

## 24 | Context容器（上）：Tomcat如何打破双亲委托机制？

2019-07-04 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)



**讲述：李号双**

时长 08:57 大小 8.20M



相信我们平时在工作中都遇到过 `ClassNotFoundException` 异常，这个异常表示 JVM 在尝试加载某个类的时候失败了。想要解决这个问题，首先你需要知道什么是类加载，JVM 是如何加载类的，以及为什么会出现 `ClassNotFoundException` 异常？弄懂

上面这些问题之后，我们接着要思考 Tomcat 作为 Web 容器，它是如何加载和管理 Web 应用下的 Servlet 呢？

Tomcat 正是通过 Context 组件来加载管理 Web 应用的，所以今天我会详细分析 Tomcat 的类加载机制。但在这之前，我们有必要预习一下 JVM 的类加载机制，我会先回答一下一开始抛出来的问题，接着再谈谈 Tomcat 的类加载器如何打破 Java 的双亲委托机制。

## JVM 的类加载器

Java 的类加载，就是把字节码格式 “.class” 文件加载到 JVM 的**方法区**，并在 JVM 的**堆区**建立一个 `java.lang.Class` 对象的实例，用来封装 Java 类相关的数据和方法。那 Class 对象又是什么呢？你可以把它理解成业务类的模板，JVM 根据这个模板来创建具体业务类对象实例。


JVM 并不是在启动时就把所有的 “.class” 文件都加载一遍，而是程序在运行过程中用到了这个类才去加载。JVM 类加载是由类加载器来完成的，JDK 提供一个抽象类 `ClassLoader`，这个抽象类中定义了三个关键方法，理解清楚它们的作用和关系非常重要。

```

1 public abstract class ClassLoader {
2
3     // 每个类加载器都有个父加载器
4     private final ClassLoader parent;
5
6     public Class<?> loadClass(String name) {
7
8         // 查找一下这个类是不是已经加载过了
9         Class<?> c = findLoadedClass(name);
10
11         // 如果没有加载过
12         if( c == null ){
13             // 先委托给父加载器去加载，注意这是个递归调用
14             if (parent != null) {
15                 c = parent.loadClass(name);
16             }else {
17                 // 如果父加载器为空，查找 Bootstrap 加载器是
18                 c = findBootstrapClassOrNull(name);
19             }
20         }
21         // 如果父加载器没加载成功，调用自己的 findClass 去
22         if (c == null) {
23             c = findClass(name);
24         }
25
26         return c;
27     }
28
29     protected Class<?> findClass(String name){
30         //1. 根据传入的类名 name，到在特定目录下去寻找类文件
31         ...
32
33         //2. 调用 defineClass 将字节数组转成 Class 对象
34         return defineClass(buf, off, len);
35     }

```

```
36
37     // 将字节码数组解析成一个 Class 对象，用 native 方法实
38     protected final Class<?> defineClass(byte[] b, int
39         ...
40     }
41 }
```



从上面的代码我们可以得到几个关键信息：

JVM 的类加载器是分层次的，它们有父子关系，每个类加载器都持有一个 parent 字段，指向父加载器。

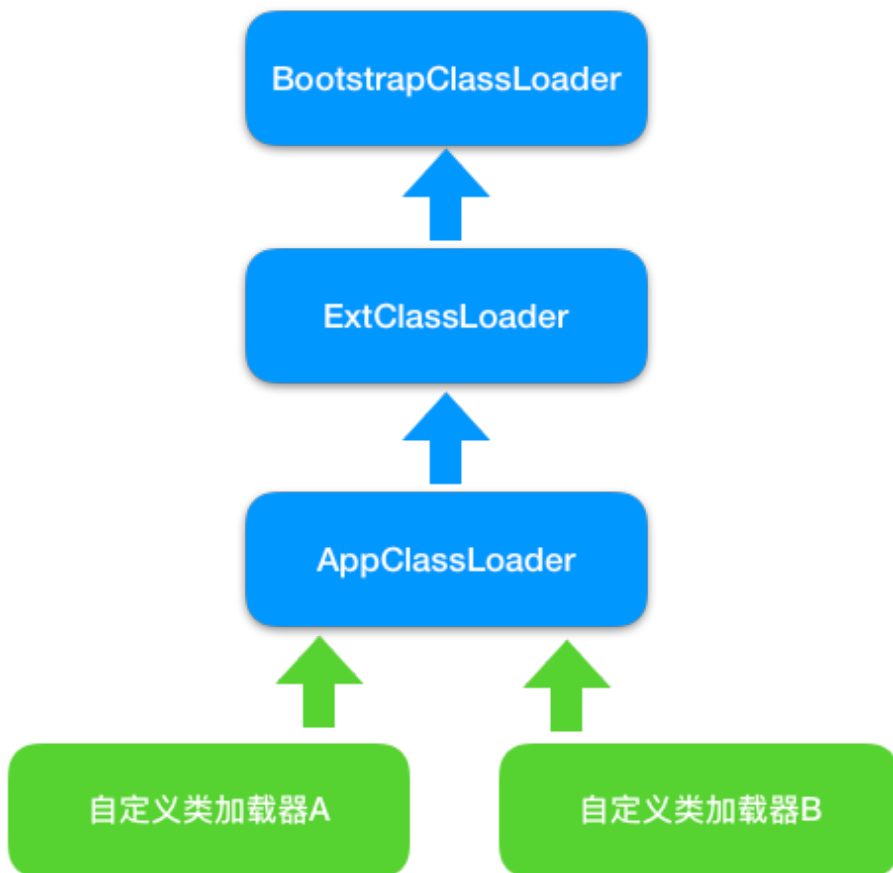
defineClass 是个工具方法，它的职责是调用 native 方法把 Java 类的字节码解析成一个 Class 对象，所谓的 native 方法就是由 C 语言实现的方法，Java 通过 JNI 机制调用。

