浙江大學

硕 士 研 究 生 读 书 报 告

题目 Websocket 的分析与应用

作者姓名 丁文龙

作者学号 21651019

指导教师 李启雷

学科专业 移动互联网与游戏开发技术

所在学院 软件学院

提交日期 二〇 17 年 1 月

The Analysis and Applications Of Websocket

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Qilei Li

By

Wenlong Ding

Zhejiang University, P.R. China

2017

摘要

本文重点探讨了 Websocket 协议，以及其发展和应用。当前互联网对于 Web 通信要求越来越高：更高的并发量，消息更加的即时可靠。传统的 HTTP 协议已经无法满足这种 web 应用需求了，虽然有 Ajax,SSE,Comet 等新技术可以实现比较有效的消息通信 ，但是它们仍然有所不足。Websocket 构建出了新的通信协议，很大程度上解决了以上技术暴露出的问题。本文首先介绍了基于 HTTP 的相关技术和应用，然后分析了 Websocket 协议，Websocket 的发展和应用。

关键词：HTTP，Websocket，instant messaging，socket.io

Abstract

This paper focuses on the Websocket protocol, and its development and relative applications. The current Internet for Web communication requirements are getting higher and higher: a higher amount of concurrency, the message should be more real-time and reliable. The traditional HTTP protocol can not meet such requirements of web application. Although there are new technologies such as Ajax, SSE and Comet, they can do more effective on message communications, but they still have some shortcomings. Websocket as a new communication protocol which tries to solve problems the above technologies exposed . This paper firstly introduces related technologies and applications based on HTTP, and then analyzes the development and application of Websocket.

Keywords： HTTP, Websocket, instant messaging, Socket.io

1 引言

在 web 通信领域从，出现了一系列的技术，这些技术的出现和应用都是对原有技术的补充，传统的 HTTP 协议已经无法满足现在快速发展的互联网应用需求了。很长时间，网络在很大程度上都是围绕着 HTTP 的请求/响应模式而构建的。客户端加载一个网页，然后直到用户点击下一页之前，什么都不会发生。AJAX 开始让网络变得更加动态，Comet 利用长轮询，客户端可打开指向服务器的 HTTP 连接，而服务器会一直保持连接打开，直到发送响应。服务器只要实际拥有新数据，就会发送响应。这些技术都是基于 HTTP 的，有个共同的问题，就是 HTTP 开销问题，这导致他们不适用于低延迟的应用。例如在实时性要求高的在线游戏中，这些技术不可能使用。

Websocket 的出现，改善了上述问题，作为一个新协议它实现了浏览器与服务器全双工通信，能更好的节省服务器资源和带宽并达到实时通讯。它和 HTTP 最大不同是：Websocket 是一种双向通信协议，在建立连接后，Websocket 服务器和 Browser/Client Agent 都能主动的向对方发送或接收数据，就像 Socket 一样；Websocket 需要类似 TCP 的客户端和服务器端通过握手连接，连接成功后才能相互通信。

学习和使用 Websocket 将使得这个技术更加的完善起来，从而能更好的指导我们在实际 Web 应用中的开发。

2 HTTP、Ajax、Comet 等技术

HTTP

HTTP 1.0 不支持 http 长连接，每次一个 http 请求响应后都关闭 tcp 连接，下个 http 请求会重新建立 tcp 连接。HTTp 1.1 支持长连接 ，和 HTTP1.0 主要区别： 在同一个 tcp 的连接中可以传送多个 HTTP 请求和响应.； 多个请求和响应可以重叠，多个请求和响应可以同时进行，更加多的请求头和响应头(比如 HTTP1.0 没有 host 的字段).

http 采用短轮询和长轮询的方法保持客户端和服务端的数据传输轮询，就是向服务器发送请求，服务器返回请求结果。短轮询就是不断的轮询的时候，每次都是即时发送，即时回复，这样来更新前端的数据，用来保证数据的前后台同步。长轮询是服务器收到

请求后如果有数据，立刻响应请求；如果没有数据就会 hold 一段时间，这段时间内如果有数据立刻响应请求；如果时间到了还没有数据，则响应 http 请求;浏览器受到 http 响应后立在发送一个同样 http 请求查询是否有数据;
对于短轮询,每次的连接断开的开销比较大;对于长轮询,当服务器端没有数据 hold 住连接时会造成浪费，容易产生服务器瓶颈。

