Comet 简介及实际使用

青云居

http://blog.csdn.net/yyri/archive/2008/02/22/2113976.aspx

Comet 概述

定义

Comet 是基于 Http 长连接的新一代互联网技术，对其较早的定义来自 http://alex.dojotoolkit.org/?p=545。

要讲 Comet，先从 Server Push 开始。

Server Push 是一种很早就存在的技术，以前在实现上主要是通过客户端的套接字，或是服务器端的远程调用。因为浏览器技术的发展比较缓慢，没有为 Server Push 的实现提供很好的支持，在纯浏览器的应用中很难有一个完善的方案去实现 Server Push 并用于商业程序。最近几年，因为 AJAX 技术的普及，以及把 IFrame 嵌在“htmlfile”的 ActiveX 组件中可以解决 IE 的加载显示问题，一些受欢迎的应用如 Meebo，Gmail+Gtalk 在实现中使用了这些新技术；同时“服务器推”在现实应用中确实存在很多需求。因为这些原因，基于纯浏览器的“服务器推”技术开始受到较多关注，Alex Russell（Dojo Toolkit 的项目 Lead）称这种基于 HTTP 长连接、无须在浏览器端安装插件的 Server Push 技术为“Comet”。目前已经出现了一些成熟的 Comet 应用以及各种开源框架；一些 Web 服务器如 Jetty 也在为支持大量并发的长连接进行了很多改进。关于 Comet 技术最新的发展状况请参考关于 Comet 的 wiki。

Wikipedia 上对 Comet 的定义如下：

> Comet is a World Wide Web application architecture in which a web server sends data to a client program (normally a web browser) asynchronously without any need for the client to explicitly request it. It allows creation of event-driven web applications, enabling real-time interaction otherwise impossible in a browser. Though the term Comet was coined in 2006, the idea is several years older, and has been called various names, including server push, HTTP push, HTTP streaming, Pushlets, Reverse Ajax, and others.
>
> Comet applications use long-lived HTTP connections between the client and server, which the server can respond to lazily, pushing new data to the client as it becomes available. This differs from the original model of the web, in which a

browser receives a complete web page in response to each request, and also from the Ajax model, in which browsers request chunks of data, used to update the current page. The effect is similar to applications using traditional Ajax with polling to detect new information on the server, but throughput is improved and latency and server load are decreased.

Comet does not refer to any specific technique for achieving this user-interaction model, but encompasses all of them—though it implies the use of browser-native technologies such as JavaScript as opposed to proprietary plugins. Several such techniques exist, with various trade-offs in terms of browser support and side effects, latency, and throughput.

传统的web应用都是基于请求-响应的模式，ajax的改进只是非全页面更新，无法解决实时性和事件驱动。ajax with polling通过定时请求可以实现伪实时，但频繁的建立和销毁连接又会耗费服务器大量资源，增加带宽使用。Comet使用Http1.1 长连接，实现实时的服务器-客户端数据推送。Comet的实现可以有两种方式，Streaming和Long-Polling。Streaming方式建立连接后，两端均不断开，使用此连接实时传输消息。Long-Polling方式一旦完成数据接收，即断开当前连接并重新建立新连接。二者相比Streaming性能最优，但即使是Long-Polling，不管是服务端负载还是对网络带宽的使用，也大大优于传统的Polling。

典型应用场景

Comet 特别适用于需要和服务器端实时交互的应用，如聊天，远程协作等类型的 Web 应用。

浏览器端兼容性

在 Comet 之前，可通过下述途径来实现类似的效果，其中部分方式现在仍有较多使用。各方式对浏览器端的兼容性如下：

1. IFrame

最早的实现方式，通过隐藏的 IFrame 元素实现对服务器的持续请求，在返回内容中输出并执行 JavaScript 以实现。优点是基本所有浏览器都支持，缺点是会导致部分浏览器的状态条一直为读取状态且鼠标状态为忙碌，影响用户体验。

