基于 WebSocket 的网络实时通信*

薛陇彬 刘钊远
(西安邮电大学计算机学院 西安 710161)

摘 要 目前 Web 实时通信方案主要基于 HTTP 协议,存在效率低、消耗大的缺点。通过分别构建基于 HTTP 协议的 AJAX 轮询和基于 WebSocket 的实时通信框架,对比测试吞吐量和延时,证实 WebSocket 在 Web 实时通信领域具有较高效率。

关键词 HTML5; WebSocket; 实时通信; AJAX
中图分类号 TP393 **DOI**:10.3969/j.issn1672-9722.2014.03.030

Network Real-time Communication Based on WebSocket

XUE Longbin LIU Zhaoyuan
(School of Computer Science, Xi'an University of Posts and Telecommunications, Xi'an 710161)

Abstract The real-time Web communication solution based on HTTP protocol has the shortcomings of low efficiency and large consumption. Then a test is made by buliding framework between AJAX polling based on HTTP and WebSocket to make contrast on network throughput and latency. The result show, that WebSocket has higher efficiency in the field of Web real-time communication.

Key Words HTML5, Websockets, real-time communication, AJAX
Class Number TP393

1 引言

随着互联网高速发展,Web 数据的即时性要求愈发强烈。目前主流 Web 实时通信是基于 HTTP 协议(超文本传输协议),主要有两种实现方式[1]:服务器推送(ServePush)和客户端拉拽(Client Pull)。

由于底层 HTTP 协议是严格遵循请求-响应模型且请求和连接是单向的[2],要保证实时通信,在浏览器和服务器之间必须频繁建立连接并响应,这种方式效率低、资源消耗大,无法实现真正的实时通信[3]。而 WebSocket 是 HTML5 新标准中的通信机制,能够实现稳定全双工实时通信,具有简便高效的特点。

本文分别对客户端拉拽中基于 AJAX 的轮询和基于 WebSocket 的通信模式进行分析和效率对比。

2 基于 AJAX 轮询的实时通信

在 AJAX[4](异步 JavaScript 和 XML)技术出现之前,传统的客户端拉拽方式为 HTML 页面的整页刷新,通过设置 meta 标签中的定时刷新来实现。如:〈meta http-equiv="refresh" content="t"/〉,以 t 秒为间隔进行刷新。

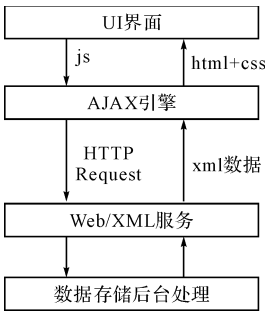

图 1 AJAX 模型通信原理结构图

整页刷新会造成页面闪烁、页面可能卡死等现象,效率极低且用户体验差。

AJAX 并不是一种新技术,而是多种 Web 技术的融合。如图 1 所示,AJAX 能够通过 XMLHttpRequest[5] 对象显式地定时与服务器

* 收稿日期:2013 年 9 月 3 日,修回日期:2013 年 10 月 17 日
作者简介:薛陇彬,男,硕士研究生,研究方向:嵌入式系统设计。刘钊远,男,教授,研究方向:嵌入式系统设计。

保持通信，自动拉取新数据[6]。并以主动异步方式将新数据更新在局部页面上，无需刷新页面。

AJAX 技术对传统拉取只能整页刷新的不足做出了改善，但以固定轮询频率向服务器发出请求，如果轮询频率设定过高，会加重网络负担和服务器负载，而过低则可能错失有效信息。并且由于 HTTP 协议的特性，每一次连接都要发送 HTTP 报头，造成很大的网络噪声。

3　基于 WebSocket 的实时通信

3.1　WebSocket 介绍

WebSocket 协议[7]并不是标准 Socket 协议，是由 IETF(互联网工程任务组)基于 HTTP 协议订制，处于 OSI 七层模型的上层，是 HTTP/1.1 协议的升级版。而 WebSocket API 由 W3C(万维网联盟)制订。WebSocket 和 WebSocket API 相结合，组成了 WebSocket 的理论和实践两个部分。

