd例程学习

1 httpd介绍

2 httpd使用验证

3 接口学习

3.1 程序流程图

3.2 关键接口

- socket（）

套接字有哪些类型？socket有哪些类型？(biancheng.net)

```
/* Create a new socket of type TYPE in domain DOMAIN, using
   protocol PROTOCOL.  If PROTOCOL is zero, one is chosen automatically.
   Returns a file descriptor for the new socket, or -1 for errors.  */
extern int socket (int __domain, int __type, int __protocol) __THROW;
```

> 注：头文件中接口后面跟的 __THROW，有什么作用？答：根据搜索的资料显示，__THROW 宏是纯粹linux平台上C库才有的东西，其他平台（如windows）上的C库是不会有的。在C里面，这个宏完全没有意义。当这个头文件被C++ 引用时，才有意义，其意义是声明这个函数支持C++里的throw异常功能。
>
> 注：这里了解一些概念。什么是socket？套接字是什么？
>
> 什么是socket？
>
> socket的原意是“插座”，在计算机通信领域，socket被翻译为“套接字”，它是计算机之间进行通信的一种约定或一种方式。通过socket这种约定，一台计算机可以接受其他计算机的数据，也可以向其他计算机发送数据。
>
> socket就是用来连接到因特尔的工具。
>
> socket的典型引用就是Web服务器和浏览器：浏览器获取用户输入的URL，向服务器发起请求，服务器分析接收到的URL，讲对应的网页内容返回给浏览器，浏览器在经过解析和渲染，就将文字、图片、视频等元素呈现给用户。
>
> 学习socket，也就是学习计算机之间如何通信，并编写出实用的程序。
>
> UNIX/Linux中的socket是什么？
>
> 在UNIX/Linux系统中，为了统一对各种硬件的操作，简化接口，不同的硬件设备也都被看成一个文件。对这些文件的操作，等同于对磁盘上普通文件的操作。
>
> UNIX/Linux中的一切都是文件！
>
> 为了表示和区分已经打开的文件，UNIX/Linux会给每个文件分配一个ID，这个ID就是一个整数，被称为文件描述符（File Descriptor）例：
>
> - 通常用0来表示标准输入文件（stdin），它对应的硬件设备就是键盘；
> - 通常用1来表示标准输出文件（stdout），它对应的硬件设备就是显示器。
>
> UNIX/Linux程序在执行任何形式的I/O操作时，都是在读取或者写入一个文件描述符。
>
> 一个文件描述符只是一个和打开的文件相关联的整数，它的背后可能是一个硬盘上的普通文件、FIFO、管道、终端、键盘、显示器，甚至是一个网络连接。

> *网络连接也是一个文件，它也有文件描述符！*
>
> *我们可以通过socket()函数来创建一个网络连接，或者说打开一个网络文件，socket()的返回值就是文件描述符。有了文件描述符，我们就可以使用普通的文件操作函数来传输数据。例：*
>
> - *用read()读取从远程计算机传来的数据；*
> - *用write()向远程计算机写入数据。*

```
int socket (int __domain, int __type, int __protocol)
```

原型是：

```
int socket(int af, int type, int protocol);
```

- *af为地址族（Address Family），也就是IP地址类型，常用的有AF_INET和AF_INET6 。AF 是“Address Family”的简写，INET是“Inetnet”的简写。AF_INET 表示 IPv4 地址，例如 127.0.0.1；AF_INET6 表示 IPv6 地址，例如 1030::C9B4:FF12:48AA:1A2B。*

  其中，`127.0.0.1`是一个特殊的IP地址，表示本机地址。

- *type为数据传输方式/套接字类型，常用的有SOCK_STREAM(流格式套接字/面向连接的套接字)和SOCK_DGRAM（数据报套接字/无连接的套接字）。*
- *protocol表示传输协议，常用的有IPPROTO_TCP和IPPTOTO_UDP，分别表示TCP传输协议和UDP传输协议。*

使用IPv4地址，参数af的值为AF_INET。如果使用SOCK_STREAM传输数据，那么满足这两个条件的协议只有TCP，如下调用socket()函数：

```
int tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);  //IPPROTO_TCP表示TCP协议
```

这种套接字称为 TCP 套接字。

如果使用 SOCK_DGRAM 传输方式，那么满足这两个条件的协议只有 UDP，因此可以这样来调用 socket() 函数：

```
int udp_socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);  //IPPROTO_UDP表示UDP协议
```

这种套接字称为 UDP 套接字。

上面两种情况都只有一种协议满足条件，可以将 protocol 的值设为 0，系统会自动推演出应该使用什么协议，如下所示：

```
int tcp_socket = socket(AF_INET, SOCK_STREAM, 0);  //创建TCP套接字
int udp_socket = socket(AF_INET, SOCK_DGRAM, 0);  //创建UDP套接字
```

- *memset()*

```
* Set N bytes of S to C.  */
extern void *memset (void *__s, int __c, size_t __n)
```

每种类型的变量都有各自的初始化方法，memset()函数可以说是初始化内存的“万能函数”，通常为新申请的内存进行初始化工作。它是直接操作内存空间，mem即“内存”（memory）的意思。该函数的原型为：

```
# include <string.h>
void *memset(void *s, int c, unsigned long n);
```

linux中memset的正确用法Simple life-CSDN博客memset在linux

函数的功能是：将指针变量s所指向的前n字节的内存单元用一个“整数”c替换，注意c是int型。s是void*型的指针变量，所以它可以为任何类型的数据进行初始化。

memset一般使用“0”初始化内存单元，而且通常是给数组或结构体进行初始化。一般的变量如char、int、float等类型的变量直接初始化即可，没必要用memset。

