华东师范大学数据科学与工程学院上机实践报告

课程名称：计算机网络原理与编程	年级：2018	上机实践成绩:
指导教师：张召	姓名：孙秋实	学号：10185501402
上机实践名称：Web Server		上机实践日期：2020/7/31
上机实践编号：Final Project	组号：	上机实践时间：

Part 1
实验目的

- **使用 Java 语言开发一个 Web 服务器，满足以下要求**
  1. 使用 ServerSocket 和 Socket 实现
  2. 使用多线程接管
  3. 在浏览器中输入 localhost:8081/index.html 能显示学号
  4. 在浏览器中输入 localhost:8081 下其他无效路径能显示 404 Not Found
  5. 使用 JMeter 进行性能测试
- **使用 Java 语言开发一个 Web 代理服务器，满足以下要求**
  1. 当你的代理服务器从一个浏览器接收到对某对象的 HTTP 请求，它生成对相同对象的一个新的 HTTP 请求并向初始服务器发送
  2. 当该代理从初始服务器接收到具有该对象的 HTTP 响应时，它生成一个包括该对象的新 HTTP 相应并发送给客户
  3. 这个代理是多线程的，使其在相同时间能够处理多个请求
- **额外任务**
  1. 进行压力测试
  2. 分析现有能支持同时连接的最大数，使其能同时支持 1000 个连接

Part 2
实验任务

- **使用 Netty 框架搭建服务器**

- **实现 Proxy**
- **使用 JMeter 进行压力测试，分析结果**

Part 3
使用环境

- IntelliJ IDEA 2019.3
- JDK 版本 11.0.6

Part 4
实验过程

Section 1
NIO 概念

NIO 即 New IO(Non-blocking IO)，NIO 和 IO 有相同的作用以及目的，但 NIO 和 IO 实现方式不同，NIO 主要用到的是块，效率要比 IO 高很多，其主要差异如下

	IO	NIO
1	面向流	面向缓冲
2	阻塞 IO	非阻塞 IO
3	无	选择器

- Java IO 和 NIO 之间第一个最大的区别是，IO 是面向流的，NIO 是面向缓冲区的。Java IO 面向流意味着每次从流中读一个或多个字节，直至读取所有字节，它们没有被缓存在任何地方。此外，它不能前后移动流中的数据。如果需要前后移动从流中读取的数据，需要先将它缓存到一个缓冲区。Java NIO 的缓冲导向方法略有不同。数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区中前后移动。这就增加了处理过程中的灵活性。但是，还需要检查是否该缓冲区中包含所有您需要处理的数据。而且，需确保当更多的数据读入缓冲区时，不要覆盖缓冲区里尚未处理的数据。
- Java IO 的各种流是阻塞的。这意味着，当一个线程调用 read() 或 write() 时，该线程被阻塞，直到有一些数据被读取，或数据完全写入。该线程在此期间不能再干任何事情了。Java NIO 的非阻塞模式，使一个线程从某通道发送请求读取数据，但是它仅能得到目前可用的数据，如果目前没有数据可用时，就什么都不会获取，而不是保持线程阻塞，所以直至数据变的可以读取之前，该线程可以继续做其他的事情。非阻塞写也是如此。一个线程请求写入一些数据到某通道，但不需要等待它完全写入，这个线程同时可以去做别的事情。线程通常将非阻塞 IO 的空闲时间用于在其它通道上执行 IO 操作，所以一个单独的线程现在可以管理多个输入和输出通道（channel）。

- Java NIO 的选择器允许一个单独的线程来监视多个输入通道，你可以注册多个通道使用一个选择器，然后使用一个单独的线程来“选择”通道：这些通道里已经有可以处理的输入，或者选择已准备写入的通道。这种选择机制，使得一个单独的线程很容易来管理多个通道。