WebSocket 能够在客户端和服务器端之间搭建类 TCPSocket 的持续、双向、有状态的通信模式，实现真正意义上的长连接模式服务器推送，满足高实时性要求。

3.2　WebSocket 连接机制和通信模型

WebSocket 的连接机制是：客户端(浏览器)通过 80 或 443 端口的 HTTP 代理和中间件向服务器发送握手请求，服务器依据 HTTP 首部判别是否为 WebSocket 请求。识别成功后，该请求将被升级为一个 WebSocket 连接。

Websocket 通信产生过程中，首先执行的是一次客户端与服务器之间的握手[8]，客户端发送请求格式类似如下报头：

```
GET /chat HTTP/1.1
Host: 127.0.0.1
Connection: Upgrade
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Origin: http://localhost
```

服务器接到请求后，返回响应格式类似如下报头：

```
HTTP/1.1
101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://localhost
Sec-WebSocket-Location: ws://localhost/chat
Sec-WebSocket-Protocol: sample
```

至此，WebSocket 连接已经成功建立。

在通信模型方面，WebSocket 模型类似于流模式(长连接)[9]：一旦连接建立便会保持当前状态，数据可以在之后的任意时间被服务器推送至客户端。然而本质上二者具有很大区别。长连接模式中，服务器发给客户端的信息都具备完整的 HTTP 请求或回复报头信息，甚至报头信息可能会超过数据，因此传输效率仍然比较低。

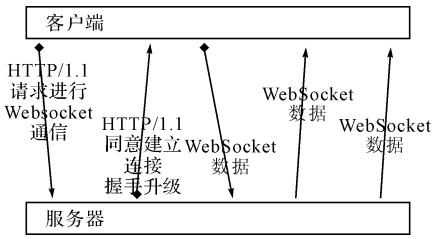

图 2　WebSocket 模型通信原理结构图

如图 2 所示，WebSocket 只在第一次握手时数据交互较复杂，握手成功后的传输阶段为纯数据，几乎没有额外请求和响应。且数据以 0x00 字节开头，以 0xFF 字节结尾的 UTF-8 格式编码，以二进制帧传送，而最小的帧仅有 2 字节。

4　测试验证

4.1　测试目的

为检验 WebSocket 的实时性和效率，分别搭建基于 AJAX 轮询和 WebSocket 的聊天室应用框架，通过运行时 Chrome 浏览器的开发测试工具的数据显示，对比在通信过程中的请求和响应信息。

4.2　测试过程

4.2.1　基于 AJAX 轮询的实时通信框架

搭建基于 Apache＋PHP＋MySQL 的后台服务器环境，处理来自客户端的 GET 或者 POST 请求。

客户端则通过 XMLHttpRequest 对象发送消息或响应服务器。以下是主要步骤伪代码：

```
//创建服务器对象
var http = require('http');
var server = http.createServer();
//设定服务器监听
server.listen(8080,'127.0.0.1');
//创建 XMLHttpRequest 对象
var xmlHttp = new XMLHttpRequest();
//根据回调函数 onreadystatechange()监听连接状态的改变
xmlHttp.onreadystatechange = function(){
    if(xmlHttp.readyState == 4){
    //当连接状态 readyState 为 4 时代表连接请求被响应，连接建立
    }
```

```
}
//使用 GET/POST 方法向服务器端发送连接请求
xmlHttp.send(data);
//服务器端根据 data 和 end 事件监听接收数据然后处理
req.addListener('data',function(){...});
req.addListener('end',function(){...});
```

服务器和客户端环境搭建完成之后,向服务器发送测试消息,通过 Chrome 开发工具中的 NetWork 监控标签显示,每次发送消息都会导致请求数目递增,并产生报头信息,而报头信息甚至大于发送的测试数据大小。

4.2.2 基于 WebSocket 的实时通信框架

搭建基于 Node.js[10] 的服务器环境,并安装 socket.io 包和 express 应用框架,Node.js 对 WebSocket 具备良好支持。

