最新版Web服务器项目详解 - 06 http连接处理（下）

原创 互联网猿 两猿社 2020-04-21 14:30

点击关注上方"两猿社"
设为"置顶或星标"，干货第一时间送达。

互联网猿|两猿社

本文内容

上一篇详解中，我们对状态机和服务器解析请求报文进行了介绍。

本篇，我们将介绍服务器如何响应请求报文，并将该报文发送给浏览器端。首先介绍一些基础API，然后结合流程图和代码对服务器响应请求报文进行详解。

基础API部分，介绍 `stat` 、 `mmap` 、 `iovec` 、 `writev` 。

流程图部分，描述服务器端响应请求报文的逻辑，各模块间的关系。

代码部分，结合代码对服务器响应请求报文进行详解。

基础API

为了更好的源码阅读体验，这里提前对代码中使用的一些API进行简要介绍，更丰富的用法可以自行查阅资料。

stat

stat函数用于取得指定文件的文件属性，并将文件属性存储在结构体stat里，这里仅对其中用到的成员进行介绍。

```
1  #include <sys/types.h>
2  #include <sys/stat.h>
3  #include <unistd.h>
4
```

```
//获取文件属性，存储在statbuf中
int stat(const char *pathname, struct stat *statbuf);

struct stat
{
    mode_t    st_mode;        /* 文件类型和权限 */
    off_t     st_size;        /* 文件大小，字节数*/
};
```

mmap

用于将一个文件或其他对象映射到内存，提高文件的访问速度。

```
void* mmap(void* start,size_t length,int prot,int flags,int fd,off_t offset);
int munmap(void* start,size_t length);
```

- start：映射区的开始地址，设置为0时表示由系统决定映射区的起始地址
- length：映射区的长度
- prot：期望的内存保护标志，不能与文件的打开模式冲突
  - PROT_READ 表示页内容可以被读取
- flags：指定映射对象的类型，映射选项和映射页是否可以共享
  - MAP_PRIVATE 建立一个写入时拷贝的私有映射，内存区域的写入不会影响到原文件
- fd：有效的文件描述符，一般是由open()函数返回
- off_toffset：被映射对象内容的起点

iovec

定义了一个向量元素，通常，这个结构用作一个多元素的数组。

```
struct iovec {
    void      *iov_base;      /* starting address of buffer */
    size_t    iov_len;        /* size of buffer */
};
```

- iov_base指向数据的地址
- iov_len表示数据的长度

writev

writev函数用于在一次函数调用中写多个非连续缓冲区，有时也将这该函数称为聚集写。

```
#include <sys/uio.h>
ssize_t writev(int filedes, const struct iovec *iov, int iovcnt);
```

- filedes表示文件描述符
- iov为前述io向量机制结构体iovec
- iovcnt为结构体的个数

若成功则返回已写的字节数，若出错则返回-1。 writev 以顺序 iov[0] ， iov[1] 至 iov[iovcnt-1] 从缓冲区中聚集输出数据。 writev 返回输出的字节总数，通常，它应等于所有缓冲区长度之和。

特别注意： 循环调用writev时，需要重新处理iovec中的指针和长度，该函数不会对这两个成员做任何处理。writev的返回值为已写的字节数，但这个返回值“实用性”并不高，因为参数传入的是iovec数组，计量单位是iovcnt，而不是字节数，我们仍然需要通过遍历iovec来计算新的基址，另外写入数据的“结束点”可能位于一个iovec的中间某个位置，因此需要调整临界iovec的io_base和io_len。

流程图

浏览器端发出HTTP请求报文，服务器端接收该报文并调用 process_read 对其进行解析，根据解析结果 HTTP_CODE ，进入相应的逻辑和模块。

其中，服务器子线程完成报文的解析与响应；主线程监测读写事件，调用 read_once 和 http_conn::write 完成数据的读取与发送。

HTTP_CODE含义

表示HTTP请求的处理结果，在头文件中初始化了八种情形，在报文解析与响应中只用到了七种。

- NO_REQUEST
  - 请求不完整，需要继续读取请求报文数据
  - 跳转主线程继续监测读事件
- GET_REQUEST
  - 获得了完整的HTTP请求
  - 调用do_request完成请求资源映射
- NO_RESOURCE
  - 请求资源不存在
  - 跳转process_write完成响应报文
- BAD_REQUEST
  - HTTP请求报文有语法错误或请求资源为目录
  - 跳转process_write完成响应报文
- FORBIDDEN_REQUEST
  - 请求资源禁止访问，没有读取权限

    - 跳转process_write完成响应报文
- FILE_REQUEST
    - 请求资源可以正常访问
    - 跳转process_write完成响应报文
- INTERNAL_ERROR
    - 服务器内部错误，该结果在主状态机逻辑switch的default下，一般不会触发

