# 基于Web Socket的网络实时通信研究

刘启光，孙向阳

悉尼工商学院，上海大学

在Web实时通信领域使用新的Web Socket协议来解决传统的Web实时通信解决方案的局限性，如轮询、长轮询、Flash插件。本文介绍了Web Socket技术的特点，分析了Web Socket协议与HTTP协议的不同，提供一个客户端与服务端Web Socket的实现，证明了Web Socket可以减少网络流量、降低网络延迟，展示了Web Socket在未来实时网络应用的前景。

关键词：Web Socket；全双工；HTTP流；长轮询；延迟

1. 介绍

在信息时代的飞速发展中，互联网已经成为了人们生活中不可缺少的一部分。人们对互联网的需求已经从Web1.0时代的信息可访问性转变为Web2.0时代的信息交互，并且在越来越多的定价系统、电子商务系统和新闻公告系统中都能看到信息的及时交互。

目前，客户端浏览器与服务器之间的通信是基于超文本传输协议（HTTP）——一个基于请求-相应和无状态的应用层协议。一个HTTP客户端发起一个请求。它建立一个传输控制协议（TCP）连接。在接收到客户端请求消息后，服务器发送一个消息作为相应并终止连接。在这种模式下，服务器无法实时将数据发送给客户端。因此，像Flash、Comet和AJAX长轮询被用来实现客户端与服务器之间的实时通讯。然而，这些技术并不能真正实现实时通讯，因为他们有的需要安装插件，有的会导致服务器负载过大。HTML5和Web Socket的出现实现了基于Web系统的实时数据传输，到目前为止，他们是解决这一问题的最佳方案。

1. 传统的Web实时通信解决方案

Web开发者完成浏览器与服务器之间实时通信的Mary解决方案是是过时的。

程序自动轮询刷新页面替代手动刷新是应用于浏览器最早的解决方案。容易实现并且不需要额外的客户端和服务器是这个解决方案的最大优势。然而，在你这个解决方案中存在明显的不足之处，它很难找出数据更新的频率，因此浏览器不能及时得到最新的数据。此外，在一段时间内没有数据更新的情况下，浏览器频繁的请求会产生不必要的网络流量，并造成不必要的服务器负担。

为了使服务器与浏览器在任何时候通信，Web开发者设计了一种被称为长轮询或者Comet的新的访问机制，通过该机制，服务器将保存浏览器的新请求一段时间而不是立刻相应。如果数据更新发生这段时间内，服务器将响应浏览器行的数据。通过这个机制，浏览器能够及时获得服务器端最新的数据。然而，如果在你大并发量下，服务器内存和计算能力将大量消耗在维持那些HTTP连接上。

开发者也尝试“HTTP流”访问机制。其主要区别是，服务器将永远不会关闭有浏览器发起的连接，服务器将使用这个连接在任何时候发送消息。在这种情况下，服务器没有信号完成连接，服务器的响应可能会被防火墙和代理服务器在网络中缓冲，使浏览器在接收数据过程中发生错误。

1. Web Socket介绍

Web Socket作为HTML的新功能，它被定义为使网页可以使用Web Socket协议与远程主机进行全双工通信的技术。它介绍了Web Socket接口并定义了一个通过单一的socket在Web上运行的全双工通信通道。HTML5 Web Socket以最小的开销有效地提供了一个socket连接到互联网。与经常用来模拟全双工通信传送实时数据的AJAX轮询和Comet相比，它大量减少了网络流量和延迟。因此，它是构建可扩展，实时网络通信系统的理想技术。

使用HTML5 Web Socket去连接一个Web客户端与远程终端，一个新的Web Socket实例应该被代表远程终端连接的有效URL初始化。Web Socket定义ws://和wss://方案分别作为Web Socket和加密Web socket连接。Web Socket建立在客户端与服务器更新HTTP协议为Web Socket协议的初次握手间。Web Socket协议使用标准的HTTP端口（80和443），因此，它被称为“代理服务器和防火墙友好的协议”。因此，HTML Web Socket不需要任何新的硬件。没有任何中间服务器（代理或者反向代理，防火墙，负载均衡路由等），只要客户端与服务器支持Web Socket协议，一个新的Web Socket连接就能够被建立。