Ajax

Ajax(Asynchronous JavaScript and XML 的简称)，由 Jesse James Garrett 首先提出。这种技术开创性地允许浏览器脚本（Javascript）发送 http 请求。传统的 web 应用想要与服务器交互，必须提交一个表单，服务器处理后返回一个全新的页面，而前后两个页面一般有很大一部分数据是相同的，这个过程传输有很多冗余的数据，浪费了带宽。Ajax 能够传输小部分数据,实现部分数据的更新到页面,减少了冗余数据的传输。Ajax 技术仍然采用的是 HTTP 协议，没能从根本上上改变 HTTP 的请求响应模式。

Ajax 基本上就是把 JavaScript 技术和 XMLHttpRequest 对象放在 Web 表单和服务器之间。当用户填写表单时，数据发送给一些 JavaScript 代码而不是 直接发送给服务器。相反，JavaScript 代码捕获表单数据并向服务器发送请求。同时用户屏幕上的表单也不会闪烁、消失或延迟。换句话说，JavaScript 代码在幕后发送请求，用户甚至不知道请求的发出。更好的是，请求是异步发送的，就是说 JavaScript 代码（和用户）不用等待服务器的响应。因此用户可以继续输入数据、滚动屏幕和使用应用程序。

然后，服务器将数据返回 JavaScript 代码（仍然在 Web 表单中），后者决定如何处理这些数据。它可以迅速更新表单数据，让人感觉应用程序是立即完成的，表单没有提交或刷新而用户得到了新数据。JavaScript 代码甚至可以对收到的数据执行某种计算，再发送另一个请求，完全不需要用户干预！这就是 XMLHttpRequest 的强大之处。它可以根据需要自行与服务器进行交互，用户甚至可以完全不知道幕后发生的一切

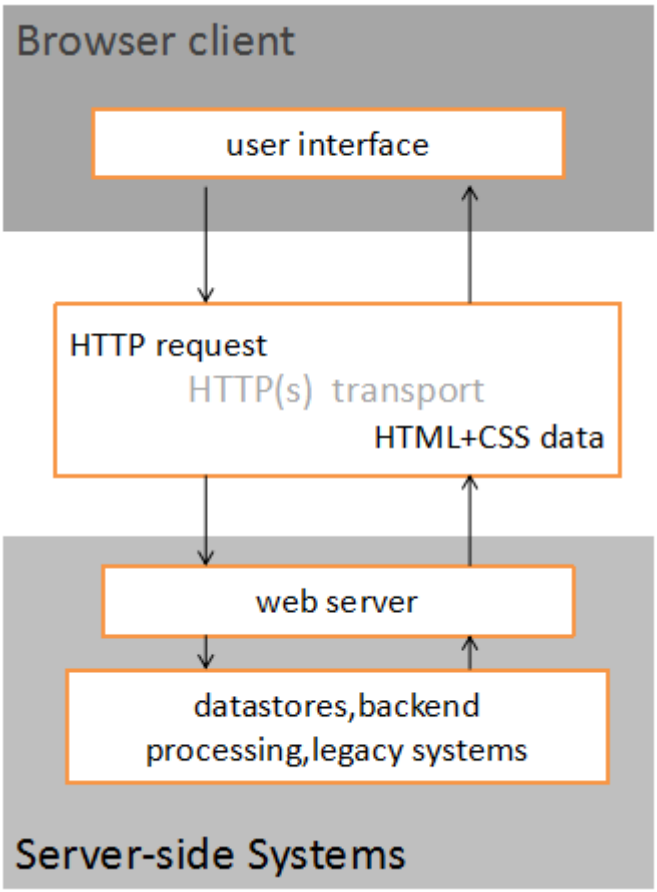
Ajax 应用程序应用到的基本技术

- HTML 用于建立 Web 表单并确定应用程序其他部分使用的字段。
- JavaScript 代码是运行 Ajax 应用程序的核心代码,帮助改进与服务器应用程序的通信。
- DHTML 或 Dynamic HTML，用于动态更新表单。我们将使用 div、span 和其他动态

