新闻

Ajax、Comet、HTML 5 Web Sockets技术比较分析

作者 Dionysios G. Synodinos 译者 沙晓兰 发布于 2008年12月24日 下午10时0分

社区 Ruby, .NET, Java 主题 RIA, Web 2.0 标签 AJAX

九十年代中期，WWW以迅猛之势转眼跻身传播信息的主要渠道之一。浏览器的身影开始无处不在，用户也随之开始适应这种信息传播方式。显然，WWW提 供的应用平台能够赢得历史上任何一个平台都无法比及的用户量。但当时很难实现这样的目标是因为一些标准（HTML、HTTP等）都不很完善，这些标准设计 的时候都没有考虑到高度交互和富客户体验。最初的一些富在线应用基本上都是由Microsoft Exchange开发组实现的。96年以来，他们曾采用IFrame为邮件服务器系统提供Outlook类型的前端应用。这些早期尝试在响应能力和整体的 用户体验方面都非常落后，但从这些应用身上却可以清楚地看到即将兴起的网络应用。1998年，团队开始为MS Exchange Server 2000编写web前端，他们开发了XMLHTTP，这个控件实现了单个web页面与服务器间的异步交互。可以看到，XMLHTTP实际上根本没有立即和 XML捆绑起来。XMLHTTP这个名字是Alex Hopmann提出的，他是后来加入开发团队的，据说名字采用这个前缀的唯一的原因是IE5当时正在准备第二个beta版本，而这个控件必须作为这个版本的MSXML库的一部分发布，这才冠上了XML。

Mozilla基金会在2002年开发他们的浏览器的一个版本时，也以XMLHttpRequest的形式实现这一新技术，这个浏览器就是后来的 Firefox。尽管当时有一些商家也曾尝试运用这些新API，但他们采用的的这种远程脚本程序的模式一直没有引起公众的注意，直到Google开始部署 基于JavaScript和XHR的一系列新型服务。当时的第一个服务是2005年2月8日Google Blog上发布的Google Maps。之后不久，XHR就一跃成为业界最炙手可热的话题。直到那时，也还没人预料到XHR给Web应用开发带来的革命性的推动，但它的成功开始让我们 转变之前对WWW的一些看法。

在Kaazing Gateway发布之际，InfoQ采访了Richard Smith，谈到关于AJAX, Comet以及蒸蒸日上的HTML 5 Web Sockets等技术的发展情况：

> Ajax为HTTP通信模型提供了很好的解决方案，它在客户端异步轮询服务器端事件。服务器事件依次排列在待处理队列中，根据轮询时间隙依次传送到 浏览器，这样模拟服务器发起的通信，在轮询时间隙间进行实时消息传递。因此，仅仅依靠Ajax,我们永远都不可能实现真正的实时通信。
>
> Comet引入的优化针对的是HTTP通信初始之时，它在HTTP基础上采用“push”通信风格。Comet提供的几项技术能够在没有客户端发送 请求的前提下让服务器主动将信息发送到浏览器。如果再增加一个额外的HTTP连接的话，Comet甚至可以在两个HTTP连接上实现双向通信。但 Comet的绊脚石在于各个浏览器提供商对XHR、iFrames——这两种实现Comet所需的数据块的支持程度不尽相同，没有统一的实现标准。另外， 无论是从网络还是开发角度来看，Comet管理两个连接的开销都很大。这些开销带来的直接影响就是Comet应用中的传输延时，限制了它们所提供的实时通 信的精确性。
>
> HTML 5 WebSocket代表的是Comet和Ajax推进HTTP通信新一轮的尝试。HTML 5 WebSocket规范中定义，在浏览器和服务器之间采用单socket全双工（或者叫双向）传输来push和pull信息。这不但可以避免Comet中 存在的连接和可移植问题，还能够提供比Ajax轮询更高效的解决方案。目前，HTML 5 WebSocket是推动web全双工实时通信的主要机制。

