评论专栏: Erik Burckart：Comet 的诱惑

诱惑与准备

级别： 中级

Erik Burckart (ejburcka@us.ibm.com), WebSphere Application Server 首席架构师, IBM

2008 年 3 月 17 日

Comet 样式的应用程序在 Web 2.0 世界正变得越来越受欢迎。不过，Comet 带来了很多挑战，而且将作为应用程序部署基础的基础设施可能尚不支持 Comet 应用程序。

来自 IBM WebSphere Developer Technical Journal.

它是翱翔天空的小鸟，它是划过长空的飞机……

Comet 是一个用于描述客户端和服务器之间的交互的术语，即使用长期保持的 HTTP 连接来在连接保持畅通的情况下支持客户端和服务器间的事件驱动的通信。自从 Alex Russell 在 *Comet: Low Latency Data for the Browser*（请参见参考资料）中定义了此术语后，Comet 就成为了一个频繁出现的 Web 2.0 词汇。Comet 样式的应用程序将连接超时值设置为最大，并充分利用基础设施来提供比其他解决方案更快的浏览器更新速度，而且数据传输量更少——这听起来非常不错。但 Comet 样式连接也有缺点，在考虑使用之前您应该对这些缺点加以了解。

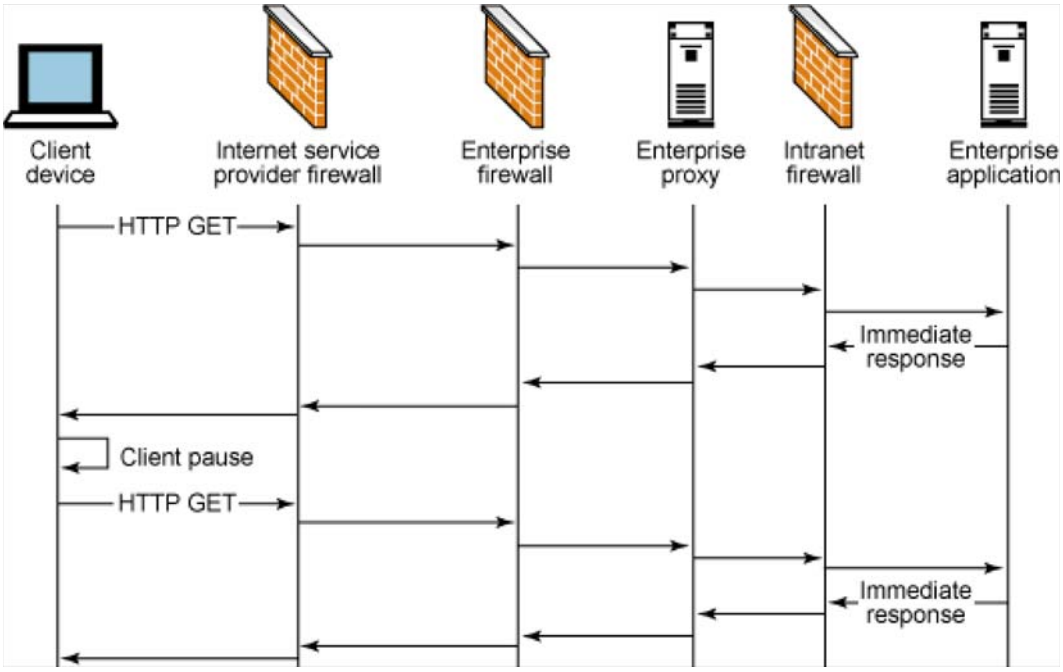
所解决的问题（以及带来的问题）

Web 2.0 领域的很多开发人员都需要处理的最为常见的问题是在服务器上为客户端生成的流化事件。解决此问题有三种常见的方式：

1. **轮询**

   在此方法中，在浏览器中运行的 Javascript™ 按配置的间隔发送请求来检查是否有其应该接收的事件。服务器的响应会立即发回：已经发生的事件，或者告知没有相应的事件。如果客户端发送请求的间隔太短，则可能会带来很大的性能影响。如果间隔太长，事件通知可能晚于客户端的期望时间到达。

