**基于Netty实现高性能弹幕系统**

主讲：鲁班

时间：2018/11/17 8:20

地址：腾讯课堂-图灵学院

**课程概要：**

一、弹幕系统概要设计

二、Netty对Http协议解析实现

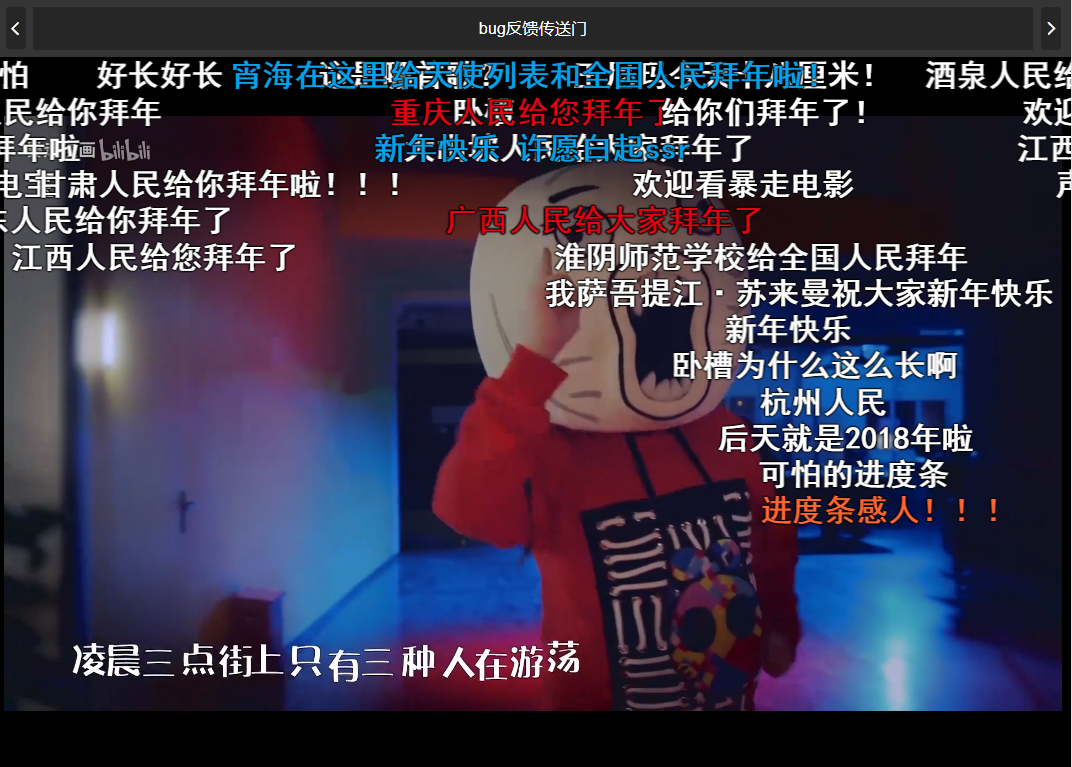
三、WebScoket协议解析实现

**讲师介绍：**



## **一、弹幕系统概要设计**

**什么是弹幕系统？**

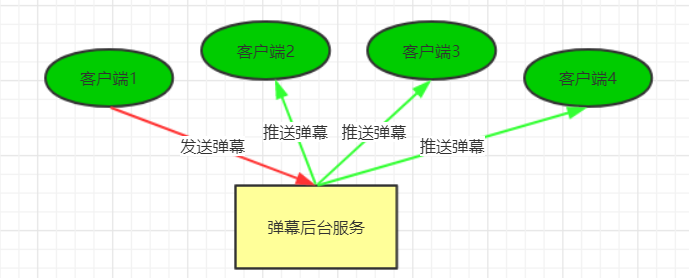


弹幕系统特点：

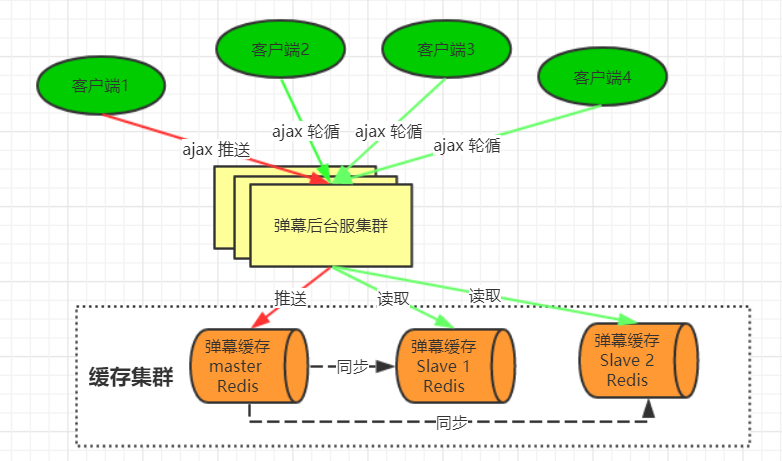
1. **实时性高**：你发我收， 毫秒之差
2. **并发量大**：一人吐槽，万人观看