当然，数组也可以直接初始化，但是memset是对较大的数组或结构体进行清零初始化的最快方式，因为它是直接对内存进行操作的。

这时有人会问：“字符串数组不是最好用'\0'进行初始化吗？那么可以用 memset 给字符串数组进行初始化吗？也就是说参数 c 可以赋值为'\0'吗？”

可以的。虽然参数 c 要求是一个整数，但是整型和字符型是互通的。但是赋值为 '\0' 和 0 是等价的，因为字符 '\0' 在内存中就是 0。所以在 memset 中初始化为 0 也具有结束标志符 '\0' 的作用，所以通常我们就写“0”。

memset 函数的第三个参数 n 的值一般用 sizeof() 获取，这样比较专业。注意，如果是对指针变量所指向的内存单元进行清零初始化，那么一定要先对这个指针变量进行初始化，即一定要先让它指向某个有效的地址。而且用memset给指针变量如p所指向的内存单元进行初始化时，n 千万别写成 sizeof(p)，这是新手经常会犯的错误。因为 p 是指针变量，不管 p 指向什么类型的变量，sizeof(p) 的值都是 4。

- htonl() htons() ntohl() ntohs()及inet_ntoa() inet_addr()

  5、【Linux网络编程】socket中htonl() htons() ntohl() ntohs()及inet_ntoa() inet_addr()的用法 - 阿牧路泽 - 博客园 (cnblogs.com)

字节序分为大端字节序和小端字节序：

大端字节序：是指一个整数的高位字节（32-31bit）存储在内存的低地址处，低位字节（0-7bit)存储在内存的高地址处。

小端字节序：是指 一个整数的高位字节（32-31bit)存储在内存的高地址处，低位字节（0-7bit)存储在内存的低地址处。

网络字节序和主机字节序

注：其中h表示“host”，n表示“net”，l表示“long”，s表示“short”，a表示“ascii”，addr表示“in_addr结构体”。

网络字节顺序NBO（Network byte Order）

按从高到低的顺序存储，在网络上使用统一的网络字节顺序，可以避免兼容性问题。

主机字节顺序（HBO，Host Byte Order）

不同机器的HBO不相同，与CPU设计有关，数据的顺序是由cpu决定的，而与操作系统无关。

如果你晓得6000端口的网络字节序是28695的 话，addrSrv.sin_port = htons(6000); 也可以直接写为addrSrv.sin_port = 28695；结果是一样的htons的作用就是**把端口号主机字节序转换为网络字节序**。

htonl()函数：将主机字节序转换为网络字节序

htons()函数：将主机字节序转换为网络字节序

ntohl()函数：网络顺序字节转换为主机顺序

ntohs()函数：网络顺序转换成主机顺序

inet_ntoa函数：接受一个in_addr结构体类型的参数并返回一个以点分十进制格式表示的IP地址字符串。

inet_addr()函数：需要一个字符串作为其参数，该字符串指定了了以点分十进制格式表示的IP地址（例如：192.168.0.16）。而且inet_addr函数会返回一个适合分配给S_addr的u_long类型的数值。

值得注意的是：程序里有如下内容：

```
name.sin_port = htons(*port);
name.sin_addr.s_addr = htonl(INADDR_ANY);
```

这里就会注意到`htons`与`htonl`的不同。

htons()把short型值转成按网络字节顺序排列的short型值
htonl()把long型值转成按网络字节顺序排列的long型值

同时了解`INADDR_ANY`参数的用法。

INADDR_ANY就是指定地址为0.0.0.0的地址，这个地址事实上表示不确定地址，或“所有地址”、“任意地址”。一般来说，在各个系统中军定义为0值。

INADDR_ANY的用法_lyzhm的博客-CSDN博客

- *bind()*

Linux bind函数详解handsome boy!_的博客-CSDN博客linux的bind函数

Linux 客户端bind函数的使用 - 寒魔影 - 博客园 (cnblogs.com)

```
int bind (int __fd, __CONST_SOCKADDR_ARG __addr, socklen_t __len)
```

```
#include<sys/socket.h>
int bind(int sockfd, struct sockaddr* addr, socklen_t addrlen)
```

bind()函数用于服务器端，服务器的网络地址和端口号通常固定不变，客户端得知服务器的地址和端口号以后，可以主动向服务器请求连接。因此服务器需要调用bind()函数绑定地址。

在套接口中，一个套接字只是用户程序与内核交互信息的枢纽，它自身没有太多的信息，没有网络协议地址和端口号等信息，在进行网络通信时，必须把一个套接字与一个地址相关联，这个过程就是地址绑定的过程。

很多时候内核会帮我们自动绑定一个地址，然后有时用户可能需要自己来完成这个绑定的过程，以满足实际应用的需要，最典型的情况是一个服务器进程需要绑定一个众所周知的地址或端口以等待客户来连接。这个事由bind的函数完成。

返回值：0——成功，-1—— 失败

参数sockfd

指定地址与哪个套接字绑定，这是一个由之前的socket函数调用返回的套接字。调用bind函数之后，该套接字与一个相应的地址关联，发送到这个地址的数据可以通过这个套接字来读取与使用。