2. Htmlfile ActiveX object

通过将 HTML IFrame 元素置于一个 ActiveX 中，规避了 1 中所提到的两个缺点。其缺点很显然，仅 IE 支持，如 Google Gtalk。另外 Zeitoun 的 comet-iframe 将提供

Javascript 对象支持内嵌 IFrame，支持 IE, FF。

3. Multipart XHR(XMLHttpRequest)

1995 年，Netscape 中增加了一个特性叫 Server Push，这个特性允许服务器端在同一个 Http Response 中发送图片或 HTML 页的新版本到浏览器端，通过在 HTML 头声明 ContentType 为 multipart/x-mixed-replace 实现，浏览器会使用新版本的 HTML 替换已有页面。不过这个特性仅在使用 Gecko 内核的浏览器中支持。

4. XHR streaming

通过自定义返回数据格式，在浏览器端捕获 onreadystatechange 事件并在 readyState=3 时回调对应 JavaScript 方法来处理数据并实现。此方式 IE 不可行。

5. XHR long polling

上述若干方法兼容性都不是很好，而 XHR long polling 在所有支持 XHR 的浏览器中都可以使用。其实现方式为：浏览器建立一个异步连接，当服务器响应后回调 JS 方法，然后重新建立新的连接并等待服务器端下一次响应。

6. Dynamic script tag long polling

该方式更好解决跨域调用问题，虽然跨域页面的互操作也可以通过代理来解决。

7. Server-sent events

HTML5 草案中的新元素 event source。

可靠性

代理服务器和防火墙可能对 Comet 应用有不利影响。一般防火墙都会断开已建立时间过长的连接。大多数 Comet 框架的解决办法是重建连接。

另外，代理服务器可能缓存被推送的数据，使应用丧失实时性。

可扩展性

因为 Comet 应用实时发送服务器端事件，一般会比其他的传统 web 应用消耗更多资源，使得其扩展性相对较差。

首先，Comet 应用需要在服务器和浏览器间维持至少一个长连接，而传统的 web 服务器按照页面-页面请求进行的设计和优化使其在无法在同一时间维持如此多的连接。这导致 Comet 应用无法在一个 web server 上处理大量请求，垂直扩展性差。

垂直扩展性差的原因在于传统的 web server 如 Apache 等的处理线程使用同步 IO。每个请求被分配一个线程去处理，并尽可能尽快处理完毕并关闭连接以处理下一个请求。但 Comet 应用建立的连接是持久的，处理请求的每个线程不能用于处理其他请求，如果有大

量长连接建立，web server 就可能无法处理新发起的请求。

Jetty 和 Tomcat 等 Web 应用服务器已为 Comet 应用进行优化(ContinuationServlet, 但是因为不是标准的 servlet 实现，不具备服务器兼容性)，使用异步 IO(JDK5 之后)来处理长连接，一定数目的线程去轮询各个连接并在到达每个连接时将所需数据推送到浏览器端。此类服务器不仅可以处理比线程服务器多的多的长连接，而且随着连接数增加，处理延迟被均摊到每个连接，对用户影响小。值得注意的是，sun 新提交的 JSR315 servlet3.0 也包括了 comet，今后将成为 J2EE 服务器的标准实现。

另外，一般 Comet 应用都是交互性的，允许多用户间通讯，因此将处理任务分派到多个 server 去处理有困难。即水平扩展性差。

对水平扩展性的一种解决方式是按照事件对服务器进行分组，将订阅某组事件的请求转发到指定的服务器上。

注意事项

1. HTTP1.1 标准中有如下规定：“A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy”，并且此规则被包括 IE 和 FF 在内的绝大多数浏览器遵守。因此，当 comet 占用一个 http 长连接，将可能导致浏览器不能为 ajax 请求创建新的连接，比如浏览器正在读取大量图片时。

2. 如果有新的公共数据需要 push，所有 servlet 要同时将数据 push 出去，在 push 之前还会有查询，这样对 cpu 内存和网络都会有较大消耗，而且是同时的，会造成共振效应。可以考虑使用一个公共的查询线程来负责，request 被放入一个队列，查询线程负责轮询所有 request，来将各自需要的数据 push 到 client，这样可以响应的并发连接数可能会多很多，且避免了共振。

