Building scalable web site

# 上品网站现状

## 请求头信息:

|  |  |
| --- | --- |
| **Connection** | **Close** |
| **Content-Language** | **zh-CN** |
| **Content-Type** | **text/html;charset=UTF-8** |
| **Date** | **Tue, 19 Nov 2013 06:17:22 GMT** |
| **Server** | **nginx/1.4.2** |
| **Set-Cookie** | **JSESSIONID=C900072941B780B396166043DC234647.tomcat21; Path=/** |
| **Vary** | **Accept-Encoding** |
| **Via** | **1.1 www.shopin.net:80 (squid/2.7.STABLE9)** |
| **X-Cache** | **MISS from www.shopin.net** |
| **Accept** | **text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8** |
| **Accept-Encoding** | **gzip, deflate** |
| **Accept-Language** | **zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3** |
| **Connection** | **keep-alive** |
| **Cookie** | **Hm\_lvt\_b1f097642bd4f6603948b19d9f4560bf=1384755710,1384759762,1384762842,1384828706; Hm\_lvt\_b14dd8757ed9f7dd76e92b0d74271018=1384507616,1384755710,1384759763,1384828706; CNZZDATA1516938=cnzz\_eid%3D1317883292-1384414985-http%253A%252F%252Fwww.shopin.net%26ntime%3D1384839984%26cnzz\_a%3D3%26ltime%3D1384828706361%26rtime%3D3; \_cnzz\_CV=; \_\_utma=87647962.790416704.1384419649.1384828706.1384839984.12; \_\_utmz=87647962.1384504450.7.2.utmcsr=172.16.103.131|utmccn=(referral)|utmcmd=referral|utmcct=/channel/la11dy.html; vizNprd=Thu, 28 Nov 2013 09:00:52 GMT\_33; http%3A//cm.g.doubleclick.net/pixel%3Fgoogle\_nid%3Dbehe%26google\_cm=http%253A//cm.g.doubleclick.net/pixel%253Fgoogle\_nid%253Dbehe%2526google\_cm; http%3A//acookie.tanx.com/cms.gif%3Fid%3D34016883%26extendata%3Dtanx=http%253A//acookie.tanx.com/cms.gif%253Fid%253D34016883%2526extendata%253Dtanx; http%3A//cm.miaozhen.atm.youku.com/cm.gif%3Fdspid%3D11114=http%253A//cm.miaozhen.atm.youku.com/cm.gif%253Fdspid%253D11114; http%3A//c.x.cn.miaozhen.com/cm.gif%3Fdspid%3D11163=http%253A//c.x.cn.miaozhen.com/cm.gif%253Fdspid%253D11163; http%3A//cm.api.baifendian.com/Mapping.do%3Fbfd\_nid%3Dbehe=http%253A//cm.api.baifendian.com/Mapping.do%253Fbfd\_nid%253Dbehe; vizNsc=Tue, 14 Jan 2014 03:44:52 GMT\_14; changesid=web2; Hm\_lvt\_e0a537ff2f1696dc1ba1f057955404e7=1384487098; \_\_pztm\_cv=3B3G5O7656D2O3LM.1384487098862.1.1384487115505.1384487098862.1384487098862; shopin\_sid=3973b847-596c-4677-9876-a1a336c2216f; referer="http://www.shopin.net/"; JSESSIONID=9FDB5F0CFF0811CA3EAD6A221F228A8F.tomcat21; Hm\_lpvt\_b1f097642bd4f6603948b19d9f4560bf=1384840676; Hm\_lpvt\_b14dd8757ed9f7dd76e92b0d74271018=1384840677; \_\_utmc=87647962; \_\_utmb=87647962.12.10.1384839984; userTicket=ed767cb0-8787-426e-a6fb-6df13c578f70** |
| **Host** | **www.shopin.net** |
| **User-Agent** | **Mozilla/5.0 (Windows NT 6.1; rv:25.0) Gecko/20100101 Firefox/25.0 FirePHP/0.7.4** |
| **x-insight** | **activate** |