**弹幕系统架构设计：**

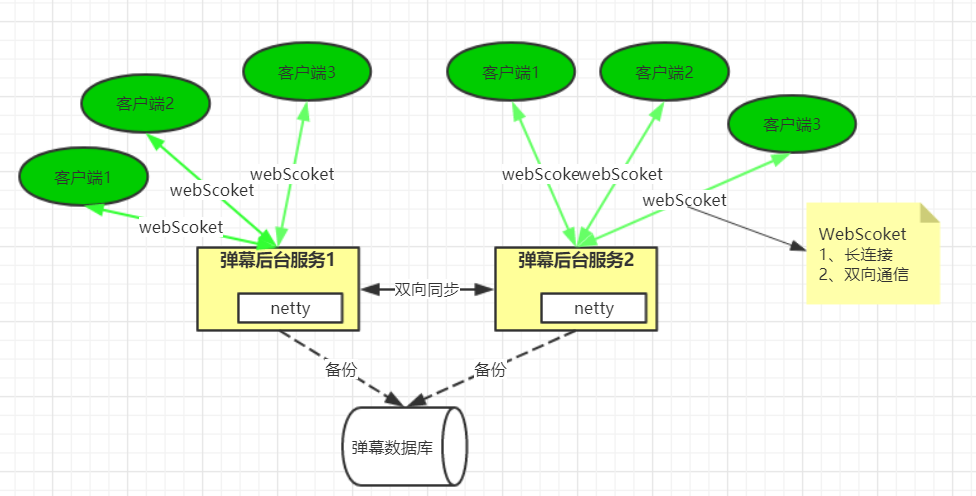
业务架构：



实现方案一：



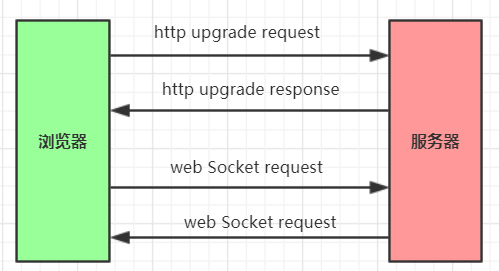
实现方案二：



**方案实现简要说明：**

## **二、NettyHttp协议解析实现**

在上述方案中 浏览器不能直接能和Netty 建立连接 其必须借助http 请求 进行协议升级才能实现服务端与客户端基于Web Socket通信，其过程如下图：



也就是说如果我们想实现弹幕就必须先实现Http服务,那么Netty如何实现Http服务呢？

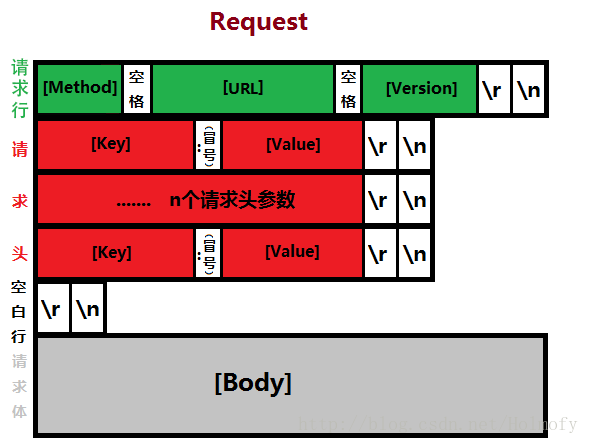
**Http协议交互过程**

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？

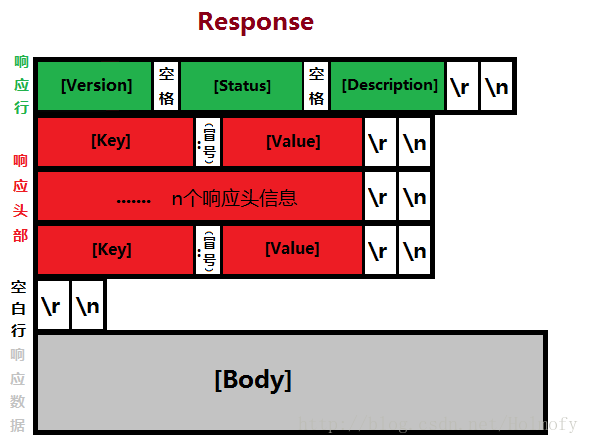
*传输数据*一般基于TCP/IP 实现，体现到开发语言上就是我们所熟悉的Socket 编程。

*交换数据*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 报文：**



**response 报文：**



### **http报文解析方案：**

1：请求行的边界是CRLF(回车)，如果读取到CRLF(回车)，则意味着请求行的信息已经读取完成。

2：Header的边界是CRLF，如果连续读取两个CRLF，则意味着header的信息读取完成。

