



点点diandian.com

博客从此简单

论坛首页 → Java企业应用论坛 → Tomcat 源代码分析之ClassLoader

全部 Hibernate Spring Struts iBATIS 企业应用 Lucene SOA Java综合 设计模式 Tomcat OO JBoss



发表回复

最成熟稳定甘特图控件,支持Java和.Net

浏览 1907 次

主题: Tomcat 源代码分析之ClassLoader

精华帖 (0) :: 良好帖 (0) :: 新手帖 (0) :: 隐藏帖 (0)

作者	正文
<div>redhat</div> <div>等级: 初级会员</div> <div></div> <div>文章: 172</div> <div>积分: 90</div> <div>来自: ...</div> <div></div>	<div>发表时间: 前天 最后修改: 32 分钟前</div> <div><div>< ></div>猎头职位: 北京: ITeye网站诚聘产品交互设计</div> <div><div>Java</div><div>Tomcat</div><div>ClassLoader</div></div> <div>Tomcat 源代码分析之ClassLoader</div> <div>此系列文章皆为Tomcat 7.0代码代码分析。</div> <div>1. ClassLoader基础知识</div> <div>1.1. Parent-Child委托模型</div> <div>我们知道Java系统中，类加载器的默认加载方式是采用Parent-Child委托方式加载类的，即就是说，先尝试使用父类加载器加载类，如果没有找到，才自己加载该类，可以看到，这是一个递归的加载过程，核心代码大致如下：</div> <div>Java代码 ☆</div> <div><pre>1. protected Class<?> loadClass(String name, boolean resolve) 2. throws ClassNotFoundException 3. { 4. synchronized (getClassLoadingLock(name)) { 5. // First, check if the class has already been loaded 6. Class c = findLoadedClass(name); 7. if (c == null) { 8. 9. try { 10. if (parent != null) { 11. c = parent.loadClass(name, false); 12. } else { 13. c = findBootstrapClassOrNull(name); 14. } 15. } catch (ClassNotFoundException e) { 16. // ClassNotFoundException thrown if class not found 17. // from the non-null parent class loader 18. } 19. 20. if (c == null) { 21. // If still not found, then invoke findClass in order 22. // to find the class. 23. ... 24. c = findClass(name);</pre></div> <div><div>相关文章:</div><div><ul style="list-style-type: none">■ 重温java之classloader体系结构 (含hotswap)■ 自定义ClassLoader■ 自定义ClassLoader遇到的问题</div><div><div>推荐群组:</div><div>高级语言虚拟机</div><div>更多相关推荐</div></div></div>

```
25.         .....
26.     }
27. }
28.     .....
29.     return c;
30. }
31. }
```

很多ClassLoader的继承类都默认了这种加载方式，其中用途比较广泛的有URLClassLoader。

1.2. 类加载器运行时模型：

当一个JVM启动时，至少有三个ClassLoader会被启动：

1. Bootstrap 类加载器

加载Java核心包，它们被置放在(<JAVA_HOME>/lib目录下，这部分是JVM的一部分，往往使用native code完成

2. Extensions类加载器

加载Java扩展包，即位于<JAVA_HOME>/lib/ext下的包，这里要注意的是，有些JVM的这个加载器和Bootstrap类加载器是同一个加载器，而Sun是把二者分开的，其实现类为sun.misc.Launcher\$ExtClassLoader。

