[**comet基于HTTP长连接技术（java即时通信，推送技术详解）**](http://www.cnblogs.com/jifeng/p/4676903.html)

Posted on 2015-07-26 00:50 [季枫](http://www.cnblogs.com/jifeng/) 阅读(7300) 评论(1) [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=4676903) [收藏](http://www.cnblogs.com/jifeng/p/4676903.html)

服务器推送技术的基础思想是将浏览器主动查询信息改为服务器主动发送信息，服务器发送一批数据，浏览器显示消息，同时保证与服务器的连接，当服务器需要再一次的发送数据，浏览器显示数据并保持连接。

comet基于HTTP长连接技术，无需安装插件。

* comet：一个概念，web push
* pushlet：comet的一个实现。
* comet4j：也是comet的一个实现（[comet4j开发指南](http://blog.csdn.net/majian_1987/article/details/8489738) [googlecode](http://code.google.com/p/comet4j/wiki/GettingStarted)）
* 就是保持长连接的策略问题，有人用jquery写了相应的util

* Pushlet基于HTTP流，这种技术常常用在多媒体视频、通讯应用中，比如QuickTime。与装载HTTP页面之后马上关闭HTTP连接的做法相反，Pushlet采用HTTP流方式将新变动的数据主动地推送到client（客户端），再此期间HTTP连接一直保持打开。有关如何在Java中实现这种Keep-alive的长连接请参看Sun提供的《HTTP Persistent Connection》和W3C的《HTTP1.1规范》

* Tomcat的comet原理其实同样很简单，它无非就是做了一件事情，它允许servlet执行完毕后的response没有被回收，我们只要拿到这个Reponse的引用并保存起来，就可以随时从Server向Client端Push数据了。每个连接一个线程的模型便非常简单。该模型对于 Comet 不大适用，但是，Java 对此同样有解决的办法。为了有效地处理 Comet，需要非阻塞 IO，Java 通过它的 NIO 库提供非阻塞 IO。两种最流行的开源服务器 Apache Tomcat 和 Jetty 都利用 NIO 增加非阻塞 IO，从而支持 Comet.

* 而非阻塞I/O和同步I/O最明显的不同就是同步I/O所有可能被阻塞的地址在非阻塞I/O中都不会被阻塞。如在读取数据时，如果数据暂时无法被读取。那么在非阻塞I/O中会立刻返回，以便程序可以执行其他的代码，然后系统会不断侦测这个未完成的读取操作，直到可以继续读数据时再来完成这个操作。非阻塞式IO的出现的目的就是为了解决这个瓶颈。而非阻塞式IO是怎么实现的呢？非阻塞IO处理连接的线程数和连接数没有联系，也就是说处理10000个连接非阻塞IO不需要10000个线程，你可以用1000个也可以用2000个线程来处理。因为非阻塞IO处理连接是异步的。当某个连接发送请求到服务器，服务器把这个连接请求当作一个请求"事件"，并把这个"事件"分配给相应的函数处理。我们可以把这个处理函数放到线程中去执行，执行完就把线程归还。这样一个线程就可以异步的处理多个事件。而阻塞式IO的线程的大部分时间都浪费在等待请求上了。
* 在comet中，为了保持长连接，如果使用阻塞时IO，则不可避免的对每一个连接保持一个线程。不同于短连接，线程可以及时释放。长连接对应的线程可能永远不能释放，这样一个server能够服务的客户端的数量就受到了线程数量上限的限制。而使用NIO可以伺候多个连接而不必要保持相应数量的线程。就解决了这个问题。

