Netty对弹幕系统应用与Dubbo框架中的应用

课程概要：

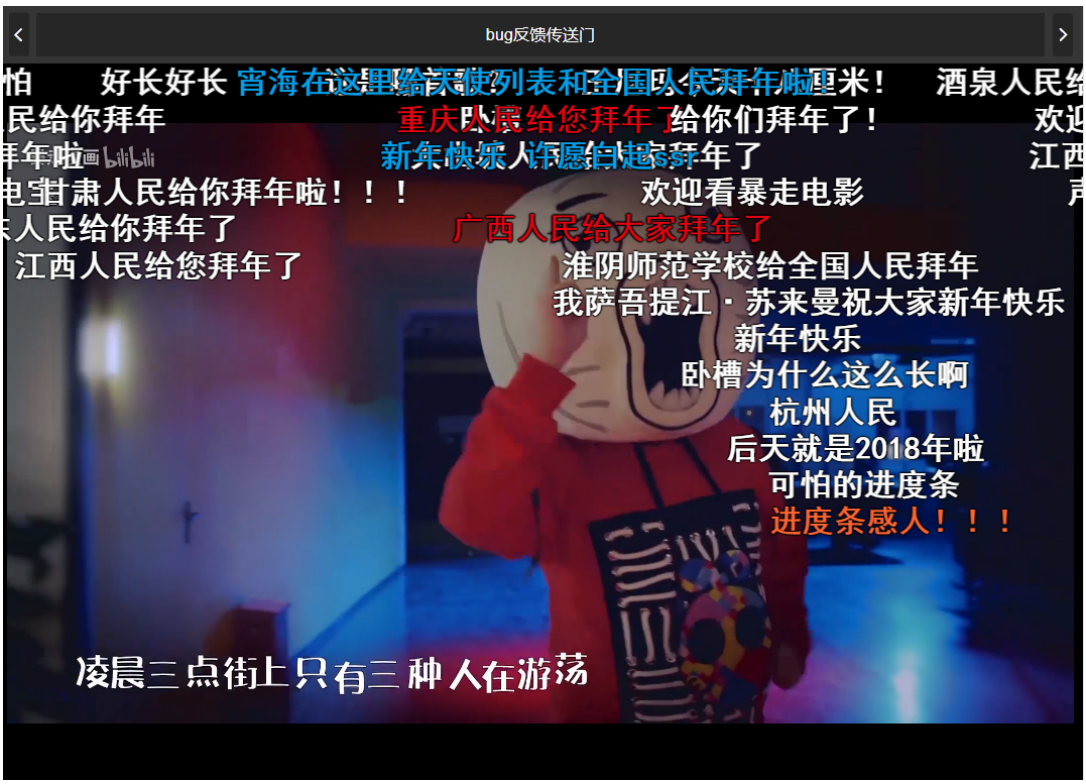
一、弹幕系统概要设计

二、WebScoket协议解析实现

三、Netty在Dubbo中应用分析

一、弹幕系统概要设计

什么是弹幕系统?

