Web端即时通讯技术盘点：短轮询、Comet、Websocket、SSE

Jack Jiang
即时通讯技术交流群：185926912，技术社区：http://52im.net

15 人赞同了该文章

1. 前言

Web端即时通讯技术因受限于浏览器的设计限制，一直以来实现起来并不容易，主流的Web端即时通讯方案大致有4种：传统Ajax短轮询、Comet技术、WebSocket技术、SSE（Server-sent Events）。本文将简要介绍这4种技术的原理，并指出各自的异同点、优缺点等。

2. 学习交流

- 更多即时通讯技术资料：52im.net/forum.php?...

- 即时通讯开发交流群：215891622 [推荐]

3. 概述

1996年IETF HTTP工作组发布了HTTP协议的1.0版本 ，到现在普遍使用的版本1.1，HTTP协议经历了17 年的发展。这种分布式、无状态、基于TCP的请求/响应式、在互联网盛行的今天得到广泛应用的协议，相对于互联网的迅猛发展，它似乎进步地很慢。互联网从兴起到现在，经历了门户网站盛行的web1.0时代，而后随着ajax技术的出现，发展为web应用盛行的web2.0时代，如今又朝着web3.0的方向迈进。反观http协议，从版本1.0发展到1.1，除了默认长连接之外就是缓存处理、带宽优化和安全性等方面的不痛不痒的改进。它一直保留着无状态、请求/响应模式，似乎从来没意识到这应该有所改变。

好在HTML5的时代已经到来，为Web端即时通讯的实现带来了WebSocket和SSE（Server-sent Events）两种技术方案。

4. Ajax短轮询：脚本发送的http请求

传统的web应用要想与服务器交互，必须提交一个表单（form），服务器接收并处理传来的表单，然后返回全新的页面，因为前后两个页面的数据大部分都是相同的，这个过程传输了很多冗余的数据、浪费了带宽。于是Ajax技术便应运而生。

Ajax是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（JS）发送http请求。Outlook Web Access小组于98年使用，并很快成为IE4.0的一部分，但是这个技术一直很小众，直到2005年初，google在他的goole groups、gmail等交互式应用中广泛使用此种技术，才使得Ajax迅速被大家所接受。

Ajax的出现使客户端与服务器端传输数据少了很多，也快了很多，也满足了以丰富用户体验为特点的web2.0时代 初期发展的需要，但是慢慢地也暴露了他的弊端。比如无法满足即时通信等富交互式应用的实时更新数据的要求。这种浏览器端的小技术毕竟还是基于http协议，http协议要求的请求/响应的模式也是无法改变的，除非http协议本身有所改变。

5. Comet：一种hack技术

以即时通信为代表的web应用程序对数据的Low Latency要求，传统的基于轮询的方式已经无法满足，而且也会带来不好的用户体验。于是一种基于http长连接的“服务器推”技术便被hack出来。这种技术被命名为Comet，这个术语由Dojo Toolkit 的项目主管Alex Russell在博文Comet: Low Latency Data for the Browser首次提出，并沿用下来。

其实，服务器推很早就存在了，在经典的client/server模型中有广泛使用，只是浏览器太懒了，并没有对这种技术提供很好的支持。但是Ajax的出现使这种技术在浏览器上实现成为可能， google的gmail和gtalk的整合首先使用了这种技术。随着一些关键问题的解决（比如 IE 的加载显示问题），很快这种技术得到了认可，目前已经有很多成熟的开源Comet框架。

以下是典型的Ajax和Comet数据传输方式的对比，区别简单明了。典型的Ajax通信方式也是http协议的经典使用方式，要想取得数据，必须首先发送请求。在Low Latency要求比较高的web应用中，只能增加服务器请求的频率。Comet则不同，客户端与服务器端保持一个长连接，只有客户端需要的数据更新时，服务器才主动将数据推送给客户端。

Comet的实现主要有两种方式，基于Ajax的长轮询（long-polling）方式和基于 Iframe 及 htmlfile 的流（http streaming）方式。

有关Comet技术的详细介绍文章请参见：《Comet技术详解：基于HTTP长连接的Web端实时通信技术》、《WEB端即时通讯：HTTP长连接、长轮询（long polling）详解》、《WEB端即时通讯：不用WebSocket也一样能搞定消息的即时性》、《开源Comet服务器iComet：支持百万并发的Web端即时通讯方案》。

5.1 基于Ajax的长轮询（long-polling）方式

浏览器发出XMLHttpRequest 请求，服务器端接收到请求后，会阻塞请求直到有数据或者超时才返回，浏览器JS在处理请求返回信息（超时或有效数据）后再次发出请求，重新建立连接。在此期间服务器端可能已经有新的数据到达，服务器会选择把数据保存，直到重新建立连接，浏览器会把所有数据一次性取回。

