一、 目的和要求

了解 HTTP 协议，实现一个简单的 Web 服务器，要求如下：

1、 用 C/C++、C#、Java 这三类编程语言中任一种开发

2、 只能基于 Socket 功能，不能使用现成的 HTTP 解析类

3、 能够让浏览器正确显示 web.zip 这个文件中的网页

二、 设计实现

2.1 业务逻辑

1、 当浏览器连接服务器时，创建一个连接套接字；

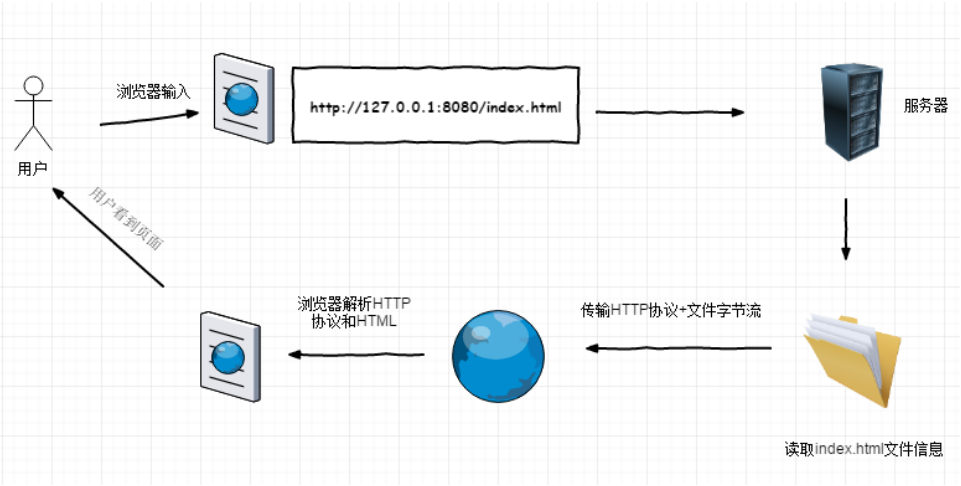
2、 从这个连接套接字接收 HTTP 请求，包括 GET 和 POST 方法；

3、 服务器对 HTTP 请求包进行解析，确定所请求的文件或者文件；

4、 如果是静态文件则直接从文件系统获取，如果是 php 程序，则启动命令行执行程序并将结果返回；

5、 组装 HTTP 响应报文；

6、 服务器通过连接套接字向浏览器发送响应报文。如果浏览器请求一个在服务器上不存在的文件，则返回 404 NOT FOUND 的差错报文。



2.2 架构与设计

项目使用 Java 进行开发，包括 2 个类，分别是 HttpServer 和 ServerThread。

HttpServer 类是程序的主入口。服务器实例化了一个 ServerSocket 对象，守听在 8080 端口并等待浏览器的连接。当浏览器连接服务器后，服务器为每一个 socket 单独创建一个线程，并使用 ArrayList 进行管理维护。主程序会等待所有子进程结束后才会退出。

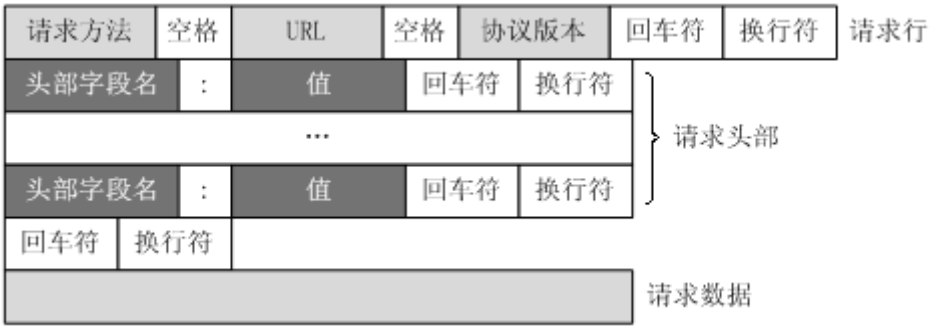
ServerThread 类继承自 Thread 类，为每一个 socket 连接提供单独的线程。这个类有 3 个变量，分别是 socket 对象、输入流和输出流。服务器从输入流中获取浏览器发来的 HTTP 请求报文，通过输出流向浏览器发送响应

报文。对于请求报文，服务器从输入流中进行逐行循环读取并解析。对于 GET 请求，服务器在读取第一行之后就跳出循环并对包头的第一行进行解析出文件路径并获取到对应的静态文件，然后组装响应包并写入到输出流中。对于 POST 请求，服务器需要读完整个包并获取到请求文件和报文内容这两个关键信息。如果请求的是 php 程序，则启动命令行执行 php-cgi 程序并将报文内容中的参数传入。然后根据程序的执行结果添加报文头并组装好响应报文写入到输入流中。