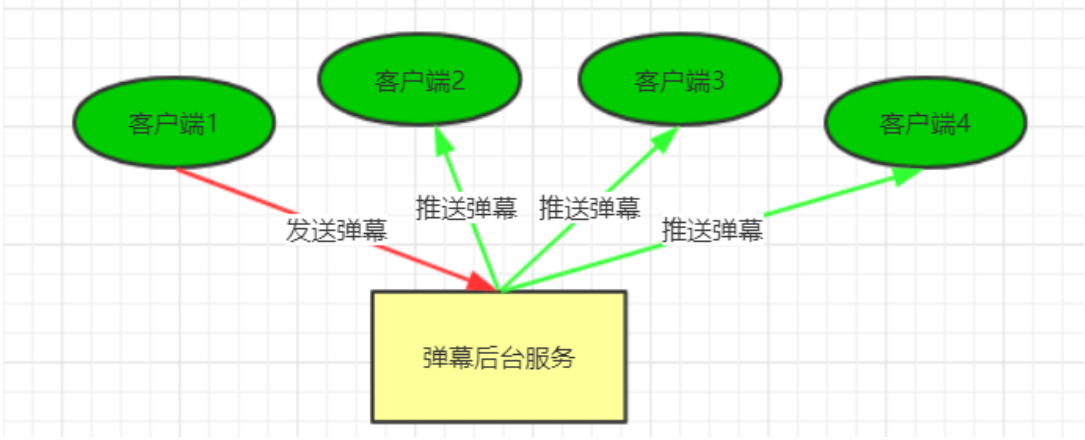


弹幕系统特点：

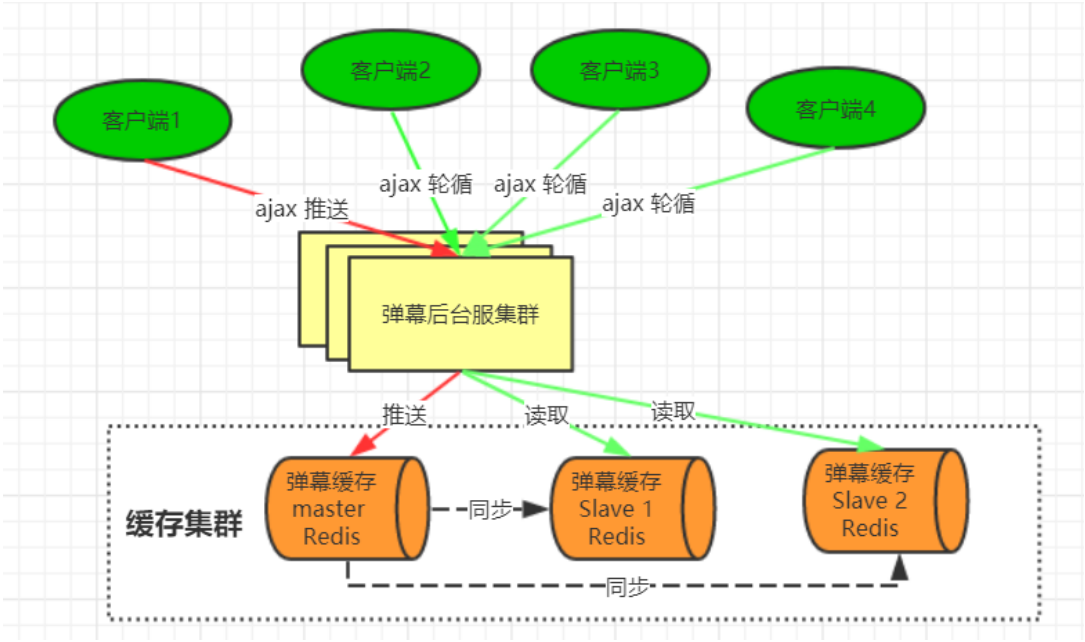
1. **实时性高**：你发我收， 毫秒之差
2. **并发量大**：一人吐槽，万人观看