3：body的长度是有Content-Length 来进行确定。

### **netty关于http 的解决方案：**

// 解析请求

很多http server的实现都是基于servlet标准，但是netty对http实现并没有基于servlet。所以在使用上比Servlet复杂很多。比如在servlet 中直接可以通过 HttpServletRequest 获取 请求方法、请求头、请求参数。而netty 确需要通过如下对象自行解析获取。

**HttpMethod：**主要是对method的封装，包含method序列化的操作

**HttpVersion:** 对version的封装，netty包含1.0和1.1的版本

**QueryStringDecoder:** 主要是对urI进行解析，解析path和url上面的参数。

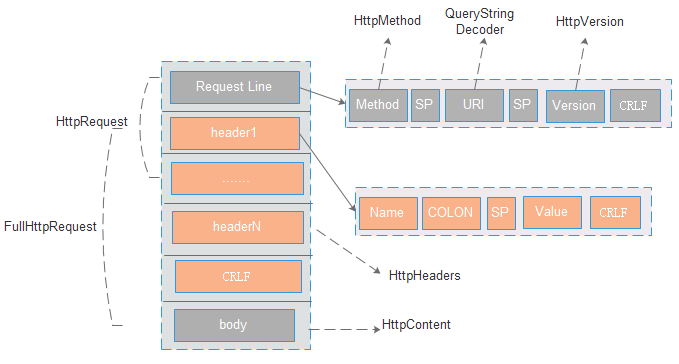
**HttpPostRequestDecoder：**对post 中body 内容进行解析获取 form 参数。

**HttpHeaders：**包含对header的内容进行封装及操作

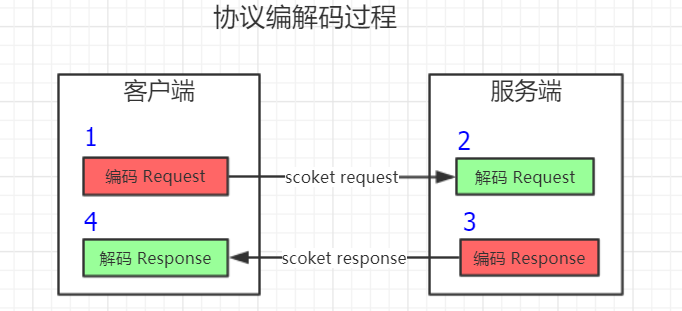
**HttpContent：**是对body进行封装，本质上就是一个ByteBuf。如果ByteBuf的长度是固定的，则请求的body过大，可能包含多个HttpContent，其中最后一个为LastHttpContent(空的HttpContent),用来说明body的结束。