HTML 元素来标记 HTML。

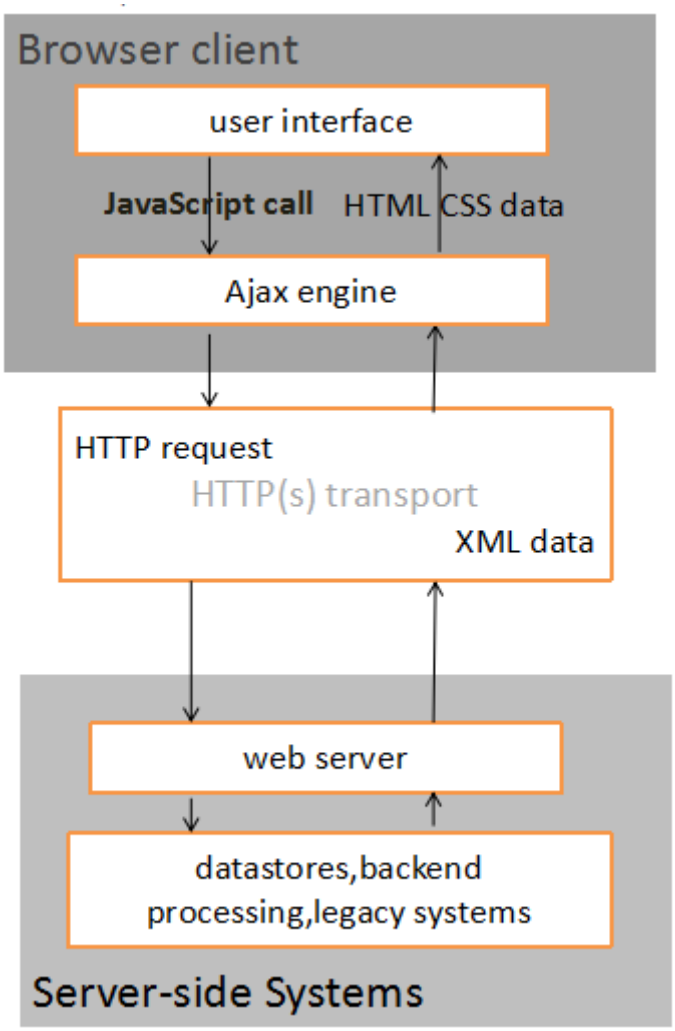
- 文档对象模型 DOM 用于(通过 JavaScript 代码)处理 HTML 结构和(某些情况下)服务器返回的 XML。

传统的 Web 应用模型如下图所示



图一：传统 Web 应用模型

基于 Ajax 的 web 应用模型如下图所示



图二：Ajax Web 应用模型

Ajax 客户端实例代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>无标题文档</title>

<script type="text/javascript">

 var xmlhttp;

 //创建异步对象

 function initXmlHttp(){
```

```
  if(window.ActiveXObject){    //IE 浏览器
    xmlhttp = new window.ActiveXObject("Microsoft.XMLHTTP");
  }
  else if(window.XMLHttpRequest){ //非 IE 浏览器
    xmlhttp = new window.XMLHttpRequest();
  }
 }
window.onload = initXmlHttp;
//发送异步请求
function sendRequest(){
  //传入一个 myname 参数 和 一个用于解决 IE 缓存问题的实时毫秒数
  xmlhttp.open("GET","AJAX_servers.aspx?myname=xg&" + new Date().getTime());
  //指定当 readyState 属性改变时的事件处理句柄 onreadystatechange
  xmlhttp.onreadystatechange = funState;
  xmlhttp.send(null);
 }
//获取异步结果
function funState(){
  if( xmlhttp.readyState == 4)
  {
    if( xmlhttp.status == 200 || //status==200 表示成功!
      xmlhttp.status == 0) //本机测试时，status 可能为 0。
    {
      var re = xmlhttp.responseText;
      //alert(re);
      document.getElementById("divShow").innerHTML = re;
    }
  }
 }
</script>
```

```
</head>
<body>
<button onclick="sendRequest();">发送</button>
<div id="divShow"></div>
</body>
</html>
```

服务端实例代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="AJAX_servers.aspx.cs"
Inherits="Web_1.AJAX_servers" %>
<%
    if (Request.HttpMethod == "GET")
    {
        string str = Request.QueryString[0];
        Response.Write(str + ": 我是来自服务器的文字！");
    }
%>
```