弹幕系统架构设计：

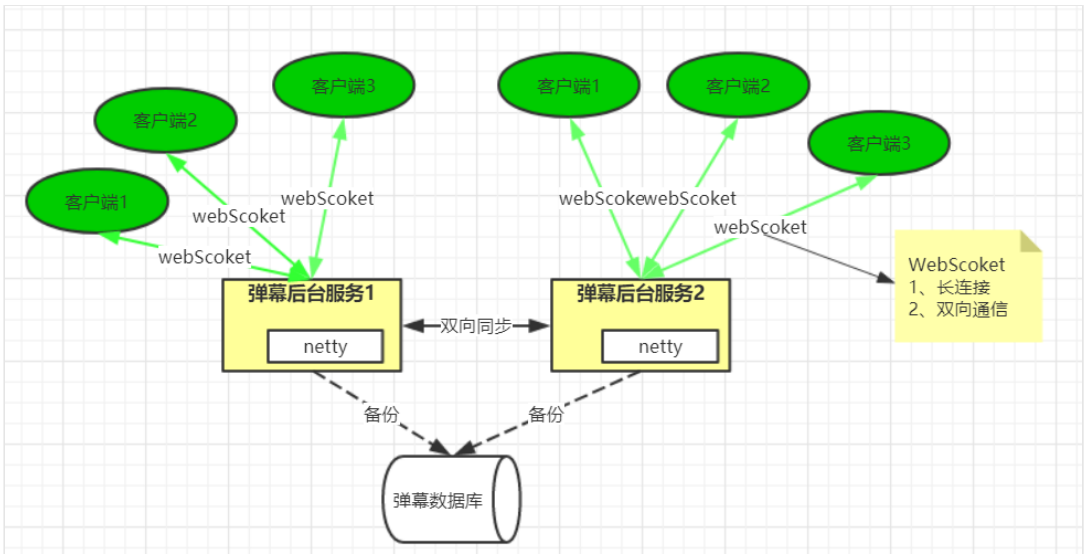
业务架构：



实现方案一：



实现方案二：



方案实现简要说明：

二、WebScoket协议解析实现

webSocket 协议简介：

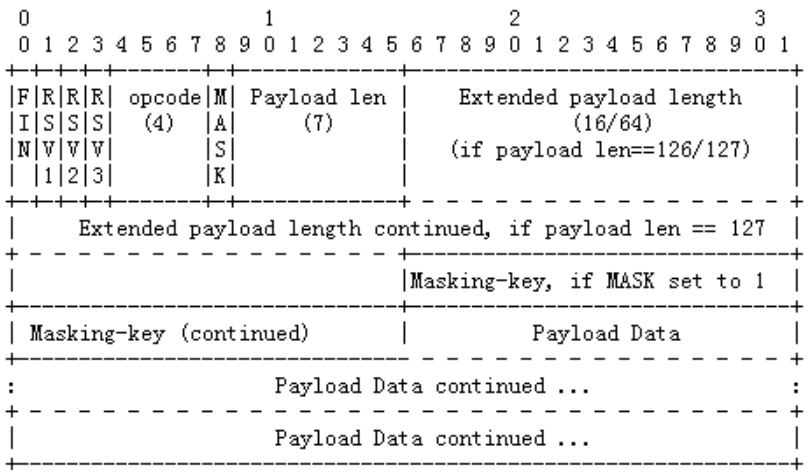
webSocket 是html5 开始提供的一种浏览器与服务器间进行全双工二进制通信协议，其基于TCP双向全双工作进行消息传递，同一时刻即可以发又可以接收消息，相比Http的半双工协议性能有很大的提升

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：



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

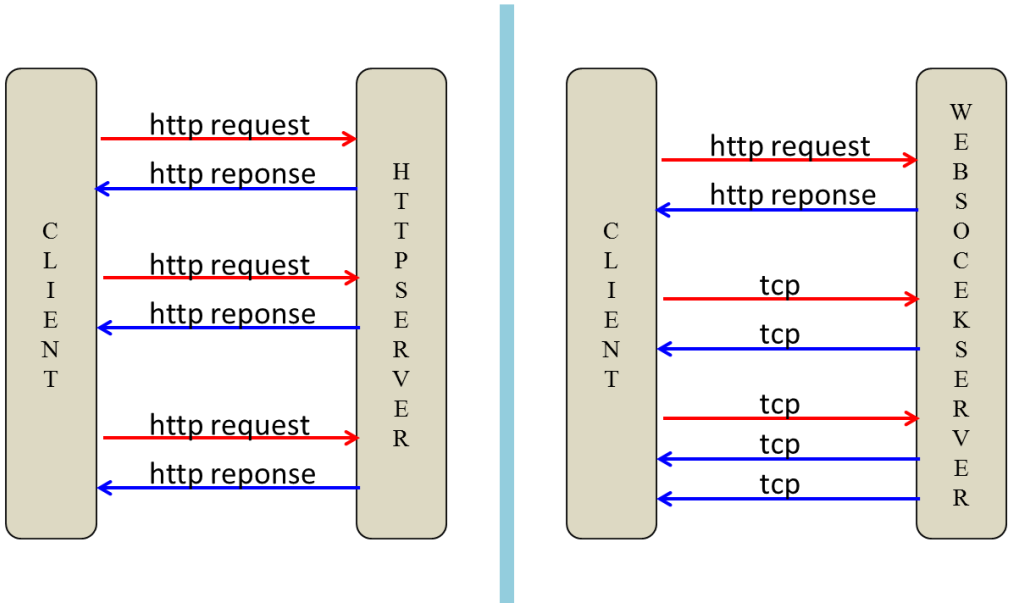
Payload data的长度，占7bits，7+16bits，7+64bits：
如果其值在0-125，则是payload的真实长度。
如果值是126，则后面2个字节形成的16bits无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。
如果值是127，则后面8个字节形成的64bits无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