图 1. 轮询

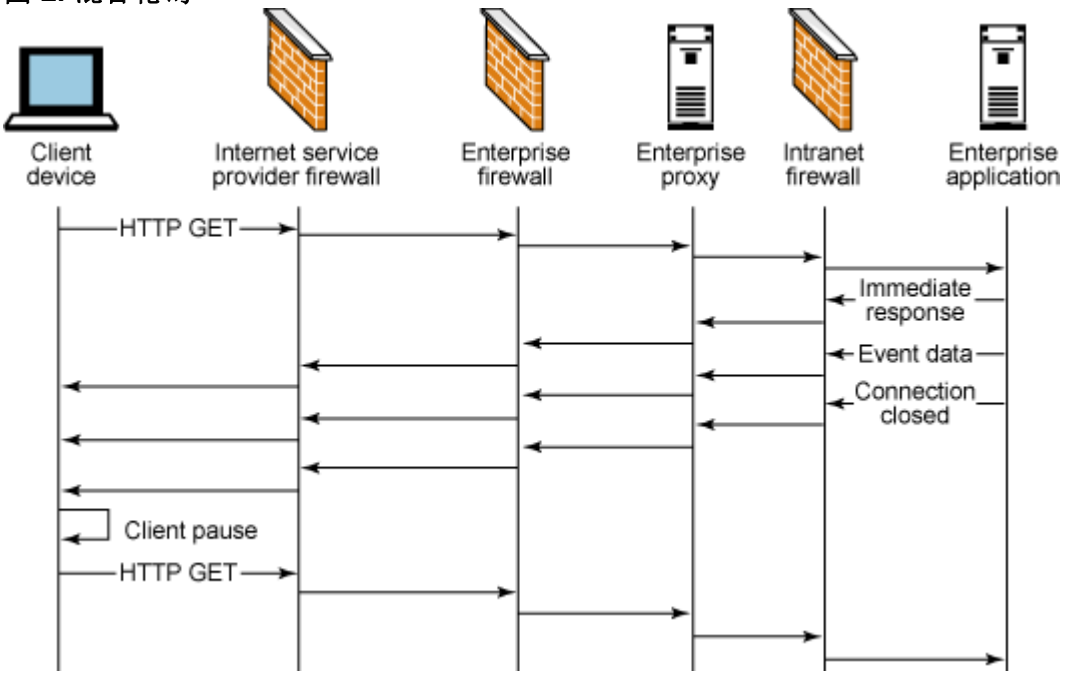


2. **Comet 长轮询或混合轮询**

   使用混合轮询的应用程序同时具备了 Comet 样式应用程序的一些优点和轮询样式的应用程序的一

些优点。客户端的浏览器中的 Javascript 发起对后端服务器的初始数据请求。服务器几乎会立即响应，但会让套接字保持开放状态，以便完成响应写入（如果在其开放期间出现了响应）。例如，服务器可能会将套接字保持开放 30 秒钟，如果在此时间段内出现了事件，会立即将其反馈给客户端。但如果在此时间段内没有事件，或者客户端需要向服务器发送更多的数据，连接将关闭，客户端将在一段时间后重新打开另一个连接。有些实现打开读取通道作为长期 HTTP GET，而另外打开写入通道作为 HTTP POST，在需要时打开和关闭。Javascript 中的 XMLHttpRequest 经常采用这种方式实现。