基于 Javascript 的一些框架，如 JQuery 等都对 Ajax 进行了相关的封装，所以使用上非常的方便。

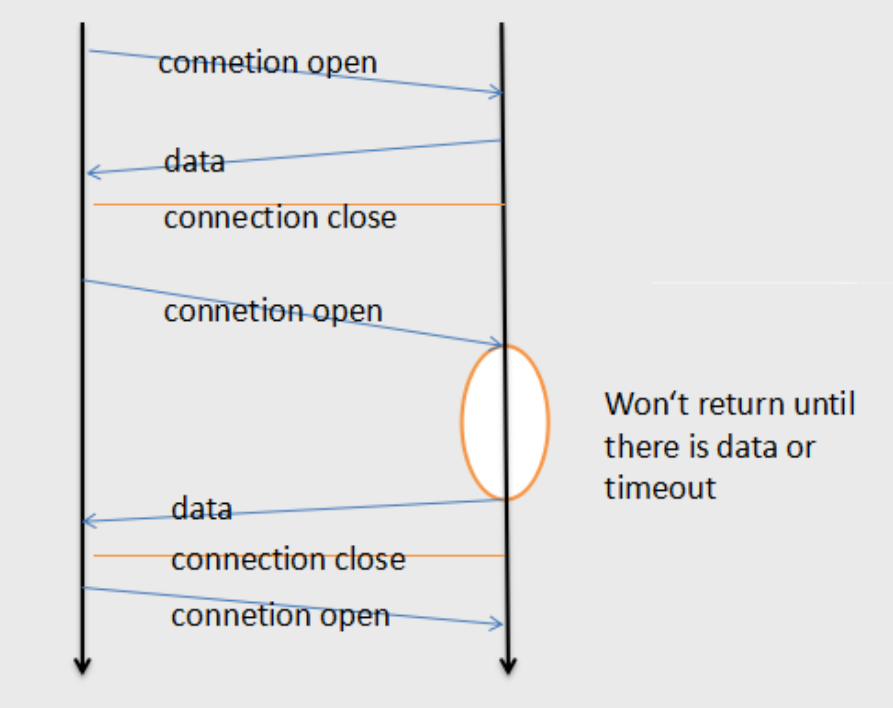
Comet“服务器推”

Comet 基于 HTTP 长连接的 web 推送技术，能使服务器实时地将更新的信息传送到客户端，而无须客户端发出请求，目前有两种实现方式，长轮询和 iframe 流。

长轮询是在打开一条连接以后保持，等待服务器推送来数据再关闭的方式。基于 AJAX 的长轮询（long-polling）方式 实现“服务器推”与传统的 AJAX 应用不同之处在于：

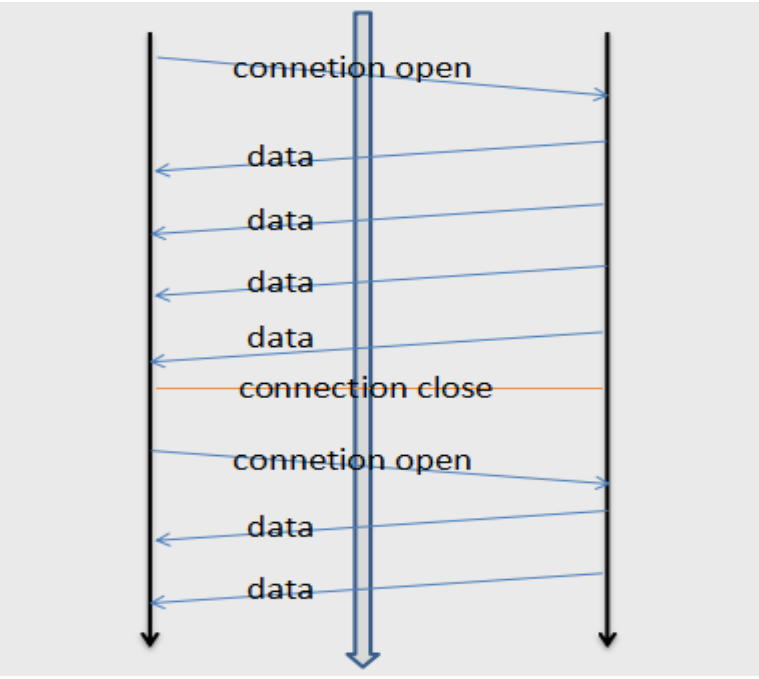
- 服务器端会阻塞请求直到有数据传递或超时才返回。
- 客户端 JavaScript 响应处理函数会在处理完服务器返回的信息后,再次发出请求，重新建立连接。
- 当客户端处理接收的数据、重新建立连接时，服务器端可能有新的数据到达；这些信息会被服务器端保存直到客户端重新建立连接，客户端会一次把当前服务器端所

有的信息取回。



图三：长轮询方式的数据传输

基于 iframe 流方式：iframe 是很早就存在的一种 HTML 标记， iframe 流方式是在页面中插入一个隐藏的 iframe,利用其 src 属性在服务器和客户端之间创建一条长链接，服务器向 iframe 传输数据（通常是 HTML，内有负责插入信息的 javascript），来实时更新页面。



图四：基于 iframe 流方式的数据传输

AJAX 方案是在 JavaScript 里处理 XMLHttpRequest 从服务器取回的数据，然后 Javascript 可以很方便的去控制 HTML 页面的显示。同样的思路用在 iframe 方案的客户端，iframe 服务器端并不返回直接显示在页面的数据，而是返回对客户端 Javascript 函数的调用，如“<script type="text/javascript">js_func(“data from server ”)</script>”。服务器端将返回的数据作为客户端 JavaScript 函数的参数传递；客户端浏览器的 Javascript 引擎在收到服务器返回的 JavaScript 调用时就会去执行代码。