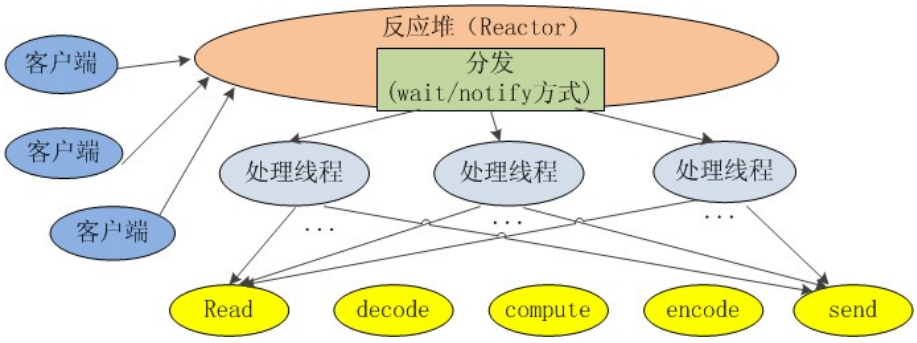


图 1: NIO 原理图（图源网络）

Section 2
使用 idea 进行项目配置

使用 idea 新建一个项目后，在 Project Structure 中进行配置，先在左侧导航栏的 Project Seeting 中选择 Modules，再选择所需库

