Tomcat7 新特性

作者：xixiaoming
原文链接：https://ld246.com/article/1500108364931
来源网站：链滴

<h2 id="TOMCAT-7新特性">TOMCAT 7 新特性</h2>
　　1. 使用随机数去防止跨站脚本攻击。
　　2. 改变了安全认证中的 jessionid 的机制，防止 session 攻击。
　　3. 内存泄露的侦测和防止
　　4. 在 war 文件外使用别名去存储静态内容。
<h2 id="TOMCAT-7的增强功能">TOMCAT 7 的增强功能</h2>
　　5. 对 Servlet 3.0,JSP 2.2 和 JSP-EL 2。2 的支持
　　6. 更容易将 Tomcat 内嵌到应用去中去，比如 JBoss
　　7. 异步日志记录
　　Tomcat 7 最显著的三个特征是 Servlet 3.0,内存检测泄露和增强的安全特性。
<h2 id="1--使用随机数去防止跨站请求伪造攻击-">1. 使用随机数去防止跨站请求伪造攻击。</h2>
　　CSRF(跨站请求伪造攻击,Cross Site Request forgery)让用户当进入一个可信任的网页时，强行执行恶意代码。
　　经典的防止 CSRF 攻击的方法是使用随机数的方式，利用随机或伪随机数嵌入到认证协议中以确保旧的不能在以后的重放攻击中被利用。
　　Tomcat 7 中有一个 servlet 过滤器，用于将随机数存储在用户每次请求处理后的 seesion 话中。这个随机数，必须作为每次请求中的一个参数。 Servlet 过滤器然后检查在请求中的这个随机是否与存储在用户 session 中的随机数是一样的。如果它们是相同的，该请求是判断来自指定的网站如果它们是不同的，该请求被认为是从其他网站发出并且会被拒绝。
　　CsrfPreventionFilter 过滤器可以在 web.xml 中进行配置，配置后，所有访问如 http://loca host:xxx/xx/xxx/的都必须带上参数，不带上参数的话会出现 403 禁止访问错误。
　　当然这种方法的缺点就是所有的链接都必须带上这个随机数。
<h2 id="2--改变了安全认证中的jessionid的机制-防止session攻击-">2. 改变了安全认证中的 jessio id 的机制，防止 session 攻击。</h2>
　　Session 劫持攻击通常是以下的情况：
　　1. 恶意攻击者先访问一个网页，由于 cookie 是以 jsession id 的方式存储在浏览器中的，即使击者不登陆，他可以伪造一个带有 jsession id 的地址，把它发给受害者，比如
<pre><code class="highlight-chroma">　　http://example.com/login?JESSIONID=qwerty
</code></pre>
　　2. 受害者点这个带有 jsessionid 的链接，提示输入验证信息之后就登陆系统。
　　3. 攻击者现在使用这个带 jsessionid 的链接，以受害者的身份登陆进系统了。
　　TOMCAT 7 对此的解决方案是一个补丁，它在验证后改变了 jsessionid，应用了 Tomcat 7 的这个补丁后：
　　• TOMCAT 默认情况下安全性不再变得脆弱，因为验证后会话发生了变化
　　• 如果用户改变了默认设置(比如应用程序不能处理变化了的 session id),风险也会降到最小因为在 Servlet 3 中，可以禁止在 url 中进行会话跟踪。
<h2 id="3--内存泄露的侦测和防止">3. 内存泄露的侦测和防止</h2>
　　开发者在部署他们写的程序到生产环境上时，经常会遇到 Permgen 错误：OutOfMemoryEr or。这是由于内存泄露而引起的。通常开发者是通过增大 permgen 内存的大小去解决或者就是重新动 tomcat。
　　TOMCAT 7 包含了一个新的特性，它通过把不能垃圾回收的引用对象移走的方法，能解决些 Permgen 内存泄露的问题。在生产环境中，最好的建议还是停掉 TOMCAT,然后清除 work 下面目录文件并且重新部署应用。
　　Mark Thomas 解析应用程序或者库程序在如下情况下会触发内存泄露：
　　• JDBC 驱动的注册
　　• 一些日志框架
　　• 在 ThreadLocals 中保存了对象但没有删除它们
　　• 启动了线程但没停止
　　而 Java API 存在内存泄漏的地方包括：
　　1. 使用 javax.imageio API (Google Web Toolkit 会用到)
　　2. 使用 java.beans.Introspector.flushCaches()
　　3. 使用 XML 解析器
　　4. 使用 RMI 远程方法调用