代码分析

do_request

`process_read` 函数的返回值是对请求的文件分析后的结果，一部分是语法错误导致的 `BAD_REQUEST` ，一部分是 `do_request` 的返回结果.该函数将网站根目录和 `url` 文件拼接，然后通过stat判断该文件属性。另外，为了提高访问速度，通过mmap进行映射，将普通文件映射到内存逻辑地址。

为了更好的理解请求资源的访问流程，这里对各种各页面跳转机制进行简要介绍。其中，浏览器网址栏中的字符，即 `url` ，可以将其抽象成 `ip:port/xxx` ， `xxx` 通过 `html` 文件的 `action` 属性进行设置。

m_url为请求报文中解析出的请求资源，以/开头，也就是 `/xxx` ，项目中解析后的m_url有8种情况。

- /
    - GET请求，跳转到judge.html，即欢迎访问页面
- /0
    - POST请求，跳转到register.html，即注册页面
- /1
    - POST请求，跳转到log.html，即登录页面
- /2CGISQL.cgi
    - POST请求，进行登录校验
    - 验证成功跳转到welcome.html，即资源请求成功页面
    - 验证失败跳转到logError.html，即登录失败页面
- /3CGISQL.cgi
    - POST请求，进行注册校验
    - 注册成功跳转到log.html，即登录页面
    - 注册失败跳转到registerError.html，即注册失败页面
- /5
    - POST请求，跳转到picture.html，即图片请求页面
- /6
    - POST请求，跳转到video.html，即视频请求页面
- /7
    - POST请求，跳转到fans.html，即关注页面

如果大家对上述设置方式不理解，不用担心。具体的登录和注册校验功能会在第12节进行详解，到时候还会针对html进行介绍。

```
//网站根目录，文件夹内存放请求的资源和跳转的html文件
const char* doc_root="/home/qgy/github/ini_tinywebserver/root";

http_conn::HTTP_CODE http_conn::do_request()
{
    //将初始化的m_real_file赋值为网站根目录
    strcpy(m_real_file,doc_root);
    int len=strlen(doc_root);

    //找到m_url中/的位置
    const char *p = strrchr(m_url, '/');

    //实现登录和注册校验
    if(cgi==1 && (*(p+1) == '2' || *(p+1) == '3'))
    {
        //根据标志判断是登录检测还是注册检测

        //同步线程登录校验

        //CGI多进程登录校验
    }

    //如果请求资源为/0，表示跳转注册界面
    if(*(p+1) == '0'){
        char *m_url_real = (char *)malloc(sizeof(char) * 200);
        strcpy(m_url_real,"/register.html");

        //将网站目录和/register.html进行拼接，更新到m_real_file中
        strncpy(m_real_file+len,m_url_real,strlen(m_url_real));

        free(m_url_real);
    }
    //如果请求资源为/1，表示跳转登录界面
    else if( *(p+1) == '1'){
        char *m_url_real = (char *)malloc(sizeof(char) * 200);
        strcpy(m_url_real,"/log.html");

        //将网站目录和/log.html进行拼接，更新到m_real_file中
        strncpy(m_real_file+len,m_url_real,strlen(m_url_real));

        free(m_url_real);
    }
    else
        //如果以上均不符合，即不是登录和注册，直接将url与网站目录拼接
        //这里的情况是welcome界面，请求服务器上的一个图片
        strncpy(m_real_file+len,m_url,FILENAME_LEN-len-1);

    //通过stat获取请求资源文件信息，成功则将信息更新到m_file_stat结构体
    //失败返回NO_RESOURCE状态，表示资源不存在
    if(stat(m_real_file,&m_file_stat)<0)
        return NO_RESOURCE;

    //判断文件的权限，是否可读，不可读则返回FORBIDDEN_REQUEST状态
    if(!(m_file_stat.st_mode&S_IROTH))
        return FORBIDDEN_REQUEST;
    //判断文件类型，如果是目录，则返回BAD_REQUEST，表示请求报文有误
    if(S_ISDIR(m_file_stat.st_mode))
        return BAD_REQUEST;

    //以只读方式获取文件描述符，通过mmap将该文件映射到内存中
    int fd=open(m_real_file,O_RDONLY);
    m_file_address=(char*)mmap(0,m_file_stat.st_size,PROT_READ,MAP_PRIVATE,fd,0);

