Web 通信 之 长连接、长轮询（long polling）

基于HTTP的长连接,是一种通过长轮询方式实现"服务器推"的技术,它弥补了HTTP简单的请求应答模式的不足,极大地增强了程序的实时性和交互性。

一、什么是长连接、长轮询？

用通俗易懂的话来说，就是客户端不停的向服务器发送请求以获取最新的数据信息。这里的“不停”其实是有停止的，只是我们人眼无法分辨是否停止，它只是一种快速的停下然后又立即开始连接而已。

二、长连接、长轮询的应用场景

长连接、长轮询一般应用与WebIM、ChatRoom和一些需要及时交互的网站应用中。其真实案例有：WebQQ、Hi网页版、Facebook IM等。

如果你对服务器端的反向Ajax感兴趣，可以参考这篇文章 DWR 反向Ajax 服务器端推的方式：http://www.cnblogs.com/hoojo/category/276235.html

欢迎大家继续支持和关注我的博客：

http://hoojo.cnblogs.com

http://blog.csdn.net/IBM_hoojo

也欢迎大家和我交流、探讨IT方面的知识。

email：hoojo_@126.com

三、优缺点

轮询：客户端定时向服务器发送Ajax请求，服务器接到请求后马上返回响应信息并关闭连接。
优点：后端程序编写比较容易。
缺点：请求中有大半是无用，浪费带宽和服务器资源。
实例：适于小型应用。

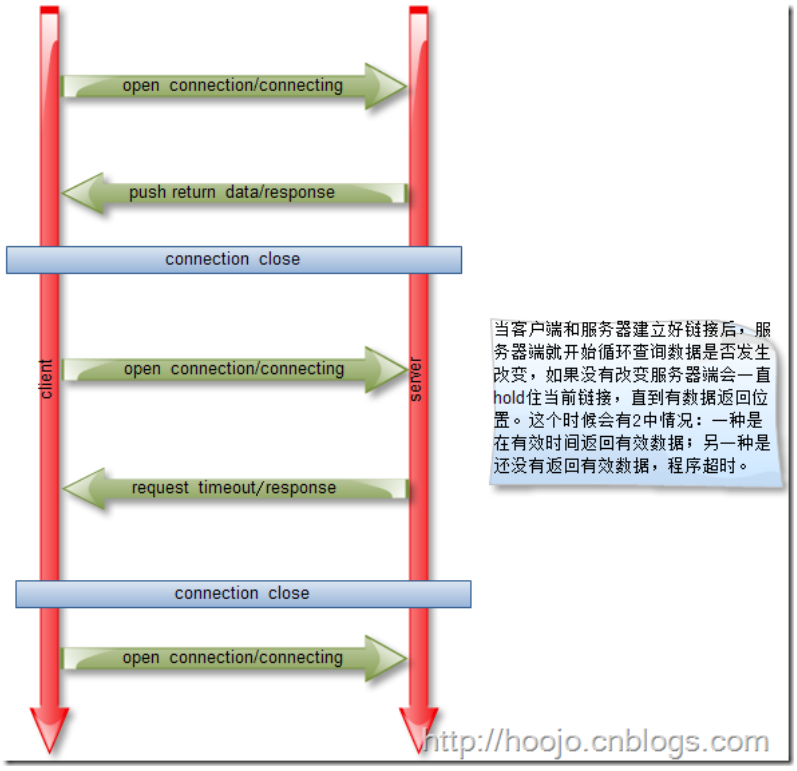
长轮询：客户端向服务器发送Ajax请求，服务器接到请求后hold住连接，直到有新消息才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。
优点：在无消息的情况下不会频繁的请求，耗费资源小。
缺点：服务器hold连接会消耗资源，返回数据顺序无保证，难于管理维护。
实例：WebQQ、Hi网页版、Facebook IM。

长连接：在页面里嵌入一个隐藏iframe，将这个隐藏iframe的src属性设为对一个长连接的请求或是采用xhr请求，服务器端就能源源不断地往客户端输入数据。
优点：消息即时到达，不发无用请求；管理起来也相对方便。
缺点：服务器维护一个长连接会增加开销。
实例：Gmail聊天

