Netty 之http协议应用实践

课题概要：

1. Http协议概述
2. 基于Netty 实现Http协议过程分析
3. 基于Netty实现弹幕系统

一、Http协议概述

1、什么是Http协议：

HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统

HTTP协议的主要特点可概括如下：

1.支持客户/服务器模式。

2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

2、Http协议交互过程

协议交互本质是指协议两端（客户端、服务端）如何传输数据？如何交换数据？

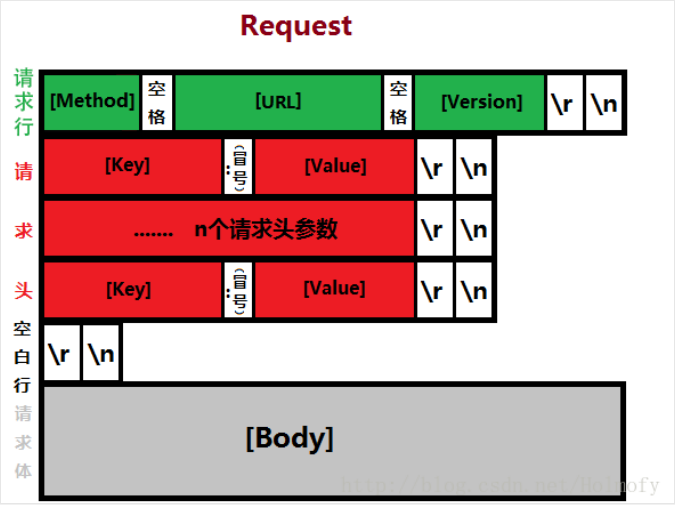
传输数据：

传输数据一般基于TCP/IP 实现，体现到开发语言上就是我们所熟悉的Socket 编程。

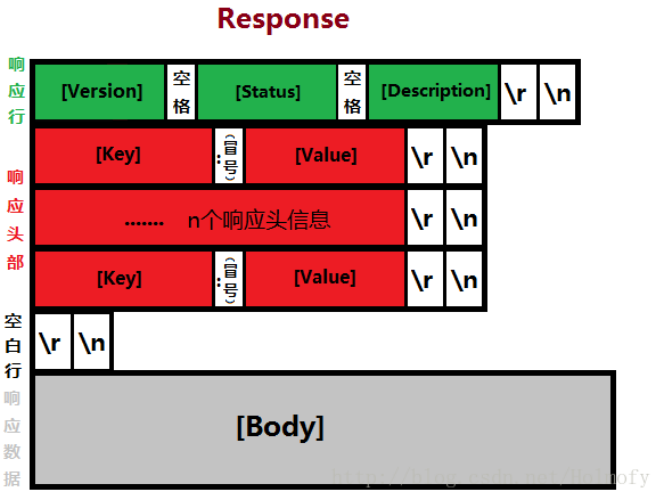
交换数据：

交换数据本质是指，两端（客户端、服务端）能各自识别对方所发送的数据。那么这就需要制定一套报文编码格式，双方以该格式编码数据发送给对方。Http 对应的Request 与Response报文格式如下图：

Request：

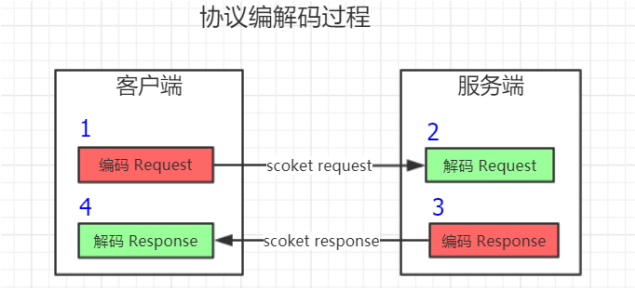


Response：



注：*我们可以通过抓包工具（fiddler）可以直接看到该报文格式*

报文约定好以后两端都需要对其进行解码和编码操作，其过程如下图：



3、Http协议内容组成

请求方法:

