Web服务器的工作原理

2015/02/12 | 分类： [技术架构](http://www.importnew.com/cat/architecture) | [5 条评论](http://www.importnew.com/15020.html#comments) | 标签： [SERVLET](http://www.importnew.com/tag/servlet)

分享到：**47**

本文由 [ImportNew](http://www.importnew.com/) - [进林](http://www.importnew.com/author/8zjl8) 翻译自 [howtodoinjava](http://howtodoinjava.com/2013/10/03/a-birds-eye-view-on-how-web-servers-work/)。欢迎加入[翻译小组](http://www.jobbole.com/groups/6/)。转载请见文末要求。

**Web服务器工作原理概述**

很多时候我们都想知道，**web容器或web服务器（比如Tomcat或者jboss）是怎样工作的？**它们是怎样处理来自全世界的http请求的？它们在幕后做了什么动作？Java Servlet API（例如ServletContext，ServletRequest，ServletResponse和Session这些类）在其中扮演了什么角色？这些都是web应用开发者或者想成为web应用开发者的人必须要知道的重要问题或概念。在这篇文章里，我将会尽量给出以上某些问题的答案。请集中精神！

**文章章节：**

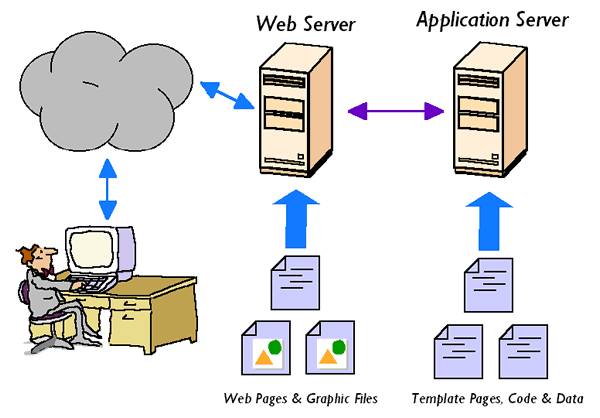
* 什么是web服务器、应用服务器和web容器？
* 什么是Servlet？他们有什么作用？
* 什么是ServletContext？它由谁创建？
* ServletRequest和ServletResponse从哪里进入生命周期？
* 如何管理Session？知道cookie吗？
* 如何确保线程安全？

**什么是web服务器，应用服务器和web容器？**

我先讨论**web服务器**和应用服务器。让我在用一句话大概讲讲：

“在过去它们是有区别的，但是这两个不同的分类慢慢地合并了，而如今在大多在情况下和使用中可以把它们看成一个整体。”

在[Mosaic浏览器](http://en.wikipedia.org/wiki/Mosaic_%28web_browser%29)（通常被认为是第一个图形化的web浏览器）和超链接内容的初期，演变出了“web服务器”的新概念，它通过[HTTP](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)协议来提供静态页面内容和图片服务。在那个时候，大多数内容都是静态的，并且HTTP 1.0只是一种传送文件的方式。但在不久后web服务器提供了[CGI](http://www.invir.com/int-prog-cgi.html)功能。这意味着我们可以为每个web请求启动一个进程来产生动态内容。现在，HTTP协议已经很成熟了并且web服务器变得更加复杂，拥有了像缓存、安全和session管理这些附加功能。随着技术的进一步成熟，我们从Kiva和[NetDynamics](http://en.wikipedia.org/wiki/NetDynamics_Application_Server)学会了公司专属的基于Java的服务器端技术。这些技术最终全都融入到我们今天依然在大多数应用开发里使用的JSP中。