findClass 方法的主要职责就是找到 “.class” 文件，可能来自文件系统或者网络，找到后把 “.class” 文件读到内存得到字节码数组，然后调用 defineClass 方法得到 Class 对象。

loadClass 是个 public 方法，说明它才是对外提供服务的接口，具体实现也比较清晰：首先检查这个类是不是已经被加载过了，如果加载过了直接返回，否则交给父加载器去加载。请你注意，这是一个递归调用，也就是说子加

载器持有父加载器的引用，当一个类加载器需要加载一个 Java 类时，会先委托父加载器去加载，然后父加载器在自己的加载路径中搜索 Java 类，当父加载器在自己的加载范围内找不到时，才会交还给子加载器加载，这就是双亲委托机制。

JDK 中有哪些默认的分类加载器？它们的本质区别是什么？为什么需要双亲委托机制？JDK 中有 3 个分类加载器，另外你也可以自定义分类加载器，它们的关系如下图所示。



**BootstrapClassLoader** 是启动类加载器，由 C 语言实现，用来加载 JVM 启动时所需要的核心类，比如 `rt.jar`、`resources.jar` 等。

**ExtClassLoader** 是扩展类加载器，用来加载 `\jre\lib\ext` 目录下 JAR 包。

**AppClassLoader** 是系统类加载器，用来加载 `classpath` 下的类，应用程序默认用它来加载类。

自定义类加载器，用来加载自定义路径下的类。

这些类加载器的工作原理是一样的，区别是它们的加载路径不同，也就是说 `findClass` 这个方法查找的路径不同。双亲委托机制是为了保证一个 Java 类在 JVM 中是唯一的，假如你不小心写了一个与 JRE 核心类同名的类，比如 `Object` 类，双亲委托机制能保证加载的是 JRE 里的那个 `Object` 类，而不是你写的 `Object` 类。这是因为 `AppClassLoader` 在加载你的 `Object` 类时，会委托给 `ExtClassLoader` 去加载，而 `ExtClassLoader` 又会委托给 `BootstrapClassLoader`，`BootstrapClassLoader` 发现自己已经加载过了 `Object` 类，会直接返回，不会去加载你写的 `Object` 类。


这里请你注意，类加载器的父子关系不是通过继承来实现的，比如 `AppClassLoader` 并不是 `ExtClassLoader` 的子类，而是说 `AppClassLoader` 的 `parent` 成员变量指向 `ExtClassLoader` 对象。同样的道理，如果你要自定义类加载器，不去继承 `AppClassLoader`，而是继承 `ClassLoader` 抽象类，再重写 `findClass` 和 `loadClass` 方法即可，Tomcat 就是通过自定义类加载器来实现自己的类加载逻辑。不知道你发现没有，如果你要打破双亲委托机制，就需要重写 `loadClass` 方法，因为 `loadClass` 的默认实现就是双亲委托机制。

# Tomcat 的类加载器

Tomcat 的自定义类加载器 `WebAppClassLoader` 打破了双亲委托机制，它**首先自己尝试去加载某个类，如果找不到再代理给父类加载器**，其目的是优先加载 Web 应用自己定义的类。具体实现就是重写 `ClassLoader` 的两个方法：`findClass` 和 `loadClass`。