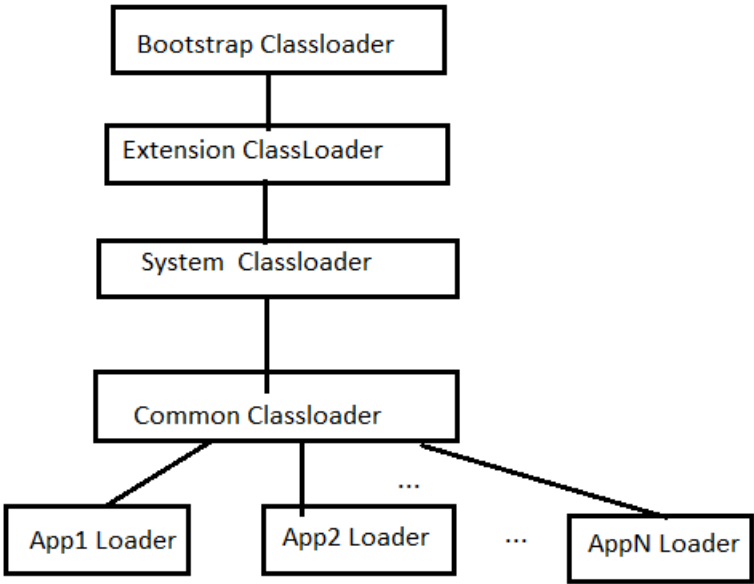
3. System类加载器

这个类加载器加载CLASSPATH下的类，Sun的默认实现是sun.misc.Launcher\$AppClassLoader。

这三者之间的父子关系是：Bootstrap类加载器是Extensions类加载器的父类加载器，而Extensions类加载器是System类加载器的父类加载器。

2. Tomcat Classloader模型

Tomcat 7.0的ClassLoader加载层次模型如下图所示：



这个模型和之前Tomcat 5.5之前有些不同，因为之前的除过Common类加载器之外，还有Catalina类加载器和Shared类加载器，这个只能导致更多的配置和概念，已经不再使用了，虽然还可以进行配置。

1. Common类加载器：加载\$CATALINA_HOME/lib和\$CATALINA_BASE/lib下的class文件和jar包以及旗下的资源文件，这些文件将会被所有Web应用共有，也会被Tomcat运行时用到。其配置在catalina.properties下，如果没有配置，则默认使用System类加载器，配置示例：

common.loader=\${catalina.home}/lib,\${catalina.home}/lib/*.jar

2. App类加载器：Web 应用的类加载器，Web应用下的资源文件，class文件盒jar包，按照Java Web规范的标准，它们位于Web应用的/WEB-INF/classes和/WEB-INF/lib文件夹下。

3. 类加载器的实现

Tomcat类加载器其实有三个类实现：StandardClassLoader，WebappClassLoader和JasperLoader，这三个类加载器都是URLClassLoader的子类。不同的是，StandardClassLoader并没有客户化URLClassLoader方法，即，它也是采用委托的方式加载类的。而其他都覆写了loadClass()方法。

- 1. StandardClassLoader类加载器会创建Common类加载器的实例。
- 2. WebappClassLoader为每个应用创建类加载器实例，对于加载Web应用的类时，我们并非推荐使用父子委托模型，但是必须保证以java.*和javax.*开头的包必须由System类加载器加载，即最终由Bootstrap加载器加载。