每次数据传送不会关闭连接，连接只会在通信出现错误时，或是连接重建时关闭（一些防火墙常被设置为丢弃过长的连接， 服务器端可以设置一个超时时间， 超时后通知客户端重新建立连接，并关闭原来的连接）。

iframe 流方式的优点是浏览器兼容好，Google 公司在一些产品中使用了 iframe 流，如 Google Talk。

- 优点：实时性好（消息延时小）；性能好（能支持大量用户）。
- 缺点：长期占用连接，丧失了无状态高并发的特点。

Comet 的应用主要体现在股票系统、实时通讯。目前一些主流网站都有类似的原理，例如： WebQQ、开心网、校内等等，它们中消息动态都是采用类似的技术，只是具体实现方式不一样。

Comet 也有一些服务器框架如 Pushlet，，Pushlet 使用了观察者模式：客户端发送请求，订阅感兴趣的事件；服务器端为每个客户端分配一个会话 ID 作为标记，事件源会把新产生的事件以多播的方式发送到订阅者的事件队列里。Pushlet 基于 HTTP 流，这种技术常常用在多媒体视频、通讯应用中，比如 QuickTime。与装载 HTTP 页面之后马上关闭 HTTP 连接的做法相反，Pushlet 采用 HTTP 流方式将新变动的数据主动地推送到 client（客户端），再此期间 HTTP 连接一直保持打开。

3 Websocket 协议

Websocket 协议分为两部分：握手和数据传输。

来自客户端的握手，示例如下

```
GET /chat HTTP/1.1
Host: server.example.com
```

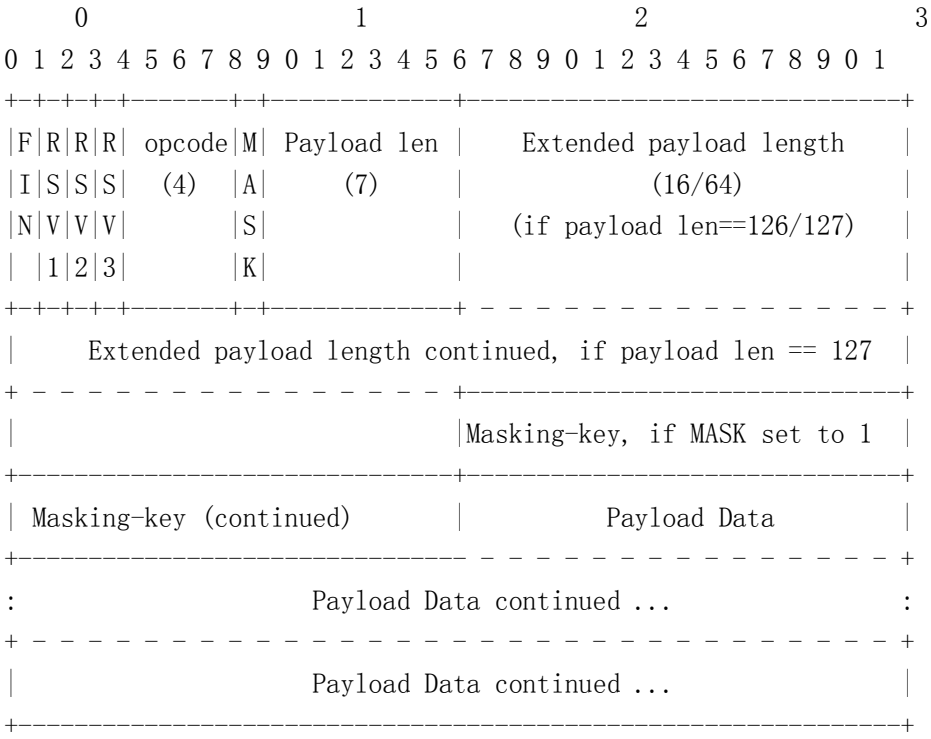
```
        Upgrade: websocket
        Connection: Upgrade
        Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
        Origin: http://example.com
        Sec-WebSocket-Protocol: chat, superchat
        Sec-WebSocket-Version: 13
```

来自服务端的握手，示例如下

```
      HTTP/1.1 101 Switching Protocols
        Upgrade: websocket
        Connection: Upgrade
        Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
        Sec-WebSocket-Protocol: chat
```

当客户端和服务端握手成功之后，就可以开始数据传输了,数据传输部分使用序列 frames 方式.