## findClass 方法

我们先来看看 `findClass` 方法的实现，为了方便理解和阅读，我去掉了一些细节：

 复制代码

```
1 public Class<?> findClass(String name) throws ClassNotFoundException
2     ...
3
4     Class<?> clazz = null;
5     try {
6         //1. 先在 Web 应用目录下查找类
7         clazz = findClassInternal(name);
8     } catch (RuntimeException e) {
9         throw e;
10    }
11
12    if (clazz == null) {
13        try {
14            //2. 如果在本地目录没有找到，交给父加载器去查找
15            clazz = super.findClass(name);
```



```
16     } catch (RuntimeException e) {
17         throw e;
18     }
19
20     //3. 如果父类也没找到，抛出 ClassNotFoundException
21     if (clazz == null) {
22         throw new ClassNotFoundException(name);
23     }
24
25     return clazz;
26 }
```



在 findClass 方法里，主要有三个步骤：

1. 先在 Web 应用本地目录下查找要加载的类。
2. 如果没有找到，交给父加载器去查找，它的父加载器就是上面提到的系统类加载器 AppClassLoader。
3. 如何父加载器也没找到这个类，抛出 ClassNotFoundException 异常。

## loadClass 方法

接着我们再来看 Tomcat 类加载器的 loadClass 方法的实现，同样我也去掉了一些细节：

```
1 public Class<?> loadClass(String name, boolean resolve)
2
3     synchronized (getClassLoadingLock(name)) {
4
5         Class<?> clazz = null;
6
7         //1. 先在本地 cache 查找该类是否已经加载过
8         clazz = findLoadedClass0(name);
9         if (clazz != null) {
10             if (resolve)
11                 resolveClass(clazz);
12             return clazz;
13         }
14
15         //2. 从系统类加载器的 cache 中查找是否加载过
16         clazz = findLoadedClass(name);
17         if (clazz != null) {
18             if (resolve)
19                 resolveClass(clazz);
20             return clazz;
21         }
22
23         // 3. 尝试用 ExtClassLoader 类加载器类加载, 为什么
24         ClassLoader javaseLoader = getJavaseClassLoader
25         try {
26             clazz = javaseLoader.loadClass(name);
27             if (clazz != null) {
28                 if (resolve)
29                     resolveClass(clazz);
30                 return clazz;
31             }
32         } catch (ClassNotFoundException e) {
33             // Ignore
34         }
35
```

```

36         // 4. 尝试在本地目录搜索 class 并加载
37         try {
38             clazz = findClass(name);
39             if (clazz != null) {
40                 if (resolve)
41                     resolveClass(clazz);
42                 return clazz;
43             }
44         } catch (ClassNotFoundException e) {
45             // Ignore
46         }
47
48         // 5. 尝试用系统类加载器 (也就是 AppClassLoader)
49         try {
50             clazz = Class.forName(name, false, parent);
51             if (clazz != null) {
52                 if (resolve)
53                     resolveClass(clazz);
54                 return clazz;
55             }
56         } catch (ClassNotFoundException e) {
57             // Ignore
58         }
59     }
60
61     //6. 上述过程都加载失败, 抛出异常
62     throw new ClassNotFoundException(name);
63 }

```



loadClass 方法稍微复杂一点, 主要有六个步骤:

1. 先在本地 Cache 查找该类是否已经加载过，也就是说 Tomcat 的类加载器是否已经加载过这个类。
2. 如果 Tomcat 类加载器没有加载过这个类，再看看系统类加载器是否加载过。
3. 如果都没有，就让 **ExtClassLoader** 去加载，这一步比较关键，目的**防止 Web 应用自己的类覆盖 JRE 的核心类**。因为 Tomcat 需要打破双亲委托机制，假如 Web 应用里自定义了一个叫 Object 的类，如果先加载这个 Object 类，就会覆盖 JRE 里面的那个 Object 类，这就是为什么 Tomcat 的类加载器会优先尝试用 ExtClassLoader 去加载，因为 ExtClassLoader 会委托给 BootstrapClassLoader 去加载，BootstrapClassLoader 发现自己已经加载了 Object 类，直接返回给 Tomcat 的类加载器，这样 Tomcat 的类加载器就不会去加载 Web 应用下的 Object 类了，也就避免了覆盖 JRE 核心类的问题。
4. 如果 ExtClassLoader 加载器加载失败，也就是说 JRE 核心类中没有这类，那么就在本地 Web 应用目录下查找并加载。
5. 如果本地目录下没有这个类，说明不是 Web 应用自己定义的类，那么由系统类加载器去加载。这里请你注意，Web 应用是通过 `Class.forName` 调用交给系统类加载器的，因为 `Class.forName` 的默认加载器就是系统类加载器。