5.从 Jar 文件中读取资源
<h2 id="4--在war文件外使用别名去存储静态内容">4. 在 war 文件外使用别名去存储静态内容</h2

　　Web 应用程序需要静态资源文件，比如象 CSS，Javascript 和视频文件、图片文件等。通都把它们打包放在 war 文件中，这将增加了 WAR 文件的大小并且导致很多重复的加载静态资源。一比较好的解决方法是使用 Apache HTTP 服务器去管理这些静态文件资源。
　　Tomcat7 允许使用新的 aliases 属性，指出静态文件资源的位置，可以通过使用 Classloader getResourceAsStream('/static/...')或者在链接中嵌入的方法让 TOMCAT 去解析绝对路径，下面是个在 context.xml 中配置的例子：
<pre><code class="highlight-chroma"><?xml version="1.0" encoding="UTF-8"?>
<Context path= /tomcat7demo" aliases="/static=/home/avneet/temp/static">
</Context >
</code></pre>
　　假设/home/avneet/temp/static 这个文件夹存放有一张图片 bg.png，如果 war 文件以 tmcat7demo 的名字部署，那么可以通过以下三个方式去访问这张图片
　　1. 直接访问
　　http://localhost:8080/tomcat7demo/static/bg.png
　　2. 在 HTML 链接中访问：< img src="/tomcat7demo/static/bg.png" / >
　　3. 通过 JAVA 代码访问： ByteArrayInputStream bais = (ByteArrayInputStream)getServletCntext().getResourceAsStream("/static/bg.png");
　　使用 aliases 的好处是可以代替 Apache 的 httpd.conf 的设置，并且可以在 servlet 容器范内访问，并且不需要 Apache。
<h2 id="5--对Servlet-3-0-JSP-2-2和JSP-EL-2-2的支持">5. 对 Servlet 3.0,JSP 2.2 和 JSP-EL 2。2 的支持</h2>
　　Servlet 3 的增强特性有：
　　• 可以在 POJO 或者过滤器 filters 中使用 annotations 注释(在 web.xml 中不再需要再进行设置)
　　• 可以将 web.xml 分块进行管理了。也就是说，用户可以编写多个 xml 文件，而最终在 web.xml 中组装它们，这将大大降低 web.xml 的复杂性增强可读性。比如， struts.jar 和 spring-mvc.jar 每个都可以有一个 web-fragment.xml。开发者不再需要在 web.xml 中去配置它们了，在 web-fragmnt.xml 中的 jar 文件会自动加载，并且 struts/spring-mvc servlets 和 filters 也会自动装配设置。<b>
　　• 异步处理 web 的请求----这个特性在 tomcat 6 中已经有了，现在在 tomcat 7 中以 Servlet 3 标准规范化了，能让使用异步 I/O 的 web 应用程序可以移植到不同的 web 容器中。异步处理使用非塞 I/O，每次的 HTTP 连接都不需要对应一个线程。更少的线程可以为更多的连接提供服务。这对于要长时间计算处理才能返回结果的情景来说是很有用的，比如产生报表，Web Servce 调用等。
　　• 安全的增强---Servlet 3.0 现在使用 SSL 去加强了会话 session 的跟踪，代替了原来的 cookie URL 重写。
<h2 id="6--更容易将Tomcat内嵌到应用去中去">6. 更容易将 Tomcat 内嵌到应用去中去</h2>
　　Tomcat 7 现在可以嵌入到应用程序中去，并可以通过程序去动态设置和启动。象在 CATALNA_HOME/conf/server.xml 中的很多配置，现在都可以用程序动态去设置了。
<pre><code class="highlight-chroma">

final String CATALINA_HOME = "/home/avneet/work/temp/tomcat7demo/";

Tomcat tomcat = ew Tomcat();

tomcat.setBaseDir(CATALINA_HOME);

tomcat.setPort(8 80);

tomcat.addWebap("/tomcat7demo", CATALINA_HOME + "/webapps/tomcat7demo.war");

tomcat.start();

System.out.println "Started tomcat");
tomcat.getServer() await(); //Keeps Tomcat running until it is shut down
//Webapp tomcat demo accessible at http://localhost:8080/tomcat7demo/
</code></pre>
<h2 id="7--异步日志记录">7. 异步日志记录</h2>
　　TOMCAT 7 现在包括了一个异步日志记录器(AsyncFileHandler)。AsyncFileHandler 继承了 FileHandler 类并能代替 FileHandler。使用 AsyncFileHandler,时，只需要在 CATALINA_HOME/conf/logging.properties 中把 FileHandler 全部替换为 AsyncFileHandler 就可以了。要注意的是异步志不能跟 log4 一起工作。
　　当有日志发向 AsyncFileHandler 时，日志被加入到队列中(java.util.concurrent.LinkedBlockingDeque)并且方法调用的信息会马上返回不需要等待 I/O 写到磁盘中。当类加载器加载 AsyncFileHandler 时，会有一个单独的线程启动，这个线程会从队列中读取日志信息并且写到磁盘中去
　　这种方法的好处是如果 I/O 速度很慢(比如日志要保存在远端的设备上)时，记录日志的请求处理过程不会显得很慢。