## **2使用Fiddler查看headers**

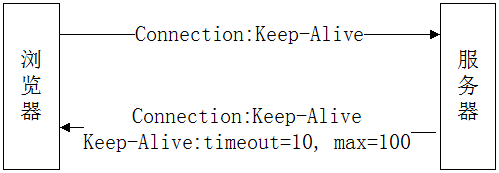
# 长连接优化

## Http持久连接

持久连接（Keep-Alive）也叫做长连接，它是一种 TCP 的连接方式，连接会被浏览器和服务器所缓存，在下次连接同一服务器时，缓存的连接被重新使用。由于 HTTP 的无状态性，人们也一直很清楚“一次性”的 HTTP 通信。持久连接则减少了创建连接的开销，提高了性能。HTTP/1.1 已经支持长连接，大部分浏览器和服务器也提供了长连接的支持。

可以想象，要想发起长连接，服务器和浏览器必须共同合作才可以。一方面浏览器要保持连接，另一方面服务器也不会断开连接。也就是说要想建立长连接，服务器和浏览器需要进行协商，而如何协商就要靠伟大的 HTTP 协议了。

### 长连接协商

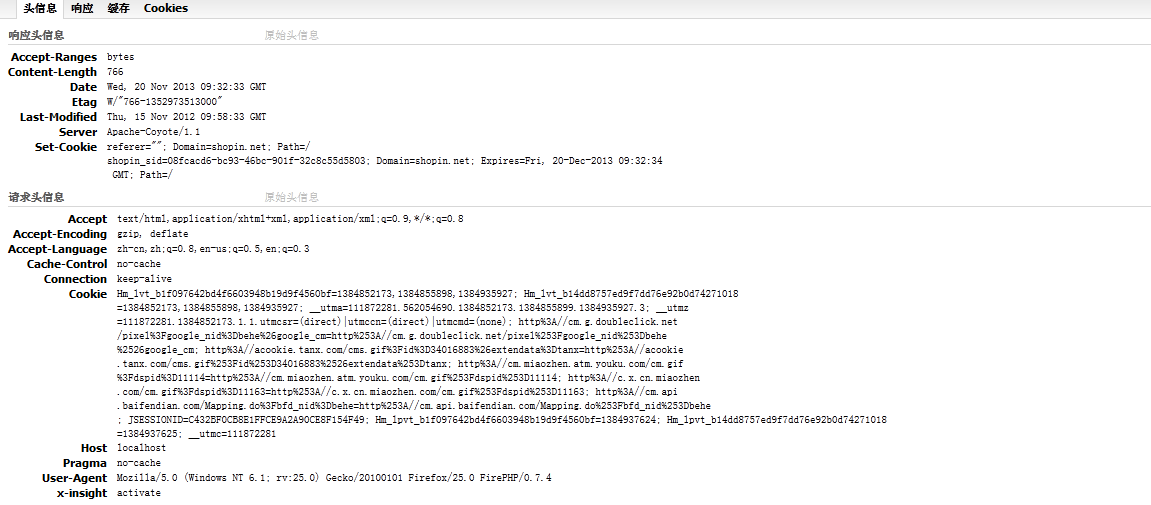


浏览器在请求的头部添加 Connection:Keep-Alive，以此告诉服务器“我支持长连接，你支持的话就和我建立长连接吧”，而倘若服务器的确支持长连接，那么就在响 应头部添加“Connection:Keep-Alive”，从而告诉浏览器“我的确也支持，那我们建立长连接吧”。服务器还可以通过 Keep-Alive:timeout=10, max=100 的头部告诉浏览器“我希望 10 秒算超时时间，最长不能超过 100 秒”。