```

```
64        //避免文件描述符的浪费和占用
65        close(fd);
66
67        //表示请求文件存在，且可以访问
68        return FILE_REQUEST;
69    }
```

process_write

根据 `do_request` 的返回状态，服务器子线程调用 `process_write` 向 `m_write_buf` 中写入响应报文。

- add_status_line函数，添加状态行：http/1.1 状态码 状态消息
- add_headers函数添加消息报头，内部调用add_content_length和add_linger函数
  - content-length记录响应报文长度，用于浏览器端判断服务器是否发送完数据
  - connection记录连接状态，用于告诉浏览器端保持长连接
- add_blank_line添加空行

上述涉及的5个函数，均是内部调用 `add_response` 函数更新 `m_write_idx` 指针和缓冲区 `m_write_buf` 中的内容。

```cpp
bool http_conn::add_response(const char* format,...)
{
    //如果写入内容超出m_write_buf大小则报错
    if(m_write_idx>=WRITE_BUFFER_SIZE)
        return false;

    //定义可变参数列表
    va_list arg_list;

    //将变量arg_list初始化为传入参数
    va_start(arg_list,format);

    //将数据format从可变参数列表写入缓冲区写，返回写入数据的长度
    int len=vsnprintf(m_write_buf+m_write_idx,WRITE_BUFFER_SIZE-1-m_write_idx,form

    //如果写入的数据长度超过缓冲区剩余空间，则报错
    if(len>=(WRITE_BUFFER_SIZE-1-m_write_idx)){
        va_end(arg_list);
        return false;
    }

    //更新m_write_idx位置
    m_write_idx+=len;
    //清空可变参列表
    va_end(arg_list);

    return true;
}

//添加状态行
bool http_conn::add_status_line(int status,const char* title)
{
    return add_response("%s %d %s\r\n","HTTP/1.1",status,title);
}

//添加消息报头，具体的添加文本长度、连接状态和空行
bool http_conn::add_headers(int content_len)
{
    add_content_length(content_len);
    add_linger();
    add_blank_line();
}

//添加Content-Length，表示响应报文的长度
bool http_conn::add_content_length(int content_len)
{
    return add_response("Content-Length:%d\r\n",content_len);
}

//添加文本类型，这里是html
bool http_conn::add_content_type()
{
    return add_response("Content-Type:%s\r\n","text/html");
}

//添加连接状态，通知浏览器端是保持连接还是关闭
bool http_conn::add_linger()
{
    return add_response("Connection:%s\r\n",(m_linger==true)?"keep-alive":"close")
}
//添加空行
bool http_conn::add_blank_line()
{
    return add_response("%s","\r\n");
}

```

```
//添加文本content
bool http_conn::add_content(const char* content)
{
    return add_response("%s",content);
}
```

响应报文分为两种，一种是请求文件的存在，通过 io 向量机制 iovec ，声明两个 iovec ，第一个指向 m_write_buf ，第二个指向 mmap 的地址 m_file_address ；一种是请求出错，这时候只申请一个 iovec ，指向 m_write_buf 。

- iovec是一个结构体，里面有两个元素，指针成员iov_base指向一个缓冲区，这个缓冲区是存放的是writev将要发送的数据。
- 成员iov_len表示实际写入的长度

```
bool http_conn::process_write(HTTP_CODE ret)
{
    switch(ret)
    {
        //内部错误，500
        case INTERNAL_ERROR:
        {
            //状态行
            add_status_line(500,error_500_title);
            //消息报头
            add_headers(strlen(error_500_form));
            if(!add_content(error_500_form))
                return false;
            break;
        }
        //报文语法有误，404
        case BAD_REQUEST:
        {
            add_status_line(404,error_404_title);
            add_headers(strlen(error_404_form));
            if(!add_content(error_404_form))
                return false;
            break;
        }
        //资源没有访问权限，403
        case FORBIDDEN_REQUEST:
        {
            add_status_line(403,error_403_title);
            add_headers(strlen(error_403_form));
            if(!add_content(error_403_form))
                return false;
            break;
        }
        //文件存在，200
        case FILE_REQUEST:
        {
            add_status_line(200,ok_200_title);
            //如果请求的资源存在
            if(m_file_stat.st_size!=0)
            {
                add_headers(m_file_stat.st_size);
                //第一个iovec指针指向响应报文缓冲区，长度指向m_write_idx
                m_iv[0].iov_base=m_write_buf;
                m_iv[0].iov_len=m_write_idx;
                //第二个iovec指针指向mmap返回的文件指针，长度指向文件大小
                m_iv[1].iov_base=m_file_address;
                m_iv[1].iov_len=m_file_stat.st_size;
```

```
                m_iv_count=2;
                //发送的全部数据为响应报文头部信息和文件大小
                bytes_to_send = m_write_idx + m_file_stat.st_size;
                return true;
            }
            else
            {
                //如果请求的资源大小为0，则返回空白html文件
                const char* ok_string="<html><body></body></html>";
                add_headers(strlen(ok_string));
                if(!add_content(ok_string))
                    return false;
            }
        }
        default:
            return false;
    }
    //除FILE_REQUEST状态外，其余状态只申请一个iovec，指向响应报文缓冲区
    m_iv[0].iov_base=m_write_buf;
    m_iv[0].iov_len=m_write_idx;
    m_iv_count=1;
    return true;
}
```

http_conn::write

服务器子线程调用 process_write 完成响应报文，随后注册 epollout 事件。服务器主线程检测写事件，并调用 http_conn::write 函数将响应报文发送给浏览器端。