一个 frames 的格式如下



FIN：1 位，是否是消息的结束帧(分片)

RSV1，RSV2，RSV3： 分别都是 1 位，预留，用于约定自定义协议。 如果双方之间没有约定自定义协议，那么这几位的值都必须为 0,否则必须断掉 WebSocket 连接；

Opcode:4 位操作码，定义有效负载数据，如果收到了一个未知的操作码，连接也必须断掉，以下是定义的操作码：

%x0 表示连续消息分片

%x1 表示文本消息分片

%x2 表未二进制消息分片

%x3-7 为将来的非控制消息片断保留的操作码

%x8 表示连接关闭 %x9 表示心跳检查的 ping

%xA 表示心跳检查的 pong

%xB-F 为将来的控制消息片断的保留操作码

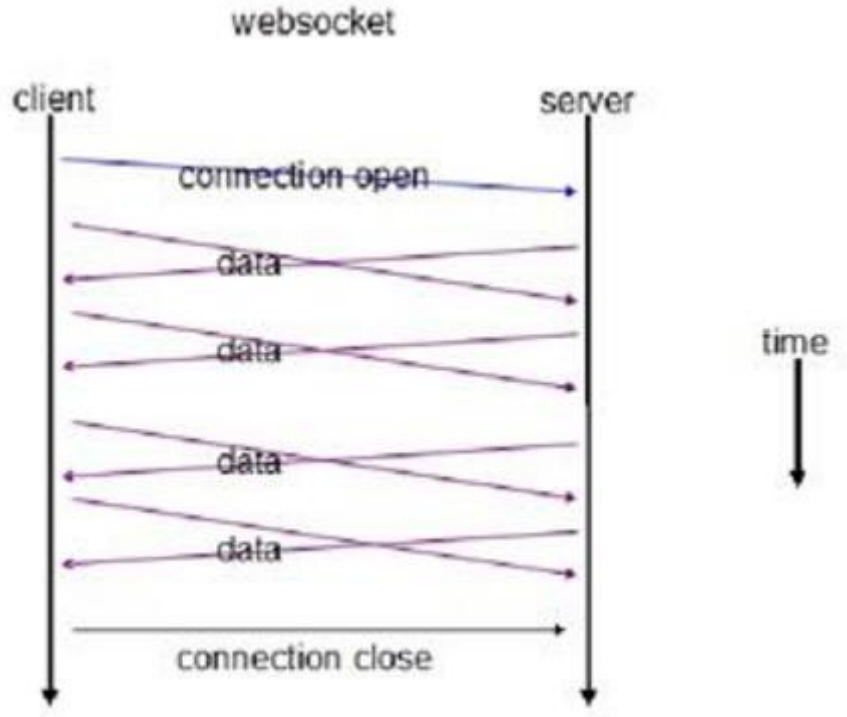
Mask： 定义传输的数据是否有加掩码,如果设置为 1,掩码键必须放在 masking-key 区域，客户端发送给服务端的所有消息，此位的值都是 1；

Payload length： 传输数据的长度，以字节的形式表示：7 位、7+16 位、或者 7+64 位。如果这个值以字节表示是 0-125 这个范围，那这个值就表示传输数据的长度；如果这个值是 126,则随后的两个字节表示的是一个 16 进制无符号数,用来表示传输数据的长度；如果这个值是 127,则随后的是 8 个字节表示的一个 64 位无符合数，这个数用来表示传输数据的长度。多字节长度的数量是以网络字节的顺序表示。负载数据的长度为扩展数据及应用数据之和,扩展数据的长度可能为 0,因而此时负载数据的长度就为应用数据的长度。

Extension data:x 位，如果客户端与服务端之间没有特殊约定，那么扩展数据的长度始终为 0，任何的扩展都必须指定扩展数据的长度，或者长度的计算方式，以及在握手时如何确定正确的握手方式。如果存在扩展数据，则扩展数据就会包括在负载数据的长度之内。

...

