# Comet/WebSocket

在互联如此发达的时代，HTTP协议成为了我们每天都会用到的，也是最为常用的协议，但HTTP协议是不是都能满足我们的日益的互联应用需求呢，这是不够的．技术总是在不断的更新，从HTTP 0.9到现在的1.1，HTML5也在逐渐进入我们的应用．

在HTTP技术的发展过程中，有一些比较重要的技术：浏览器缓存，网页压缩，AJAX异步请求．今天为大家呈现是服务器端的推技术．将服务器的数据主动送去浏览器客户端．

服务器的推技术会应用场景：监控系统，股市行情，在线聊天室，网页游戏，实时报价系统．我们比较常见的如：IM应用：meebo．QQ，同样它也可以应用于网页游戏．

我们把服务器推技术分成两个部分来说，一个是基于AJAX来实现的Comet方式，一个是HTML5中的WebSocket实现方式，当然还可以通过FLASH的方式来实现，这个是目前许多网页游戏的实现方式．

下面是两个例子，一个基于comet实现，一个基于websocket实现

Comet简单聊天系统：<http://www.zeitoun.net/articles/comet_and_php/start>

<http://uu1758.com/comet/index.html>

WebSocket：<http://websocket.org/echo.html>

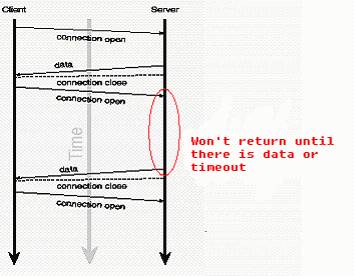
## AJAX Comet

维基百科：Comet is a web application model in which a long-held HTTP request allows a web server to push data to a browser, without the browser explicitly requesting it. Comet是一种用于web的技术，能使服务器能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，长轮询和流。

**长轮询(long polling)**

在以往的网页聊天室实现方式中，通常是由定时器来实现，在浏览器客户端设置一个定时器，当客户端到时了设定的时间间隔时，由客户端重新发起请求，实现页面数据的刷新．这种方式的实现存在许多的问题，如果数据变化太快，用户不能及时了解的数据的变化，从另一方面来讲如果请求的数据没有变化，那么就会消耗一些无谓请求和系统资源．这种频繁轮询的方式是存在问题的，长轮询将会解决这个问题．

长轮询的方式思路：首先由客户端发出AJAX请求，服务端在收到该请求如果数据还没有准备好，服务器不会马上响应客户端，而是将请求挂起．阻塞的请求一直到有数据产生或者连接超时才会有响应给到客户端。而客户端在接收到服务端的指令之后会马上发出新的请求，没有数据的话服务器再次挂起请求．



 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。

这种方式的实现还需要解决一个超时问题．也就是在客户端和服务器端之间需要保持住这个连接，要保持这个连接就可以通过心跳的方式来维护．服务器端在阻塞读时会设置一个时限，时限到时,服务器端发给客户端发送一个信息，这个信息可以没有任何数据，我们可以称它为心跳信息。此时如果客户端已经关闭，服务器往通道写数据会出现异常，需要发送错误信息通知客户端，同时释放资源、关闭连接。

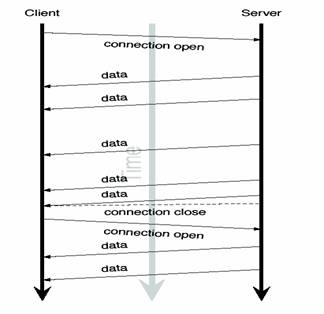
这种方式的实现，主要在于客户端的进步，AJAX异步请求的出现，为我们解决了这个问题．

**IFrame流**

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。

这种方式是使用了iframe的机制，然后使得这个iframe请求一个特定的URL，并通过对这个页面的加载不断的从服务端抓回数据，这里从服务端抓回的数据大多是对页面当前JavaScript函数的引用和操作。

这个方案的一个明显不足之处是页面会一直显示正在加载，在我测试时Google的Chrome浏览器上会一直显示加载．而在IE上可以通过google的htmlfile的ActiveX控件来解决这个问题。



## HTML5 WebSocket

在展开WebSocket之前，我们需要明确的是IE9以及IE9之前的所有版本都不支持WebSocket，我们如果要测试请使用WebKit内核的浏览器，如Chrome.