1. Web Socket连接与HTTP连接的比较

客户端与服务器之间的通信经常是基于HTTP连接，它需要附上客户端的请求头和服务器的响应头，根据HTTP协议定义，这些头中包含了一些传输控制信息，如协议类型、协议版本、浏览器类型、传输语言、编码、超时时间、Cookie和Session。在像Firebug和Turning on Live HTTP Headers软件的帮助下，请求头和响应头能够被非常清楚地观察到。下面是一个请求头和响应头被定义的例子：

From client (browser) to server: GET /long-polling HTTP/1.1 Host: www.kaazing.com User-Agent: Mozilla/5.0 (X11; U; Linux x86\_64;

en-US; rv:1.9) Gecko/2008061017 Firefox/3.0 Accept:

text/html,application/xhtml+xml,application/xml;q = 0.9, \*/\*; q = 0.8

Accept-Language: en-us,en;q = 0.5 Accept-Encoding: gzip,deflate Accept-Charset: ISO-8859-1,utf-8;q = 0.7,\*;q = 0.7 Keep-Alive: 300

Connection: keep-alive Cache-Control: max-age = 0 Referer: http://www.example.com/

From server to client (browser): Date: Tue, 16 Aug 2008 00:00:00 GMT Server: Apache/2.2.9 (Unix) Content-Type: text/plain Content-Length: 12

Hello World

从上面的两个头中可以看到，除了数据“Hello World”，在整个浏览器与服务器的交互过程中，头中的大部分数据对于终端用户是无用的，更不用说Cookie和Session（在大部分网站中，这两个包含的信息比控制信息要多）。此外，这些头的类型将被包含在每次交互中。因此，如果采用轮询或则Comet解决方案，它必将浪费大量的带宽，产生大量的网络流量。此外，在处理请求和响应时构建和分析头将占用一些时间，并导致一定程度的延迟。轮询和Comet的这些缺点已经表明，这些技术必将被其他的实时通信技术替代。让我们回到Web Socket连接。

Web Socket 使用HTTP升级机制升级到Web Socket协议。Web Socket的握手机制与HTTP是一致的。因此，HTTP服务能够与Web Socket共享默认的HTTP和HTTPS端口（80和443）。为了建立一个新的Web Socket连接，HTTP协议将在客户端与服务器最初握手期间升级为Web Socket协议。一旦连接被建立，基于全双工模式的Web Socket将被来回传播在客户端与服务端之间。初次握手的头如下：

From client (browser) to server: GET /text HTTP/1.1 Upgrade: WebSocket Connection: Upgrade

Copyright © 2012 SciRes.

IJCNS

Host: www.websocket.org

Fromservertoclient(browser): HTTP/1.1 101 WebSocket Protocol Handshake Upgrade: WebSocket Connection: Upgrade

Hello world

它清除的显示了，Web Socket连接的头中包含的控制信息要比HTTP连接少很多。另一方面，Web Socket协议规定Web Socket连接头中不允许携带Cookie和Session，并且最重要的是，一旦连接被建立成功，客户端能够和服务端自由通信，并且仅仅两个比特的控制信息，被附属在终端用户所需的数据上，它们被UTF-8编码，一个比特是“\x00”定位开始，另一个是“\xFF”定位结束。这种Web Socket的定义大量减少了带宽和处理Web Socket连接头所用的时间，使得网络流量更少和延迟更低。这是使用Web Socket建立基于Web的实时通信比轮询和Comet更适合的原因。

从安全角度来看，Web Socket协议和HTTP协议都能实现安全传输。Wss和https是他们各自独立的安全协议。因此，Web Socket在网络流量、延迟和安全方面被认为是理想实时通信技术。