* Tomcat提供了CometProcessor接口，有这种特定标记的Servlet，Tomcat会做特殊处理，Tomcat不会把它当做普通Servlet实行完毕后，会回收request和response。注意：实现CometProcessor接口后不用在servlet中写doGet,doPoset方法，所有事件在BEGIN,READ,END,ERROR中实现。
* 从event拿到的request和response不会在begin和end/error之间不会被释放，一直有效。可以用来传递消息。Note that the response object and dependent OutputStream and Writer are still not synchronized, so when they are accessed by multiple threads, synchronization is mandatory.
* BEGIN：初始化参数和获取request和response，结束时，request is commited
* READ（只有POST方法，才会触发该事件）：有数据从request进来，可以从request读取数据。在此事件之外不允许读取request的数据。On some platforms, like Windows, a client disconnect is indicated by a READ event. Reading from the stream may result in -1, an IOException or an EOFException. Make sure you properly handle all these three cases. If you don't catch the IOException, Tomcat will instantly invoke your event chain with an ERROR as it catches the error for you, and you will be notified of the error at that time.
* END: End may be called to end the processing of the request. Fields that have been initialized in the begin method should be reset. After this event has been processed, the request and response objects, as well as all their dependent objects will be recycled and used to process other requests. End will also be called when data is available and the end of file is reached on the request input (this usually indicates the client has pipelined a request).
* ERROR: Error will be called by the container in the case where an IO exception or a similar unrecoverable error occurs on the connection. Fields that have been initialized in the begin method should be reset. After this event has been processed, the request and response objects, as well as all their dependent objects will be recycled and used to process other requests.（END，ERROR之后request和response就不要再用了）

* ventSubType.TIMEOUT: The connection timed out (sub type of ERROR); note that this ERROR type is not fatal, and the connection will not be closed unless the servlet uses the close method of the event.
* EventSubType.CLIENT\_DISCONNECT: The client connection was closed (sub type of ERROR). method of the event.
* EventSubType.IOEXCEPTION: An IO exception occurred, such as invalid content, for example, an invalid chunk block (sub type of ERROR).
* EventSubType.WEBAPP\_RELOAD: The web application is being reloaded (sub type of END).
* EventSubType.SESSION\_END: The servlet ended the session (sub type of END).

BEGIN-READ-READ-Error/TIMEOUT。随时可以event.close()。来终止连接。

* writer.flush();writer.close();长轮询和流风格的comet的差别只是取决于是否有第二句（长轮询需要client端在response关闭后再重连）

If you are using the NIO connector, you can set individual timeouts for your different comet connections. To set a timeout, simply set a request attribute like the following code shows:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | CometEvent event.... event.setTimeout(30\*1000); |  |
|  |  |  |

or

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | event.getHttpServletRequest().setAttribute("org.apache.tomcat.comet.timeout", new Integer(30 \* 1000)); |  |
|  |  |  |

This sets the timeout to 30 seconds. Important note, in order to set this timeout, it has to be done on the BEGIN event. The default value is soTimeout

**简单的Comet servlet代码示例：**

[复制代码](javascript:void(0);)

http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif  
 1 import java.io.\*;  
 2 import javax.servlet.ServletException;  
 3 import javax.servlet.http.\*;  
 4 import org.apache.catalina.CometEvent;  
 5 import org.apache.catalina.CometProcessor;  
 6 import org.apache.catalina.CometEvent.EventType;  
 7   
 8 public class cometServlet extends HttpServlet implements CometProcessor {  
 9      
10         public void event(CometEvent e) throws IOException, ServletException {  
11                if(e.getEventType() == EventType.BEGIN) {  
12                       // fill in code handling here  
13                    HttpServletResponse response = e.getHttpServletResponse();  
14                    PrintWriter out = response.getWriter();  
15                    out.write("Hello world");  
16                    out.flush();   
17                    //System.out.println("message sent");  
18                }  
19                if(e.getEventType() == EventType.READ) {  
20                   // fill in code handling here  
21                }  
22                // and continue handing other events  
23         }  
24 }

[复制代码](javascript:void(0);)

在此源代码中，仅完成向客户端发送Hello World字符串的功能，关键点，out.flush()不可缺少

**客户端javascript相关代码：**

[复制代码](javascript:void(0);)

http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif  
 1 <script>  
 2 function CometEx() {  
 3   var request =  new XMLHttpRequest();  
 4   request.open("GET", 'http://localhost:8080/cometEx/cometServlet', true);  
 5   request.onreadystatechange = function() {  
 6     if (request.readyState == 3 && request.status == 200) {  
 7            alert(request.responseText);       
 8     }  
 9   }  
10   request.send(null);  
11 }  
12 </script>

[复制代码](javascript:void(0);)

服务器端代码类似与普通Ajax代码，其中，需要注意的是：request.readyState值如果设置为4，浏览器会处于长期等待状态，而收不到响应消息，设置为3后，firefox浏览器正常，但IE不能正常获得消息