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：

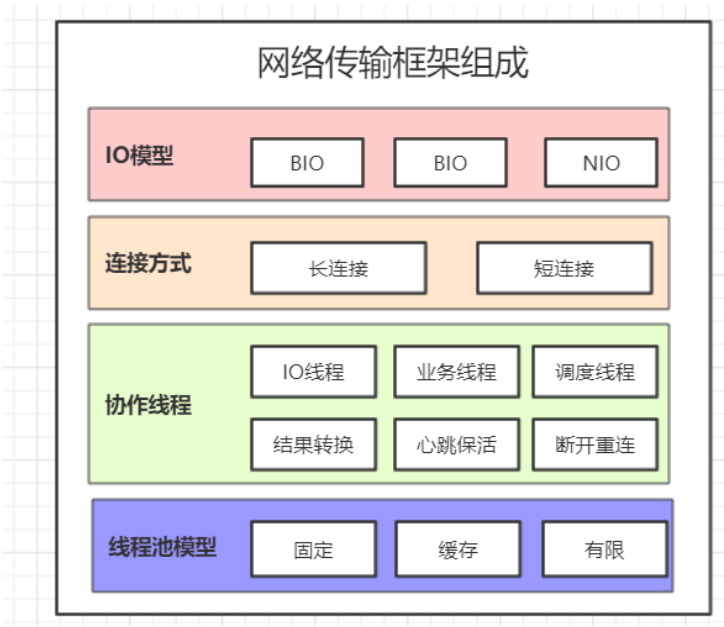
- ☐ http协议后台实现
- ☐ webSocket协议后台实现
- ☐ 弹幕系统前台实现