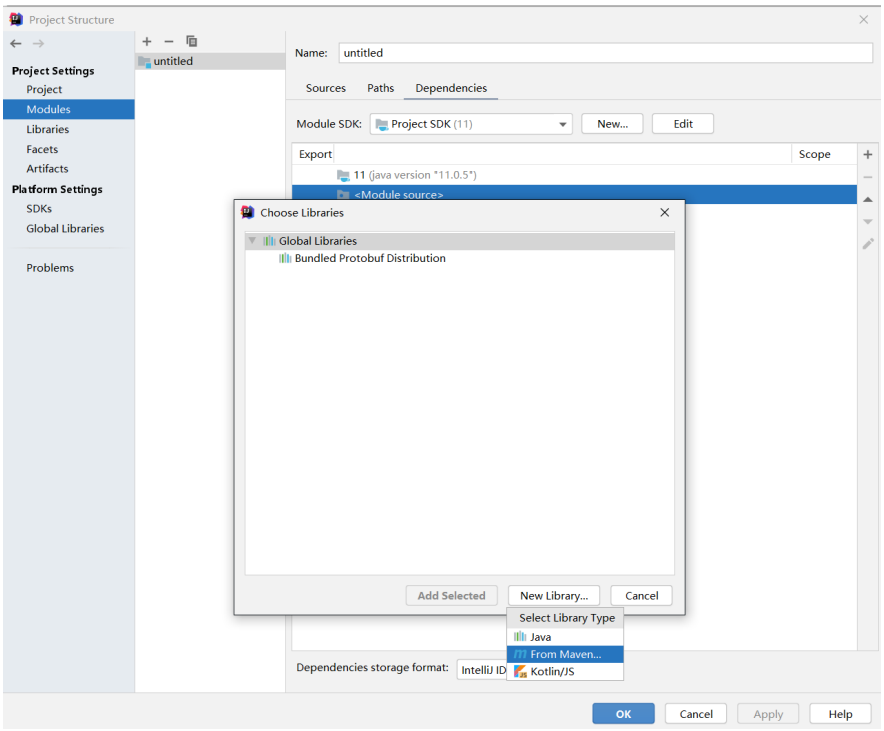


图 2: 使用 maven 型 library

随后，搜索我们所需要的 netty 框架并安装

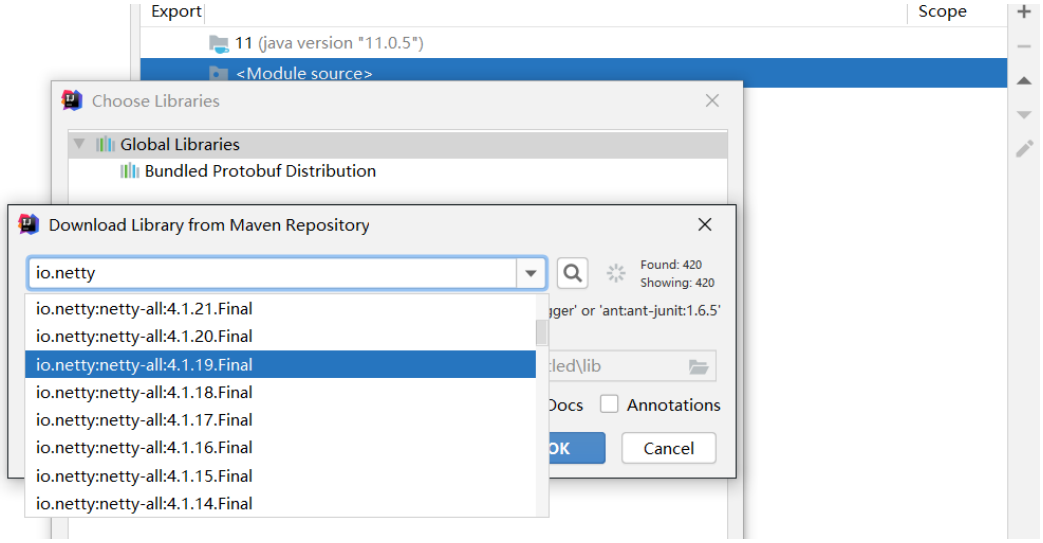


图 3: 选择 io.netty-all

随后进入项目，在文件目录下可以看到已经配置好的 netty 框架

Section 3
实现 Java Web Server

Channel 和 IO 中的 Stream 是同等概念。但是，Stream 是单向的，如 InputStream, OutputStream，而 Channel 是双向的，既可以用来进行读操作，又可以用来进行写操作。

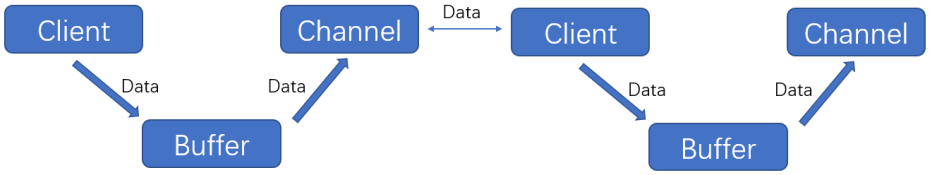


图 4: 缓冲区原理图

首先是 MyHttpServer 类

由于使用了 Netty 框架，所以先导入一些相关的库

```
import io.netty.handler.codec.http.*;
import io.netty.util.CharsetUtil;
import io.netty.buffer.Unpooled;
import io.netty.channel.ChannelFutureListener;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.SimpleChannelInboundHandler;
```

```
public MyHttpServer(int port){
    this.port=port;
}
EventLoopGroup bossGroup = new NioEventLoopGroup();
EventLoopGroup workerGroup = new NioEventLoopGroup();
```

创建一个线程来接收客户端连接，并且使用 EventLoop 接收客户端的连接，然后将 Socket 交给 WorkerEventLoopGroup 做 I/O 处理。

```
public void initChannel(SocketChannel MyChannel)throws Exception{
    MyChannel.pipeline().addLast(new HttpServerHandler());//Http业务实现
    MyChannel.pipeline().addLast(new HttpObjectAggregator(65535));
    MyChannel.pipeline().addLast(new HttpResponseEncoder());
    MyChannel.pipeline().addLast(new HttpRequestDecoder());
    MyChannel.pipeline().addLast(new ChunkedWriteHandler());
    //对于大文件需要支持CHUNKED方式写

}
```

借助 Netty 4.16 来构建服务器端, 其 Channel 的 pipeline 设定如上所示

```
.option(ChannelOption.SO_BACKLOG,128)
.childOption(ChannelOption.SO_KEEPALIVE,true);
//异步处理
        ChannelFuture MyFuture=MyServerBootstrap.bind(port).sync();
        MyFuture.channel().closeFuture().sync();
```

设置线程队列中等待连接的个数，然后初始化通道对象，通过 ChannelFuture 可以获取到 Channel，从而利用 Channel 在通道上进行读写和关闭。

```
finally {
    bossGroup.shutdownGracefully();
    workerGroup.shutdownGracefully();
    }
```

最后要释放资源（“优雅”退出），优雅退出可以：

- 尽快的释放 NIO 线程、句柄等资源；
- 如果使用 flush 做批量消息发送，需要将积攒在发送队列中的待发送消息发送完成；
- 正在 write 或者 read 的消息，需要继续处理；
- 设置在 NIOEventLoop 线程调度器中的定时任务，需要执行或者清理。

接下来是服务端处理程序，主要处理用户的请求

```
    import io.netty.buffer.Unpooled;
    import io.netty.channel.ChannelFutureListener;
    import io.netty.channel.ChannelHandlerContext;
    import io.netty.channel.SimpleChannelInboundHandler;
    import io.netty.handler.codec.http.*;
    import io.netty.util.CharsetUtil;
```