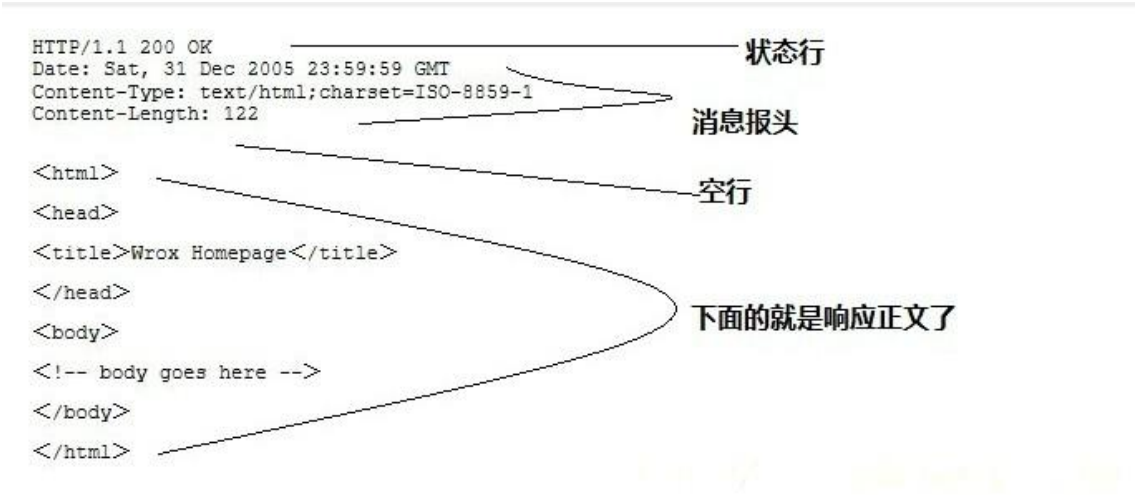
2.3 重难点解决

2.3.1 HTTP 消息结构

HTTP 请求消息由请求行（request line）、请求头部（header）、空行和请求数据四个部分组成。



HTTP 响应由状态行、消息报头、空行和响应正文四个部分组成。



2.3.2 如何解析 php 文件

解析静态文件应该是这个实验中最难的一个部分。与静态文件不同，服务器需要执行 php 文件然后再将结果返回。刚开始我以为只要我在程序中使用 Runtime.exec()方法调用外部 php 程序执行对应的 php 文件就能解决这个问题。但后来碰到了一个问题就是我如何将 php 文件中所需的 post 参数传入。因为我们知道，执行 php 文件中传入的命令行参数是通过全局变量$argv 获取，但是 php 文件中的参数却是$_post，这就不太好操作了。

后来查看了 Nginx 的处理方式，发现它是将动态请求转向后端 php-fpm 来处理。因此我对这一过程进行了一个简单的复现。

（1） cgi 协议

为了解决不同的语言解释器(如 php、python 解释器)与 webserver 的通信，于是出现了 cgi 协议。只要你按

照 cgi 协议去编写程序，就能实现语言解释器与 webwerver 的通信。如 php-cgi 程序。

（2） fast-cgi 的改进

有了 cgi 协议，解决了 php 解释器与 webserver 通信的问题，webserver 终于可以处理动态语言了。

但是，webserver 每收到一个请求，都会去 fork 一个 cgi 进程，请求结束再 kill 掉这个进程。这样有 10000 个请求，就需要 fork、kill php-cgi 进程 10000 次。这样很浪费资源。

于是，出现了 cgi 的改良版本，fast-cgi。fast-cgi 每次处理完请求后，不会 kill 掉这个进程，而是保留这个进程，使这个进程可以一次处理多个请求。这样每次就不用重新 fork 一个进程了，大大提高了效率。