6. 如果上述加载过程全部失败，抛出 `ClassNotFoundException` 异常。

从上面的过程我们可以看到，Tomcat 的类加载器打破了双亲委托机制，没有一上来就直接委托给父加载器，而是先在本地目录下加载，为了避免本地目录下的类覆盖 JRE 的核心类，先尝试用 JVM 扩展类加载器 `ExtClassLoader` 去加载。那为什么不先用系统类加载器 `AppClassLoader` 去加载？很显然，如果是这样的话，那就变成双亲委托机制了，这就是 Tomcat 类加载器的巧妙之处。

## 本期精华

今天我介绍了 JVM 的类加载器原理和源码剖析，以及 Tomcat 的类加载器是如何打破双亲委托机制的，目的是为了优先加载 Web 应用目录下的类，然后再加载其他目录下的类，这也是 Servlet 规范的推荐做法。

要打破双亲委托机制，需要继承 `ClassLoader` 抽象类，并且需要重写它的 `loadClass` 方法，因为 `ClassLoader` 的默认实现就是双亲委托。

## 课后思考

如果你并不想打破双亲委托机制，但是又想定义自己的类加载器来加载特定目录下的类，你需要重写 findClass 和 loadClass 方法中的哪一个？还是两个都要重写？

不知道今天的内容你消化得如何？如果还有疑问，请大胆的在留言区提问，也欢迎你把你的课后思考和心得记录下来，与我和其他同学一起讨论。如果你觉得今天有所收获，欢迎你把它分享给你的朋友。

 极客时间

## 深入拆解 Tomcat & Jetty

从源码角度深度探索 Java 中间件

李号双  
eBay 技术主管



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 23 | Host容器：Tomcat如何实现热部署和热加载？

下一篇 24 | Jetty的线程模型与线程池管理

## 精选留言 (22)

写留言

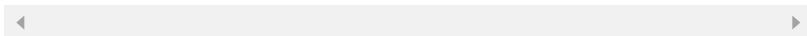


Liam

2019-07-04

老师能讲下什么是上下文加载器吗，什么情况下会用到它？这个和双亲委派有关吗

作者回复: 线程上下文加载器其实是线程私有数据，跟线程绑定的属性



1

3



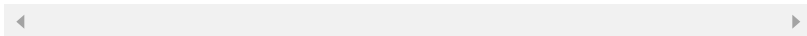
QQ怪

2019-07-05

双亲委派模型其实不是叫单亲委派更好？

展开

作者回复: 还真是



1

1

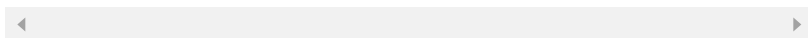


妥协

2019-07-04

看到过这样一句话，一直没想明白："类的唯一性由类加载器实例和类的全名一同确定的，即使同一串字节流由不同的类加载器加载，也是不同的实例"，每个类加载器加载前都会判断是否已经加载过，同名的类判断加载过了，不是不会在加载吗？

作者回复: 每个类记载“实例”只判断自己是否加载过



1

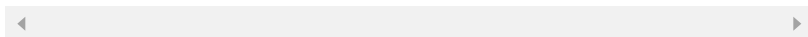


nightmare

2019-07-04

tomcat的类加载机制老师剖析的很透彻，先扩展类加载器加载，这样避免自己覆盖JRE中的类然后再自定义的加载器加载，最后应用加载器加载，有一个疑问，就是比如我一个tomcat部署了多个web应用，如果都有spring的jar包，由于自定义的类加载器先加载spring的jar包，这...  
展开 ∨

作者回复: 你说的很对，Spring的类应该是由sharedclassloader来加载，所以不应该把Spring的包放到Web应用的路径下，应该放到Tomcat指定的共享目录下



1





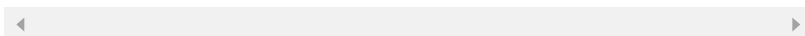
新世界

2019-07-04

由于沿用双亲委派重写findClass即可，找不到最后到固定目录下查找，不需要重写loadClass，还有一点不明白，tomcat为什么要打破双亲委派定义自己的classloader，不定义不行吗？