序号	方法	描述
1	GET	请求指定的页面信息，并返回实体主体。
2	HEAD	类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
3	POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
4	PUT	从客户端向服务器传送的数据取代指定的文档的内容。
5	DELETE	请求服务器删除指定的页面。
6	CONNECT	HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
7	OPTIONS	允许客户端查看服务器的性能。
8	TRACE	回显服务器收到的请求，主要用于测试或诊断。

部分请求头：

请求头	说明
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

部分响应头：

响应头	说明说明
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	实体报头域用于指明实体正文的长度，以字节方式存储的十进制数字来表示响应正文长度
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

部分响应状态：

状态码	说明
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

二、基于Netty 实现Http协议过程分析

如何实现实现Http协议：

通过上小节内容我们知道实现Http协议分为三部分：

1. 远程数据传输
2. 报文编解码
3. 业务处理

但如果是开发基于Http普通应用，完全没有必要重复造轮子，有现成的容器或组已经实现以经实现上述第1、2部，我们只需实现业务即可。现比较成熟的中间件有：Tomcat、Jetty、Jboos。这些中间有个缺点是较重，如果需要轻量实现可采用：netty 或JDK自还http 实现

JDK Http源码参见：com.sun.net.httpserver.HttpServer

Netty 实现Http Demo演示：

- [ ] 初始ServerBootstrap
- [ ] 通过ChannelInitializer 初始 pipeline
- [ ] 基于SimpleChannelInboundHandler HttpServer处理类

实现过程分析

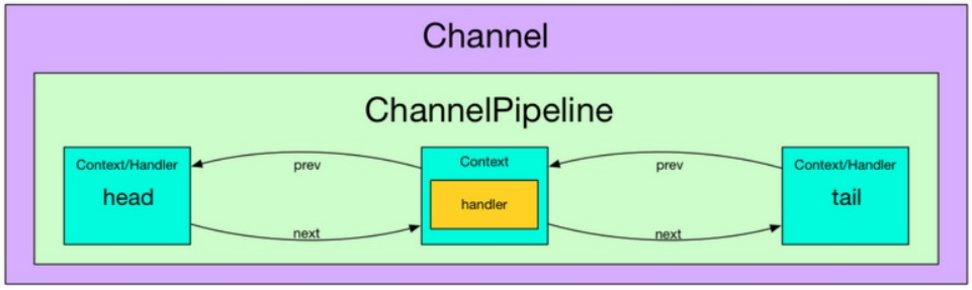
1. 建立连接读取消息流
2. 解码Request
3. 业务处理
4. 编码Response
5. 返回消息关闭连接

Netty是如何连接上述5个步骤的?

Channel 与 ChannelPipeline

1. **Channel**:
   a. ServerSocketChannel
   b. SocketChannel
2. **pipeline**:一个pipeline 当中包含了多个ChandlerHandler,而且是有顺序的
3. **ChandlerHandler**
   a. HttpRequestDecode:解码请求
   b. HttpResponseEncode :编码返回结果

在 Netty 中每个 Channel 都有且仅有一个 ChannelPipeline 与之对应, 它们的组成关系如下:

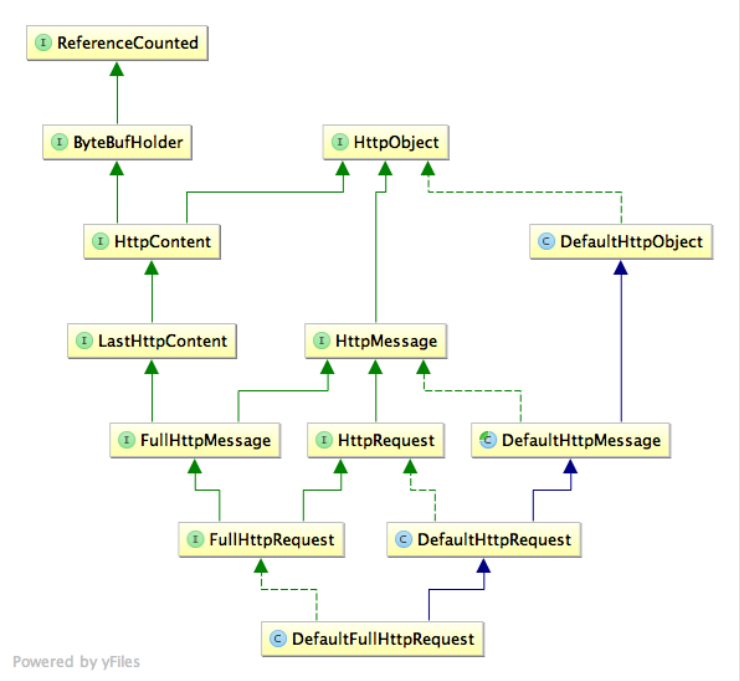


通过上图我们可以看到, 一个 Channel 包含了一个 ChannelPipeline, 而 ChannelPipeline 中又维护了一个由 ChannelHandlerContext 组成的双向链表. 这个链表的头是 HeadContext, 链表的尾是 TailContext, 并且每个 ChannelHandlerContext 中又关联着一个 ChannelHandler.

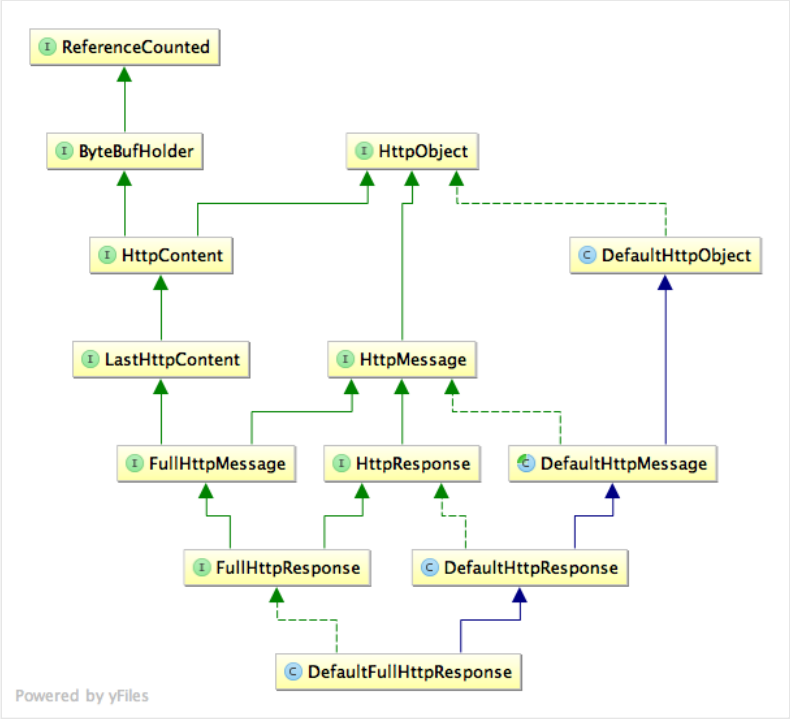
程序中如何初始pipleline?

```
bootstrap.childHandler(new ChannelInitializer<NioSocketChannel>() {
    @Override
    protected void initChannel(NioSocketChannel ch) throws Exception {
        ch.pipeline().addLast("http-decoder", new HttpRequestDecoder());
        ch.pipeline().addLast("http-aggregator", new HttpObjectAggregator(65536));
        ch.pipeline().addLast("http-encoder", new HttpResponseEncoder());
        ch.pipeline().addLast("http-server", new HttpServerHandler());
    }
});
```

HttpRequest 在netty 当中的表示结构

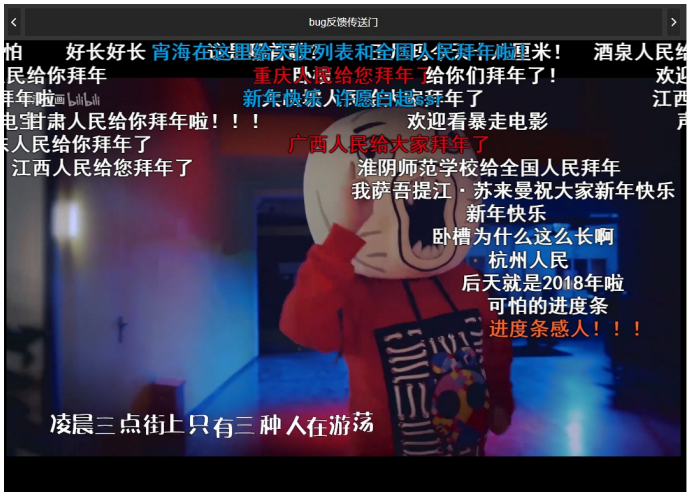


HttpResponse在netty 当中的结构:



三、基于Netty实现弹幕系统

什么是弹幕系统?