同样地，先导入 Netty 框架所需的库

```
String Current_Uri=fullHttpRequest.uri();
        String ReportMessage="<html><head><title>计算机网络与编程原理大作业</title></head><body>打印URI:
            "+Current_Uri+"</body></html>";
        String Command_Shutdown="/shutdown";
        String Command_StuNum="/index.html";
```

然后是获取请求的 URI

```
if (Current_Uri.equals(Command_Shutdown)){
            channelHandlerContext.close();
            System.exit(1);
        }
        if (Current_Uri.equals(Command_StuNum)){
            ReportMessage="<html><head><title>QiushiSun</title></head><body>My Student ID:10185501402</
                body></html>";
        }
        else if(!Current_Uri.equals("")){
            ReportMessage="<html><head><title>QiushiSun</title></head><body>404 Not Found</body></html>"
                ;
        }
```

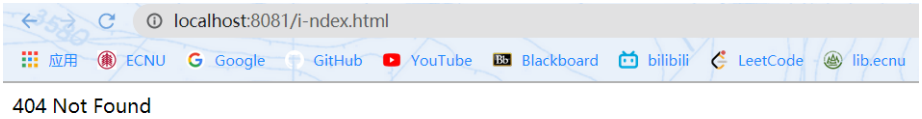
完成一些用户的请求

```
FullHttpResponse Http_Response=new DefaultFullHttpResponse(
                HttpVersion.HTTP_1_1,
                HttpResponseStatus.OK,
                Unpooled.copiedBuffer(ReportMessage, CharsetUtil.UTF_8));
        Http_Response.headers().set(HttpHeaderNames.CONTENT_TYPE,"text/html;charset=UTF-8");
        channelHandlerContext.writeAndFlush(Http_Response).addListener(ChannelFutureListener.CLOSE);
```

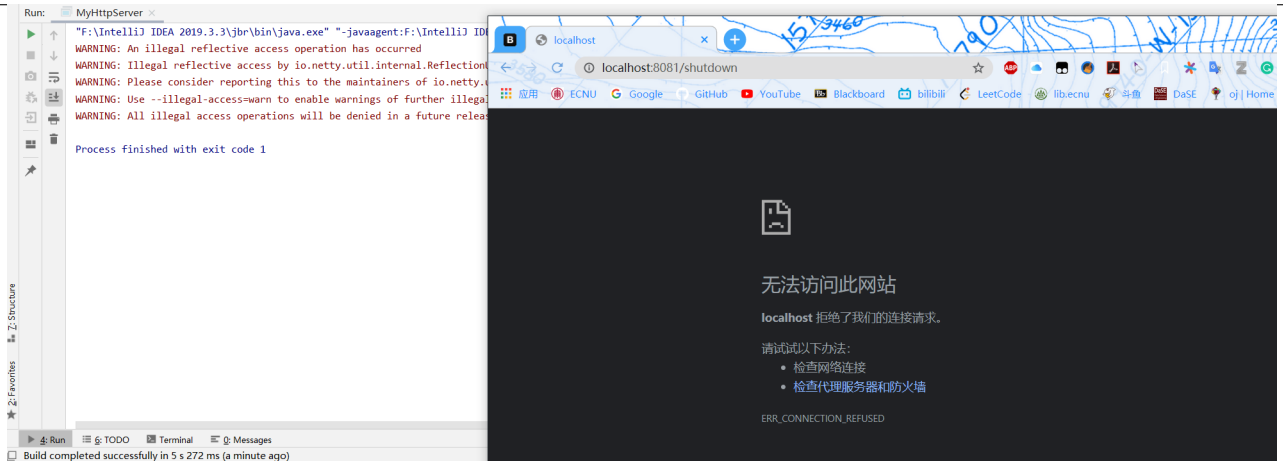
创建 Http 响应

服务器最终实验效果如下所示：

访问 index.html 能显示学号



其他路径则显示 404 Not Found

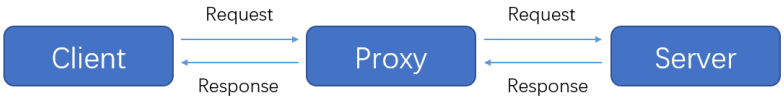


最后，在路径中输入 shutdown 关闭 Java Web 服务器

Section 4
实现 Java Web Server Proxy

实现一个代理服务器，Proxy 是介于用户浏览器和 Web 服务器之间的中间人，启动代理服务器之后，浏览器不直接到 Web 服务器去取回网页而是向代理服务器发出请求，Request 信号会先送到代理服务器，由代理服务器取回浏览器所需要的数据，并传送给你的浏览器。