核心代码如下：

Java代码 

```
1. public Class loadClass(String name, boolean resolve)
2.     throws ClassNotFoundException {
3.
4.     .....
5.
6.     // (0) Check our previously loaded local class cache
7.     clazz = findLoadedClass0(name);
8.     if (clazz != null) {
9.         if (log.isDebugEnabled())
10.            log.debug(" Returning class from cache");
11.         if (resolve)
12.            resolveClass(clazz);
13.         return (clazz);
14.     }
15.
16.     // (0.1) Check our previously loaded class cache
17.     clazz = findLoadedClass(name);
18.     if (clazz != null) {
19.         if (log.isDebugEnabled())
20.            log.debug(" Returning class from cache");
21.         if (resolve)
22.            resolveClass(clazz);
23.         return (clazz);
24.     }
25.
26.     // (0.2) Try loading the class with the system class loader, to prevent
27.     //       the webapp from overriding J2SE classes
28.     //保证以java.*和javax.*开头的包必须由System类加载器加载
29.     try {
30.         clazz = system.loadClass(name);
31.         if (clazz != null) {
32.             if (resolve)
33.                 resolveClass(clazz);
34.             return (clazz);
35.         }
36.     } catch (ClassNotFoundException e) {
37.         // Ignore
38.     }
39.
40.     .....
41.
42.     boolean delegateLoad = delegate || filter(name);
43.
44.     // (1) Delegate to our parent if requested
45.     //Mbeans加载时，是否采用delegate方式
46.     if (delegateLoad) {
```

```


47.         if (log.isDebugEnabled())
48.             log.debug(" Delegating to parent classloader1 " + parent);
49.         ClassLoader loader = parent;
50.         if (loader == null)
51.             loader = system;
52.         try {
53.             clazz = loader.loadClass(name);
54.             if (clazz != null) {
55.                 if (log.isDebugEnabled())
56.                     log.debug(" Loading class from parent");
57.                 if (resolve)
58.                     resolveClass(clazz);
59.                 return (clazz);
60.             }
61.         } catch (ClassNotFoundException e) {
62.             ;
63.         }
64.     }
65.
66.     // (2) Search local repositories
67.     // 先本地加载，不委托父加载器加载
68.     if (log.isDebugEnabled())
69.         log.debug(" Searching local repositories");
70.     try {
71.         clazz = findClass(name);
72.         if (clazz != null) {
73.             if (log.isDebugEnabled())
74.                 log.debug(" Loading class from local repository");
75.             if (resolve)
76.                 resolveClass(clazz);
77.             return (clazz);
78.         }
79.     } catch (ClassNotFoundException e) {
80.         ;
81.     }
82.
83.     // 最后使用父加载器加载
84.     if (!delegateLoad) {
85.         if (log.isDebugEnabled())
86.             log.debug(" Delegating to parent classloader at end: " + parent);
87.         ClassLoader loader = parent;
88.         if (loader == null)
89.             loader = system;
90.         try {
91.             clazz = loader.loadClass(name);
92.             if (clazz != null) {
93.                 if (log.isDebugEnabled())
94.                     log.debug(" Loading class from parent");
95.                 if (resolve)
96.                     resolveClass(clazz);
97.                 return (clazz);
98.             }
99.         } catch (ClassNotFoundException e) {
100.            ;
101.        }
102.    }
103.
104.    throw new ClassNotFoundException(name);
105. }

```

可以看到，并非简单的父子委托方式加载类

3. JasperLoader是为了加载jsp编译成的servlet而创建的类加载器，它覆写了loadClass()方法，除过加载org.apache.jsp.*的文件外，其他的均使用父加载器加载。

核心代码如下：

Java代码 

```
1. public Class loadClass(final String name, boolean resolve)
2.     throws ClassNotFoundException {
3.
4.     Class clazz = null;
5.
6.     // (0) Check our previously loaded class cache
7.     clazz = findLoadedClass(name);
8.     if (clazz != null) {
9.         if (resolve)
10.            resolveClass(clazz);
11.         return (clazz);
12.     }
13.
14.     // (.5) Permission to access this class when using a SecurityManager
15.     .....
16.
17.     if( !name.startsWith(Constants.JSP_PACKAGE_NAME) ) {
18.         // Class is not in org.apache.jsp, therefore, have our
19.         // parent load it
20.         clazz = parent.loadClass(name);
21.         if( resolve )
22.            resolveClass(clazz);
23.         return clazz;
24.     }
25.     //所有的以org.apache.jsp.*开头的文件都由该类加载器加载。
26.     return findClass(name);
27. }
```

今天就讲到这里了。

[查看图片附件](#)

声明：ITeye 文章版权属于作者，受法律保护。没有作者书面许可不得转载。

推荐链接

- [中国建设银行诚聘英才！](#)
- [.Net程序员不用羡慕Android程序员，微软移动战役打响](#)
- [想进外企，出国，跳槽，找工作？英语不好，快来充电吧](#)
- [100万现金奖培训出来的程序员](#)

[返回顶楼](#)

 [主页](#)

 [资料](#)

 [短信](#)

 [留言](#)

 [关注](#)

arong

等级: 