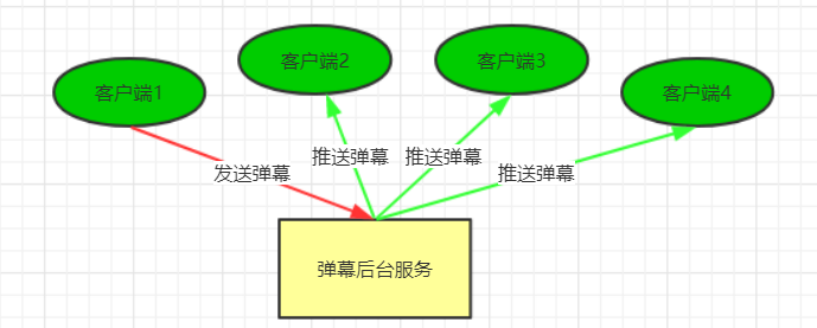


弹幕系统特点:

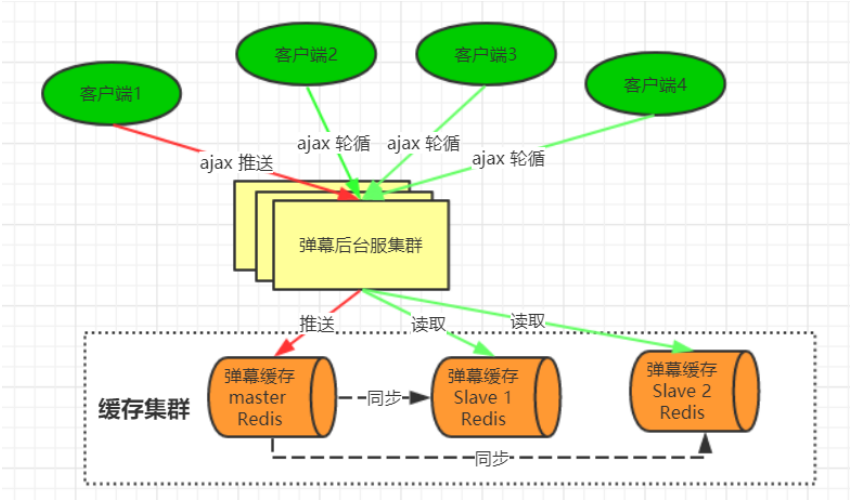
1. **实时性高**: 你发我收, 毫秒之差
2. **并发量大**: 一人吐槽, 万人观看

弹幕系统架构设计:

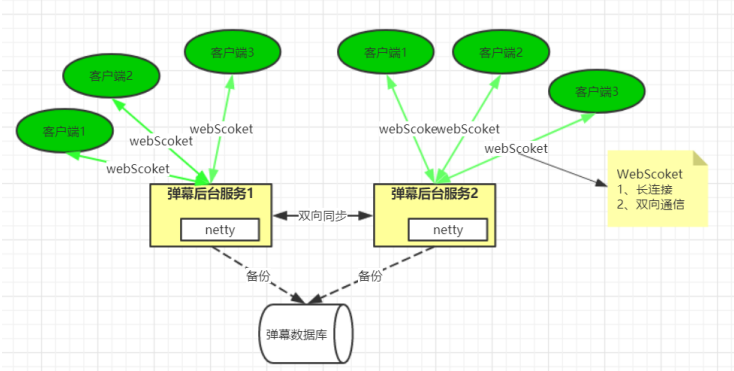
业务架构:



实现方案一：



实现方案二：



Netty实现代码讲解：