原理如下图所示




```
1 import java.net.ConnectException;
2 import java.net.Socket;
3 import java.net.SocketException;
4 import java.util.ArrayList;
5 import java.util.List;
6 import java.util.Scanner;
```

首先导入一些需要的包

```
1 private void ToURI(List<String> Buffer)
2 {
```

```
    for (int k = 0; k < Buffer.size(); k++) {
        String CurrentLine = Buffer.get(k);
        String head = CurrentLine.substring(0, CurrentLine.indexOf(' '));
        int Header = head.length() + 1;
        if(CurrentLine.substring(Header, Header + 7).equals("http://")) {
            String CurrentURI = CurrentLine.substring(CurrentLine.indexOf('/', Header + 7));
            Buffer.set(k, head + " " + CurrentURI);
            break;
        }
    }
}
```

我们需要一个转换 URI 的函数

以下是转发部分的主要代码，创建 SocketServer 监听端口，接着根据 HOST 头建立代理服务器与目标服务器的连接，然后转发数据。HTTPS 请求需要特殊处理, 因为 CONNECT 请求并不需要转发，要返回一个 HTTP 200 的响应建立隧道，之后才进行转发。

```
@Override
  public void run() {
    try {
      OutputStream Out = Dest.getOutputStream();
      InputStream In = Src.getInputStream();
      if(buff != null && buff.size() > 0){
        for(String string : buff)
        {
          Out.write((string + "\r\n").getBytes());
          Out.flush();
        }
        Out.write("\r\n".getBytes());
        Out.flush();
      }
      byte[] BufferSize = new byte[2048];
      int Proxyed_Len;
      while ((Proxyed_Len = In.read(BufferSize)) != -1) {
        Out.write(BufferSize, 0, Proxyed_Len);
        Out.flush();
      }
      Out.close();
      In.close();
    } catch (IOException ie) {
      System.err.println("Data Forwarding Failure " + ie);
    }
  }
```

ProxyHandler 的主要部分，启动一个线程，对被代理网站的数据进行转发，每次存储入缓冲区后交给 Server 端，然后清空，往复直至完成所有数据转发

```
public DataForwardHandler(List<String> Proxy_Buffer, Socket src, Socket dest) {
    this.buff = Proxy_Buffer;
    this.Dest = dest ;
    this.Src = src ;
}
```

进行数据转发，首先建立一个 DataForwardHandler

```
public void run() {
    try {
        byte[] BufferSize = new byte[2048];
        int Proxyed_Len=0;
        OutputStream Out = Dest.getOutputStream();
        InputStream In = Src.getInputStream();
        if(buff != null && buff.size() > 0){
            for(String string : buff)
            {
                Out.write((string + "\r\n").getBytes());
                Out.flush();
            }
            Out.write("\r\n".getBytes());
            Out.flush();
        }
        while ((Proxyed_Len = In.read(BufferSize)) != -1) {
            Out.write(BufferSize, 0, Proxyed_Len);
            Out.flush();
        }
        Out.close();
        In.close();
    } catch (IOException df) {
        System.err.println("Data Forwarding Failure " + df);
    }
}
```

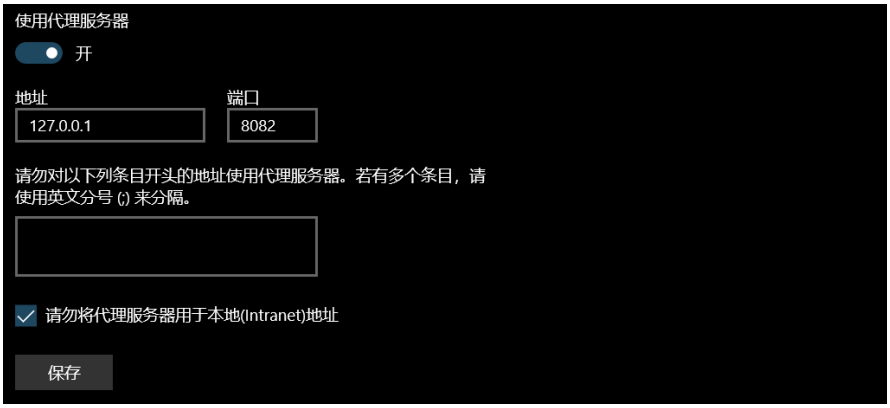
实现数据转发

```
catch (ConnectException c) {
    System.err.println("I/O Time-Out");
} catch (SocketException se) {
    System.err.println("错误！无法连接" + Proxyed_Host + "----->" + Proxyed_Port);
} catch (Exception e) {
    System.err.println("Error" + e);
}
```