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
```

三、Netty在Dubbo中应用分析

网络传输的实现组成



1. **IO模型：**
   a. BIO 同步阻塞
   b. NIO 同步非阻塞
   c. AIO 异步非阻塞
2. **连接模型：**
   a. 长连接
   b. 短连接
3. **线程分类：**
   a. IO线程
   b. 服务端业务线程
   c. 客户端调度线程
   d. 客户端结果exchange线程。
   e. 保活心跳线程
   f. 重连线程
4. **线程池模型：**
   a. 固定数量线程池
   b. 缓存线程池
   c. 有限线程池

Dubbo 长连接实现与配置

初始连接：

引用服务增加提供者==>获取连接===》是否获取共享连接==>创建连接客户端==》开启心跳检测状态检查定时任务===》开启连接状态检测

源码见：com.alibaba.dubbo.rpc.protocol.dubbo.DubboProtocol#getClients

心跳发送：

在创建一个连接客户端同时也会创建一个心跳客户端，客户端默认基于60秒发送一次心跳来保持连接的存活，可通过 heartbeat 设置。

源码见：

com.alibaba.dubbo.remoting.exchange.support.header.HeaderExchangeClient#startHeatbeatTimer

断线重连：

每创建一个客户端连接都会启动一个定时任务每两秒中检测一次当前连接状态，如果断线则自动重连。

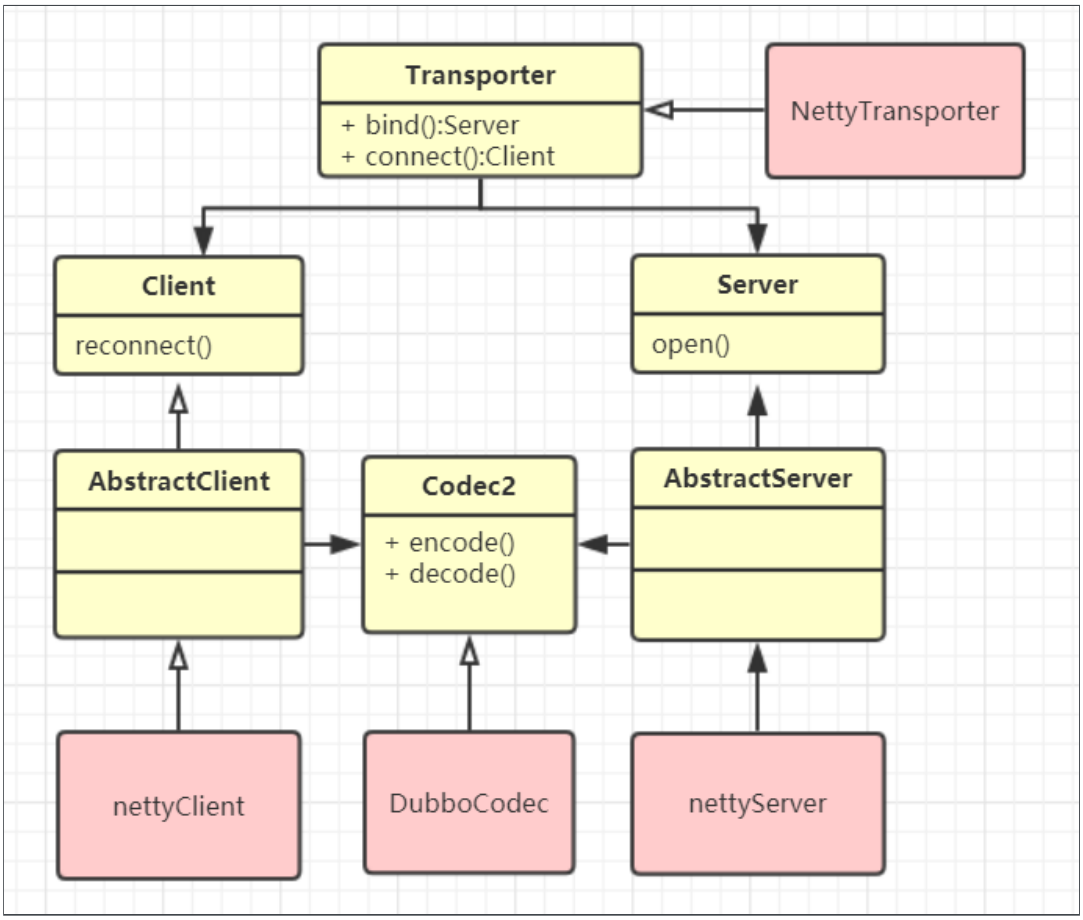
源码见：com.alibaba.dubbo.remoting.transport.AbstractClient#initConnectStatusCheckCommand

连接销毁:

基于注册中心通知，服务端断开后销毁

源码见：com.alibaba.dubbo.remoting.transport.AbstractClient#close()

dubbo传输uml类图:

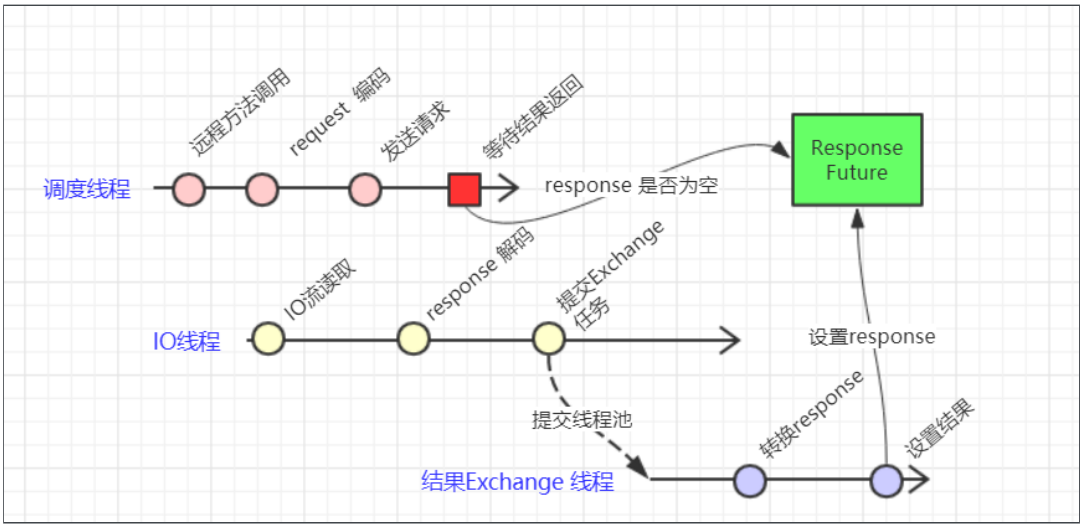


Dubbo 传输协作线程

1. **客户端调度线程**：用于发起远程方法调用的线程。