WebSocket 是HTML5中一种新的协议，定义了客户端和服务器通讯的WebSocket方式，在得到浏览器支持以后，实现了浏览器与服务器的双向通讯，WebSocket将会取代Comet成为服务器推送的方法．

WebSocket的思想：使用原有HTTP协议已经满足不了我们的需求了，于是我们重新建设了一套与HTTP平行的WebSocket协议．在浏览器客户端我们可以以短连接的方式访问WEB服务，也可以访问WebSocket服务．WebSocket顾名思义它就是一个建立在WEB通道上的一个Socket，通过这个Socket我们将更灵活的处理我们的业务需求．如：我们的服务器推应用．WebSocket把这个应用从HTTP中剥离出来，以一个单独服务的形式进行部署，这样也更灵活，同时WebSocket的数据我更小，流量更省．

WebSocket在使用过程中主要分成两个步骤：一是握手过程，二是数据收发过程．

### 握手过程

通过ws协议发送握手请求，我们抓包数据如下，

GET /chat HTTP/1.1

Host: server.example.com 8000

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

Sec-WebSocket-Origin: http://example.com

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 8

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

通过数据我们首先要明白的是ws:// server.example.com:8000/chat这个WebSocket服务器它是不是建立在80端口之上，而是用了我们ws自己的端口．它是另外一条数据通道，在数据交互之前，客户端要先发送请求头信息到服务端，通过握手来建立连接，整个过程都是使用了HTTP协议，也就是通过HTTP协议来建立WebSocket，然后再在socket上面做数据交互．

服务端会对请求进行检查，如果是合法的请求头，则会发送握手信息到客户端。同时为每个客户端建产一个socket用来通信。这个socket有输入流，也有输出流，是一个全双工的tcp通信端口。当然客户端也会对这个连接进行合法性判断，如果数据不合法，直接丢弃．

在这里有一个重要的Header就是Sec-WebSocket-Accept，这个Header需要进行加密计算，在ws协议中如此说明：<http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17>

Page 7:

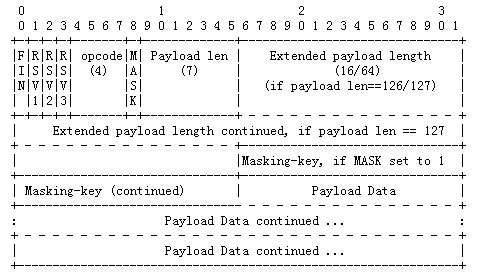
|Sec-WebSocket-Key| header field had the value "dGhlIHNhbXBsZSBub25jZQ==", the server would concatenate the string "258EAFA5-E914-47DA-95CA-C5AB0DC85B11" to form the string "dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA-C5AB0DC85B11".The server would then take the SHA-1 hash of this, giving the value 0xb3 0x7a 0x4f 0x2c 0xc0 0x62 0x4f 0x16 0x90 0xf6 0x46 0x06 0xcf 0x38 0x59 0x45 0xb2 0xbe 0xc4 0xea. This value is then base64-encoded (see Section 4 of [RFC4648]), to give the value "s3pPLMBiTxaQ9kYGzzhZRbK+xOo=". This value would then be echoed in the |Sec-WebSocket-Accept| header field.

步骤：

1. 获取到请求头中的Sec-WebSocket-Key的值；
2. 再把GUID "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"加到获取到的Sec-WebSocket-Key的值的后面
3. 然后拿这个字符串做SHA-1 hash计算
4. 最后再把得到的结果通过base64加密，得到的数据就是要返回给客户端的Sec-WebSocket-Accept的http响应头的值

### 数据收发过程

ws协议Page30：



FIN: 为1则为结束包,不是话,则会继续往下读数据.