Flash Socket：在页面中内嵌入一个使用了Socket类的 Flash 程序JavaScript通过调用此Flash程序提供的Socket接口与服务器端的Socket接口进行通信，JavaScript在收到服务器端传送的信息后控制页面的显示。
优点：实现真正的即时通信，而不是伪即时。
缺点：客户端必须安装Flash插件；非HTTP协议，无法自动穿越防火墙。
实例：网络互动游戏。

四、实现原理

所谓长连接，就是要在客户端与服务器之间创建和保持稳定可靠的连接。其实它是一种很早就存在的技术，但是由于浏览器技术的发展比较缓慢，没有为这种机制的实现提供很好的支持。所以要达到这种效果，需要客户端和服务器的程序共同配合来完成。通常的做法是，在服务器的程序中加入一个死循环，在循环中监测数据的变动。当发现新数据时，立即将其输出给浏览器并断开连接，浏览器在收到数据后，再次发起请求以进入下一个周期，这就是常说的长轮询（long-polling）方式。如下图所示，它通常包含以下几个关键过程：



1. 轮询的建立
建立轮询的过程很简单，浏览器发起请求后进入循环等待状态，此时由于服务器还未做出应答，所以HTTP也一直处于连接状态中。
2. 数据的推送
在循环过程中，服务器程序对数据变动进行监控，如发现更新，将该信息输出给浏览器，随即断开连接，完成应答过程，实现“服务器推”。
3. 轮询的终止
轮询可能在以下3种情况时终止：
3.1. 有新数据推送
当循环过程中服务器向浏览器推送信息后，应该主动结束程序运行从而让连接断开，这样浏览器才能及时收到数据。
3.2. 没有新数据推送
循环不能一直持续下去，应该设定一个最长时限，避免WEB服务器超时（Timeout），若一直没有新信息，服务器应主动向浏览器发送本次轮询无新信息的正常响应，并断开连接，这也被称为“心跳”信息。
3.3. 网络故障或异常
由于网络故障等因素造成的请求超时或出错也可能导致轮询的意外中断，此时浏览器将收到错误信息。
4. 轮询的重建
浏览器收到回复并进行相应处理后，应马上重新发起请求，开始一个新的轮询周期。

五、程序设计

1、普通轮询 Ajax方式

客户端代码片段

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="author" content="hoojo & http://hoojo.cnblogs.com">
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <%@ include file="/tags/jquery-lib.jsp"%>

        <script type="text/javascript">
            $(function () {

                window.setInterval(function () {

                    $.get("${pageContext.request.contextPath}/communication/user/ajax.mvc",
                        {"timed": new Date().getTime()},
                        function (data) {
                            $("#logs").append("[data: " + data + " ]<br/>");
                        });
                }, 3000);

            });
        </script>
    </head>
```

```
    <body>
        <div id="logs"></div>
    </body>
</html>
```

客户端实现的就是用一种普通轮询的结果，比较简单。利用setInterval不间断的刷新来获取服务器的资源，这种方式的优点就是简单、及时。缺点是链接多数是无效重复的；响应的结果没有顺序（因为是异步请求，当发送的请求没有返回结果的时候，后面的请求又被发送。而此时如果后面的请求比前面的请求要先返回结果，那么当前面的请求返回结果数据时已经是过时无效的数据了）；请求多，难于维护、浪费服务器和网络资源。

服务器端代码

```
@RequestMapping("/ajax")
public void ajax(long timed, HttpServletResponse response) throws Exception {
     PrintWriter writer = response.getWriter();

     Random rand = new Random();
     // 死循环 查询有无数据变化
     while (true) {
         Thread.sleep(300); // 休眠300毫秒，模拟处理业务等
         int i = rand.nextInt(100); // 产生一个0-100之间的随机数
         if (i > 20 && i < 56) { // 如果随机数在20-56之间就视为有效数据，模拟数据发生变化
             long responseTime = System.currentTimeMillis();
             // 返回数据信息，请求时间、返回数据时间、耗时
             writer.print("result: " + i + ", response time: " + responseTime + ", request time: " + timed + ", use time: " + (resp
             break; // 跳出循环，返回数据
         } else { // 模拟没有数据变化，将休眠 hold住连接
             Thread.sleep(1300);
         }
     }

