**预研技术**

# Websocket

1. What

WebSocket protocol 是HTML5一种新的协议。它实现了浏览器与服务器全双工通信(full-duplex)。一开始的握手需要借助HTTP请求完成。

1. Why

Web 应用的交互过程通常是客户端通过浏览器发出一个请求，服务器端接收请求后进行处理并返回结果给客户端，客户端浏览器将信息呈现，这种机制对于信息变化不是特别频繁的应用尚可，但对于实时要求高、海量并发的应用来说显得捉襟见肘。

传统的方式：

ajax轮询

http long pull

缺点：

1、都是在不断地建立HTTP连接，然后等待服务端处理，可以体现HTTP协议的另外一个特点，被动性。服务端不能主动联系客户端，只能有客户端发起。

2、消耗资源 ajax轮询需要很快的处理速度和资源 long pull需要很高的并发

1. connection

websocket握手

握手请求

GET /echo HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Version: 13

Origin: http://example.com

握手响应  
HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

接下来就是完全按照Websocket协议进行了

Websocket的优点:

在传统的方式上，要不断的建立，关闭HTTP协议，由于HTTP是非状态性的，每次都要重新传输identity info（鉴别信息），来告诉服务端你是谁。浪费处理时间，而且还会在网路传输中消耗过多的流量。但是Websocket只需要一次HTTP握手，所以说整个通讯过程是建立在一次连接/状态中，也就避免了HTTP的非状态性，服务端会一直知道你的信息，直到你关闭请求。并且由客户主动询问，转换为服务器（推送）有信息的时候可以主动推送。

非 WebSocket 模式传统 HTTP 客户端与服务器的交互如下图所示：

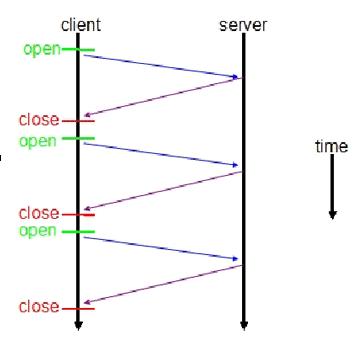


图 1. 传统 HTTP 请求响应客户端服务器交互图

使用 WebSocket 模式客户端与服务器的交互如下图：

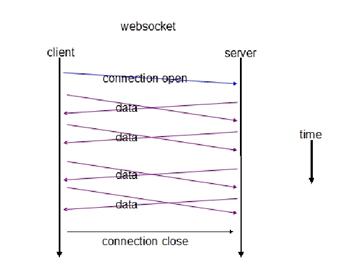
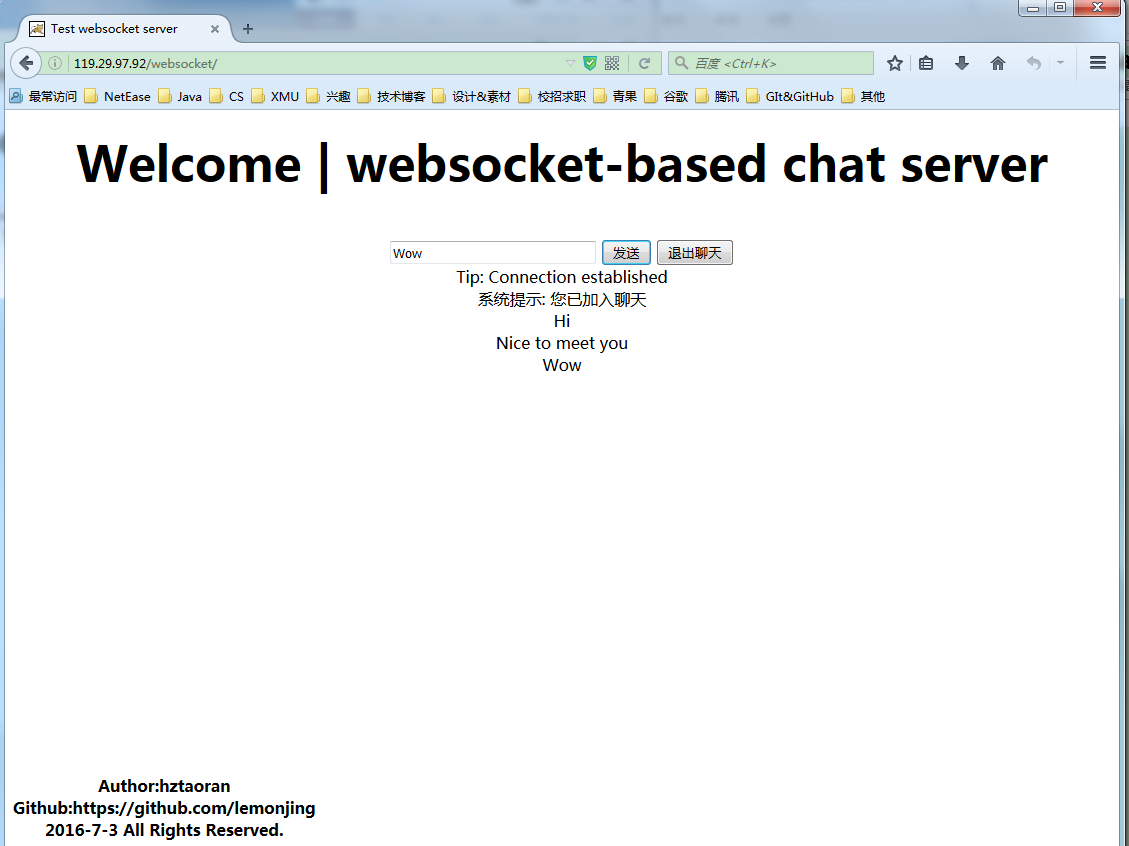