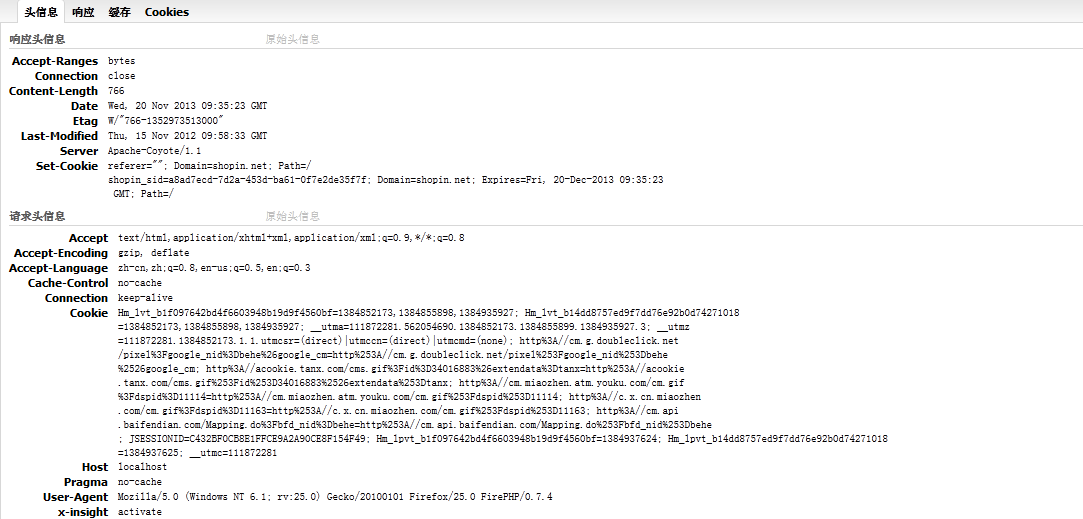
在 Tomcat 里是允许配置长连接的，配置 conf/server.xml 文件，配置 Connector 节点，该节点负责控制浏览器与 Tomcat 的连接，其中与长连接直接相关的有两个属性，它们分别是：keepAliveTimeout，它表示在 Connector 关闭连接前，Connector 为另外一个请求 Keep Alive 所等待的微妙数，默认值和 connectionTimeout 一样；另一个是 maxKeepAliveRequests，它表示 HTTP/1.0 Keep Alive 和 HTTP/1.1 Keep Alive / Pipeline 的最大请求数目，如果设置为 1，将会禁用掉 Keep Alive 和 Pipeline，如果设置为小于 0 的数，Keep Alive 的最大请求数将没有限制。也就是说在 Tomcat 里，默认长连接是打开的，当我们想关闭长连接时，只要将 maxKeepAliveRequests 设置为 1 就可以。