该函数具体逻辑如下：

在生成响应报文时初始化byte_to_send，包括头部信息和文件数据大小。通过writev函数循环发送响应报文数据，根据返回值更新byte_have_send和iovec结构体的指针和长度，并判断响应报文整体是否发送成功。

- 若writev单次发送成功，更新byte_to_send和byte_have_send的大小，若响应报文整体发送成功，则取消mmap映射,并判断是否是长连接.
  - 长连接重置http类实例，注册读事件，不关闭连接，
  - 短连接直接关闭连接
- 若writev单次发送不成功，判断是否是写缓冲区满了。
  - 若不是因为缓冲区满了而失败，取消mmap映射，关闭连接
  - 若eagain则满了，更新iovec结构体的指针和长度，并注册写事件，等待下一次写事件触发（当写缓冲区从不可写变为可写，触发epollout），因此在此期间无法立即接收到同一用户的下一请求，但可以保证连接的完整性。

```
bool http_conn::write()
{
    int temp = 0;

    int newadd = 0;

    //若要发送的数据长度为0
    //表示响应报文为空，一般不会出现这种情况
    if(bytes_to_send==0)
    {
        modfd(m_epollfd,m_sockfd,EPOLLIN);
```

```
12          init();
13          return true;
14      }
15
16      while (1)
17      {
18          //将响应报文的状态行、消息头、空行和响应正文发送给浏览器端
19          temp=writev(m_sockfd,m_iv,m_iv_count);
20
21          //正常发送，temp为发送的字节数
22          if (temp > 0)
23          {
24              //更新已发送字节
25              bytes_have_send += temp;
26              //偏移文件iovec的指针
27              newadd = bytes_have_send - m_write_idx;
28          }
29          if (temp <= -1)
30          {
31              //判断缓冲区是否满了
32              if (errno == EAGAIN)
33              {
34                  //第一个iovec头部信息的数据已发送完，发送第二个iovec数据
35                  if (bytes_have_send >= m_iv[0].iov_len)
36                  {
37                      //不再继续发送头部信息
38                      m_iv[0].iov_len = 0;
39                      m_iv[1].iov_base = m_file_address + newadd;
40                      m_iv[1].iov_len = bytes_to_send;
41                  }
42                  //继续发送第一个iovec头部信息的数据
43                  else
44                  {
45                      m_iv[0].iov_base = m_write_buf + bytes_to_send;
46                      m_iv[0].iov_len = m_iv[0].iov_len - bytes_have_send;
47                  }
48                  //重新注册写事件
49                  modfd(m_epollfd, m_sockfd, EPOLLOUT);
50                  return true;
51              }
52              //如果发送失败，但不是缓冲区问题，取消映射
53              unmap();
54              return false;
55          }
56
57          //更新已发送字节数
58          bytes_to_send -= temp;
59
60          //判断条件，数据已全部发送完
61          if (bytes_to_send <= 0)
62          {
63              unmap();
64
65              //在epoll树上重置EPOLLONESHOT事件
66              modfd(m_epollfd,m_sockfd,EPOLLIN);
67
68              //浏览器的请求为长连接
69              if(m_linger)
70              {
71                  //重新初始化HTTP对象
72                  init();
73                  return true;
74              }
75              else
76              {
77                  return false;
78              }
```

```
79              }
80          }
81      }
```

书中原代码的write函数不严谨，这里对其中的Bug进行了修复，可以正常传输大文件。

后续，我会写一篇推文，对大文件传输Bug定位、解决思路的代码实现进行介绍。

如果本文对你有帮助， 阅读原文 star一下服务器项目，我们需要你的星星^_^.

完。

Web服务器-原始版本 13

Web服务器-原始版本 · 目录

上一篇
最新版Web服务器项目详解 - 05 http连接处理（中）

下一篇
最新版Web服务器项目详解 - 07 定时器处理非活动连接（上）

阅读原文

喜欢此内容的人还喜欢

项目搞成现在这个样子，我有责任
两猿社