以上是关于web服务器的。现在我们来讨论**应用服务器**。

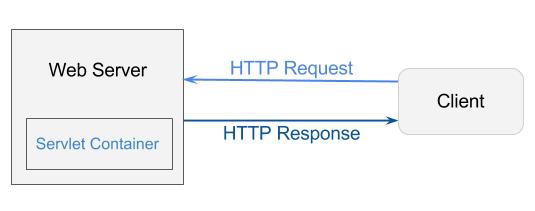
在同一时期，应用服务器已经存在并发展很长一段时间了。一些公司为Unix开发了[Tuxedo](http://en.wikipedia.org/wiki/Tuxedo_%28software%29)（面向事务的中间件）、**TopEnd**、**Encina**等产品，这些产品都是从类似IMS和CICS的主机应用管理和监控环境衍生而来的。大部分的这些产品都指定了“封闭的”产品专用通信协议来互连胖客户机(“fat” client)和服务器。在90年代，这些传统的应用服务器产品开始嵌入HTTP通信功能，刚开始要利用网关来实现。不久后它们之间的界线开始变得模糊了。

同时，web服务器越来越成熟，可以处理更高的负载、更多的并发和拥有更好的特性；应用服务器开始添加越来越多的基于HTTP的通信功能。所有的这些导致了web服务器与应用服务器的界线变得更窄了。

目前，“应用服务器”和“web服务器”之间的界线已经变得模糊不清了。但是人们还把这两个术语区分开来，作为强调使用。

当有人说到“web服务器”时，你通常要把它认为是以HTTP为核心、web UI为向导的应用。当有人说到“应用服务器”时，你可能想到“高负载、企业级特性、事务和队列、多通道通信（HTTP和更多的协议）”。但现在提供这些需求的基本上都是同一个产品。

以上就是关于web服务器和应用服务器的全部内容。现在我们来看看第三个术语，即web容器。



在Java方面，[web容器](http://en.wikipedia.org/wiki/Web_container)一般是指Servlet容器。Servlet容器是与Java Servlet交互的web容器的组件。web容器负责管理Servlet的生命周期、把URL映射到特定的Servlet、确保URL请求拥有正确的访问权限和更多类似的服务。综合来看，**Servlet容器就是用来运行你的Servlet和维护它的生命周期的运行环境。**

**什么是Servlet？他们有什么作用？**

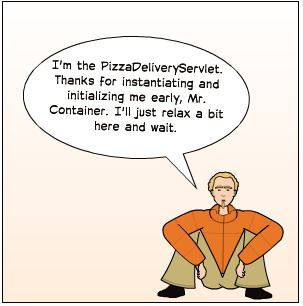
在Java里，**Servlet使你能够编写根据请求动态生成内容的服务端组件**。事实上，Servlet是一个在javax.servlet包里定义的接口。它为Servlet的生命周期声明了三个基本方法——init()、service()和destroy()。每个Servlet都要实现这些方法（在SDK里定义或者用户定义）并在它们的生命周期的特定时间由服务器来调用这些方法。

类加载器通过**懒加载（lazy-loading）或者预加载（eager loading）**自动地把Servlet类加载到容器里。每个请求都拥有自己的线程，而一个Servlet对象可以同时为多个线程服务。当Servlet对象不再被使用时，它就会被JVM当做垃圾回收掉。

懒加载的Servlet



预加载的Servlet



**什么是ServletContext？它由谁创建？**

当Servlet容器启动时，它会部署并加载所有的web应用。当web应用被加载时，Servlet容器会一次性为**每个应用创建Servlet上下文（ServletContext）**并把它保存在内存里。Servlet容器会处理web应用的web.xml文件，并且一次性创建在web.xml里定义的Servlet、Filter和Listener，同样也会把它们保存在内存里。当Servlet容器关闭时，它会卸载所有的web应用和ServletContext，所有的Servlet、Filter和Listner实例都会被销毁。

从Java文档可知，ServletContext定义了一组方法，**Servlet使用这些方法来与它的Servlet容器进行通信**。例如，用来获取文件的MIME类型、转发请求或者编写日志文件。**在web应用的部署文件（deployment descriptor）标明“分布式”的情况下**，web应用的每一个虚拟机都拥有一个上下文实例。在这种情况下，**不能把Servlet上下文当做共享全局信息的变量**（因为它的信息已经不具有全局性了）。可以使用外部资源来代替，比如数据库。