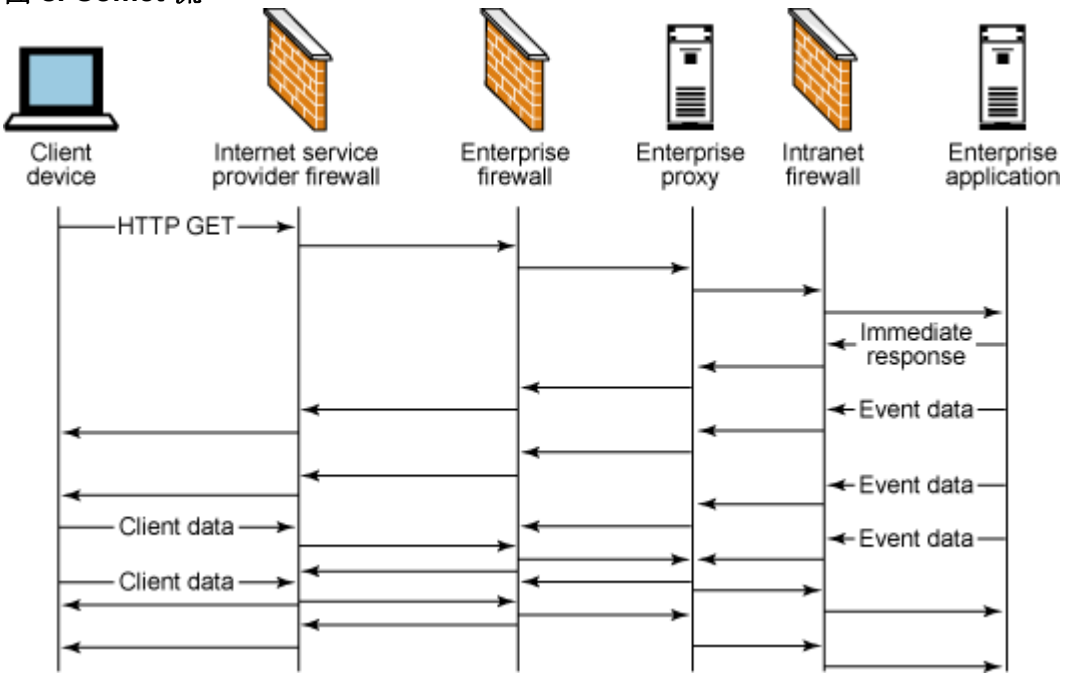
图 2. 混合轮询



3. **Comet 流或 Forever Frame**

在 Comet 流样式应用程序中，客户端打开连接并发送经过编码的数据块，而服务器也将发送响应中经过编码的数据块。创建初始连接后，两个方向的数据传输开销都非常小。只要可能，连接就会保持开放状态，每个字节集仅包含成块的编码部分开销，这些都是表示所发送数据的大小的十六进制数和一个换行回车，通常少于 10 个字节。

图 3. Comet 流



Comet 流和混合轮询的优势

Comet 流和混合轮询应用程序在初始连接建立后都允许双向通信。Comet 流是目前唯一可用于在不使用浏览器插件情况下提供即时双向通信的技术模式。混合轮询允许在连接保持的情况下出现事件时立即通知。混合轮询和其他在浏览器中使用 Javascript 的模式需要使用服务器端逻辑来在客户端连接获取事件前对其进行缓存。Comet 流的一些好处简要总结如下：

- 在无需插件的情况下提供浏览器支持。
- 直接通知客户端；不需要缓存事件。
- 减少服务器上的 CPU 开销。
- 从发送事件和信息的字节数量而言，开销小得多。

Comet 流的缺点

可惜的是，大部分 Internet 基础设施尚未针对目前大量使用 Comet 流样式应用程序做好准备。我之所以这样说，是因为有以下原因：

- **标准制定工作滞后**

  HTTP 协议设计为在开始发送请求时，请求发送所使用的连接在响应完全发回之前都几乎完全锁定。这就意味着在 Comet 流样式通信持续期间，从客户端设备到后端服务器的每个系统上都至少有一个连接处于不可用状态。

- **同步与异步**

  Internet 基础设施的很多部分本质上是同步性质的。这包括服务器（如不同的防火墙）和编程模型（如 HTTP Servlet）。因此，目前的 Internet 基础设施通常并不能为 CS 样式应用程序的广泛应用提供所需的基础，因为其中的大部分系统的线程数量都有限，而每个 Comet 连接都会锁定一个线程。

- **有限的连接数量**

  代理服务器和防火墙将对大量 Comet 流应用程序的引入造成阻碍。为什么呢？目前的操作系统通常并没有做好普遍使用 Comet 流的准备。由于上面提到的关于 HTTP 的问题，Comet 流应用程序的每个用户都将锁定 HTTP 请求通过的每个防火墙和代理中的两个连接和文件描述符。很多操作系统（以及构建于这些操作系统上的软件）都限制为任何时间整个计算机的连接数少于 65,000 个。这就意味着这些系统可能会在 CPU 或内存不够之前就耗尽可用连接数，而需要更多系统——但是现有系统上仍然有大量的 CPU 和内存可用。