2. **客户端结果Exchange线程：**当远程方法返回response后由该线程填充至指定ResponseFuture，并叫醒等待的调度线程。
3. **客户端IO线程：**由传输框架实现，用于request 消息流发送、response 消息流读取与解码等操作。

4. **服务端IO线程**：由传输框架实现，用于request消息流读取与解码 与Response发送。
5. **业务执行线程**：服务端具体执行业务方法的线程

客户端线程协作流程：

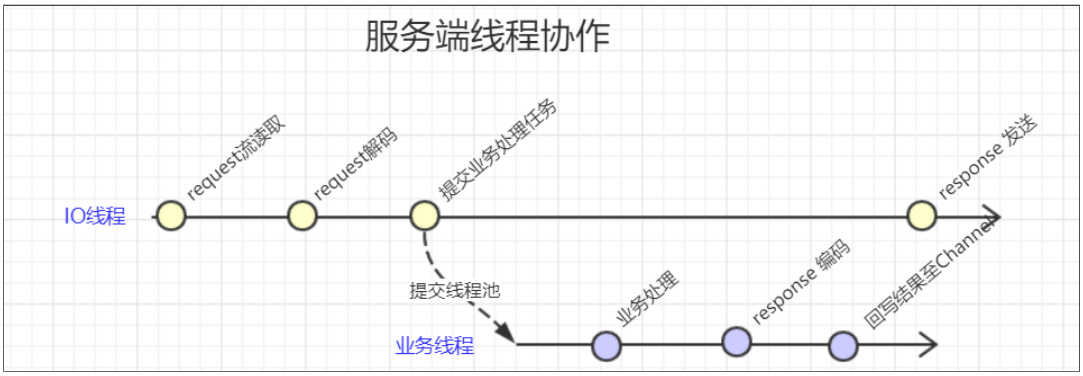


1. **调度线程**
   a. 调用远程方法
   b. 对request 进行协议编码
   c. 发送request 消息至IO线程
   d. 等待结果的获取
2. **IO线程**
   a. 读取response流
   b. response 解码
   c. 提交Exchange 任务
3. **Exchange线程**
   a. 填写response值 至 ResponseFuture
   b. 唤醒调度线程，通知其获取结果

调用调试：

客户端的执行线程:
1、业务线程
1) DubboInvoker#doInvoke(隐示传公共参数、获取客户端、异步、单向、同步（等待返回结果）)
2)AbstractPeer#send// netty Client客户端发送消息 写入管道
3)DubboCodec#encodeRequestData // Request 协议编码
2、IO线程
DubboCodec#decodeBody //Response解码
AllChannelHandler#received //// 派发消息处理线程
3、调度线程
DefaultFuture#doReceived // 设置返回结果

服务端线程协作：



1. **IO线程：**
   a. request 流读取
   b. request 解码
   c. 提交业务处理任务
2. **业务线程：**
   a. 业务方法执行
   b. response 编码
   c. 回写结果至channel

线程池

1. **fixed：**固定线程池,此线程池启动时即创建固定大小的线程数，不做任何伸缩，
2. **cached：**缓存线程池,此线程池可伸缩，线程空闲一分钟后回收，新请求重新创建线程
3. **Limited：**有限线程池,此线程池一直增长，直到上限，增长后不收缩。