**ServletRequest和ServletResponse从哪里进入生命周期？**

Servlet容器包含在web服务器中，web服务器监听来自特定端口的HTTP请求，这个端口通常是80。当客户端（使用web浏览器的用户）发送一个HTTP请求时，**Servlet容器会创建新的HttpServletRequest和HttpServletResponse对象**，并且把它们传递给已经创建的Filter和URL模式与请求URL匹配的Servlet实例的方法，所有的这些都使用同一个线程。

request对象提供了获取HTTP请求的所有信息的入口，比如请求头和请求实体。response对象提供了控制和发送HTTP响应的便利方法，比如设置响应头和响应实体（通常是JSP生成的HTML内容）。当HTTP响应被提交并结束后，request和response对象都会被销毁。

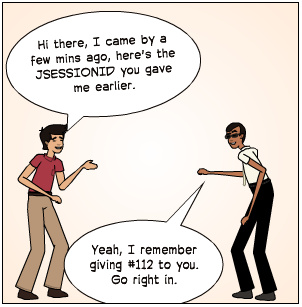
**如何管理Session？知道cookie吗？**

当客户端第一次访问web应用或者第一次使用request.getSession()获取HttpSession时，Servlet容器会创建Session，生成一个long类型的唯一ID（你可以使用session.getId()获取它）并把它保存在服务器的内存里。Servlet容器同样会在HTTP响应里设置一个Cookie，cookie的名是**JSESSIONID**并且cookie的值是session的唯一ID。

根据[HTTP cookie规范](http://www.faqs.org/rfcs/rfc2965.html)（正规的web浏览器和web服务器必须遵守的约定），在cookie的有效期间，客户端（web浏览器）之后的请求都要把这个cookie返回给服务器。Servlet容器会利用带有名为JSESSIONID的cookie检测每一个到来的HTTP请求头，并使用cookie的值从服务器内容里获取相关的HttpSession。

HttpSession会一直存活着，除非超过一段时间没使用。你可以在web.xml里设定这个时间段，**默认时间段是30分钟**。因此，如果客户端已经超过30分钟没有访问web应用的话，Servlet容器就会销毁Session。之后的每一个请求，即使带有特定的cookie，都再也不会访问到同一个Session了。servletcontainer会创建一个新的Session。

现有的Session



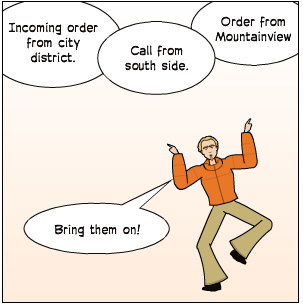
新的Session



另外，**在客户端的session cookie拥有一个默认的存活时间，这个时间与浏览器的运行时间相同**。因此，当用户关闭浏览器后（所有的标签或者窗口），客户端的Session就会被销毁。重新打开浏览器后，与之前的Session关联的cookie就再也不会被发送出去了。再次使用request.getSession()会返回一个全新的HttpSession并且使用一个全新的session ID来设置cookie。

**如何确保线程安全?**

你现在应该已经知道所有的请求都在共享Servlet和Filter。这是Java的一个很棒的特性，它是[多线程的](http://howtodoinjava.com/category/core-java/multi-threading/)并且不同的线程（即HTTP请求）可以使用同一个实例。否则，对每一个请求都重新创建一个实体会耗费很多的资源。



你同样要知道，你不应该使用Servlet或者Filter的实例变量来存放任何的请求或者会话范围内的数据。这些数据会被其他Session的所有请求共享。这是非线程安全的！下面的例子说明了这个问题：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class MyServlet extends HttpServlet  {      private Object thisIsNOTThreadSafe; //Don't to this        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException      {          Object thisIsThreadSafe;            thisIsNOTThreadSafe = request.getParameter("foo"); // BAD!! Shared among all requests!          thisIsThreadSafe = request.getParameter("foo"); // OK, this is thread safe.      }  } |

不要这样做，这会导致软件出bug。

所有的话题已经讲完了。敬请期待更多的文章。建议使用电子邮件订阅来获取文章更新的通知。

**学习愉快！！**