文章: 74

积分: 112

 我现在离线

发表时间: 昨天

学习了，期待。。。。

我是yangguang
等级: 初级会员



文章: 3
积分: 30
我现在离线

发表时间: 23 小时前

好东西，学习中！

leonayx123
等级: 初级会员



性别:
文章: 69
积分: 30
来自: 杭州
我现在离线

发表时间: 17 小时前 最后修改: 17 小时前

少了两个貌似。

一个 `sharedClassLoader`用来加载app可见的包
一个 `serverClassLoader`用来加载tomcat服务必须的包
我没看过tomcat的源码，
但我感觉，`sharedClassLoader`应该是appClassloader的parent
不知道是不是

网上看到的资料，加载servlet的classloader（appclassloader?）。似乎比较特殊，
不是按标准的 先让parent加载，而是自己先加载。
有点模糊，有见解的话，麻烦给我解惑下。

redhat
等级: 初级会员



文章: 172
积分: 90
来自: ...
我现在离线

发表时间: 13 小时前

leonayx123 写道

少了两个貌似。

一个 `sharedClassLoader`用来加载app可见的包
一个 `serverClassLoader`用来加载tomcat服务必须的包
我没看过tomcat的源码，
但我感觉，`sharedClassLoader`应该是appClassloader的parent
不知道是不是

网上看到的资料，加载servlet的classloader（appclassloader?）。似乎比较特殊，
不是按标准的 先让parent加载，而是自己先加载。
有点模糊，有见解的话，麻烦给我解惑下。

redhat 写道

这个模型和之前Tomcat 5.5之前有些不同，因为之前的除过Common类加载器之外，还有Catalina类加载器和Shared类加载器，这个只能导致更多的配置和概念，已经不再使用了，虽然还可以进行配置。

这个已经提到了，是以前的，本文是按照7.0说明的。
Catalina类加载器和Shared类加载器不同，
Catalina类加载器加载Catalina servlet（tomcat）需要的jar包和class文件
Shared类加载器，加载所有web app共有的
common相当于Shared类加载器+Catalina类加载器。
但是这个如上述所说，很繁复，所以现在只有一种了。

返回顶楼

主页

资料

短信

留言

关注

回帖地址

0

0

请登录后投票

xpjsky
等级: 初级会员



性别:
文章: 2
积分: 30
来自: 杭州

我现在离线

发表时间: 2 小时前

系统类加载器SUN给出的实现应该是sun.misc.Launcher\$AppClassLoader，sun.misc.Launcher\$ExtClassLoader是扩展类加载器的实现

返回顶楼

主页

资料

短信

留言

关注

回帖地址

1

0

请登录后投票

redhat
等级: 初级会员



文章: 172
积分: 90
来自: ...

我现在离线

发表时间: 30 分钟前

xpjsky 写道

系统类加载器SUN给出的实现应该是sun.misc.Launcher\$AppClassLoader，sun.misc.Launcher\$ExtClassLoader是扩展类加载器的实现

确实写错了，把ext和sys写成同一个了，谢谢指正！

返回顶楼

主页

资料

短信

留言

关注

回帖地址

0

0

请登录后投票

发表回复

[论坛首页](#) → [Java企业应用版](#)

跳转论坛:

Java

- 陕西: [ThoughtWorks诚聘西安Senior Java Applicatio](#)
- 北京: [杰嘉迪诚聘Java程序员](#)
- 上海: [用友汽车软件诚聘软件研发工程师（实习岗位）](#)
- 北京: [美团诚聘资深研发工程师（Java）](#)

 [上海: 用友汽车软件诚聘高级Java程序员](#)

 [北京: ITeye官方招聘和猎头招聘ITeye网站招聘Java](#)

[广告服务](#) | [ITeye黑板报](#) | [联系我们](#) | [友情链接](#)

© 2003-2011 ITeye.com. [[京ICP证110151号](#) 京公网安备110105010620]
百联优力(北京)投资有限公司 版权所有
上海炯耐计算机软件有限公司 提供商务支持