Web 应用中的服务器推送技术研究综述

Survey on Server – Push Technology of Web Applications

孙清国[1,2]　朱　玮[1]　刘华军[3]　张　鹏[3]　(1. 中国科学院长春光学精密机械与物理研究所 吉林 长春 130033；2. 中国科学院研究生院 北京 100039；3. 91550 部队装备部 辽宁 大连 116023)

摘　要：很多 Web 应用如即时通讯,股票行情系统,都需要将服务器发生的变化实时传送到客户端而无须客户端不停地刷新、发送请求。本文首先对现存的服务器推送技术进行了分析和总结,着重介绍了 Comet – 基于 HTTP 长轮询、事件驱动的服务器推送技术,最后对实现 Comet 应用需要面对的相关问题进行了讨论。

关键词：服务器推送 Comet AJAX 长轮询

1　引言

随着 Web 技术的流行,越来越多的应用基于 B/S 架构。传统模式的 Web 系统以客户端发出请求、服务器端响应的方式工作。随着人们对 web 应用的要求越来越多,这种方式并不能满足很多现实应用的需求,譬如:监控系统、即时通信系统、即时报价系统等等。在这些需求当中,服务器软件需要向客户端主动发送消息或信息。当服务器某些事情发生的时候,服务器需要主动地向客户端实时地发送消息,将最新的数据或事件通知给用户。在传统的 C/S 系统中,这种需求没有任何问题,因为客户端和服务器之间通常存在着持久的连接,这个连接可以双向传递各种数据。而基于 HTTP 协议的 Web 应用却不行。在 Web 世界中,服务器永远是被动地发送数据,前提是客户端必须先发送请求。浏览器其实并不知道服务器的信息什么时候会有改变。于是,针对这种问题的解决方案变的越来越重要。

2　背景

2.1 Web 应用模型

传统的 Web 应用模型如图 1 所示,在这种结构下,用户工作界面是通过 WWW 浏览器来实现,极少部分事务逻辑在前端(Browser)实现,主要业务逻辑在服务器端(Server)实现,形成所谓三层(3 – tier)结构。客户端向服务端发送 HTTP 请求,Web server 接收这个请求,根据不同情况,访问相应的数据库、Web Service 或者其他业务系统,经过业务逻辑计算,向客户端游览器发送 HTML 响应。这样就大大简化了客户端电脑载荷,减轻了系统维护与升级的成本和工作量,降低了用户的总体成本。

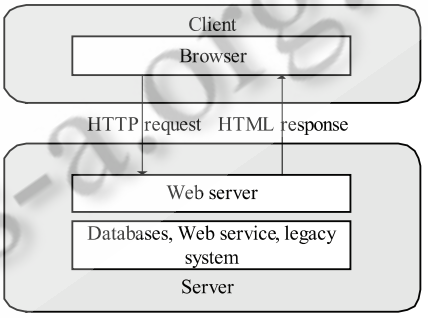

图 1　Web 应用模型

但是根据这个模型,服务端仅在客户端发出请求后才把数据发送回去。对于每个客户端请求,浏览器进行连接到 Web Server 的 Http 连接的初始化工作,当数据返回后这个连接便被关闭了。这样在在客户端用户看到的数据是“静态的”,“过时”的数据。在服务端,发生的数据变化、事件响应,客户端完全不知情。这样,问题便产生了。

2.2 现有解决方案

2.2.1 HTML refresh

最简单的解决办法可能就是 HTML refresh。这种

办法是根据 web 应用模型的限制，为了得到最新的数据或响应，客户端不得不连续向服务端发送请求，即刷新。HTML 的 <meta> 标签中可以设置 HTML 文件自动刷新的频率如：

<META HTTP – EQUIV = " Refresh" CONTENT = " 4 ; URL = http : // www. google. com" >

如果服务段发生了变化，客户端就会得到最新的响应。但是，随之而来的问题是，间隔多久刷新一次哪？过度频繁的刷新对服务器端的压力是毫无疑问的，尤其是并发比较大的应用。而刷新间隔过长，又可能造成客户端响应的不及时，问题又没有被完全解决。在现实应用中，可能会根据客户端对数据响应的及时性需求和服务端的性能作一个折中考虑。

2.2.2 基于插件技术服务器推送技术

采用浏览器的插件技术（ActiveX、Applet、Flash），浏览器安装插件，基于套接口传送信息，或是使用 RMI、CORBA 进行远程调用，来实现服务器推送数据。但是浏览器插件技术本身又有许多问题，例如跨平台问题和插件版本兼容性问题。另外，客户端必须安装相应的插件，这无疑失去了 B/S 结构的一些优点，系统的升级和维护都加大了成本。基于 ActiveX、Applet、Flash 的服务器推送技术有不同的实现，本文不作过多讨论。