参数addr

指定地址。这是一个地址结构，并且是一个已经经过填写的有效的地址结构。调用bind之后这个地址与参数sockfd指定的套接字关联，从而实现上面所说的效果。

参数addrlen

正如大多数socket接口一样，内核不关心地址结构，当它复制或传递地址给驱动的时候，它依据这个值来确定需要复制多少数据。这已经称为socket接口中最常见的参数之一。

bind函数并不是总是需要调用的，只有用户进程想与一个具体的地址或端口相关联的时候才需要调用这个函数。

如果用户进程没有这个需要，那么程序可以依赖内核的自动的选址机制来完成自动地址选择，而不需要调用bind的函数，同时也避免不必要的复杂度。

一般情况下，对于服务器进程问题需要调用bind函数，对于客户进程则不需要调用调用bind函数。

需要注意的是，在调用bind函数时一般不要将端口号设置为小于1024的值，因为1—1024是保留端口号，你可以选择大于1024中的任何一个没有被占用的端口号。

- listen()

linux下listen函数zhnlion的博客-CSDN博客linux listen函数

```
#include <sys/types.h>
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

listen: 监听来自客户端的tcp socket的链接请求；

参数sockfd是被listen函数作用的套接字

参数backlog是侦听队列的长度。在进程正在处理一个连接请求的时候，可能还存在其他的连接请求。

因为TCP连接是一个过程，所以可能存在一种半连接的状态，有时由于同时尝试连接的用户过多，使得服务器进程无法快速完成连接请求。如果这个情况出现了，服务器进程希望内核如何处理呢？内核会在自己的进程空间里维护一个队列以跟踪这些完成的连接但服务器进程还没有接手处理的连接（还没有调用accept函数的连接），这样的一个队列内核不可能让其任意大，所以必须有一个大小的上限。这个backlog告诉内核使用这个数值作为上限。

返回值	成功	失败	是否设置errno
	0	-1	是

错误信息：

EADDRINUSE：另一个socket也在监听同一个端口。
EBADF：参数sockfd为非法的文件描述符。
ENOTSOCK：参数sockfd不是文件描述符。
EOPNOTSUPP：套接字类型不支持listen操作。

其实一开始只是把这个函数当作监听网络连接请求，然而并非如此简单，在例程中，listen()函数后，startup函数直接返回了socket，然后socket就被accept()函数作为参数调用了。显然listen()如果只是监听网络连接，那么我们直接返回socket，说明listen并不只是监听网络，同时也对socket做了某些变化。

listen函数使用主动套接口变为被连接套接口，使得一个进程可以接受其他进程的请求，从而称为一个服务器进程。在TCP服务器编程中listen函数把进程变为一个服务器，并指定相应的套接字变为被动连接。

所以，listen函数一般在调用bind之后，调用accept之前调用。

参数sockfd被listen函数作用的套接字，sockfd之前由socket函数返回。在被socket函数返回的套接字fd之时，它是一个主动连接的套接字，也就是此时系统假设用户会对这个套接字调用connect函数，期待它主动与其他进程连接，然而在服务器编程中，用户希望这个套接字可以接受外来的连接请求，也就是被动等待用户来连接。由于系统默认时认为一个套接字是主动连接的，所以需要通过某种方式来告诉系统，**用户进程通过系统调用listen来完成这件事。**

listen函数-暗夜linux-ChinaUnix博客

- recv()

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

ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

不论是客户还是服务器应用程序都用recv函数从TCP连接的另一端接收数据。

参数sockfd：指定接收端套接字描述符；

参数buf：指明一个缓冲区，该缓冲区用来存放recv函数接收到的数据；

参数len：指明buf的长度；

参数flags：一般置0，。其实它还有另外一个选择是MSG_PEEK，用这个参数的时候，只从socket里读出数据，但并不会从socket里面把数据移除。

*当 应用程序调用recv函数时，recv先等待sockfd的发送缓冲中的数据被协议传送完毕，如果协议在传送sockfd的发送缓冲中的数据时出现网络错误，那么recv函数返回SOCKET_ERROR,如果sockfd的发送缓冲中没有数据或者数据被协议成功发送完毕后，**recv先检查套接字sockfd的接收缓冲区，如果sockfd接收缓冲区中没有数据或者协议正在接收数据，那么recv就一直等待，直到协议把数据接收完毕。**recv函数就把s的接收缓冲中的数据copy到buf中（注意协议接收到的数据可能大于buf的长度，所以在这种情况下要调用几次recv函数才能把s的接收缓冲中的数据copy完。recv函数仅仅是copy数据，真正的接收数据是协议来完成的），recv函数返回其实际copy的字节数。如果recv在copy时出错，那么它返回SOCKET_ERROR；如果recv函数在等待协议接收数据时网络中断了，那么它返回0。*

- *ISSpace()*

```
isspace()
              checks for white-space characters.   In  the  "C"  and
              "POSIX"  locales,  these are: space, form-feed ('\f'),
              newline ('\n'), carriage return ('\r'), horizontal tab
              ('\t'), and vertical tab ('\v').

```

功能：判断字符c是否为空白符

说明：当c为空白符时，返回非零值，否则返回零。（空白符指空格、水平制表、垂直制表、换页、回车和换行符。）

- *strcasecmp()函数*

```
#include <strings.h>

       int strcasecmp(const char *s1, const char *s2);

       int strncasecmp(const char *s1, const char *s2, size_t n);