客户端则利用 WebSocket 对象监听各种事件是否被触发,如连接、接收、发送、关闭等,做出相应处理。以下是主要步骤伪代码:

```
//服务器端建立连接监听
io.on('connection', function (socket) {
  //通知客户端已连接
  socket.emit('open');
//服务器端设置 message 事件的监听
socket.on('message', function(msg){
}
//客户端连接服务器
socket = io.connect('http://localhost:3000');
//客户端收到服务器端连接确认
socket.on('open',function(){
});
//客户端设置 message 事件监听
socket.on('message',function(json){
//显示消息等操作
});
```

服务器和客户端环境搭建完成之后,向服务器发送测试消息,通过 Chrome 开发工具的 NetWork 监控标签显示,只有第一次发送消息后产生了请求,之后的消息发送并不会使请求数目递增,报头信息也只在第一次握手时产生。

通过表 1 数据可以对比 AJAX 轮询和 WebSocket 每次连接需要的平均资源开销。

表 1 AJAX 轮询和 WebSocket 平均资源开销对比

实现方式	需传输数据帧(个)	平均数据帧大小(Byte)
	建立连接	请求连接
AJAX 轮询	8	871
WebSocket	1	54

可以看出 AJAX 轮询在建立和保持连接时,请求数量更多且报头信息较大,甚至大于消息本身;而 WebSocket 只需要一次握手建立即可建立稳定连接,报头信息较小且只在握手期间产生,之后便可以二进制帧格式传输纯数据。

5 测试结果分析

5.1 网络吞吐量

在基于 HTTP 协议的实时通信中,最大的开销来自于每次请求和响应都要发送完整的报头信息。表 1 显示平均数据帧大小对比,AJAX 轮询下为 871B。

依图 3 所示,在三种用例下观察当连接数增大时,AJAX 轮询中请求与数据造成的吞吐量变化:

用例 A:1000 客户端每秒轮询一次,网络吞吐量为(871×1000)=871000 字节=6968000 比特/秒(6.6Mbps)。

用例 B:10000 客户端每秒轮询一次,网络吞吐量为(871×10000)=8710000 字节=69680000 比特/秒(66Mbps)。

用例 C:100000 客户端每秒轮询一次,网络吞吐量为(871×100000)=87100000 字节=696800000 比特/秒(665Mbps)。

可以看到当连接用户数逐渐增大至一定数量级,由于 AJAX 轮询所带来的网络流量十分惊人。

而使用 WebSocket,每个消息都是一个 WebSocket 帧,只有 2B 的开销。同样观察 WebSocket 下的吞吐量变化:

用例 A:1000 客户端每秒轮询一次,网络吞吐量为(2×1000)=2000 字节=16000 比特/秒(0.015Mbps)。

用例 B:10000 客户端每秒轮询一次。网络吞吐量为(2×10000)=20000 字节=160000 比特/秒(0.153Mbps)。

用例 C:100000 客户端每秒轮询一次。网络吞吐量为(2×10000)=200000 字节=1600000 比特/秒(1.526Mbps)。

可以看到随着客户端数量增加,WebSocket 产生的吞吐量远远低于 AJAX 轮询。

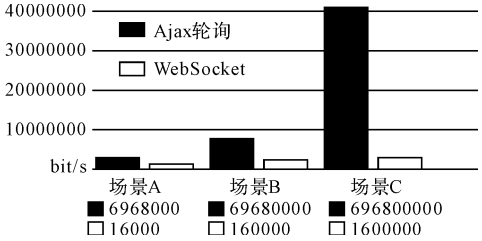

图 3 AJAX 轮询与 WebSocket 网络吞吐量对比

5.2 网络延迟

通过分析 AJAX 轮询和 WebSocket 的响应模式，图 4 给出了两种模式下的网络延迟对比。

AJAX 轮询模式下，服务器和浏览器之间平均延时为 50ms，表 1 中的数据显示，由于需要保持连接，频繁的请求和响应的执行造成多次延时，并且延时期间服务器无法向浏览器发送任何消息，造成资源浪费。