Richard认为，AJAX和Comet这两种方式都有各自的局限：

> 要通过Ajax来模拟服务器端起始的通信就需要轮询机制，而这一机制不顾应用的状态改变盲目检查更新，结果就是CPU周期和内存毫无必要地过早或者 太晚侦测服务器的更新，客户和服务器两端的资源利用状况因此都相当差劲。所以，传统的Ajax应用必须根据服务器上事件的发布率不停地调整轮询时间隙的长 短，才能改善各个请求的准确度。另外，高频率轮询会加重网络承载，拖累服务器；低频率轮询又会错过更新和传送一些失去时效的信息。无论哪种情况，消息传递过程都无法避免传输延时。短间隙轮询开销很大，因为要支持这样的小型服务器ping需要大量服务器资源。
>
> Comet维护服务器和浏览器之间的持续连接和长时间有效的HTTP请求，以此来尝试“push”型通信。这种连接下，服务器可以发送事件，但连接是由客户端浏览器发起。逆流请 求也可以看做是浏览器向服务器发送的请求，这需要一个额外的HTTP连接。Comet因此可以利用跨两个HTTP连接的双向通信。但是，维护这两个连接会 消耗服务器端大量资源，因为这也意味着服务一个顾客需要消耗双倍资源。而且，浏览器的配置往往都是通过域来限制HTTP连接。Comet的应用因此更为复 杂，还常常会要求运用一些像多路复用那样复杂的技术，再