**HttpRequest：**主要包含对Request Line和Header的组合

**FullHttpRequest：** 主要包含对HttpRequest和httpContent的组合



### **Netty Http的请求处理流程：**



从图中可以看出做为服务端的Netty 就是在做 编码和解码操作。其分别通过以下两个ChannelHandler对象实现：

**HttpRequestDecoder :**用于从byteBuf 获取数据并解析封装成HttpRequest 对象

**HttpResponseEncoder：**用于将业务返回数据编码成 Response报文并发送到ByteBuf。

将以上两个对象添加进 Netty 的 pipeline 即可实现最简单的http 服务。

**通过一个示例演示Http实现：**

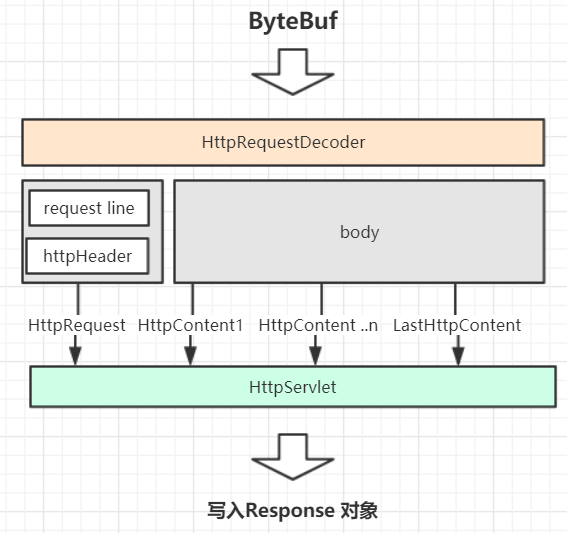
编写serverBootstrap

初始pipeline

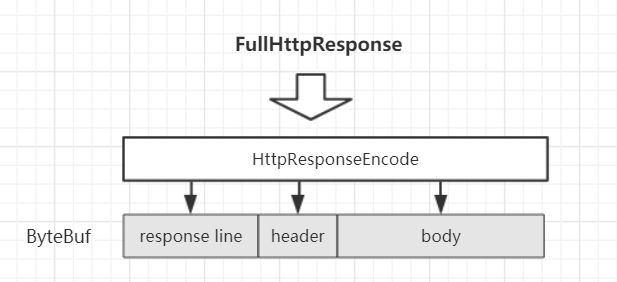
编写ServletHandler

**Http完整流程介绍：**

Decoder 流程：



encode 流程：



## **三、WebScoket协议实现**

**webSocket 协议简介：**

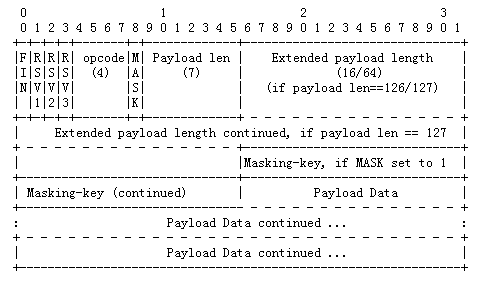
webSocket 是html5 开始提供的一种浏览器与服务器间进行全双工二进制通信协议，其基于TCP双向全双工作进行消息传递，同一时刻即可以发又可以接收消息，相比Http的半双工协议性能有很大的提升，

**webSocket特点如下：**

1. 单一TCP长连接，采用全双工通信模式
2. 对代理、防火墙透明
3. 无头部信息、消息更精简
4. 通过ping/pong 来保活
5. 服务器可以主动推送消息给客户端，不在需要客户轮询

**WebSocket 协议报文格式：**

我们知道，任何应用协议都有其特有的报文格式，比如Http协议通过 空格 换行组成其报文。如http 协议不同在于WebSocket属于二进制协议，通过规范进二进位来组成其报文。具体组成如下图：



**Http协议报文：**

**报文说明：**

FIN

标识是否为此消息的最后一个数据包，占 1 bit

RSV1, RSV2, RSV3: 用于扩展协议，一般为0，各占1bit

Opcode

数据包类型（frame type），占4bits

0x0：标识一个中间数据包

0x1：标识一个text类型数据包

0x2：标识一个binary类型数据包

0x3-7：保留

0x8：标识一个断开连接类型数据包

0x9：标识一个ping类型数据包

0xA：表示一个pong类型数据包

0xB-F：保留

MASK：占1bits

用于标识PayloadData是否经过掩码处理。如果是1，Masking-key域的数据即是掩码密钥，用于解码PayloadData。客户端发出的数据帧需要进行掩码处理，所以此位是1。