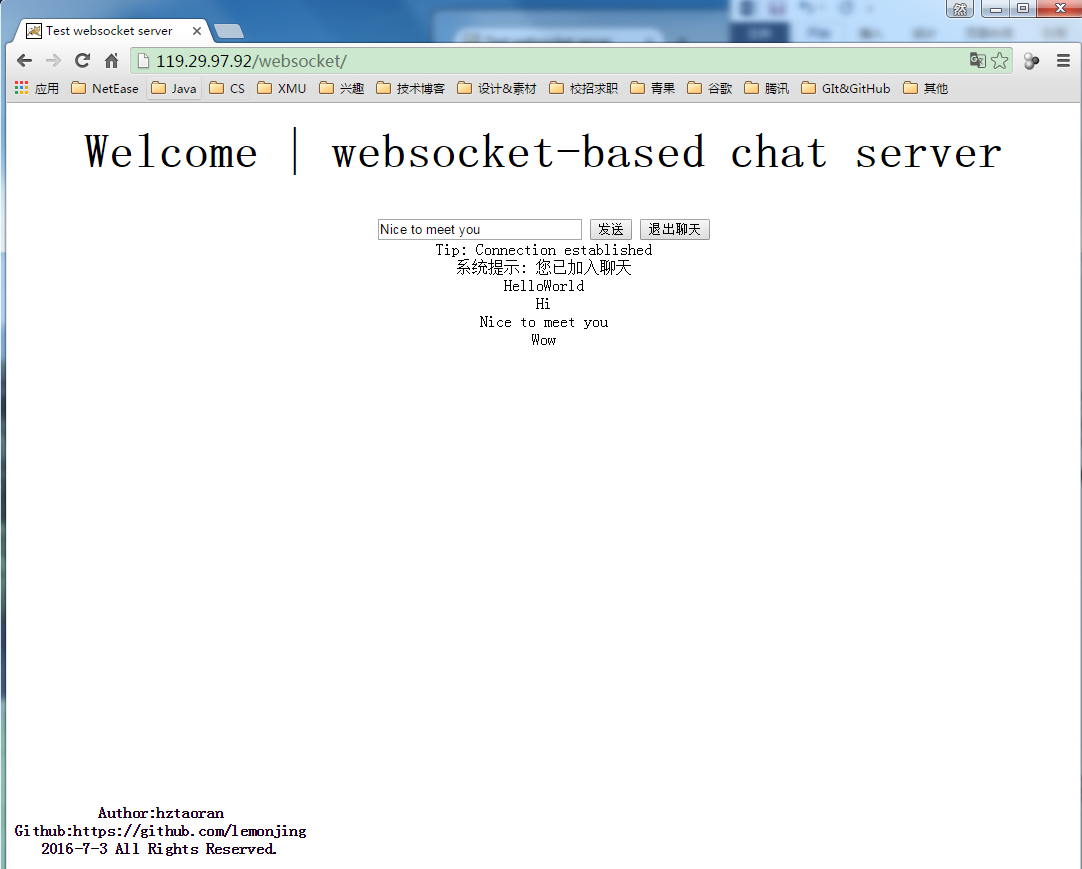
图 2.WebSocket 请求响应客户端服务器交互图

预研Demo

客户端1



客户端2

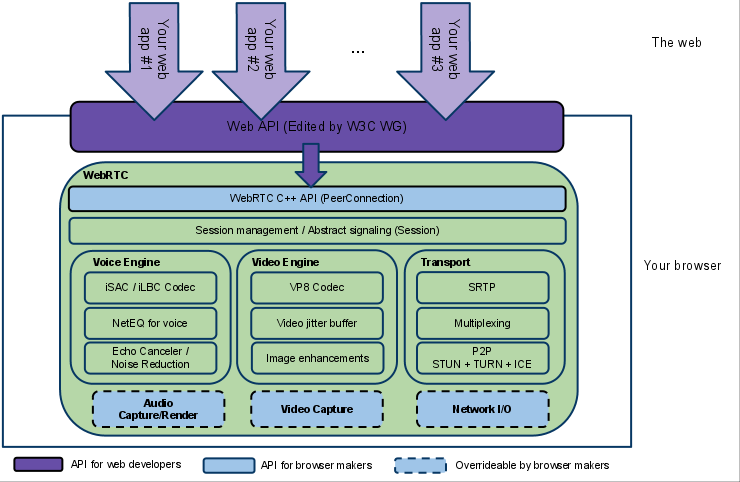


# WebRTC

1、简介和基本概念

WebRTC，名称源自网页实时通信（Web Real-Time Communication）的缩写，是一个支持网页浏览器进行实时语音通信或视频通信的技术。

对浏览器来说，WebRTC其实就是提供了3个API：  
MediaStream (即getUserMedia)，用于获取媒体数据，例如来自摄像头和麦克风的视频流和音频流  
RTCPeerConnection，用于peer跟peer之间呼叫和建立连接以便传输音视频数据流  
RTCDataChannel 用于peer跟peer之间传输音视频之外的一般数据