首先将 maxKeepAliveRequests 设置为 20，keepAliveTimeout 为 10000，通过 Firefox 查看请求头部

### 服务器打开长连接



### 服务器关闭长连接



|  |
| --- |
|  |
| **对比可以发现，Tomcat 关闭长连接后，在服务器的请求响应中，明确标识了：Connection close, 它告诉浏览器服务器并不支持长连接。那么长连接究竟可以带来怎么样的性能提升，我们用数据说话。我们依然使用 AB 工具，它可以使用一个 -k 的参数，模拟浏览器使用 HTTP 的 Keep-Alive 特性。** AB测试长连接 **测试脚本：ab – k -c 1000 -n 10000** [**http://www.shopin.net/favicon.ico**](http://www.shopin.net/favicon.ico)  **关闭长连接时：**  **Concurrency Level: 1000**  **Time taken for tests: 5.067 seconds**  **Complete requests: 10000**  **Failed requests: 0**  **Write errors: 0**  **Keep-Alive requests: 0**  **Total transferred: 44600000 bytes**  **HTML transferred: 42130000 bytes**  **Requests per second: 1973.64 [#/sec] (mean)**  **Time per request: 506.678 [ms] (mean)**  **Time per request: 0.507 [ms] (mean, across all concurrent requests)**  **Transfer rate: 8596.13 [Kbytes/sec] received**  **打开长连接时，maxKeepAliveRequests 设置为 50：**  **Concurrency Level: 1000**  **Time taken for tests: 1.671 seconds**  **Complete requests: 10000**  **Failed requests: 0**  **Write errors: 0**  **Keep-Alive requests: 10000**  **Total transferred: 44650000 bytes**  **HTML transferred: 42130000 bytes**  **Requests per second: 5983.77 [#/sec] (mean)**  **Time per request: 167.119 [ms] (mean)**  **Time per request: 0.167 [ms] (mean, across all concurrent requests)**  **Transfer rate: 26091.33 [Kbytes/sec] received**  **结果**  **结果一定会让您大为惊讶，使用长连接和不使用长连接的性能对比，对于 Tomcat 配置的 maxKeepAliveRequests 为 50 来说，竟然提升了将近 5 倍。可见服务器默认打开长连接是有原因的** HTTP 协议的合理使用**背景** **很多程序员都将精力专注在了技术实现上，他们认为性能的高低完全取决于代码的实现，却忽略了已经成型的某些规范、协议、工具。最典型的就是在 Web 开发上，部分开发人员没有意识到 HTTP 协议的重要性，以及 HTTP 协议可以提供程序员另一条性能优化之路。通过简单的在 JSP 的 request 对象中添加响应头部，往往可以迅速提升程序性能，一切实现代码仿佛都成浮云。本系列文章的宗旨也在于让程序员编最少的代码，提升最大的性能。**  **本文提出一个这样的需求，在文章前面部分提到的 test.jsp 中，它的一部分功能是显示服务器的当前时间。现在我们希望这个动态网页允许被浏览器缓存，这似乎有点不合理，但是在很多时候，虽然是动态网页，但是却只执 行一次（比如有些人喜欢将网页的主菜单存入数据库，那么他肯定不希望每次加载菜单都去读数据库）。浏览器缓存带来的性能提升已经众人皆知了，而很多人却并 不知道浏览器的缓存过期时间、缓存删除、什么页面可以缓存等，都可以由我们程序员来控制，只要您熟悉 HTTP 协议，就可以轻松的控制浏览器。**  **我们访问上面提及的 test.jsp。用 Firebug 查看请求情况，发现每次请求都会重新到服务器下载内容，这不难理解，因此 test.jsp 是动态内容，每次服务器必须都执行后才可以返回结果**  **安装代码片段**  **<%**  **SimpleDateFormat f2=new SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss");**  **String ims = request.getHeader("If-Modified-Since");**  **if (ims != null)**  **{**  **try**  **{**  **Date dt = f2.parse(ims.substring(0, ims.length()-4));**  **if (dt.after(new Date(2009, 1, 1)))**  **{**  **response.setStatus(304);**  **return;**  **}**  **} catch(Exception e)**  **{**  **}**  **}**  **response.setHeader("Last-Modified", f2.format(new Date(2010, 5, 5)) + " GMT");**  **%>**  上述代码的意图是：服务器获得浏览器请求头部中的 If-Modified-Since 时间，这个时间是浏览器询问服务器，它所请求的资源是否过期，如果没过期就返回 304 状态码，告诉浏览器直接使用本地的缓存就可以 修改后的头信息   增加了 Last-Modified 头部，这个头部告诉浏览器可以将此页面缓存。  按下 F5（必须是 F5 刷新），F5 会强制 Firefox 加载服务器内容，并且发出 If-Modified-Since 头部。得到的报头结果    可以看到，图 12 的底部已经提示所有内容都来自缓存。浏览器的请求头部多出了 If-Modified-Since，以此询问服务器从缓存时间起，服务器是否对资源进行了修改。服务器判断后发现没有对此资源（test.jsp）修 改，就返回 304 状态码，告诉浏览器可以使用缓存。  我们在上面的实验中，用到了 HTTP 协议的相关知识，其中涉及了 If-Modified-Since、Last-Modified、304 状态码等，事实上与缓存相关的 HTTP 头部还有许多，诸如过期设置的头部等。熟悉了 HTTP 头部，就如同学会了如何与用户的浏览器交谈，也可以利用协议提升您的程序性能。这也是本文为何一直强调 HTTP 协议的重要性。那么对于 test.jsp 这个小网页来说，基于缓存的方案提升了多少性能呢？ **AB测试**测试脚本: AB 是个很强大的工具，他提供了 -H 参数，允许测试人员手动添加 HTTP 请求头部，  D:\Program Files\Apache Software Foundation\Apache2.2\bin>ab -c 1000 -n 10000 -H  "If-Modified-SinceSun, 05 Jun 3910 00:00:00 GMT0 GMT" http://localhost/mobile/te  st.jsp 修改后的测试报告  **修改前的测试报告**   分别对比 Document Length、Requests per second 以及 Transfer rate 这三个指标。可以发现没使用缓存的 Document Length（下载内容的长度）是 362 字节，而使用了缓存的长度为 0。在吞吐率方面，使用缓存是不使用缓存的 3 倍左右。同时在传输率方面，缓存的传输率比没缓存的小。这些都是用到了客户端缓存的缘故。 |

|  |
| --- |
|  |