采用 websocket 协议，客户端和服务端的交互入下图所示



图五：Websocket 连接连接和传输数据

Websocket 在客户端和服务端建立连接后，双方可以畅通的发送数据，不需要重新发起连接请求，知道双方确定关闭连接。

握手部分

- websocket 是 独立的基于TCP的协议，其跟http协议的关系仅仅是 WebSocket 的握手被 http 服务器当做 Upgrade request http 包处理。 websocket 有自己的握手处理。 TCP 连接建立后，client 发送 websocket 握手请求. 请求包需求如下：
- 必须是有效的 http request 格式

  HTTP request method 必须是 GET，协议应不小于 1.1 如： Get /chat HTTP/1.1
- 必须包括 Upgrade 头域，并且其值为“websocket”
- 必须包括"Connection" 头域，并且其值为 "Upgrade"
- 必须包括"Sec-WebSocket-Key"头域,其值采用 base64 编码的随机 16 字节长的字符序列， 服务器端根据该域来判断 client 确实是 websocket 请求而不是冒充的，如 http。响应方式是，首先要获取到请求头中的 Sec-WebSocket-Key 的值，再把这一

GUID"258EAFA5-E914-47DA-95CA-C5AB0DC85B11"加到获取到的 Sec-WebSocket-Key 的值的后面，然后拿这个字符串做 SHA-1 hash 计算，然后再把得到的结果通过 base64 加密,就得到了返回给客户端的 Sec-WebSocket-Accept 的 http 响应头的值。如果请求来自浏览器客户端，还必须包括 Origin 头域。该头域用于防止未授权的跨域脚本攻击，服务器可以从 Origin 决定是否接受该 WebSocket 连接。

- 必须包括"Sec-webSocket-Version" 头域，当前值必须是 13.
- 可能包括"Sec-WebSocket-Protocol"，表示 client（应用程序）支持的协议列表，server 选择一个或者没有可接受的协议响应之。
- 可能包括"Sec-WebSocket-Extensions"， 协议扩展， 某类协议可能支持多个扩展，通过它可以实现协议增强
- 可能包括任意其他域，如 cookie

Server 接手到握手请求后应处理该请求包括：

- 处理请求包括处理 GET 方法
- 验证 Upgrader 头域
- 验证 Connection 头域
- 处理 Sec-WebSocket-Key 头域，方法见上；
- 处理 Sec-WebSocket-Version
- 处理 Origin 头域，可选，浏览器必须发送该头域
- 处理 Sec-WebSocket-Protocol 头域，可选
- 处理 Sec-WebSocket-Extensions 头域，可选
- 处理其他头域，可选

Server 发送握手响应，这里只介绍服务器接受该连接情况下，包括：

- http Status-Line
- Upgrade 头域 ，值必须是"websocket"
- Conntion 头域，值必须是：“Upgrade”
- Sec-WebSocket-Accept” 头域，该头域的值即处理 Sec-WebSocket-Key" 域后的结果。
- 可选的"Sec-WebSocket-Protocol"头域
- 可选的"Sec-WebSocket-Extensions"头域

数据传输

在 WebSocket 协议中，数据使用帧序列来传输。基本帧协议定义了带有操作码（opcode）的帧类型、负载长度、和用于“扩展数据”与“应用数据”及它们一起定义的“负载数据”的指定位置。某些字节和操作吗保留用于未来协议的扩展。

分片的主要目的是允许当消息开始但不必缓冲该消息时发送一个未知大小的消息。如果消息不能被分片，那么端点将不得不缓冲整个消息以便在首字节发生之前统计出它的长度。对于分片，服务器或中间件可以选择一个合适大小的缓冲，当缓冲满时，写一个片段到网络。 第二个分片的用例是用于多路复用，一个逻辑通道上的一个大消息独占输出通道是不可取的，因此多路复用需要可以分割消息为更小的分段来更好的共享输出通道。

控制帧由操作码确定，其中操作码最重要的位是 1。当前定义的用于控制帧的操作码包括 0x8 （Close）、0x9（Ping）、和 0xA（Pong）。 操作码 0xB-0xF 保留用于未来尚未定义的控制帧。控制帧用于传达有关 WebSocket 的状态。控制帧可以插入到分片消息的中间。所有控制帧必须有一个 125 字节的负载长度或更少， 必须不被分段。

4 Websocket 扩展应用

实时消息推送

利用 Node.js 构建的 HTTP 服务器，充分利用 node.js 的单线程，非阻塞 I/O 调用等特性，使得其并发性非常强大。在消息的推送过程中，传输的实体是消息本省，不需要发送 http 头信息等，并且连接是保持的直到有一方断开。Websocket 对比 HTTP，Webscoekt 发送的数据将更少，并能保持长连接，这将大大降低消息传送的延时。

Socket.io

Socket.io 是完全由 JavaScript 实现、基于 Node.js、支持 WebSocket 的协议用于实时通信、跨平台的开源框架，它包括了客户端的 JavaScript 和服务器端的 Node.js。当然，Socket.io 除了支持 WebSocket 通讯协议外，还支持许多种轮询（Polling）机制以及其它实时通信方式，并封装成了通用的接口，并且在服务端实现了这些实时机制的