DESCRIPTION
       The  strcasecmp() function performs a byte-by-byte comparison of the
strings s1 and s2, ignoring the case  of  the  characters.   It returns an
integer less than, equal to, or greater than zero if s1 is found,
respectively, to be less  than,  to match, or be greater than s2.
```

对字符串s1和s2进行逐字节比较，忽略字符的大小写。如果分别发现S1小于、匹配或大于S2，则返回一个小于、等于或大于零的整数。

字符串大小的比较是以ASCII码表上的顺序来决定，此顺序亦为字符的值。strcmp()首先将s1 第一个字符值减去s2 第一个字符值，若差值为0则再继续比较下个字符，若差值不为0 则将差值返回。

若参数s1和s2字符串相等返回0，s1大于s2则返回大于0的值，s1小于s2则返回小于0的值。

- *strcat()*

```
char *strcat(char *dest, const char *src);
```

strcat函数用法-百度经验 (baidu.com)

Strcat()函数将src字符串附加到DEST字符串，覆盖DEST末尾的终止空字节('\0')，然后添加终止空字节。字符串不能重叠，且DEST字符串必须有足够的空间来显示结果。如果。DEST不够大，程序行为不可预测；缓冲区溢出是攻击安全程序的首选途径。

- *strcmp（）*

```
#include <string.h>

      int strcmp(const char *s1, const char *s2);
```

函数的作用是：比较两个字符串s1和s2。如果分别发现s1小于、等于或大于s2，则返回一个小于、等于或大于零的整数。(区分大小写)

- *fgets()*

```
 #include <stdio.h>
      char *fgets(char *s, int size, FILE *stream);
```

参考链接：

linux C 文件操作之fgets（） - 陪你赏日出 - 博客园 (cnblogs.com)

linux文件操作-标准I/O操作--fgets与gets专注one的博客-CSDN博客fgets linux

fgets()从流中读入最多一个小于一个大小的字符，并将它们存储到s指向的缓冲区中。读取在EOF或换行符之后停止。如果读取换行符，则将其存储到缓冲区中。缓冲区中最后一个字符之后的终止空字节('\0')。

参数：

s:字符型指针，指向存储读入数据的缓冲区的地址。

n:从流中读入n-1个字符。

stream：指向读取的流。

返回值：

1. *当n<0时返回NULL，即空指针。*
2. *当n=1时，返回空串“”。*
3. *如果读入成功，则返回缓冲区的地址。*
4. *如果读入错误活遇到文件结尾（EOF），则返回NULL。*

在用fgets（）读入数据时，先定义一个字符数组或字符指针，如果定义了字符指针，那么一定要初始化。

- *feof()*

```
int feof(FILE *stream);
```

函数feof()测试stream指向的流的文件结束指示符，如果设置了，则返回非零值。文件结束指示符只能由函数clearerr()清除。

- *atoi()*

```
#include <stdlib.h>
      int atoi(const char *nptr);
```

把字符串转换成整型数据。atoi()函数的功能：将字符串转换成整型数；atoi()会扫描参数nptr字符串，跳过前面的空格字符，直到遇上数字或正负号才开始做转换，而再遇到非数字或字符串时（'\0'）才结束转化，并将结果返回（返回转换后的整型数）。

参考链接：

字符串函数---atoi()函数详解及实现（完整版）lanzhihui的博客-CSDN博客_atoi

- ***fork()***

```
#include <sys/types.h>
#include <unistd.h>
      pid_t fork(void);
```

linux中fork（）函数详解 - 学习记录园 - 博客园 (cnblogs.com)

首先一个进程，包括代码、数据和分配给进程的资源。

fork()函数通过系统调用创建一个与原来进程几乎完全相同的进程，也就是两个进程可以做完全相同的事，但如果初始参数或者传入的变量不同，两个进程也可以做不同的事。

一个进程调用fork()函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。

特点：*fork调用的一个奇妙之处在于它仅仅被调用一次，却能够返回两次，他可能有三种不同的返回值：*

1）在父进程中，fork返回新创建子进程的进程ID;

2）在子进程中，fork返回0;

3）如果出现错误，fork返回一个负值。

- ***dup2()***

```
#include <unistd.h>
      int dup(int oldfd);
      int dup2(int oldfd, int newfd);
```

linux之dup和dup2函数解析 - 李学文 - 博客园 (cnblogs.com)

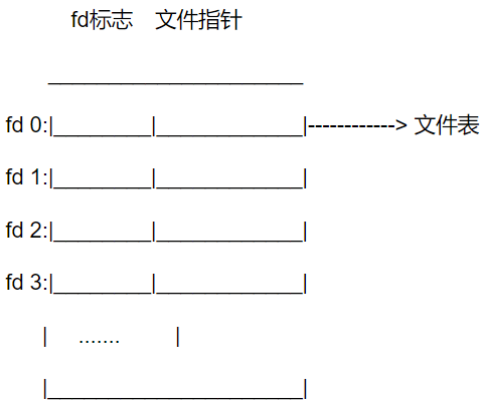
文件描述符在内核中数据结构

一个进程在此存在期间，会有一些文件被打开，从而返回一些文件按描述符，从shell中运行一个进程，默认会有3个文件描述符存在（0，1，2），

0与进程的标准输入关联；

1与进程的标准输出相关联；

2与进程的标准错误输出相关联。



文件表中包含：文件状态标志、当前文件偏移量、v节点指针。我们需要知道每个打开的文件描述符（fd标识）在进程表中都有自己的文件表项，由文件指针指向。

man文档都简明的用一句话来表达这个函数的作用：复制一个现存的文件描述符。

当调用dup函数时，内核在进程中创建一个新的文件描述符，此描述符是当前可用文件描述符的最小数值，这个文件描述符指向oldfd所拥有的文件表项。

dup2和dup的区别就是可以用newfd参数指定新描述符的数值，如果newfd已经打开，则先将其关闭。如果newfd等于oldfd，则dup2返回newfd，而不关闭它。dup2函数返回的新文件描述符同样与参数oldfd共享同一文件表项。

- send ()