或者就是要管理多个域。

有一些Comet解决方案试图降低长轮询技术导致的资源消耗，但这一技术发送太多的HTTP请求/回复头信息。比如，服务器发送的 每个事件，都通过客户浏览器为连接提供服务，这迫使浏览器必须和服务器重新建立连接。这一动作又引发了另一个客户端请求以及服务器在长轮询时间隙中发送回复。很多时候，回复中的HTTP头内容完全超过了传送的信息。

```
From client (browser) to server:
GET /long-polling HTTP/1.1\r\n
Host: www.kaazing.com\r\n
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\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
\r\n


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

用Comet开发有很多挑战。实现你自己的解决方案不是不可能，但这需要开发JavaScript类库，借用frame和XHR Streaming等很多技术来维护持续连接。这时候，问题就在于不同的浏览器对这些技术有不同的实现，更糟的是，它们往往都依赖服务器端代码推送 JavaScript代码段，不仅增加了整个实现的复杂程度，还引入了移植性的问题。

还好，现在有几个框架提供了这些传输的抽象，简化了Comet开发。其中最著名的就是SitePen的Cometd，其实现完全参 考Bayuex规范。Bayuex规范定义了Comet的publish-subscribe模型。Jetty近期版本也包含了基于Java的服务器端的 Bayeux实现。

Bayuex和Cometd着实简化了Comet，然而它的API和wire协议还是有很多争议。Comet Daily上有一个“Coliding Comet: Battle of the Bayuex”系列就专门深入讨论关于Bayuex的各种问题。

Richard还认为 HTML 5 WebSockets可以从当前的各种解决方案中获得很多帮助：

尽管Comet和Ajax都可以实现提供桌面应用功能的终端用户体验，而且传输延时也可以缩短到用户无法感知的程度，但仍然只有Web Sockets才能真正为浏览器提供精确、高效的流事件，保证传输延时可以微乎其微，直至忽略不计。这是目前为止通过web发送实时信息最出色的解决方 案。它不仅通过单个TCP/IP连接提供完整的异步双工道流通信，而且新的HTTP头的应用也非常有利，更重要的是它能够支持浏览器和源服务的消息采用同样的格式。

多数Comet实现都依赖Bayeux协议。该协议要求源服务发出的消息必须转换成Bayeux协议支持的格式，这一并不必要的转 换反而使得整个系统更加复杂，开发员不得不在服务器端处理一种消息格式（比如JMS、IMAP、XMPP等），在客户端又要处理另一种消息格式（比如 Bayeux、JSON）。而且实现将源协议转换到Bayeux的代码硬是要在发送消息之前对消息本身进行解析和处理，这又给系统增添了不必要的性能负 载。采用Web Sockets的话，就不会有因为转换代码而增加系统的复杂性，也就不用为这方面的性能担忧。

WebSockets经常遇到的一个问题是它是否可行。目前来看，浏览器本身没法直接支持这项技术。但再过几个月就肯定可以了，像 WebKit、Firefox和Opera这样的浏览器从来就对HTML 5的特性——比如Canvas、postMessage、离线存储和服务器端发送信息(SSE)等反应迅速，及时添加相应的支持。

WebSockets还需要服务端一定程度的支持，因为现存HTTP连接更新到新连接需要HTTP的一个起始“握手”。Kaazing Gateway开源项目实现了第一个支持这一动作的服务器，并且拥有能够支持成千上万持续连接所需的

> 扩展性。 Kaazing Gateway的供应商Kaazing还提供了一个可以让当前所有web浏览器都支持WebSockets的JavaScript类库。所以，目前对 WebSocket的支持也可说是准备就绪了。

为了支持HTML 5 WebSockets，Kaazing发布了Kaazing Gateway 8.09_2 Atlantis，这是一个开源HTML 5 WebSocket服务器，可以在Mozilla公共许可的衍生许可—— OSI approved Common Public Attribution License (CPAL)下使用：

> Kaazing Gateway提供JavaScript类库来模拟HTML 5 WebSocket，开发员现在就可以开始运用WebSockets，结合WebSocket接口创建的应用在当前甚或是未来的浏览器上都可以部署。
>
> Kaazing Gateway背后的超高性能服务器的单个节点能够支持成千上万的并发连接。多实例通过传统的HTTP负载平衡或者DNS round robin算法集群分类，因此能够支持无数个持续客户连接。除了大量的连接之外，Kaazing Gateway的高性能和分级事件驱动构架（SEDA）还推动它本身能够处理高数据吞吐量。
>
> Kaazing Gateway的Atlantis发布还为流行的消息服务（诸如Apache ActiveMQ、RabbittMQ）和XMPP服务（诸如OpenFire、Jabberd和其它一些流行的聊天服务器）打包了JavaScript 客户端。这样，创建那些基于web的聊天应用或是stock matrixes、网上交易平台、在线游戏等消息发送应用就更为简单了。

计划中的Kaazing Gateway 8.12发布把目标瞄准了更多的HTML 5特性，例如服务器端发送事件（Server-sent Event）、更先进的安全服务，以及对XMPP（Jabber）、STOMP (比如ActiveMQ、RabbitMQ或OpenMQ)等的扩展支持：

> 它所提供的类库能够让目前的浏览器都支持HTML 5服务器发送事件，引进了HTML 5 postMessage的支持，无疑方便了跨文档的消息传递。Kaazing的HTML 5类库还包括对HTML 5离线存储的支持，提供简易、基于DOM的存储解决方案。Kaazing Gateway及其客户类库现在还为跨站请求支持W3C访问控制，这一机制能够让客户端启动跨站请求，比较多的提法是跨站 XmlHttpRequests。
>
> 除了对HTML 5的扩展支持以外，Kaazing Gateway 8.12还提供更高级的XMPP特性，比如群聊。这一发布版本还引进了STOMP-JMS适配器，因此，结合Kaazing Gateway还能适配任何现存的Java消息服务（JMS）（例如JBoss Messaging、Tibco EMS、OpenMQ、SwiftMQ、WebSphere MQ等等）。

您可以在InfoQ上找到更多关于 **AJAX**、**Comet**和**富网络应用**的信息。

查看英文原文:HTML 5 Web Sockets vs. Comet and Ajax

没有回复 **回复**