5.2 基于 Iframe 及 htmlfile 的流（http streaming）方式

Iframe是html标记，这个标记的src属性会保持对指定服务器的长连接请求，服务器端则可以不停地返回数据，相对于第一种方式，这种方式跟传统的服务器推则更接近。

在第一种方式中，浏览器在收到数据后会直接调用JS回调函数，但是这种方式该如何响应数据呢？可以通过在返回数据中嵌入JS脚本的方式，如“<script type="text/javascript">js_func(“data from server ”)</script>”，服务器端将返回的数据作为回调函数的参数，浏览器在收到数据后就会执行这段JS脚本。

但是这种方式有一个明显的不足之处：IE、Morzilla Firefox 下端的进度栏都会显示加载没有完成，而且 IE 上方的图标会不停的转动，表示加载正在进行。Google 的天才们使用一个称为“htmlfile”的 ActiveX 解决了在 IE 中的加载显示问题，并将这种方法应用到了 gmail+gtalk 产品中。

6. Websocket：未来的解决方案1

如果说Ajax的出现是互联网发展的必然，那么Comet技术的出现则更多透露出一种无奈，仅仅作为一种hack技术，因为没有更好的解决方案。Comet解决的问题应该由谁来解决才是合理的呢？浏览器，html标准，还是http标准？主角应该是谁呢？本质上讲，这涉及到数据传输方式，http协议应首当其冲，是时候改变一下这个懒惰的协议的请求/响应模式了。

W3C给出了答案，在新一代html标准html5中提供了一种浏览器和服务器间进行全双工通讯的网络技术Websocket。从Websocket草案得知，Websocket是一个全新的、独立的协议，基于TCP协议，与http协议兼容、却不会融入http协议，仅仅作为html5的一部分。于是乎脚本又被赋予了另一种能力：发起websocket请求。这种方式我们应该很熟悉，因为Ajax就是这么做的，所不同的是，Ajax发起的是http请求而已。

与http协议不同的请求/响应模式不同，Websocket在建立连接之前有一个Handshake（Opening Handshake）过程，在关闭连接前也有一个Handshake（Closing Handshake）过程，建立连接之后，双方即可双向通信。

有关WebSocket的详细介，请参见即时通讯网有关WebSocket的系列文章：《WebSocket详解（一）：初步认识WebSocket技术》、《WebSocket详解（二）：技术原理、代码演示和应用案例》、《WebSocket详解（三）：深入WebSocket通信协议细节》。

从浏览器支持角度来看，WebSocket已经近在眼前，但仍有一段较长的路要走，特别是在中国这个IE6、7、8依然盛行的国家，旧版本浏览器的消亡需要很长一段时间，在完全实现浏览器全兼容前，Comet技术可能仍然是最好的解决方案。不过，当前也已存在一些比较成熟的封装方案来解决这种兼容性限制，比如：开源的Socket.io，详见《Socket.IO介绍：支持WebSocket、用于WEB端的即时通讯的框架》。

7. SSE：未来的解决方案2

SSE（Server-Sent Event，服务端推送事件）是一种允许服务端向客户端推送新数据的HTML5技术。与由客户端每隔几秒从服务端轮询拉取新数据相比，这是一种更优的解决方案。

与WebSocket相比，它也能从服务端向客户端推送数据。那如何决定你是用SSE还是WebSocket呢？概括来说，WebSocket能做的，SSE也能做，反之亦然，但在完成某些任务方面，它们各有千秋。

WebSocket是一种更为复杂的服务端实现技术，但它是真正的双向传输技术，既能从服务端向客户端推送数据，也能从客户端向服务端推送数据。

WebSocket和SSE的浏览器支持率差不多，大多数主流桌面浏览器两者都支持。在Android 4.3以及更早的版本中，系统默认浏览器两者都不支持，Firefox和Chrome则完全支持；Android 4.4中，系统默认浏览器两者都支持；Safari从5.0开始支持SSE（iOS系统从4.0开始），但直到6.0才正确地支持WebSocket（6.0之前的Safari所实现的WebSocket协议存在安全问题，所以一些主流浏览器已经禁用了基于这个协议的实现）。

与WebSocket相比，SSE有一些显著的优势。个人认为它最大的优势就是便利：不需要添加任何新组件，用任何你习惯的后端语言和框架就能继续使用。你不用为新建虚拟机、弄一个新的IP或新的端口号而劳神，就像在现有网站中新增一个页面那样简单。我喜欢把这称为既存基础设施优势。

