netty之WebSocket协议应用

课程概要：

1. webSocket协议概述
2. Netty中WebSocket实现
3. WebSocket 应用场景

一、webSocket 协议概述

提问：

假设我们要实现一个WEB版的聊天室可以采用哪些方案？

1. Ajax轮询去服务器取消息
   客户端按照某个时间间隔不断地向服务端发送请求，请求服务端的最新数据然后更新客户端显示。这种方式实际上浪费了大量流量并且对服务端造成了很大压
2. Flash XMLSocket
   在 HTML 页面中内嵌入一个使用了 XMLSocket 类的 Flash 程序。JavaScript 通过调用此 Flash 程序提供的套接口接口与服务器端的套接口进行通信。JavaScript 在收到服务器端以 XML 格式传送的信息后可以很容易地控制 HTML 页面的内容显示。

以上方案的弊端

Ajax 轮询：

1. Http为半双工协议，也就是说同一时刻，只有一个方向的数据传送
2. Http消息冗长，包含请求行、请求头、请求体。占用很多的带宽和服务器资源。
3. 空轮询问题

Flash XMLSocket ：

1. 客户端必须安装 Flash 播放器；
2. 因为 XMLSocket 没有 HTTP 隧道功能，XMLSocket 类不能自动穿过防火墙；
3. 因为是使用套接口，需要设置一个通信端口，防火墙、代理服务器也可能对非 HTTP 通道端口进行限制；

为了解决上述弊端，Html5定义了WebSocket协义能更好的节省服务器资源和宽带达到实时通信的目的。

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。

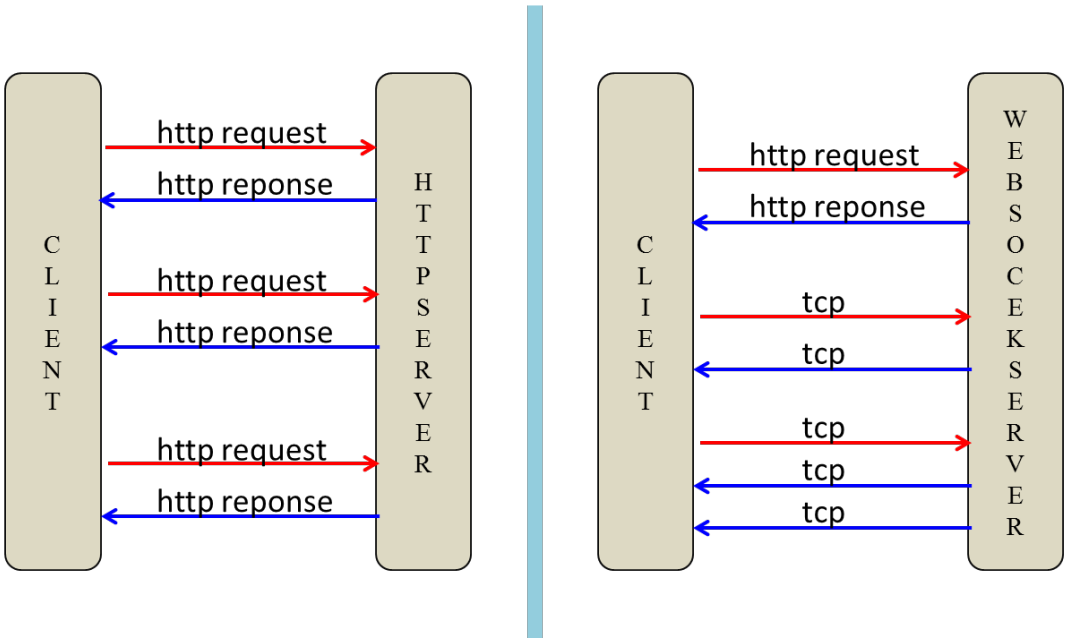
如果值是127，则后面8个字节形成的64bits无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

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
 };
```

首先：申请一个WebSocket对象，并传入WebSocket地址信息，这时client会通过Http先发起握手请求。消息格式如下：

```
   GET /chat HTTP/1.1
    Host: server.example.com
    Upgrade: websocket //告诉服务端需要将通信协议升级到websocket
    Connection: Upgrade
    Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== //浏览器base64加密的密钥,server端收到后需要提取Sec-WebSocket-Key 信息,然后加密。
    Origin: http://example.com
    Sec-WebSocket-Protocol: chat, superchat //表示客户端请求提供的可供选择的子协议
    Sec-WebSocket-Version: 13 //版本标识
```

其次：服务端响应、并建立连接

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
```

```
Sec-WebSocket-Accept: SIEylb7zRYJAEgiqJXaOW3V+ZWQ=
```

然后：握手成功促发客户端 onOpen 事件

连接状态查看：

通过ws.readyState 可查看当前连接状态可选值如下：

1. CONNECTING (0)：表示还没建立连接；
2. OPEN (1)： 已经建立连接，可以进行通讯；
3. CLOSING (2)：通过关闭握手，正在关闭连接；
4. CLOSED (3)：连接已经关闭或无法打开；

二、Netty中WebSocket实现

WebSocketDemo演示：

- ☐ 服务端编写输出webSocketPage.html 页
  - ☐ 添加 聚合器 HttpObjectAggregator
  - ☐ 添加分包器ChunkedWriteHandler
- ☐ 添加WebSocket协议处理器 WebSocketServerProtocolHandler
- ☐ 编写WebSocketServerHandler
- ☐ 编写客户端js脚本

WebSocket 协议中的Decode 与Encode哪去了？

1. WebSocketServerProtocolHandler ==》handlerAdded()
2. WebSocketServerProtocolHandshakeHandler ==》channelRead()
3. WebSocketServerHandshaker==》handshake()
4. WebSocket13FrameDecoder==》decode()

Decoder与Encode 过程详解：

三、WEB 应用场景

WebSocket的应用场景举例：

多人联机贪吃蛇效果演示：

游戏架构思路：

架构当中的算法：