其他解决方案

鉴于浏览器对 Comet 的直接支持较差，有部分开发者使用了 flash 和 java applet 等浏览器插件来实现实时响应。只要安装了相应插件，该方式不用考虑浏览器的差异，通用性较好，且不用占用 HTTP 连接。

A 项目中 Comet 的使用

在 A 项目中根据 Bayeux 协议实现进行封装和修改，完成 A 功能需求。

在后台代码中，通过增加消息体内容的类型定义，覆写 Bayeux 协议中定义的事件回调函数，实现 A 的数据发送：

- 覆写 subscribe 回调函数，实现对订阅者的缓存，并对新用户进行历史留言发送
- 覆写 publish 回调函数，实现 IP 屏蔽、屏蔽词过滤、已发布消息缓存、更新访谈最高在线人数和消息发布的操作；
- 覆写协议实现的默认初始化函数，实现频道创建、订阅和发布的安全验证；
- 增加消息体中 IP 地址、发布时间、发言类型、用户数量等的定义，并在消息接收 Servlet 和后台发布中对原始消息体进行封装和补充。

为了保证效率，后台程序尽量使用缓存和线程轮询，通过 JDK 语法和 Concurrent 包的工具类对竞争资源进行同步。

前台系统，使用 Dojo 对 JSON 消息体进行封装发送和解析，优化历史消息显示策略，实时进行新消息发送和接收。

- 创建 JavaScript 类，实现对页面操作的封装。
- 对不同的访谈定义不同的频道编号，并分别生成各自静态页，便于今后的服务拆分和管理。
- 为不影响前台浏览器效率，对接收到的消息进行缓存，批量进行页面操作，避免频繁操作(700 次/S)导致浏览器 CPU 占用率过高。

Dojo和Bayeux简介

Dojo 是一套重量级的，强大的面向对象的 JavaScript 库，包括 dojo 和 dojox 两大部分。为开发富客户端应用提供了一套现成的工具库。

DWR 也已加入 Dojo，Struts2 也整合了 dojo，且很多 IDE 也提供过了对 DOJO 的支持和良好的调试环境，包括 Eclipse WTP+ATF，Netbeans5 以上。

Bayeux 是 Dojo 基金会定义的一个协议，应 Comet 应用的需要而产生，其目标是消除不同 Web Server 的 Comet 实现间的差异，以及不同浏览器脚本使用 Comet 的差异并提高已有代码的重用率。Bayeux 是第一个比较全面的实现 Comet 的协议。特别是对 long-polling，callback-polling，iframe 这几种 comet 的实现手段都能支持。

Bayeux 使用 JSON 作为数据封装格式，使用 long polling 方式获取服务端数据，以 publish/subscribe 方式发布和订阅消息。

备注

Multipart/x-mixed-replace 示例

```
<%
response.setContentType("multipart/mixed;boundary=BOUNDARY");
int i = 1;
while(i > 0) {
 out.println("--BOUNDARY\r\n");
%>

<html>
<head><title>百度一下，你就知道</title>
</head>
<body>
<%
    out.println("push "+i+"<br/>");
    out.flush();
    try {
        Thread.currentThread().sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

%>
</body>
</html>
<%
i++;
}
%>
```

Pushlet 简介

Pushlet 是一个开源的 Comet 框架，在设计上有很多值得借鉴的地方，对于开发轻量级的 Comet 应用很有参考价值。

Pushlet 使用了观察者模型：客户端发送请求，订阅感兴趣的事件；服务器端为每个客户端分配一个会话 ID 作为标记，事件源会把新产生的事件以多播的方式发送到订阅者的事件队列里。

pushlet 提供了基于 AJAX 的 JavaScript 库文件用于实现长轮询方式的“服务器推”；还提供了基于 iframe 的 JavaScript 库文件用于实现流方式的“服务器推”。