SSE的第二个优势是服务端的简洁。相对而言，WebSocket则很复杂，不借助辅助类库基本搞不定（我试过，令人痛苦）。

因为SSE能在现有的HTTP/HTTPS协议上运作，所以它能直接运行于现有的代理服务器和认证技术。而对WebSocket而言，代理服务器需要做一些开发（或其他工作）才能支持，在写这本书时，很多服务器还没有（虽然这种状况会改善）。SSE还有一个优势：它是一种文本协议，脚本调试非常容易。事实上，在本书中，我们会在开发和测试时用curl，甚至直接在命令行中运行后端脚本。

不过，这就引出了WebSocket相较SSE的一个潜在优势：WebSocket是二进制协议，而SSE是文本协议（通常使用UTF-8编码）。当然，我们可以通过SSE连接传输二进制数据：在SSE中，只有两个具有特殊意义的字符，它们是CR和LF，而对它们进行转码并不难。但用SSE传输二进制数据时数据会变大，如果需要从服务端到客户端传输大量的二进制数据，最好还是用WebSocket。

WebSocket相较SSE最大的优势在于它是双向交流的，这意味向服务端发送数据就像从服务端接收数据一样简单。用SSE时，一般通过一个独立的Ajax请求从客户端向服务端传送数据。相对于WebSocket，这样使用Ajax会增加开销，但也就多一点点而已。如此一来，问题就变成了“什么时候需要关心这个差异？”如果需要以1次/秒或者更快的频率向服务端传输数据，那应该用WebSocket。0.2次/秒到1次/秒的频率是一个灰色地带，用WebSocket和用SSE差别不大；但如果你期望重负载，那就有必要确定基准点。频率低于0.2次/秒左右时，两者差别不大。

从服务端向客户端传输数据的性能如何？如果是文本数据而非二进制数据（如前文所提到的），SSE和WebSocket没什么区别。它们都用TCP/IP套接字，都是轻量级协议。延迟、带宽、服务器负载等都没有区别，除非……呃？除非什么？

当你在享用SSE的既存基础设施优势，并在客户端和服务端脚本之间设了一个网络服务器，区别就显现出来了。一个SSE连接不仅使用一个套接字，还会占用一个Apache线程或进程，如果用PHP，它会为这个连接专门创建一个PHP新实例。Apache和PHP会使用大量的内存，这会限制服务器所能支持的并行连接数。所以，要做到用SSE在数据传输性能上和WebSocket完全一样，需要写一个自己的后端服务器，当然，那些在任何情况下都会用自己的服务器并使用Node.js的人，会觉得这有什么稀奇的。

说一下WebSocket在旧版本浏览器上的兼容。当前，大约超过2/3的浏览器支持这些新技术，移动端浏览器的支持率会低一些。依惯例，每当需要双向套接字时，就会用到Flash，并且WebSocket的向后兼容通常是用Flash来做，这已经相当复杂了，如果浏览器上没有Flash，情况更糟。概括来说，WebSocket难兼容，SSE易兼容。有关SSE的专项介绍文章请参见：《SSE技术详解：一种全新的HTML5服务器推送事件技术》。

（本文同步发布于：52im.net/thread-336-1-1...）

系列资料

Web端即时通讯新手入门贴：

《新手入门贴：详解Web端即时通讯技术的原理》

关于Ajax短轮询：

找这方面的资料没什么意义，除非忽悠客户，否则请考虑其它3种方案即可。

有关Comet技术的详细介绍请参见：

《Comet技术详解：基于HTTP长连接的Web端实时通信技术》

《WEB端即时通讯：HTTP长连接、长轮询（long polling）详解》

《WEB端即时通讯：不用WebSocket也一样能搞定消息的即时性》

《开源Comet服务器iComet：支持百万并发的Web端即时通讯方案》

有关WebSocket的详细介绍请参见：

《WebSocket详解（一）：初步认识WebSocket技术》

《WebSocket详解（二）：技术原理、代码演示和应用案例》

《WebSocket详解（三）：深入WebSocket通信协议细节》

《Socket.IO介绍：支持WebSocket、用于WEB端的即时通讯的框架》

《socket.io和websocket 之间是什么关系？有什么区别？》

有关SSE的详细介绍文章请参见：

《SSE技术详解：一种全新的HTML5服务器推送事件技术》

更多WEB端即时通讯文章请见：

52im.net/forum.php?...

> 作者：Jack Jiang (点击作者姓名进入Github)
>
> 出处：52im.net/space-uid-1.ht...
>
> 交流：欢迎加入即时通讯开发交流群 215891622
>
> 讨论：52im.net/
>
> Jack Jiang同时是【原创Java Swing外观工程BeautyEye】和【轻量级移动端即时通讯框架MobileIMSDK】的作者，可前往下载交流。
>
> 本博文 欢迎转载，转载请注明出处（也可前往 我的52im.net 找到我）。

编辑于 2016-07-14