}
```

服务器端实现，这里就模拟下程序监控数据的变化。上面代码属于SpringMVC 中controller中的一个方法，相当于Servlet中的一个doPost/doGet方法。如果没有程序环境适应servlet即可，将方法体中的代码copy到servlet的doGet/doPost中即可。

服务器端在进行长连接的程序设计时，要注意以下几点：

1. 服务器程序对轮询的可控性

由于轮询是用死循环的方式实现的，所以在算法上要保证程序对何时退出循环有完全的控制能力，避免进入死循环而耗尽服务器资源。

2. 合理选择"心跳"频率

从图1可以看出，长连接必须由客户端不停地进行请求来维持，所以在客户端和服务器间保持正常的"心跳"至为关键，参数POLLING_LIFE应小于WEB服务器的超时时间，一般建议在10～20秒左右。

3. 网络因素的影响

在实际应用时，从服务器做出应答，到下一次循环的建立，是有时间延迟的，延迟时间的长短受网络传输等多种因素影响，在这段时间内，长连接处于暂时断开的空档，如果恰好有数据在这段时间内发生变动，服务器是无法立即进行推送的，所以，在算法设计上要注意解决由于延迟可能造成的数据丢失问题。

4. 服务器的性能

在长连接应用中，服务器与每个客户端实例都保持一个持久的连接，这将消耗大量服务器资源，特别是在一些大型应用系统中更是如此，大量并发的长连接有可能导致新的请求被阻塞甚至系统崩溃，所以，在进行程序设计时应特别注意算法的优化和改进，必要时还需要考虑服务器的负载均衡和集群技术。

```
[data: result: 35, response time: 1380095839335, request time: 1380095837341, use time: 1994 ]
[data: result: 22, response time: 1380095842676, request time: 1380095842340, use time: 336 ]
[data: result: 55, response time: 1380095850873, request time: 1380095847341, use time: 3532 ]
[data: result: 42, response time: 1380095854273, request time: 1380095852341, use time: 1932 ]
[data: result: 47, response time: 1380095857672, request time: 1380095857340, use time: 332 ]
[data: result: 33, response time: 1380095864272, request time: 1380095862341, use time: 1931 ]
[data: result: 25, response time: 1380095872665, request time: 1380095872341, use time: 324 ]
[data: result: 42, response time: 1380095874074, request time: 1380095867341, use time: 6733 ]
[data: result: 43, response time: 1380095877673, request time: 1380095877340, use time: 333 ]
[data: result: 29, response time: 1380095882673, request time: 1380095882342, use time: 331 ]
[data: result: 46, response time: 1380095887808, request time: 1380095887389, use time: 419 ]
[data: result: 32, response time: 1380095893009, request time: 1380095892340, use time: 669 ]
[data: result: 22, response time: 1380095900885, request time: 1380095897341, use time: 3544 ]
```

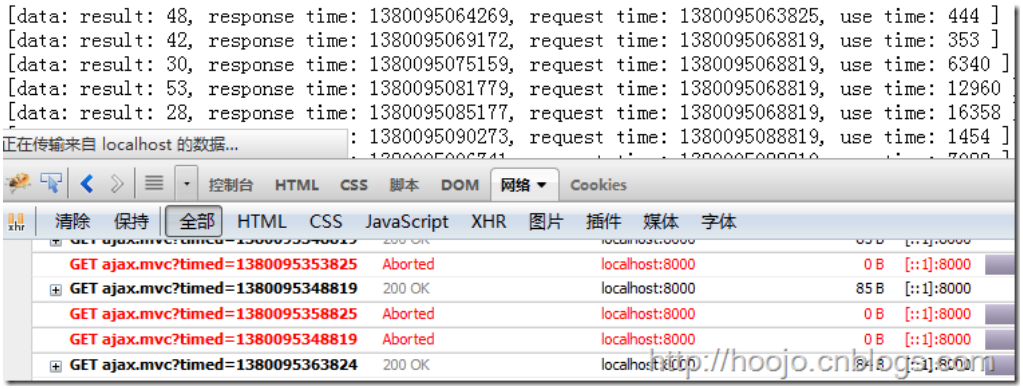

上图是返回的结果，可以看到先发出请求，不一定会最先返回结果。这样就不能保证顺序，造成脏数据或无用的连接请求。可见对服务器或网络的资源浪费。

2、普通轮询 iframe方式

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <%@ include file="/tags/jquery-lib.jsp"%>

        <script type="text/javascript">
            $(function () {
```

```
                window.setInterval(function () {
                    $("#logs").append("[data: " + $($("#frame").get(0).contentDocument).find("body").text() + " ]<br/>");
                    $("#frame").attr("src", "${pageContext.request.contextPath}/communication/user/ajax.mvc?timed=" + new Date().ge
                    // 延迟1秒再重新请求
                    window.setTimeout(function () {
                        window.frames["polling"].location.reload();
                    }, 1000);
                }, 5000);

            });
        </script>
    </head>

    <body>
        <iframe id="frame" name="polling" style="display: none;"></iframe>
        <div id="logs"></div>
    </body>
</html>
```