而在 WebSocket 模式下，连接一旦经一次握手升级为 WebSocket，消息会在到达服务器后立即返回浏览器。在一次发送的 50ms 延时后，WebSocket 连接将会始终保持打开状态，无需重复向服务器发送请求，因此降低了网络延迟。同时减少因延时造成的消息阻塞和资源浪费，提高了实时性。

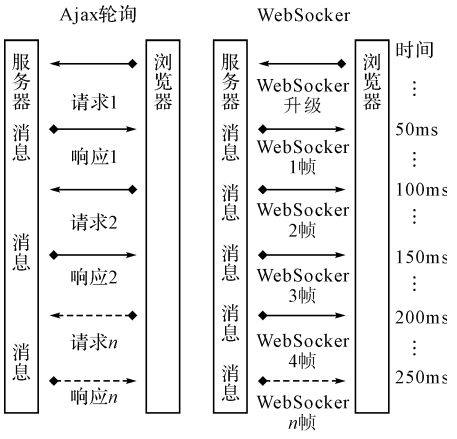

图 4　AJAX 轮询和 WebSocket 网络延迟对比

综上，根据模型对比以及结果分析，WebSocket 可以实现网络吞吐量大量缩减，并由于其稳定的长连接模式，通信延时也进一步缩减。

6 结语

目前基于 HTTP 的实时方案被广泛应用，但由于 HTTP 并不是为实时、全双工通信设计的，因此实时性不够理想。通过对比测试，结果显示 WebSocket 能够减少实时通信中的吞吐量和延时。相信随着 WebSocket 协议的进一步完善，基于 WebSocket 的实时方案将被大量应用。

参考文献

[1] ClientPull/ServerPush. [EB/OL]. (2005)[2013. 5. 13] http://www. kiv. zcu. cz/~ledvina/vyuka/books/HTMLnya/ch38. htm.

[2] 吴晓东，王鹏. Html5 的通信机制及效率的研究[J]. 长春理工大学学报，2011，34(4)：159-160.
WU Xiaodong, WANG Peng. Research on Communication Mechanism and Efficiency of Html5[J]. Journal of Changchun University of Science and Technology, 2011, 34(4): 159-160.

[3] Pimentel V, Nickerson B. G. Communicating and Displaying Real-Time Data with Web-Socket[J]. IEEE Computer Society, 2012, 4: 45.

[4] 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)：3826-2827.
WEN Zhaosong, YI Renwei, YAO Hanbing. WebSocket Based Real Time Web Application Solution[J]. Computer Knowledge and Technology, 2012, 8(16): 3826-2827.

[5] 游丽贞，郭宇春，李纯喜. Ajax 引擎的原理和应用[J]. 微计算机信息，2006，22(2-3)：206.
YOU Lizhen, GUO Yuchun, LI Chunxi. The principle and application of Ajax engine[J]. Control & Automation, 2006, 22(2-3): 206.

[6] 柯昌正，黄厚宽. Ajax 技术的原理与应用[J]. 铁路计算机应用，2007，16(1)：1-2.
KE Changzheng, HUANG Houkuan. Principle and application of Ajax technology[J]. Railway Computer Application, 2007, 16(1): 1-2.

[7] W3C. TheWebSocketAPI[EB/OL]. (2009. 5. 2)[213. 10] http://www. w3. org/TR/2009/WD-sebsockets-20090423/

[8] Peter Lubbers, Brian Albers, Frank Salim. Pro HTML5 Programming[M]. 北京：人民邮电出版社，2012：234-239.
Peter Lubbers, Brian Albers, Frank Salim. Pro HTML5 Programming[M]. Beijing: Posts and Telecom Press, 2012: 234-239.

[9] 郭欣. 基于 HTML5 的通用 WebIM 组件的前端设计与实现[D]. 西安：西北工业大学，2011：21.
GUO Xin. Front-end Design and Implementation of General WebIM Component based on HTML5[D]. Xi'an: Xi'an Technological University, 2011: 21.

[10] 肖在昌，杨文晖，刘兵. 基于 WebSocket 的实时技术[J]. 网络与通信，2012(12)：41.
XIAO Zaichang, YANG Wenhui, LIU Bing. Real-Time Technology of Web Based on WebSocket[J]. Network and communication, 2012(12): 41.