相应代码。Socket.IO 实现的 Polling 通信机制包括 Adobe Flash Socket、AJAX 长轮询、AJAX multipart streaming、持久 Iframe、JSONP 轮询等。Socket.IO 能够根据浏览器对通讯机制的支持情况自动地选择最佳的方式来实现网络实时应用。

Socket.IO 已经具有众多强大功能的模块和扩展 API,如(session.socket.io)(http session 中间件，进行 session 相关操作）、socket.io-cookie（cookie 解析中间件）、session-web-sockets（以安全的方式传递 Session）、socket-logger（JSON 格式的记录日志工具）、websocket.MQ（可靠的消息队列）、socket.io-mongo（使用 MongoDB 的适配器）、socket.io-redis（Redis 的适配器）、socket.io-parser（服务端和客户端通讯的默认协议实现模块）等。

Socket.IO 实现了实时、双向、基于事件的通讯机制,它解决了实时的通信问题，并统一了服务端与客户端的编程方式。例如一个简单的服务端示例代码

```
var io = require('socket.io').listen(8080);
io.sockets.on('connection', function (socket) {
        socket.emit('news', { hello: 'world' });
        socket.on('my other event', function (data) {
        console.log(data);
});
});
```

客户端示例代码

```
<script src="socket.io.min.js"></script>
<script>
  var socket = io.connect('http://localhost:8080');
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my: 'data' });
  });
</script>
```

其它框架中使用 Websocket

1. Netty

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。

2. Undertow

Undertow 是一个采用 Java 开发的灵活的高性能 Web 服务器，提供包括阻塞和基于 NIO 的非堵塞机制。

3. Jetty

Jetty 是一个开源的 Servlet 容器,它为基于 Java 的 web 容器,例如 JSP 和 Servlet 提供运行环境。Jetty 是使用 Java 语言编写的，它的 API 以一组 JAR 包的形式发布。开发人员可以将 Jetty 容器实例化成一个对象，可以迅速为一些独立运行（stand-alone）的 Java 应用提供网络和 web 连接。

4. Vert.x

Vert.x 框架基于事件和异步，依托于全异步 Java 服务器 Netty，并扩展了很多其他特性，以其轻量、高性能、支持多语言开发。

5. Grizzly

Grizzly 是一种应用程序框架，专门解决编写成千上万用户访问服务器时候产生的各种问题。使用 JAVA NIO 作为基础，并隐藏其编程的复杂性。容易使用的高性能的 API。带来非阻塞 socketd 到协议处理层。利用高性能的缓冲和缓冲管理使用高性能的线程池。

6. spray-websocket

Spray 是一个开源的 REST/HTTP 工具包和底层网络 IO 包，基于 Scala 和 Akka 构建。轻量级、异步、非堵塞、基于 actor 模式、模块化和可测试是 spray 的特点，spray-websocket 扩展了 spray,实现了 Websocket。

7. Go

Go 实现 Websocket。

5 小结

通过对HTPP、Ajax、Comet等的分析，熟悉了相关web通信的技术，了解各个技术的优缺点，websocket带来了新的协议，对原来的http协议是个有力的改进。再分析过程中，总结为几下几点

1) 在研究新协议前，需要总结分析原先的协议和相关技术，只有在深入理解原有技术的基础上，才有可能发现问题，并提出先的改进措施。
2) 在研究新协议时，需要对比原先协议，明白个中新技术，还要了解新技术的一些应用，在应用过程中，可以发现更多的知识，这样使得技术更加的发展。

参考文献

[1] The WebSocket Protocol rfc[J/OL]https://tools.ietf.org/html/rfc6455

[2] 齐华，李佳，刘军：基于 Websocket 的消息实时推送设计与实现[J]微处理机，2016，3

[3]温照松，易仁伟，姚寒冰：基于 WebSocket 的实时 Web 应用解决方案[J]电脑知识与技术，2012，8（16）

[4] Gerard Nicolas,Karim Sbata,Elie Najm：Websocket Enabler: achieving IMS and Web services end-to-end convergence[J]ACM，2011,27(1)：81-96

[5]陆晨，冯向阳，苏厚勤：HTML5 WebSocket 握手协议的研究与实现[J]计算机应用与软件，2015，32(1)

[6] Gerard Nicolas,Karim Sbata,Elie Najm:Websocket Enabler: achieving IMS and Web services end-to-end convergence[J]ACM,2011,27(1):81-96

[7]Mikko Pohja: Server Push with Instant Messaging[J]ACM，2009，653-658

[8]Vanessa Wang，Frank Salim，Peter Moskovits：The Definitive Guide to HTML5 WebSocket[M]Apress，2013-3