（3） php-fpm

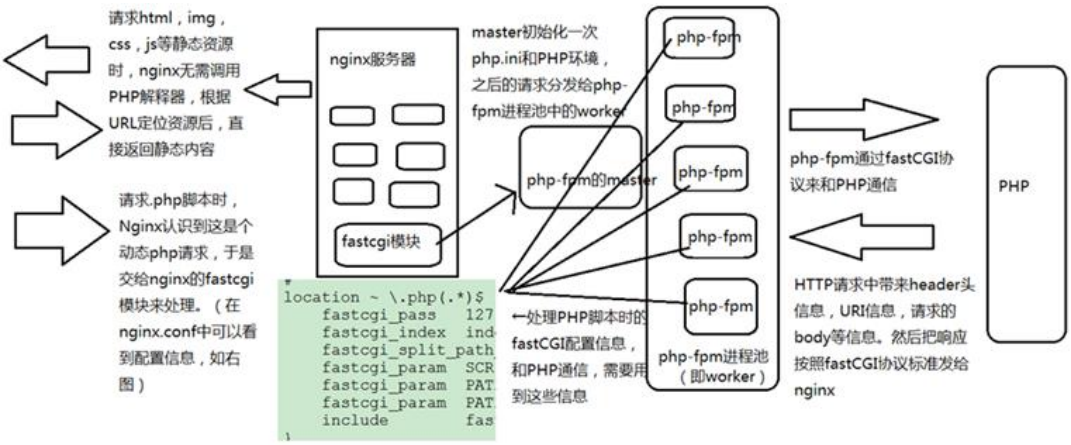
php-fpm 即 php-Fastcgi Process Manager.

php-fpm 是 FastCGI 的实现，并提供了进程管理的功能。

进程包含 master 进程和 worker 进程两种进程。

master 进程只有一个，负责监听端口，接收来自 Web Server 的请求，而 worker 进程则一般有多个(具体数量根据实际需要配置)，每个进程内部都嵌入了一个 PHP 解释器，是 PHP 代码真正执行的地方。

（4） Nginx 与 php-fpm 的工作流程：



（5） Windows 下使用 php-cgi 复现这一过程

```
set REDIRECT_STATUS=true
set SCRIPT_FILENAME=C:\Users\pilgr\IdeaProjects\WebServer\web\login.php
set REQUEST_METHOD=POST
set GATEWAY_INTERFACE=CGI/1.1
set CONTENT_LENGTH=14
set CONTENT_TYPE=application/x-www-form-urlencoded
echo  "loginName=carl" | D:\Wampserver\wamp64\bin\php\php7.2.10\php-cgi.exe
pause
```

在 dos 通过 set 和在 linux 通过 export 设置环境变量，在 php 中可以通过$_SERVER['环境变量名']去获取。

$_SERVER['QUERY_STRING']会被解析到$_GET 全局变量中

$_SERVER['HTTP_COOKIE']会被解析到$_COOKIE 全局变量中

$_SERVER['REQUEST_METHOD'] =”POST”

$_SERVER['CONTENT_LENGTH'] =16

$_SERVER['CONTENT_TYPE']=”application/x-www-form-urlencoded”设置了这三个变量，php 会从管道中读取 16 个字节，并解析到$_POST 全局变量。

2.3.3 readLine()可能造成的阻塞问题

在处理 POST 报文时，发现服务器总是收不到浏览器发来的报文体，当连接断开时，才会收到报文体。debug 后发现每次 readLine 函数读完报文头部之后的空行后，就会被阻塞。查阅文档后了解到，该方法读取一行文本，当遇到换行符”\n”,回车符”\r”或者回车符后面紧跟着换行符时，该行结束并返回。没有数据时，将会一直处于等待状态。因此在进行网络连接时，应该避免使用该方法。

最后我采用的处理方法是在使用 readLine()函数读取到前面的内容，最后一行使用 readLine 循环读取并拼接成字符串就可以避免阻塞，顺利拿到报文内容。

三、 运行环境

CPU: Intel(R)Core(TM)i5-5200U CPU@2.20GHz

Memory: 8GB(DDR3L 1600MHZ)

Operating System: Windows 10

Compiler: jdk1.8.0

IDE: IntelliJ IDEA

四、 操作方法与实验步骤

1. 运行 HttpServer.java；
2. 打开浏览器，输入 http://localhost:8080/web/index.html；
3. 输入登录名和密码，点击登录。

注意事项：

1. HttpServer 是程序的主入口，IDE 建议使用 IntelliJ IDEA；
2. 运行此程序前，请将 php-cgi 添加到环境变量中；
3. 默认服务器的资源文件在项目根目录下。如果不在同一位置，请在代码中修改相应的位置。

五、 实验结果与测试

六、 开发体会与小结

这次中期实验很有意义，让我对服务器的工作原理特别是对 php 的动态解析过程有了更加深入的了解。以前做过支持静态文件代理的服务器，所以这次实验并没有给我带来太多的困难。

这次实验有 2 个关键点，首先是服务器如何解析浏览器发来的 Http 请求报文并组装响应报文，这需要我对 Http 的消息结构有一个全面的了解。其次是如何动态解析 PHP，最后我采用的是类似于 Nginx 的处理方式，

使用 cgi 协议实现服务器与 php 的交互。

这次实验还有值得改进的地方是我在管理每一个 socket 线程时使用的是 ArrrayList，但是这种方式在连接数增加的情况下会浪费很多线程资源。之后有时间的话，我会重构这部分代码，使用线程池来维护这些线程。