```
#include <sys/types.h>
#include <sys/socket.h>
       ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

对send(),recv()函数的全面理解Linux脚本之家 (jb51.net)

不论是客户还是服务器应用程序都用send函数来向TCP连接的另一端发送数据。

客户端程序一般用send函数向服务器发送请求，而服务器则通常用send函数来向客户端程序发送应答。

该函数的第一个参数指定发送端套接字描述符

第二个参数指明一个存放应用程序要发送数据的缓冲区；

第三个参数指明实际要发送的数据的字节数；

第四个参数一般置0。

- pipe()函数

Linux系统编程pipe()小麦大大的博客-CSDN博客linux pipe

linux pipe()函数 - 简书 (jianshu.com)

linux编程之pipe()函数 - 拦云 - 博客园 (cnblogs.com)

Linux系统编程pipe()致守的博客-CSDN博客linux pipe()

```
#include <unistd.h>
       int pipe(int pipefd[2]);
```

管道是一种把两个进程之间的标准输入和标准输出连接起来的机制，从而提供一种让多个进程间通信的方法，当进程创建管道时，每次都需要提供两个文件描述符来操作管道。其中一个对管道进行写操作，另一个对管道进行写操作。对管道的读写与一般的IO系统函数一致，使用write()函数写入数据，使用read()读出数据。

管道又分为匿名管道（anonymous pipe）和命名管道（named pipe/FIFO）

具体内容参考如下连接：

浅谈Linux管道（pipe）_LittleMagics的博客-CSDN博客

匿名管道有如下特点：

> 1. 半双工，数据再同一时刻只能在一个方向流动；
> 2. 数据只能从管道的一段写入，从另一端读出；
> 3. 写入管道中的数据遵循先入先出的规则；
> 4. 管道所传送的数据是无格式的，这要求管道的读写方与写入方必须事先约定好数据的格式，如多少字节算一个消息等；
> 5. 管道不是普通的文件，不属于某个文件系统，其只存在于内存中；
> 6. 管道在内存中对应一个缓冲区，不同的系统其大小不一定相同。
> 7. 从管道读数据是一次性操作，数据一旦被读走，它从管道中被抛弃，释放空间以便写更多的数据。
> 8. 管道没有名字，只能在具有公共祖先的进程（父进程与子进程，或者两个兄弟进程，具有亲缘关系）之间使用（非常重点）。

从本质上来讲，管道也是一种文件，但他又和一般的文件有所不同，可以克服使用文件进行通信的两个问题，这个文件只存在内存中。

函数说明：pipe()会建立管道，并将文件描述词由参数filedes数组返回。

filedes[0]为管道里的读取端；

filedes[1]则为管道的写入端。

返回值：若成功则返回零，否则返回-1，错误原因存在于errno中。

- putenv()

```
#include <stdlib.h>
    int putenv(char *string);
```

linux 环境变量函数getenv()和putenv()的使用 (shuzhiduo.com)

putenv()用来改变环境变量的值。执行成功返回0，否则返回-1。string字符串必须按照"名字=值"的格式输入。

- getsockname()

```
#include <sys/socket.h>
       int getsockname(int sockfd, struct sockaddr *addr, socklen_t
*addrlen);
```

函数调用成功，则返回0，如果调用出错，则返回-1。

UNIX网络编程——getsockname和getpeername函数 - 留下的只是回忆 - 博客园 (cnblogs.com)

linux getsockname和getpeername使用JDSH0224的博客-CSDN博客getpeername linux

类似的接口有`getpeername`，`getpeername`只有在连接建立以后才调用，否则不能正确获得对方地址和端口，所以它的参数描述字一般是已连接描述字而非监听套接口描述字。

没有连接的UDP不能调用getpeername，但是可以调用getsockname和TCP一样，它的地址和端口不是再调用socket就指定了，而是再第一次调用sendto函数以后。

已经连接的UDP，再调用connect以后，这2个函数（getsockname，getpeername）都是可以用的。但是这时的意义不大，因为已经连接（connect）的UDP已经知道对方的地址。

需要这两个函数的理由：

1. 在一个没有调用bind的TCP客户上，connect成功返回后，getsockname用于返回由内核赋予该连接的本地IP地址和本地端口号。
2. 在以端口号为0调用bind（告知内核去选择本地临时端口号）后，getsockname用于返回由内核赋予的本地端口号。
3. 在一个以通配IP地址调用bind的TCP服务器上，与某个客户的连接一旦建立（accept成功返回），getsockname就可以用于返回由内核赋予该连接的本地IP地址。在这样的调用中，套接字描述符参数必须是一连接套接字的描述符，而不是监听套接字的描述符。
4. 当一个服务器的是由调用过accept的某个进程通过调用exec执行程序时，它能够获取客户身份的唯一途径便是调用getpeername。