WebRTC 架构组件

**信令服务器**

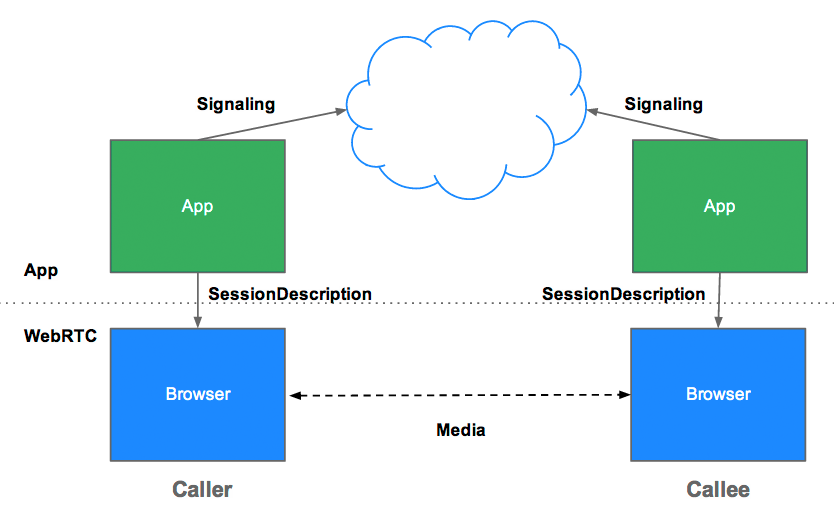
WebRTC提供了浏览器到浏览器（点对点）之间的通信，但并不意味着WebRTC不需要服务器。暂且不说基于服务器的一些扩展业务，WebRTC至少有两件事必须要用到服务器：  
1. 浏览器之间交换建立通信的元数据（信令）必须通过服务器  
2. 为了穿越NAT和防火墙

为什么需要信令？

我们需要通过一系列的信令来建立浏览器之间的通信。而具体需要通过信令交换哪些内容呢？这里大概列了一下：  
1. 用来控制通信开启或者关闭的连接控制消息  
2. 发生错误时用来彼此告知的消息  
3. 媒体流元数据，比如像解码器、解码器的配置、带宽、媒体类型等等  
4. 用来建立安全连接的关键数据  
5. 外界所看到的的网络上的数据，比如IP地址、端口等