这里的客户端程序是利用隐藏的iframe向服务器端不停的拉取数据，将iframe获取后的数据填充到页面中即可。同ajax实现的基本原理一样，唯一不同的是当一个请求没有响应返回数据的情况下，下一个请求也将开始，这时候前面的请求将被停止。如果要使程序和上面的ajax请求一样也可以办到，那就是给每个请求分配一个独立的iframe即可。下面是返回的结果：



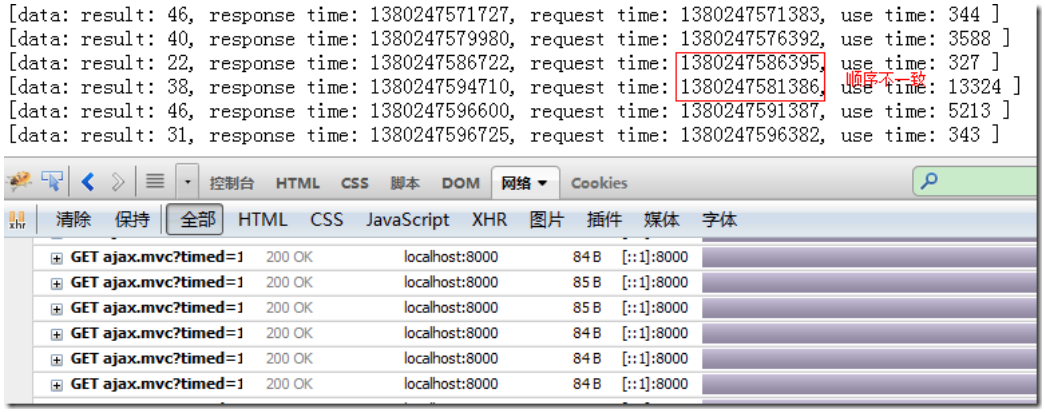
其中红色是没有成功返回请求就被停止（后面请求开始）掉的请求，黑色是成功返回数据的请求。

3、长连接iframe方式

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="author" content="hoojo & http://hoojo.cnblogs.com">
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <%@ include file="/tags/jquery-lib.jsp"%>

        <script type="text/javascript">
            $(function () {

                window.setInterval(function () {
                    var url = "${pageContext.request.contextPath}/communication/user/ajax.mvc?timed=" + new Date().getTime();
                    var $iframe = $('<iframe id="frame" name="polling" style="display: none;" src="' + url + '"></iframe>');
                    $("body").append($iframe);

                    $iframe.load(function () {
                        $("#logs").append("[data: " + $($iframe.get(0).contentDocument).find("body").text() + " ]<br/>");
                        $iframe.remove();
                    });
                }, 5000);

            });
        </script>
    </head>

    <body>

        <div id="logs"></div>
```

```
    </body>
</html>
```

这个轮询方式就是把刚才上面的稍微改下，每个请求都有自己独立的一个iframe，当这个iframe得到响应的数据后就把数据push到当前页面上。使用此方法已经类似于ajax的异步交互了，这种方法也是不能保证顺序的、比较耗费资源、而且总是有一个加载的条在地址栏或状态栏附件（当然要解决可以利用htmlfile，Google的攻城师们已经做到了，网上也有封装好的lib库），但客户端实现起来比较简单。



如果要保证有序，可以不使用setInterval，将创建iframe的方法放在load事件中即可，即使用递归方式。调整后的代码片段如下：

```
<script type="text/javascript">
    $(function () {
        (function iframePolling() {
            var url = "${pageContext.request.contextPath}/communication/user/ajax.mvc?timed=" + new Date().getTime();
            var $iframe = $('<iframe id="frame" name="polling" style="display: none;" src="' + url + '"></iframe>');
            $("body").append($iframe);

            $iframe.load(function () {
                $("#logs").append("[data: " + $($iframe.get(0).contentDocument).find("body").text() + " ]<br/>");
                $iframe.remove();