从程序上看，在我们的端口号为0的情况下，为他动态分配端口，调用的就是`getsockname`函数。

```
if (*port == 0)  /* if dynamically allocating a port */
 {
  int namelen = sizeof(name);
  if (getsockname(httpd, (struct sockaddr *)&name, &namelen) == -1)
   error_die("getsockname");
  *port = ntohs(name.sin_port);
 }
```

- accept()

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
```

accept函数常用在服务器端接受从客户端发出的请求信息，服务器程序一旦决定接收来自客户端的请求，就需要向系统申请相关的缓冲空间。

> （通常情况下，请求可以是客户端请求读取服务端的一个文件，或是请求调用服务器上的一个函数，但无论哪一种都需要服务器进程在本地为其提供一定的缓存空间。如果是文件，就会从服务器端的硬盘中通过系统调用将文件中的内容读取到内存中（缓存）；如果是远程调用，便会调用服务器的某个进程，进程的运行时需要创建变量和从系统堆栈上面获取运行空间的。）
>
> 系统为进程划分缓冲空间的方式与系统通过文件描述符来为文件划分空间的方式相似，所以，一旦服务器accept客户端的请求成功，便会返回一个新的套接字描述符；而这个套接字描述符所指向的服务器本地系统中的缓冲空间，是用于client端与server端通信交换数据的缓冲空间。它与服务器启动的时候通过socket函数返回的套接字描述符所对应的空间不同，后者是用来接收并缓存来自各个不同client端的连接信息的缓冲队列的数据缓冲空间。

参数：

1. s：是服务器端通过正确调用socket->bind->listen函数之后的用于指向存放多个客户端缓冲队列缓冲区的套接字描述符；
2. addr：是用来保存发起连接请求的主机的地址与端口的结构体变量，就是存放服务器接受请求的客户端的网络地址与端口的结构体变量；
3. addrlen：用来传入第二个参数类型长度

返回值：

如果函数执行正确，将会返回新的套接字描述符，用于指向与当前通信的客户端交换数据的缓冲区的套接字描述符。

[c++,linux]网络编程之accept-夏目玲子-ChinaUnix博客

Linux accept系统调用Blue summer的博客-CSDN博客accept linux

- *stat()*

```
#include <sys/stat.h>
#include <unistd.h>
int stat(const char *file_name, strcut stat *buf);
```

通过文件名file获取文件信息，并保存在buf所指的结构体stat中

返回值：执行成功则返回0，失败则返回-1，错误代码存于errno

linux stat函数讲解(整理)unix linux脚本之家 (jb51.net)

Linux stat函数讲解-qiushui_007-ChinaUnix博客

关于Linux下的stat()函数rain_xyw的博客-CSDN博客linux stat函数

- *fopen()*

```
#include <stdio.h>
      FILE *fopen(const char *pathname, const char *mode);
      FILE *fdopen(int fd, const char *mode);
      FILE *freopen(const char *pathname, const char *mode, FILE *stream);
```

*对于计算机来说，I/O 代表计算机和外界的交互，交互的对象可以是人或其他设备。而对于程序来讲，I/O覆盖的范围更广。一个程序的I/O指代了程序与外界的交互，包括文件，管道，网络，命令行。I/O指代任何**操作系统理解为文件**的事物。许多操作系统都将葛总具有输入和输出概念的实体——包括设备，磁盘文件，命令行等统称为文件。*

C语言文件操作通过一个FILE 结构的指针来进行，fopen()函数返回一个FILE结构指针，

在操作系统层面上，文件操作也由一个类似于FILE的概念，在Linux里，叫做文件描述符（file description）. 在windows里，叫做句柄（handle)。用户通过某个函数打开文件获得句柄。设计句柄的原因在于句柄可以防止用户随意读写操作系统内核的文件对象。文件句柄总是和内核的文件对象相关联的，但是如何关联，细节用户并不可知。内核可以通过句柄来计算出内核里文件对象的地址。

在内核中，每个进程都有一个私有的打开文件列表，这个列表是一个指针数组，每个元素都指向内核的打开的文件对象。而fd就是这个数组的下标，当用户打开一个文件时，内核会在内部生成一个打开的文件对象，并在这个表里找到一个空的项，让这个项指向一个打开的文件对象，并返回这一项的下标作为fd。这个表处于内核，并且用户无法访问，因此即使用户有fd，也无法打开。 而文件对象的地址，只能通过系统提供的函数来操作。

在C语言中，通过FILE结构操作文件，FILE结构必定和fd有一一对应的关系。

linux输出流文件描述符,Linux下 文件描述符（fd）与 文件指针（FILE） _管墨迪的博客-CSDN博客*

深度讲解linux中fopen()函数Arranh的博客-CSDN博客fopen linux

4 源码分析

4.1 资料搜集

会搜集网络上关于httpd源码分析的帖子，从中做出摘录学习；

> *链接：HTTP服务器的本质:tinyhttpd源码分析及拓展 - 知乎 (zhihu.com)*
>
> *1. http报文*
>
> *http请求由3部分组成看，分别是：起始行、消息报头、请求正文*

```
Request Line<CRLF>
Header-Name: header-value<CRLF>
Header-Name: header-value<CRLF>
//一个或多个，均以<CRLF>结尾
<CRLF>
body//请求正文
```

起始行以一个方法符号开头，以空格分开，后面跟着请求的URL和协议的版本，格式如下：