进行异常处理

```java
public static void main(String[] args)
{
    Local_Port = 8082;
    try {
        new MyLocalServer().start();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

在主函数中设定端口并启动，还需要在系统中设置代理端口

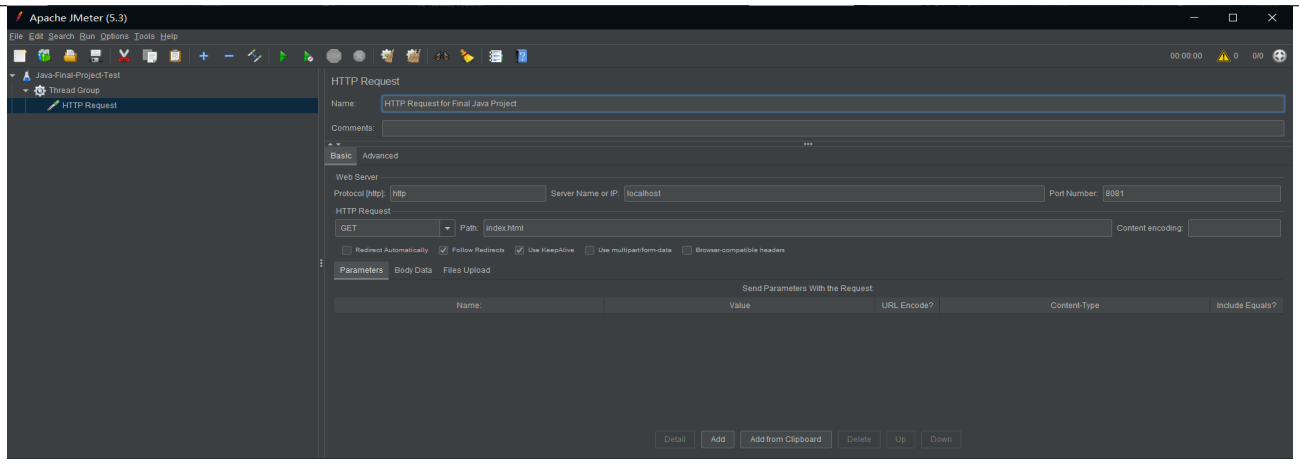