1. Web Socket技术的实现
2. **客户端Web Socket的实现**

客户端的实现是比较简单的。下面是W3C工作组给出的Web Socket接口定义：

[Constructor(in DOMString url, in optional DOM- String protocol)]

interface WebSocket { readonly attribute DOMString URL; // ready state const unsigned short CONNECTING = 0; const unsigned short OPEN = 1; const unsigned short CLOSED = 2; readonly attribute unsigned short readyState; readonly attribute unsigned long bufferedAmount; // networking

attribute Function onopen; attribute Function onmessage; attribute Function onclose;

oolean send(in DOMString data); Copyright © 2012 SciRes.

void close(); };

根据定义的接口和构造方法，一个新的Web Socket实例能够通过两个参数初始化，一个必要的参数是一个有效的网络地址，另一个可选参数是协议类型。在浏览器中，Web Socket对象由JavaScript操作。一个新的Web Socket能够一段简单的代码创建：

var myWebSocket = new WebSocket (“ws://www.websocket.org”);

“ws”代表Web Socket连接，与此相关的是“wss”，它代表安全的Web Socket连接。在初始化之前，不需要检查客户端浏览器是否支持客户端技术：

if (“WebSocket” in window)

{var ws = new WebSocket (“ws://example.com/service”);}

else {alert(“WebSockets NOT supported here”);}

在发送消息之前，处理Web socket连接一系列事件的几个方法应该被注册，如，连接简历成功，接受消息，关闭消息。

myWebSocket.onopen = function(evt) {alert(“Connection open ...”); };

myWebSocket.onmessage = function(evt) {alert( “Re- ceived Message: “+ evt.data);};

myWebSocket.onclose = function(evt) {alert(“Connection closed.”);};

对于发送消息，只需要调用方法，然后以消息内容为默认参数。消息发送后，调用断开连接方法关闭连接。

myWebSocket.postMessage(“Hello Web Socket! “);

myWebSocket. disconnect(); stockTickerWebSocket. disconnect();

**B．服务端Web Socket实现**

相对于客户端，服务端是复杂的，大多数客户端上的操作，如生成请求头、解析请求头、提取有用数据都是由浏览器自动完成的，但是这些在服务器都没有对应的实现，并需要开发者们手动开发。服务端Web Socket实现主要取决于Socket编程，这些在C#，Java和C++中是常见的。在本文中，坐着使用C#实现服务端Socket编程。

首先，应该创建一个监听器，用于监听新的网络请求。

private Socket serverListener = new Socket (Address- Family. InterNetwork, SocketType. Stream, rotocolType. IP);

接收功能是负责监听新来的请求，因此，它应该被放在一个一直运行着的循环中用于获得客户端在任何时候的请求。

while (true) { Socket sc = serverListener.Accept(); //get a new connection if (sc != null) { ... }//process the request }

当收到一个新的连接请求，类似的，在整个通信过程中几个处理接收消息，发送消息，关闭连接的方法需要被注册。

ci.ReceiveData += new ClientSocketEvent (Ci\_Re- ceiveData);

ci.BroadcastMessage += new BroadcastEvent (ci. SendMessage);

ci.DisConnection += new ClientSocketEvent (Ci\_Dis- Connection)

然后调用BeginReceive方法去接收客户端请求消息，并尝试与客户端浏览器握手，如果握手成功，则全双工通信开始。

ci.ClientSocket.BeginReceive(ci.receivedDataBuffer, 0, ci.receivedDataBuffer.Length, 0, new AsyncCallback (ci.StartHandshake), ci.ClientSocket.Available);

在上面的代码中，StartHandshake方法根据客户端请求负责生成握手信息。在这方法中，两个关键字的值“Sec-Web- Socket-Key1” 和 “Sec-WebSocket-Key2”将从请求头中获取，并且在Web socket协议中对这两个值进行MD5运算定义，最后返回结果，MD5结果是为了保护整个握手过程中的数据。如果握手成功，新的连接将被放进连接池等待下次使用。

listConnection.Add(ci);

值得注意的是，当发送消息时，将字符“\x00” 放在消息的开始，“\xFF”放在消息的结尾，并且在读取消息的时候移除这两个字符。另外，消息在使用前应该进行UTF-8编码和解码。

public void SendMessage(MessageEntity me) { ClientSocket.Send(new byte[] {0x00});