```
Method Request-URL HTTP-Verson CRLF
```

其中Method 表示请求方法；

Request-URL是一个统一资源标识符；

HTTP-Version表示请求的HTTP协议版本；

CRLF表示回车和换行（除了作为结尾的CRLF外，不允许出现单独的CR或LF字符；

- 请求方法（所有方法全为大写）有多种，各个方法的解释如下：
    - GET 请求获取Request-URL所标识的资源；
    - POST 在Request-URL 所标识的资源后附加新的数据；
    - HEAD 请求获取由Request-URL 所标志的资源的响应消息报头；
    - PUT 请求服务器存储一个资源，并用Request-URI作为其标识
    - DELETE 请求服务器删除Request-URI所标识的资源
    - TRACE 请求服务器回送收到的请求信息，主要用于测试或诊断
    - CONNECT 保留将来使用
    - OPTIONS 请求查询服务器的性能，或者查询与资源相关的选项和需求

[tinyhttp源码分析 - 简书 (jianshu.com)]

这个作者手画了一个流程图，但是最值得学习，是他提出的几个问题，很值得借鉴，如下：

- 如何去处理并发的http请求？

  源码给出的思路是，对每一个来的请求， 创建一个线程去处理并发的请求；

- 对于一个http请求，如果从请求里面解析到关键的字段信息，比如http method ,是get, post, put, delete, 还是head? url是什么？如果是post类型的请求，post的参数是在http请求的正文里面的，那么怎么读取出来他们？他们的长度是如何确定的？

判断http请求header的每一行的标志是：\r\n

判断http请求header和请求正文的标志是：两个\r\n如上图所示。

对于一个post请求，请求的正文里面是post的请求数据，header里面的content-length指明了post请求的数据的长度。

- *对于带有参数的get方法，和post方法，你的服务器如何去处理?*

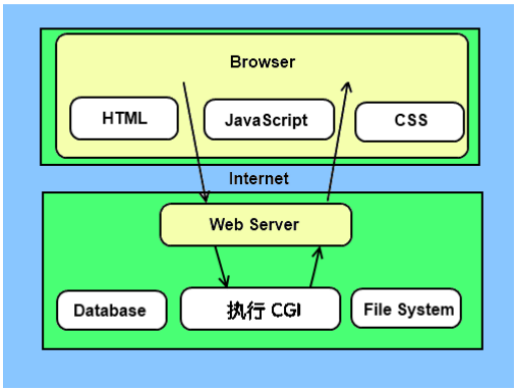
带有参数的get请求，和post请求，服务器没有办法简单的返回一个静态的文件，服务器需要在服务器端将相应的页面"计算"出来，然后返回给浏览器。

加入浏览器发送了一个请求 /index?id=100, 请求id=100的人的主页；那么，服务器需要"计算"出来id=100的这个人的主页的页面。这个需要cgi来帮忙，cgi可以理解位在服务器端可以执行的小脚本。服务器收到这个请求之后，执行index.cgi(这个文件是提前写好的，专门来处理这样的请求)，服务器执行index.cgi, 参数是id=100,然后"计算"出网页的数据，返回给浏览器。

链接:Python CGI 编程 | 菜鸟教程 (runoob.com)

CGI(Common Gateway Interface)通用网关接口，它是一段程序，运行在服务器上：HTTP服务器，提供同客户端HTML页面的接口。

CGI架构图



5 附录

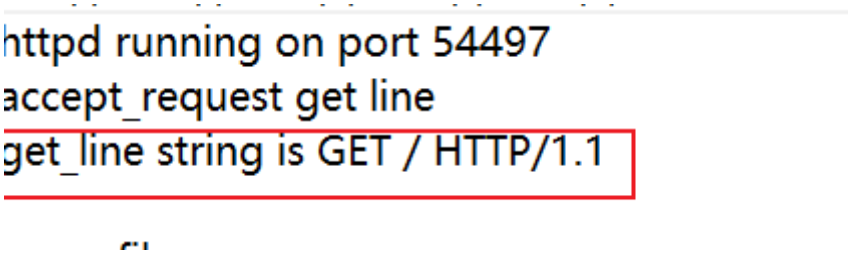
5.1 HTTP报文格式

HTTP有两种报文：请求报文和响应报文

5.2 请求报文

- 请求行

请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成，它们用空格分隔。例如：



- HTTP 协议的请求方法有 GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。这里介绍最常用的 GET 方法和 POST 方法。

  GET：当客户端要从服务器中读取文档时，使用 GET 方法。GET 方法要求服务器将 URL 定位的资源放在响应报文的数据部分，回送给客户端。使用 GET 方法时，请求参数和对应的值附加在 URL 后面，利用一个问号（“?”）代表 URL 的结尾与请求参数的开始，传递参数长度受限制。例如，/index.jsp?id=100&op=bind。

  POST：当客户端给服务器提供信息较多时可以使用 POST 方法。POST 方法将请求参数封装在 HTTP 请求数据中，以名称 / 值的形式出现，可以传输大量数据。

  协议版本的格式为：HTTP / 主版本号. 次版本号，常用的有 HTTP/1.0 和 HTTP/1.1。

- 请求头部

  请求头部由关键字 / 值对组成，每行一对，关键字和值用硬问冒号":"分隔。请求头部通知服务器有关客户端请求的信息。常见的请求头如下：

  - Host 接受请求的服务器地址，可以是 IP: 端口号，也可以是域名
  - User-Agent 发送请求的应用程序名称
  - Connection 指定与连接相关的属性，如 Connection:Keep-Alive
  - Accept-Charset 通知服务端可以发送的编码格式
  - Accept-Encoding 通知服务端可以发送的数据压缩格式
  - Accept-Language 通知服务端可以发送的语言