Opcode: 0x01:文本,0x02:二进制编码 0x08连接关闭 09心跳ping 0A 心跳pong

MASK：设置为1,都会有数据masking-key, 0则没有masking-key

注意的是在客户端往服务器送数据时，一定要求有掩码，而服务器下发给客户端掩码则是可有可无

                           必须有掩码

           客户端----------------------->服务器

                  <-----------------------

                          掩码（可选）

Payload len: 占用的长度可能：7 bits或者 7+16 bits或者 7+64 bits

如果为 11111102 后2个字节为具体的内容长度

如果为 1111111后8字节为具体的内容长度

如果小于11111102 则本身就是长度,不用再往为取内容长度.

例子：

helloworld

S🡪C： 81:0a:68:65:6c:6c:6f:77:6f:72:6c:64

C🡪S： 81:8a:84:85:db:de:ec:e0:b7:b2:eb:f2:b4:ac:e8:e1

我们是发现数据包相对于HTTP协议，它的数据量相当的小了．

协议中例子：

o A single-frame unmasked text message

\* 0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f (contains "Hello")

o A single-frame masked text message

\* 0x81 0x85 (0x37 0xfa 0x21 0x3d) (0x7f 0x9f 0x4d 0x51 0x58) (contains "Hello")

我们会发现，上面有”Hello”与下面的”Hello”它的编码会不一样，下面的数据具体的解析是这样：(0x81 0x85)为头，(0x37 0xfa 0x21 0x3d)为mask key，(0x7f 0x9f 0x4d 0x51 0x58)才是具体内容．这个数据是 Hello的ascii码和掩码做异或运算算出来的。 将内容的第零个字符'H'和第零个掩码异或， 第一个字符'e'和第一个掩码异或......

在c语言表示就是res = str[n]^mask[n%4]

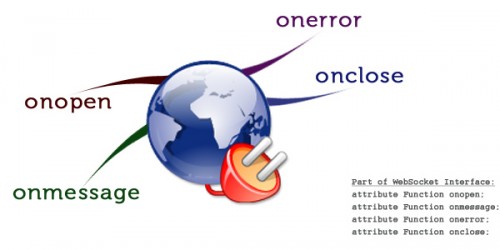
0x37 0011 0111

0x48 0100 1000

-------------------------

0x7F 0111 1111

### 客户端实现：



var ws = new WebSocket("**ws://localhost:8000/comet.do**");

ws.onmessage = function(evt) {

valueLabel.innerHTML = evt.data;

};

ws.onclose = function() {

};

ws.onopen = function() {

ws.send("Hello, Server!");

};

目前的IE等浏览器不支持WebSocket，要提供WebSocket的事件处理、返回传输、在服务器端使用一个统一的API，那么该怎么办呢？幸运的是，Guillermo Rauch创建了一个Socket.IO技术。

<script src="http://cdn.socket.io/stable/socket.io.js"></script>

另外淘宝的实现：

JS+Nginx+Nginx\_HTTP\_Push\_Module：

这个方案是Nginx的一个第三方模块，Nginx本身就实现了非阻塞I/O的特性，所以才能承载很高的并发。Nginx\_HTTP\_Push\_Module这个模块似乎是有一些内存方面的BUG，在淘宝的博客中有提到，他们对这块做了优化处理．

参考:

websocket protocol:

<http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17>

基于html5的WebSocket使用方式。承接jetty配置

<http://yangzc106.iteye.com/blog/1030473>

HTML5 WebSockets 基础使用教程

<http://kb.cnblogs.com/page/72745/>

Comet研究

<http://lync.in/research-on-comet/>

小例子

<http://www.zeitoun.net/articles/comet_and_php/start>

用Python实现一个简单的WebSocket服务器

<http://www.kuqin.com/networkprog/20111127/315464.html>

websocket通讯协议（10版本）简介

<http://songpengfei.iteye.com/blog/1178310>