```
"F:\IntelliJ IDEA 2019.3.3\jbr\bin\java.exe" "-javaagent:F:\IntelliJ IDEA 2019.3.3\lib\idea_rt.jar=61637:F:\IntelliJ IDEA 2019.3.3\bin" -Dfile.encoding=UTF-8 -classpath C:\U
This Proxy is Running On Port:8082
fonts.googleapis.com ---> 443  istrue
cn.bing.com ---> 443  istrue
www.bing.com ---> 443  istrue
fonts.gstatic.com ---> 443  istrue
api.mousegesturesapi.com ---> 443  istrue
api.mousegesturesapi.com ---> 443  istrue
fonts.gstatic.com ---> 443  istrue
beacons.gvt2.com ---> 443  istrue
ibaidu.htt5.com ---> 80  isfalse
ibaidu.htt5.com ---> 80  isfalse
ibaidu.htt5.com ---> 80  isfalse
ibaidu.htt5.com ---> 80  isfalse
ibaidu.htt5.com ---> 80  isfalse
ibaidu.htt5.com ---> 80  isfalse
```

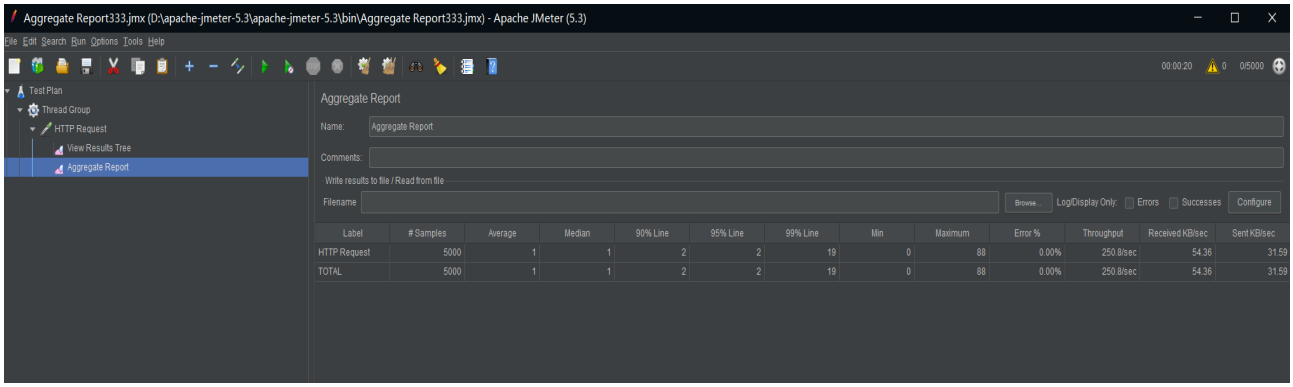
测试结果如上图所示，能代理一部分网站，但是有些动态网站和 Https 网站无法代理

Section 5
性能测试

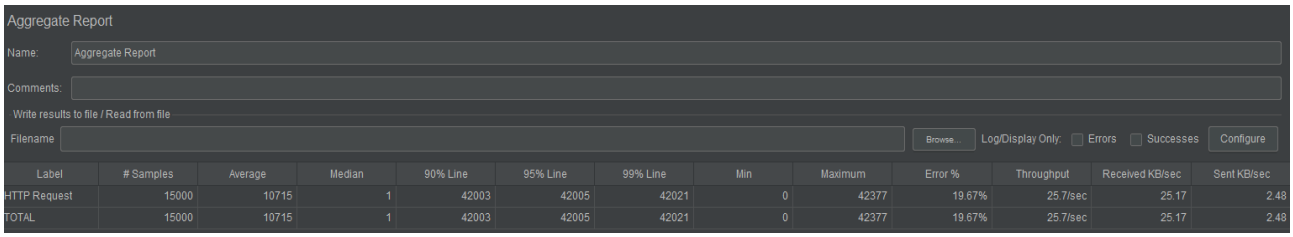
使用 JMeter 进行性能测试



选择 index.html 进行 GET 性能测试



读取信息，记录 Error 和 Throughput（下图仅用于举例）



以下是设置 $ramp-up\ period$ 为 20，$number\ of\ threads$ 分别为 100，500，1000，2000，5000，10000，20000 时的性能测试数据

	性能测试数据	
	Error(%)	Throughput
100(Threads)	0	5.3(tps)
500(Threads)	0	13.4(tps)
1000(Threads)	0	12.1(tps)
2000(Threads)	0	18.5(tps)
5000(Threads)	0	251.5(tps)
10000(Threads)	6.3	107.6(tps)
20000(Threads)	7.6	149.4(tps)

Part 5
实验总结

感觉还是有点难度的，为了提高性能选择使用 Java NIO 实现，使用了 Netty 框架让整个开发变得简单了一些（但中间调试还是花了很久），第二问的代理本来也想用 Netty 框架实现，但是我尝试了很多次后依然没有成功，遂放弃，跟着网上的一些帖子学了传统的代理服务器写法。IntelliJ idea 装包的时候遇到了点问题，可能是 JDK 版本不同导致安装框架时报错多次，后面是自己下载好解压进项目里才解决。在性能测试这一块 Netty 表现真的挺好的，可以支持远超过所要求的 1000 个连接同时进行，当然这是架构的功劳，不过也算是学会了一种新技术。我看网上一些论坛里讨论的都是“百万连接”，显然水平还没有达到那个程度，还需要继续努力。

Part 5
参考资料

主要参考资料如下

- https://www.jianshu.com/p/7ebb533c0ae7
- https://github.com/netty/netty
- https://www.yiibai.com/netty/
- https://uestc-toy.github.io/2020/03/17