JavaScript 库做了很多封装工作：

定义客户端的通信状态：STATE_ERROR、STATE_ABORT、STATE_NULL、

STATE_READY、STATE_JOINED、STATE_LISTENING；

保存服务器分配的会话 ID，在建立连接之后的每次请求中会附上会话 ID 表明身份；

提供了 join()、leave()、subscribe()、 unsubsribe()、listen() 等 API 供页面调用；

提供了处理响应的 JavaScript 函数接口 onData()、onEvent()…

网页可以很方便地使用这两个 JavaScript 库文件封装的 API 与服务器进行通信。

客户端与服务器端通信信息格式:

pushlet 定义了一套客户与服务器通信的信息格式，使用 XML 格式。定义了客户端发送请求的类型：join、leave、subscribe、unsubscribe、listen、refresh；以及响应的事件类型：data、join_ack、listen_ack、refresh、heartbeat、error、abort、subscribe_ack、unsubscribe_ack。

服务器端事件队列管理:

pushlet 在服务器端使用 Java Servlet 实现，其数据结构的设计框架仍可适用于 PHP、C 编写的后台客户端。

Pushlet 支持客户端自己选择使用流、拉（长轮询）、轮询方式。服务器端根据客户选择的方式在读取事件队列（fetchEvents）时进行不同的处理。“轮询”模式下 fetchEvents() 会马上返回。“流”和“拉”模式使用阻塞的方式读事件，如果超时，会发给客户端发送一个没有新信息收到的“heartbeat”事件，如果是“拉”模式，会把“heartbeat”与“refresh”事件一起传给客户端，通知客户端重新发出请求、建立连接。

客户服务器之间的会话管理:

服务端在客户端发送 join 请求时，会为客户端分配一个会话 ID， 并传给客户端，然后客户端就通过此会话 ID 标明身份发出 subscribe 和 listen 请求。服务器端会为每个会话维护一个订阅的主题集合、事件队列。

服务器端的事件源会把新产生的事件以多播的方式发送到每个会话（即订阅者）的事件队列里。

参考资料

1. http://www.javaeye.com/topic/28020?page=1
2. http://www.ibm.com/developerworks/cn/web/wa-lo-comet/index.html
3. http://docs.codehaus.org/display/JETTY/Continuations
4. http://www.pushlets.com/doc/protocol-all.html

5. http://www.matrix.org.cn/resource/article/2007-01-16/bcc2c490-a502-11db-8440-755941c7293d.html
6. http://ajaxpatterns.org/HTTP_Streaming //usefull
7. http://tomcat.apache.org/tomcat-6.0-doc/aio.html //advanced IO & comet
8. http://www.duduwolf.com/wiki/2007/440.html
9. https://grizzly.dev.java.net/ //Grizzly is an NIO frameowork for building scalable application (NIO和IO的主要区别是非阻塞的)
10. http://www.ibm.com/developerworks/cn/java/j-jettydwr/index.html //jetty ContinuationServlet 简介 和dwr reverse ajax
11. http://alex.dojotoolkit.org/?p=545 //Comet: Low Latency Data for the Browser
12. http://www.zeitoun.net/index.php?2007/06/22/46-how-to-implement-comet-with-php // a php comet demo

Comet 讨论

1. Joe Walker (23 October 2007). "Why Comet is of Growing Importance". Comet Daily. Retrieved 4 Dec 2007.
2. Rodney Gedda (14 October 2005). "Desktop apps coming to the Web: Google". Computerworld. Retrieved 29 Nov 2007.
3. Paul Graham (November 2005). "Web 2.0". paulgraham.com. Retrieved 29 Nov 2007.
4. Alex Russell (3 March 2006). "Comet: Low Latency Data for the Browser". Continuing Intermittent Incoherency. Retrieved 29 Nov 2007.
5. Greg Wilkins (6 November 2007). "Comet is Always Better Than Polling". Comet Daily. Retrieved 29 Nov 2007.
6. Jacob Rus (11 December 2007). "The Future of Comet: Part 1, Comet Today". Comet Daily. Retrieved 14 Dec 2007.
7. Andrew Betts (2007). "Browser Techniques". Meteor website. Retrieved 29 Nov 2007.
8. Maciej Stachowiak (26 June 2007). "Re: XMLHttpRequest and readyState==3". Webkit-dev mailing list. Retrieved 29 Nov 2007.
9. Alex Russell (12 February 2006). "What else is buried down in the