2.2.3 无插件服务器推送技术

服务器在事件发生时发送消息给客户端，而不需要客户端进行询问。这样客户端将不再周期性地检查服务器，它能够继续其它的工作并处理事件发生后被 server 推送过来的数据。现在的服务器推送技术是保持原有的 HTTP 协议不变，在服务器端改变处理方式，使得服务器能够使用浏览器已经打开的 HTTP 连接，主动向浏览器发送消息。这里关键的技术是要保持原有的 HTTP 连接不断。一旦拥有持久的连接，服务器就可以根据自己的数据更新，随时地向客户端发送最新的信息。

Alex Russell[1] 称这种基于 HTTP 长连接、无须在浏览器端安装插件、事件驱动的服务器推送技术为“Comet”。

3 基于 AJAX 的服务器推送技术

因为浏览器技术上的种种限制，没有为服务器推送技术的实现提供很好的支持，在无插件的纯浏览器的应用中很难有一个完善的方案去实现服务器推送技术并用于商业程序。

作为一种广泛使用的 Web 应用程序开发技术，AJAX（Asynchronous JavaScript and XML）技术变得日益流行，随之而来的是一些通用 AJAX 使用模式。例如，AJAX 经常用于对用户输入做出响应，然后使用从服务器获得的新数据修改页面的部分内容。但是，有时 Web 应用程序的用户界面需要进行更新以响应服务器端发生的异步事件，而不需要用户操作。例如，显示到达 AJAX 应用程序的最新股票行情，或者在即时订票系统中显示服务器上票务数据的改变。由于只能由浏览器建立 Web 浏览器和服务器之间的 HTTP 连接，服务器无法在改动发生时将变化“推送”给浏览器。

这种方式的缺点是只有在用户明确地刷新页面或是转移到新页面时被显示的页面才被更新。由于传输这个页面需要一段较长的时间，刷新页面便要花费一段较长的延迟。为了解决这个问题，AJAX 被用来通过使 Web 浏览器只对页面中需要改变的部分发出请求的方式减少数据的更新量。因为传输数据总量被减少了，延迟也相应减少了，整个 Web 站点的反应能力增强了。这样基于 AJAX 的异步数据调用，部分解决了客户端数据延迟的问题。

但是随之而来的一个问题是：client 必须在服务器发生数据变化之后发出一些数据请求。那么如何获知服务器端发生变化，AJAX 何时向服务器发送请求哪？

AJAX 应用程序可以使用三种基本的方法解决这一问题：一种方法是浏览器每隔一段时间向服务器发出轮询（poll）以进行更新，类似 HTML refresh，另一种方法是服务器始终打开与浏览器的连接并在数据可用时发送给浏览器，即应用基于长轮询（long – polling）方式 Comet 模型。最后一种是和长轮询方式比较接近的基于 HTTP streaming 方式。

3.1 基于 AJAX poll 实现

如图 2 所示，这种拉取方式的频率要足够高才能保证很高的数据精确度，但高频率可能会导致多余的检查，从而导致较高的网络流量。而另一方面，低频率则会导致错过更新的数据。理想地，拉取的时间间隔应该等于服务器状态改变的速度。

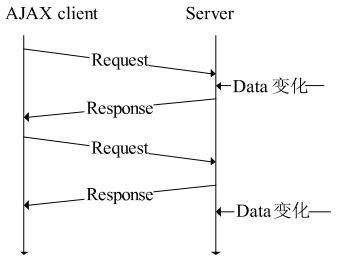

图 2　基于 AJAX 轮询实现

轮询方法的主要缺点是:当扩展到更多客户机时,将生成大量的通信量。每个客户机必须定期访问服务器以检查更新,这为服务器资源添加了更多负荷。最坏的一种情况是对不频繁发生更新的应用程序使用轮询,在这种情况下,相当数量的客户机轮询是没有必要的,服务器对这些轮询的响应只会是“数据未发生变化”。虽然可以通过增加轮询的时间间隔来减轻服务器负担,但是这种方法会产生不良后果,即延迟客户机对服务器事件的获得,这又相当于回到了问题的开始。当然,简单的应用程序可以做出某种权衡,从而获得可接受的轮询方法。

3.2 基于长轮询(long – polling)的服务器推送技术,Comet

AJAX 的出现使得 JavaScript 可以调用 XMLHttpRequest 对象异步发出 HTTP 请求,JavaScript 响应处理函数根据服务器返回的信息对 HTML 页面的显示进行更新。使用 AJAX 实现基于长连接的服务器推送与传统的 AJAX 应用不同之处在于:

(1) 服务器端会阻塞请求直到有数据传递或超时才返回,即维持长连接。

(2) 客户端响应处理程序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。

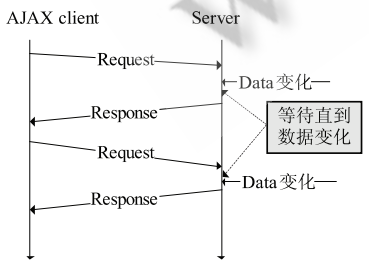

图 3　基于 AJAX 长轮询实现

如图 3 所示,相对于轮询(poll),这种长轮询方式也可以称为“拉”(pull)。因为这种方案基于 AJAX,具有以下一些优点:请求异步发出;无须安装插件。

尽管如此,吸引人们使用 Comet 策略的其中一个优点是其显而易见的高效性。客户机不会像使用轮询方法那样生成烦人的通信量,并且事件发生后可立即发布给客户机。但是保持长期连接处于打开状态也会消耗服务器资源。当等待状态的 servlet 持有一个持久性请求时,该 servlet 会独占一个线程。这将限制 Comet 对传统 servlet 引擎的可伸缩性,因为客户机的数量会很快超过服务器栈能有效处理的线程数量。这是实现基于长连接的 Comet 的一个挑战。

3.3 Http streaming 实现

如图 4 所示,基于 HTTP streaming 的 AJAX 实现与基于长连接的 Comet 实现非常相似。不同的地方是 HTTP streaming 重复使用同一个连接,而且永不关闭这个连接,服务器和浏览器之间的 TCP 连接会一直保持连接状态,直到其中一方发送了一条明显的“关闭连接”的消息,或者有超时以及网络错误发生。这样服务器端和客户端之间可以减少网络负担。Google 的 Gmail 在邮件更新接口中使用了这种技术。

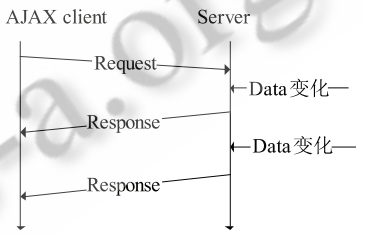

图 4　基于 Http streaming 实现

基于 HTTP streaming 的 AJAX 实现同样会面对 Comet 所要面对的性能问题。如果服务器频繁的发生数据变化,服务器会不断地向客户端发送响应,AJAX 程序有可能不能处理过多的响应,造成客户端数据的丢失。另外,服务端对于大量的用户的请求,所要承载的压力也是相当巨大的。

4　Web 服务器性能改进

在实际的应用中实现基于长连接的 Comet 或者 HTTP streaming,考虑系统的稳定性和性能是非常重要的,所以必须要解决一些细节问题。

首先,HTTP 1.1 规范中规定,客户端与服务器端建立超过两个的 HTTP 连接,新的连接会被阻塞。这意味着,如果 AJAX 应用程序打开了两个长连接,任何新的 HTTP 请求被前两个长连接阻塞,而无法到达服务器。因此,当采用基于长连接的服务器推送技术时,应该只有一个连接使用那样的技术而确保还有另外一个连接用来接收其它的请求

其次,一般 Web 服务器会为每个连接创建一个线程,如果在大型的应用中使用基于长连接的 Comet 模型,服务器端需要维护大量并发的长连接。AJAX 的应用使请求的出现变得频繁,而 Comet 则会长时间占用一个连接,上述的服务器模型在新的应用背景下会变得非常低效,线程池里有限的线程数甚至可能会阻塞新的连接。在这种应用背景下,应该在服务器端为长连接作一些改进。

HTTP 1.1 与 1.0 规范有一个很大的不同:1.0 规范下服务器在处理完每个 Get/Post 请求后会关闭套接口连接;而 1.1 规范下服务器会保持这个连接,在处理两个请求的间隔时间里,这个连接处于空闲状态。Java 1.4 引入了支持异步 IO 的 java. nio 包。当连接处于空闲时,为这个连接分配的线程资源会返还到线程池,可以供新的连接使用;当原来处于空闲的连接的客户发出新的请求,会从线程池里分配一个线程资源处理这个请求[2]。这种技术在连接处于空闲的机率较高、并发连接数目很多的场景下对于降低服务器的资源负载非常有效。Jetty, GlassFish 和 Tomcat 根据以上原理分别以自己的方式提高了 web 服务器的性能。