- **按照设计工作**

  有些系统根本就没有设计为处理这种类型的应用程序。这些系统会采用缓存 HTTP 块之类的方式处理，直到达到特定大小才将其发送给客户端。有些防火墙在接收到完整的请求并能够完整地对其进行检查后才会将 HTTP 请求发送到服务器。有时候可以通过设置更改此行为，但其他时候则不行。

- **浏览器并没有做好处理流的准备**

  现有的 API 没有针对发送简单数据块和接收简单数据块进行优化。这有时候是通过实现新客户端 Applet 或类似的东西来实现的，但目前很难对其进行支持。大部分人都返回到混合轮询机制，在独立连接中发送数据。

为了更好地支持 Comet 流，需要进行什么工作？

我们认为，我们可以进行很多工作来更好地支持 Comet 样式应用程序。首先，我们需要处理上面的问题。

- **标准**

  HTTP 管道在很多人眼里不可行，因为它不允许在单个连接上按顺序发送数据块。通过序列号和请求标识，浏览器、防火墙、代理或应用服务器可以在单个连接上传送多个 Comet 流请求。

- **同步与异步**

  需要对我们的应用程序和基础设施进行更新，从而以异步方式处理所有东西。这就包括 HTTP Servlet 规范，这方面可以从 Jetty 中的 Continuations 支持或其他 SIP Servlet 的异步本质得到启发。我们大部分人都预期 Servlet 3.0 规范将采用异步方式进行处理。

- **有限的连接数量**

  如果有更为高级的 HTTP 管道支持，则可能就不需要处理这个问题了。但是，如果没有这方面的改进，系统则需要比目前更多的连接数量可用。

- **更好的 JavaScript API**

需要更好的 API 来处理 Comet 流样式服务器端应用程序的成块消息的发送和接收。

处理了这些问题后，Comet 流将得到更广泛的接受，因为此模式可以帮助提高基于 Web 的界面的互操作性。股票报价、聊天、体育赛事比分以及很多应用程序都将得到广泛的应用，而且不会对基础设施造成太大的压力。到那个时候，我估计可能需要对 Internet 上的每个服务器软件进行检查，确定其是否能够应对这个新的基于 Comet 流的世界。

我们是否应该使用混合轮询？

混合轮询支持快速（但并不一定是即时的）通知。此方法之所以受到今天基础设施的青睐，是因为有以下这些原因：

- Javascript API（具体来说就是 XMLHttpRequest）能更好地支持此方法。
- 混合轮询在服务器基础设施上更容易实现，因为请求之间有断层。

两个间隔（客户端轮询间隔和服务器在没有响应的情况下保持连接的时长）中较短的间隔将决定混合轮询将在基础设施上造成的资源使用影响。如果可能，请将服务器保持时间设置为最小，而将客户端轮询时间设置为最大。

现在我们能做些什么？

- 将您的 Comet 流使用限制为仅用于需要即时事件通知的应用程序。
- 使用支持更快事件通知但不会对基础设施造成大负荷的方法，如混合轮询。调整混合轮询时间，以让连接尽可能多地关闭。
- 了解您所购买的新基础设施的限制。例如，如果您购买防火墙，如果知道以下信息将很有帮助：其可以处理的最大连接数、是否支持 Comet 样式应用程序以及所需的功能（如入侵检测）是否需要对消息进行缓存。

参考资料

学习

- 您可以参阅本文在 developerWorks 全球站点上的 英文原文
- Comet: Low Latency Data for Browsers
- Ajax for Java developers: Write scalable Comet applications with Jetty and Direct Web Remoting

获得产品和技术

- IBM WebSphere Application Server Feature Pack for Web 2.0

关于作者

Erik Burckart 是 WebSphere Application Server 产品的首席架构师。他毕业于匹兹堡大学信息科学学院，曾在该学院学习电信、软件开发和人机交互。凭借在 WebSphere Application Server 的 SIP Servlet 方面的工作，他加入了 SIP Servlet 1.1 (JSR 289) Expert Group，并在组合先进的 Java EE 平台与最新的 SIP Servlet 规范方面做出了大量贡献。

IBM 公司保留在 developerWorks 网站上发表的内容的著作权。未经IBM公司或原始作者的书面明确许可，请勿转载。如果您希望转载，请通过 提交转载请求表单 联系我们的编辑团队。