```
serve_file
get_line string is Host: 127.0.0.1:54497

serve_file
get_line string is Connection: keep-alive

serve_file
```

```
get_line string is sec-ch-ua: " Not;A Brand";v="99", "Microsoft Edge";v="97", "Chromium";v="97"

serve_file
get_line string is sec-ch-ua-mobile: ?0

serve_file
get_line string is sec-ch-ua-platform: "Linux"

serve_file
get_line string is Upgrade-Insecure-Requests: 1

serve_file
get_line string is User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.20 Safari/537.36 Edg/97.0.1072.21

serve_file
get_line string is Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

serve_file
get_line string is Sec-Fetch-Site: none

serve_file
get_line string is Sec-Fetch-Mode: navigate

serve_file
get_line string is Sec-Fetch-User: ?1

serve_file
get_line string is Sec-Fetch-Dest: document

serve_file
get_line string is Accept-Encoding: gzip, deflate, br

serve_file
get_line string is Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6

serve_file
get_line string is

```

- 空行

  最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器一下不再有请求头。

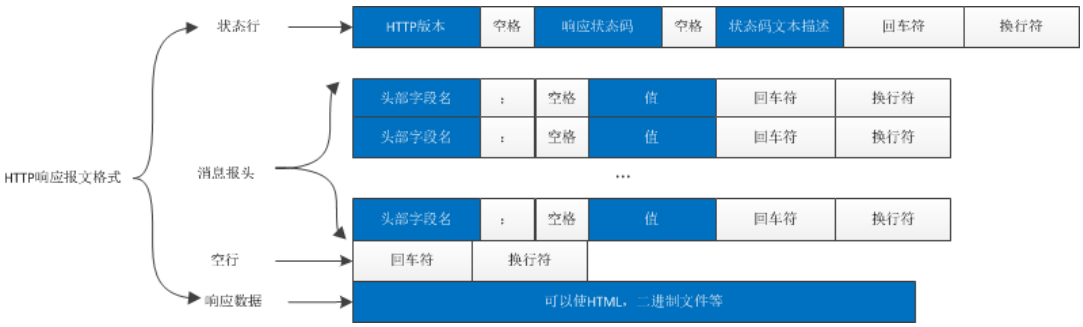
- 请求数据

  请求数据

  请求数据不在 GET 方法中使用，而是在 POST 方法中使用。POST 方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是 Content-Type 和 Content-Length。

```
accept_request get line
get_line string is POST /color.cgi HTTP/1.1

execute_cgi 2
get_line string is Host: 127.0.0.1:54497

execute_cgi 3
get_line string is Connection: keep-alive

execute_cgi 3
get_line string is Content-Length: 12

execute_cgi 3
get_line string is Cache-Control: max-age=0

execute_cgi 3
get_line string is sec-ch-ua: " Not;A Brand";v="99", "Microsoft Edge";v="97", "Chromium";v="97"

execute_cgi 3
get_line string is sec-ch-ua-mobile: ?0

execute_cgi 3
get_line string is sec-ch-ua-platform: "Linux"

execute_cgi 3
get_line string is Upgrade-Insecure-Requests: 1

execute_cgi 3
get_line string is Origin: http://127.0.0.1:54497

execute_cgi 3
get_line string is Content-Type: application/x-www-form-urlencoded
```

5.3 响应报文



HTTP响应报文主要由状态行、响应头部、响应正文3部分组成。

- *状态行*

  状态行由 3 部分组成，分别为：协议版本，状态码，状态码描述，之间由空格分隔。状态代码为 3 位数字，200~299 的状态码表示成功，300~399 的状态码指资源重定向，400~499 的状态码指客户端请求出错，500~599 的状态码指服务端出错（HTTP/1.1 向协议中引入了信息性状态码，范围为 100~199）。这里列举几个常见的：

  - *200：响应成功*
  - *302：跳转，跳转地址通过响应头中的 Location 属性指定（JSP 中 Forward 和 Redirect 之间的区别）*
  - *400：客户端请求有语法错误，不能被服务器识别*
  - *403：服务器接收到请求，但是拒绝提供服务（认证失败）*
  - *404：请求资源不存在*
  - *500：服务器内部错误*

- *响应头部*

  与请求头部类似，为响应报文添加了一些附加信息。常见响应头部如下：

  - *Server：服务器应用程序软件的名称和版本*
  - *Content-Type：响应正文的类型（是图片还是二进制字符串）*
  - *Content-Length：响应正文长度*
  - *Content-Charset：响应正文使用的编码*
  - *Content-Encoding：响应正文使用的数据压缩格式*

- *Content-Language：响应正文使用的语言*

6 问题解决

- **代码运行时，页面颜色无法改变**

> 页面颜色无法改变，主要原因就是缺少CGI，我们的color.cgi文件中，需要给出正确的perl命令的路径。
>
> Centos运行tinyhttpd源码运行与分析（解决代码运行+页面颜色无法改变的问题）_九筒的博客-CSDN博客