Payload length

Payload data的长度，占7bits，7+16bits，7+64bits：

如果其值在0-125，则是payload的真实长度。

如果值是126，则后面2个字节形成的16bits无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

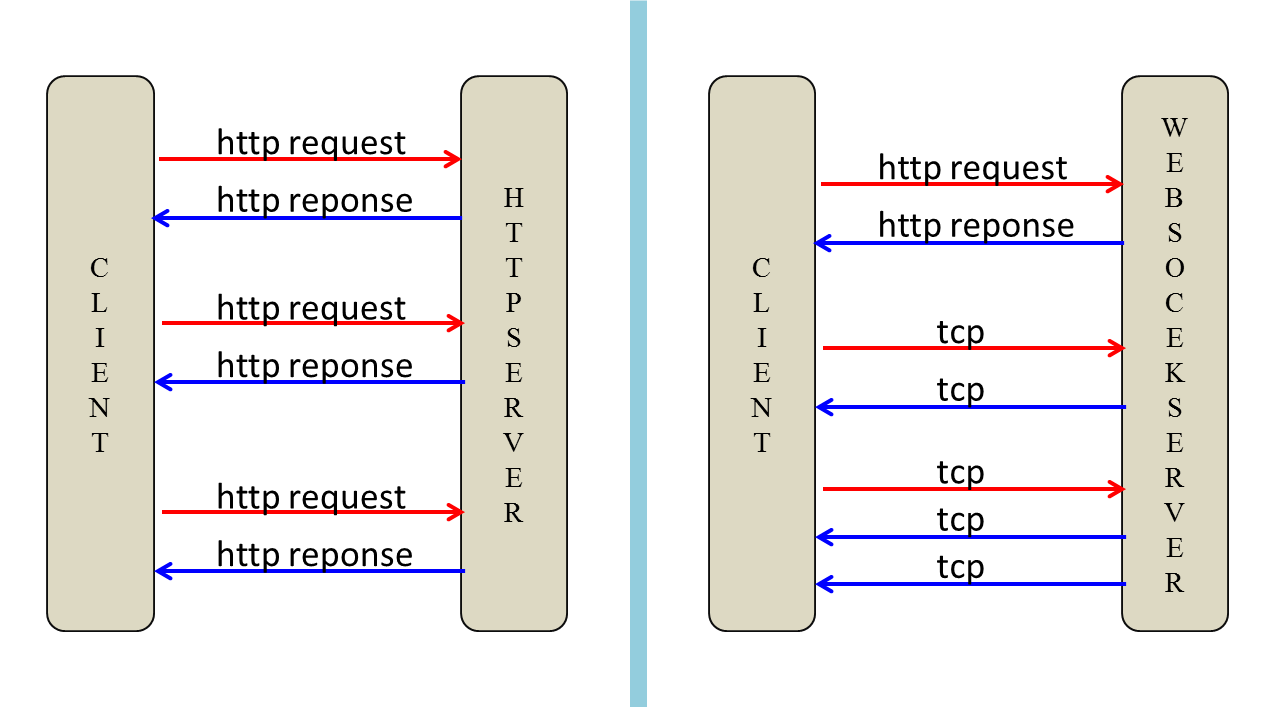
如果值是127，则后面8个字节形成的64bits无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

Payload data

应用层数据

**WebSocket 在浏览当中的使用**

Http 连接与webSocket 连接建立示意图：



通过javaScript 中的API可以直接操作WebSocket 对象，其示例如下：

var ws = new WebSocket(“ws://localhost:8080”);

ws.onopen = function()// 建立成功之后触发的事件 {

console.log(“打开连接”); ws.send("ddd"); // 发送消息

};

ws.onmessage = function(evt) { // 接收服务器消息

console.log(evt.data);

};

ws.onclose = function(evt) {

console.log(“WebSocketClosed!”); // 关闭连接 };

ws.onerror = function(evt) {

console.log(“WebSocketError!”); // 连接异常

};

**弹幕系统实现讲解：**

Http 协议后台实现：

webSocket 协议后台实现

弹幕系统前台实现

**弹幕系统实时演示：**

#启动服务

#查看当前端口连接数

netstat -nat|grep -i "8880"|wc -l

#查看指定进程线程数

pstree -p 3000 | wc -l