信令处理过程需要客户端能够来回传递消息,这个过程在webrtc里面是没有实现的。

在建立连接之前，浏览器之间显然没有办法传递数据。所以我们需要通过服务器的中转，在浏览器之间传递这些数据，然后建立浏览器之间的点对点连接。但是WebRTC API中并没有实现这些。

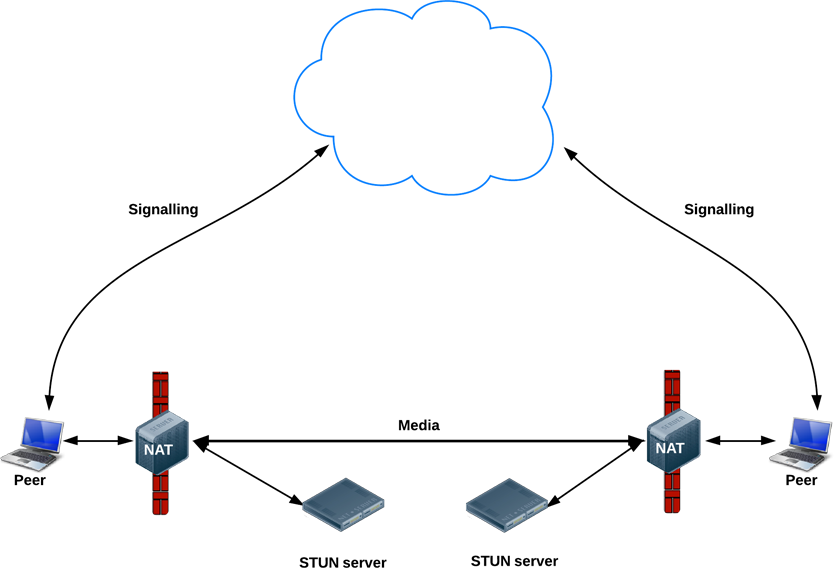


信令服务器

实现：websocket

**打洞服务器**

STUN、TURN



一旦peer跟peer之间建立好连接就可以直接传输音视频数据流，并不需要借助第三方服务器中转。

2、WebRTC封装库

WebRTC的目的是为了简化基于浏览器的实时数据通信的开发工作量，  
但实际应用编程还是有点复杂，尤其调用RTCPeerConnection必须对如何建立连接、  
交换信令的流程和细节有深入的理解。为此可以使用WebRTC的封装库SimpleWebRTC

1. Demo

安装nodejs

安装信令服务器的模块

npm install express  
npm install yetify  
npm install getconfig  
npm install node-uuid  
npm install socket.io

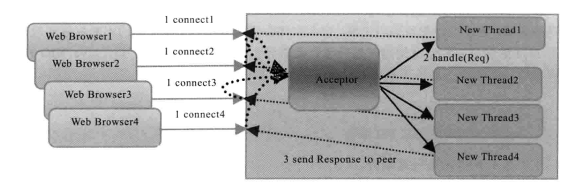
# Kurento

# Netty

1. 传统的BIO编程

Netty是一个高性能、异步事件驱动的NIO框架，它提供了对TCP、UDP和文件传输的支持，作为一个异步NIO框架，Netty的所有IO操作都是异步非阻塞的，通过Future-Listener机制，用户可以方便的主动获取或者通过通知机制获得IO操作结果。

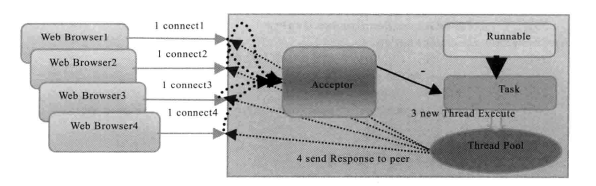
作为当前最流行的NIO框架，Netty在互联网领域、大数据分布式计算领域、游戏行业、通信行业等获得了广泛的应用，一些业界著名的开源组件也基于Netty的NIO框架构建。



一客户端一线程

缺点：缺乏弹性收缩，客户端并发访问量增大后，线程膨胀，导致线程堆栈溢出，并最终使得服务器宕机或假死，停止服务。

1. 伪异步IO



M:N的伪异步IO模型

1. NIO(JDK1.4)

引入了Channel、Buffer和Selector

缺点：

API能力较弱。目录的级联创建和递归遍历需要手动实现

文件操作是同步阻塞调用，不支持异步文件读写

1. NIO2.0(AIO、JDK1.7)

提供AIO功能，支持基于文件的异步IO操作和针对网络套接字的异步操作