depths of Google's amazing JavaScript?". Continuing Intermittent Incoherency. Retrieved 29 Nov 2007.

For advice about avoiding garbage collection bugs when implementing Comet with htmlfiles, see:

Michael Carter (25 October 2007). "HTTP Streaming and Internet Explorer". Comet Daily. Retrieved 29 Nov 2007.

Michael Carter (18 November 2007). "IE ActiveX("htmlfile") Transport, Part II". Comet Daily. Retrieved 29 Nov 2007.

10. Johnny Stenback, et al. (March–April 2004). "Bug 237319 – Add support for server push using multipart/x-mixed-replace with XMLHttpRequest". Mozilla Bugzilla bug tracker. Retrieved 29 November 2007. Also see:

    Alex Russell (6 August 2005). "Toward server-sent data w/o iframes". Continuing Intermittent Incoherency. Retrieved 29 Nov 2007.

11. Rob Butler, et al. (June 2006). "Bug 14392: Add support for multipart/x-mixed-replace to XMLHttpRequest". Webkit Bugzilla bug tracker. Retrieved 29 Nov 2007.
12. Alex Russell (21 December 2006). "Adventures In Comet and Multipart Mime". Continuing Intermittent Incoherency. Retrieved 29 Nov 2007.
13. Andrew Betts (4 December 2007). "Cross Site Scripting Joy". Comet Daily. Retrieved 14 Dec 2007.
14. Bob Ippolito (5 December 2005). "Remote JSON – JSONP". from __future__ import *. Retrieved 30 Nov 2007.
15. Alex Russell (22 July 2006). "Cross Domain Comet". Continuing Intermittent Incoherency. Retrieved 30 Nov 2007.
16. HTTP 1.1 specification, section 8.14. W3C. Retrieved 30 Nov 2007.
17. Alessandro Alinone (December 2005). "Changing the Web Paradigm". Lightstreamer white paper. Retrieved 14 Dec 2007.
18. Michael Carter (24 October 2007). "Comet for Highly Scalable Applications". Presentation at The Ajax Experience Boston. (Link goes to PDF slides). Retrieved 30 Nov 2007.
19. Alessandro Alinone (19 October 2007). "Comet and Push Technology". Comet Daily. Retrieved 14 Dec 2007.
20. Just Van Den Broecke (3 January 2007). "Pushlets: Send events from

servlets to DHTML client browsers". JavaWorld. Retrieved 14 Dec 2007.

21. Dion Almaer (29 September 2005). "Jotspot Live: Live, group note-taking" (interview with Abe Fettig). Ajaxian. Retrieved 15 Dec 2007.
    Matt Marshall (15 December 2006). "Renkoo launches event service — in time to schedule holiday cocktails". Venture Beat. Retrieved 15 Dec 2007.
22. Alex Russell, et al. (2007). Bayeux Protocol specification, 1.0 draft 1. Dojo Foundation. Retrieved 14 Dec 2007.
23. Jesse James Garrett (18 February 2005). "Ajax: A New Approach to Web Applications". Adaptive Path. Retrieved 29 Nov 2007.

进阶资料

1. Ian Hickson, et al. (2005–2007). HTML 5 specification, Section 6.2: Server-sent DOM events. WHATWG. Retrieved 14 Dec 2007
2. Rohit Khare (August 2005). "Beyond AJAX: Accelerating Web Applications with Real-Time Event Notification". KnowNow white paper (link from the Wayback Machine). Retrieved 14 Dec 2007