4.1 Jetty Continuations[2]机制

Jetty[3] 6 的目的是扩展大量同步连接,使用 Java 语言的非阻塞 I/Ojava. nio)库并使用一个经过优化的输出缓冲架构。Jetty 还为处理长期连接提供了一些技巧:该特性称为 Continuations。

一个处理 AJAX request 的 java Filter 或者 Servlet 可以向一个 Continuation 对象发出请求,这个 Continuation 对象用来挂起这个 AJAX request 和释放当前的线程。当超时或者一个 resume 方法被调用,request 会立刻恢复。

Continuations 使用 SelectChannelConnector 处理请求。这个连接器构建在 java. nio API 之上,因此使它能够不用消耗每个连接的线程就可以持有开放的连接。当使用 SelectChannelConnector 时,ContinuationSupport. getContinuation( )将提供一个 SelectChannelConnector. RetryContinuation 实例当对 RetryContinuation 调用 suspend( )时,它将抛出一个特殊的运行时异常—RetryRequest— 该异常将传播到 servlet 以外并通过过滤器链传回,并由 SelectChannelConnector 捕获。但是发生该异常之后并没有将响应发送给客户机,请求被放到处于等待状态的 Continuation 队列中,而 HTTP 连接仍然保持打开状态。此时,为该请求提供服务的线程将返回 ThreadPool,用以为其他请求提供服务[4]。这样服务器的性能将得到很大的提高。

4.2 Grizzly[5]

在 JDK 1.4 推出 NIO 之后,Grizzly 利用 NIO 的新特性,来编写高性能的 HTTP 引擎。Grizzly 最早被用于 Sun Java System Application Server, Platform Edition 8.1。随后成为开源软件 GlassFish[6]的一部分。

GlassFish 在其 HTTP 引擎 Grizzly 的内部就实现了异步请求处理(Asynchronous Request Processing, ARP),使得异步请求处理的速度与同步请求处理一样迅速快捷。在 ARP 的基础之上,Grizzly 还实现了一个服务器推送技术的引擎。这个引擎满足了日益流行的 AJAX 应用对服务器连接资源的消耗,使得 GlassFish 成为部署 AJAX 应用最理想的平台之一。

在 Grizzly 中,对异步请求处理完全是依靠 NIO 的特点,将处理线程与 HTTP 连接进行了分离,来确保系统的高性能和高可扩展性。但是这个实现在很多的方面还有待提高和完善。

5 总结

根据上述分析,针对当前基于 Web 应用中的服务器推送技术的实现模型,有各自的优势和不足。从目前的技术方案来说,基于长连接 Comet 模型实现 Web 应用中的服务器推送技术也许更有吸引力。但实现这样一个 Comet 模型并不简单,随之而来的是,Bayeux[7]协议出现了。Bayeux 一种基于 JSON 的、基于事件分发订阅模型协议[8]。Bayeux 协议的目的是使事件分发快捷、简单。可以由任何 Comet 客户端和服务器端实现。目前客户端的 Dojo[9]、服务器端的 Jetty Cometd[10]已经实现了对这个协议的支持。

但在实际的应用中,选择哪种方案还是需要一个不小的挑战。但选择的标准应该从以下几方面考虑:

(1) 网络通讯数据量

(2) 响应速度

(3) 服务器性能

(4) 游览器兼容性

(5) 开发成本

(6) 方案扩展性

虽然基于不同方案的服务器推送技术正在不断成熟,尤其很多的开源项目在这个领域非常活跃,但要解决的问题还有很多。本文从 Web 应用模型入手,阐述了当前 Web 应用对服务器推送技术的需求,并介绍、比较了常用的服务器推送方案,着重介绍了 Comet – 基于长轮询、事件驱动的服务器推送技术。未来在服务器推送技术上应该会从性能问题、基于标准和易用性等方面作进一步的研究。

参考文献

1 Comet: Low Latency Data for the Browser. http://alex.dojotoolkit.org/? p=545.

2 Jetty Continuations. http://docs.codehaus.org/display/JETTY/Continuations.

3 Jetty. http://jetty.mortbay.org/.

4 Write scalable Comet applications with Jetty and Direct Web Remoting. http://www.ibm.com/developerworks/java/library/j – jettydwr/? S_TACT = 105AGX52&S_CMP = cn – a – j.

5 Grizzly. https://grizzly.dev.java.net/.

6 GlassFish. https://glassfish.dev.java.net/.

7 Bayeux Protocol – – Bayeux 1.0draft1. http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html.

8 Cometd, Bayeux, and Why They're Different. http://alex.dojotoolkit.org:80/? p=573.

9 dojo . http://dojotoolkit.org/.

10 cometd: The Scalable Comet Framework. http://cometd.com/.