                // 递归
                iframePolling();
            });
        })();
    });
</script>
```

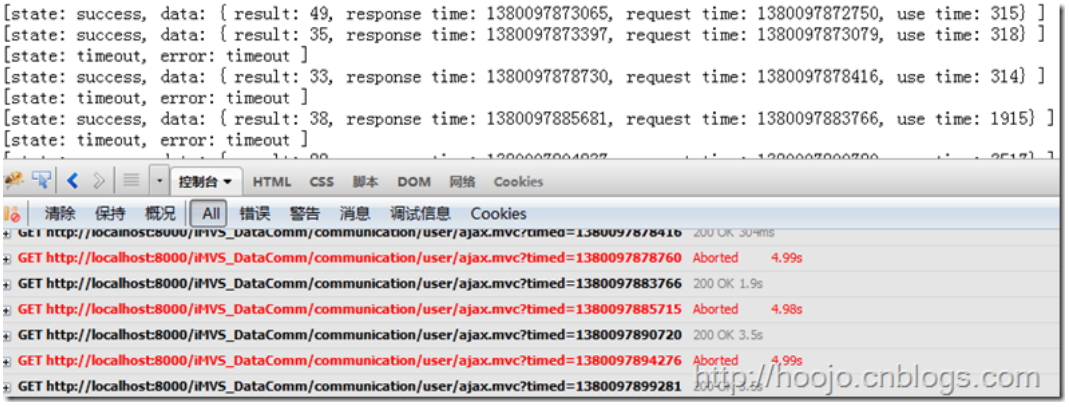
这种方式虽然保证了请求的顺序，但是它不会处理请求延时的错误或是说很长时间没有返回结果的请求，它会一直等到返回请求后才能创建下一个iframe请求，总会和服务器保持一个连接。和以上轮询比较，缺点就是消息不及时，但保证了请求的顺序。

4、ajax实现长连接

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <%@ include file="/tags/jquery-lib.jsp"%>

        <script type="text/javascript">
            $(function () {

                (function longPolling() {

                    $.ajax({
                        url: "${pageContext.request.contextPath}/communication/user/ajax.mvc",
                        data: {"timed": new Date().getTime()},
                        dataType: "text",
                        timeout: 5000,
                        error: function (XMLHttpRequest, textStatus, errorThrown) {
                            $("#state").append("[state: " + textStatus + ", error: " + errorThrown + " ]<br/>");
                            if (textStatus == "timeout") { // 请求超时
                                    longPolling(); // 递归调用

                                // 其他错误，如网络错误等
                                } else {
                                    longPolling();
                                }
```

```
                        },
                    success: function (data, textStatus) {
                        $("#state").append("[state: " + textStatus + ", data: { " + data + "} ]<br/>");

                        if (textStatus == "success") { // 请求成功
                            longPolling();
                        }
                    }
                });
            })();

        });
    </script>
</head>

<body>
```

上面这段代码就是才有Ajax的方式完成长连接，主要优点就是和服务器始终保持一个连接。如果当前连接请求成功后，将更新数据并且继续创建一个新的连接和服务器保持联系。如果连接超时或发生异常，这个时候程序也会创建一个新连接继续请求。这样就大大节省了服务器和网络资源，提高了程序的性能，从而也保证了程序的顺序。



六、总结

现代的浏览器都支持跨域资源共享（Cross-Origin Resource Share，CORS）规范，该规范允许XHR执行跨域请求，因此基于脚本的和基于iframe的技术已成为了一种过时的需要。

把Comet做为反向Ajax的实现和使用的最好方式是通过XMLHttpRequest对象，该做法提供了一个真正的连接句柄和错误处理。当然你选择经由HTTP长轮询使用XMLHttpRequest对象（在服务器端挂起的一个简单的Ajax请求）的Comet模式，所有支持Ajax的浏览器也都支持该种做法。

基于HTTP的长连接技术，是目前在纯浏览器环境下进行即时交互类应用开发的理想选择，随着浏览器的快速发展，html5将为其提供更好的支持和更广泛的应用。在html5中有一个websocket 可以很友好的完成长连接这一技术，网上也有相关方面的资料，这里也就不再做过多介绍。

来源： http://www.cnblogs.com/hoojo/p/longPolling_comet_jquery_iframe_ajax.html