ClientSocket. Send(Encoding. UTF8. GetBytes (Json- Convert.SerializeObject(me)));

ClientSocket.Send(new byte[] { 0xff }); }

最后，当通信结束时，调用DisConnection方法关闭连接。

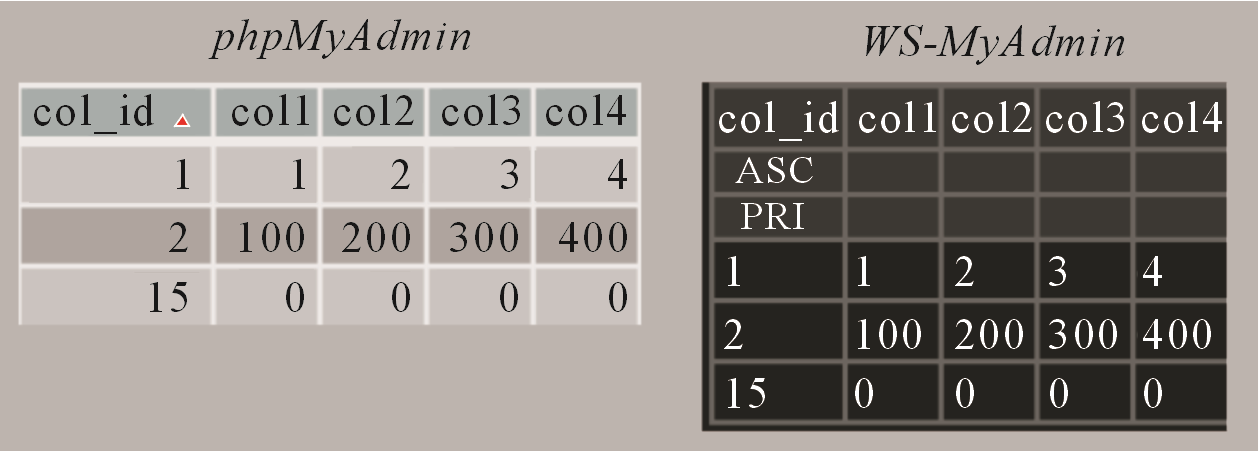
1. Web Socket性能分析

效率是实时数据传输的关键问题；也是一个协议是否适合进行实时传输的

重要评估标准。在异步传输中测试了Web Socket的性能。这个测试分为两部分，第一部分是在基于HTTP请求的phpMyAdmin 页面中对一个5列3行的数据进行排序，另一部分是在基于Web Socket通信的单独网页上对相同大小的表进行排序。整个测试过程使用Google Chrome5作为客户端浏览器，使用Wireshark Network Protocol Analyzer软件作为监测数据包和比特流变化的工具。最后得到以下结果（图1）：

通过以上数据（表1），Socket连接的效率比HTTP连接高10倍。另一方面，查阅Peter Lubbers和Frank Greco’s关于比较Ajax轮询和Web Socket的测试，它能推断出Web Socket的在网络流量和延迟的性能比HTTP要好很多，特别是在大并发量的情况下。

图1，比较HTTP连接和socket连接的流量和时间



Client to server Server to client Total

HTTP 33,662 45,600 79,262

Number of packets HTTP Web Socket

83 5

77 8 160 13

Number of bits Web Socket

372 7456 7828

Time (second) HTTP Web Socket

~2.5 ~0.25

Table 1. Data table used in test.

Copyright © 2012 SciRes.

1. 总结

实时数据传输是基于Web信息系统的必然趋势。Web Socket作为下一个Ajax将在互联网上被广泛使用。目前，最受欢迎的IE8及其较低版本的浏览器还不支持Web Socket。然而，Kaazing公司已经开发了一个在低版本浏览器能够将Ajax轮询转换为Web Socket的智能网关。Web Socket协议和Web Socket API仍然在不断更新。或许，Web Socket将成为“C10K”问题的完美解决方案。