展开 ∨

作者回复: Servlet规定这样做的，优先加载web应用目录下的类，只要这个类不覆盖jre核心类



👍 1



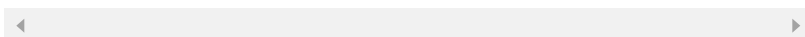
Mq

2019-07-04

李老师，为什么要打破双亲委托

展开 ∨

作者回复: 文中其实有解释，servlet规范建议这么做



👍 1



Cy190622

2019-07-06

老师，您好。麻烦请教几个问题，希望您有时间解答，谢谢：

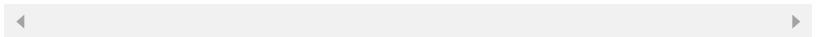
1.双亲委托机制在ClassLoader类中，我看到仅显示了子加载器到父加载器的过程，没有自加载器加载的过程。具体子加载器加载过程是代码那部分体现。...

展开▼

作者回复: 1，这两行代码接下来就是自己加载的代码：

```
c = findBootstrapClassOrNull(name);
    }
    }
    // 如果父加载器没加载成功，调用自己的 findClass
去加载
    if (c == null) {
        c = findClass(name);
    }
```

2，其实就是AppClassLoader，有人叫它系统类加载器



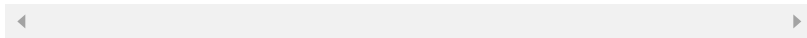
罗乾林

2019-07-06

我想打破双亲委托机制，能保证不同版本的类共存，就像一个tomcat下多个工程，使用了不同版本的spring，各加载各的互不影响。如果不打破双亲委托机制，都交由

AppClassLoader去加载，那么相同包名相同类名的类就被判定已经加载过了，达不到加载不同版本的功能。由...  
展开 ∨

作者回复: 对的，下一篇就说隔离

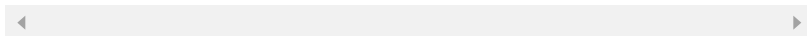


**Mr.差不多**

2019-07-05

双亲委派规则是 当父加载器找不到此文件时才交给子加载器去加载。那么我觉得Tomcat重写loadClass方法其实也是这个逻辑。假设现在有一个类是需要在WebAppClassLoader加载的，那么它会先查询是否在AppClassLoader加载过，如果没有那么查看是否在...  
展开 ∨

作者回复: WebAppClassLoader不会首先委托给AppClassLoader去加载，而是ExtClassLoader。这是根本区别



**-W.LI-**

2019-07-05

老师好!Tomcat这边就是跳过了一个 AppClassLoader 加载器打破了双亲委托模型。

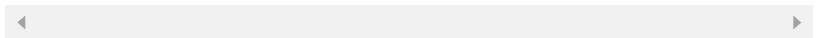
如果我需要加载一个系统类加载器加载的类是不是就 classNotFound了。

Tomcat打破双亲委派模型是由于Servlet规范，这样的...

展开 ∨

作者回复: 1, Tomcat加载器的最后一步是交给父加载器，会传导到AppClassLoader。

2, 建议看源码 😊



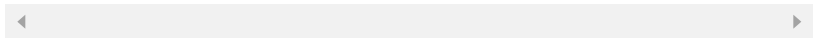
**nightmare**

2019-07-04

我明白了，比如可以把多个项目共享的jar包放到 \${CATALINA\_HOME}/shared目录下，让 sharedclassloader来加载，并且是所有context的web应用共享的，而都有的放在web路径下，先让扩展类加载器加载，避免覆盖jre中的类，再让自定义的web加载器来...

展开 ∨

作者回复: 👍





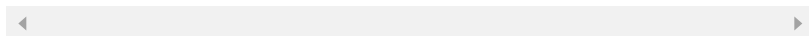
妥协

2019-07-04

老师，为什么Tomcat的类加载器的findclass函数在本地路径找不到后，要交给父类加载器去查找，如果查找到了，那不是由Tomcat类加载器加载了嘛？而不是父类加载器加载了

展开 ∨

作者回复: 这就是目的，优先自己来加载



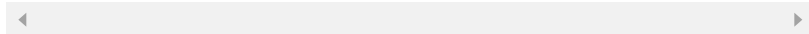
-W.LI-

2019-07-04

李老师好!问个线上问题。服务好像存在内存泄露，可是本地和测试环境试验下来FGC能正常回收内存(本地内存只给了1G)，线上本来是4G内存，后来升级到8G。每天都会被吃掉300M内存，一个多星期一共发生了6次FGC可是从监控屏幕看，没有看见内存释放的痕迹，ygc大...

展开 ∨

作者回复: jmap -dump:live,format=b,file=heap-dump.bin <pid> 生成heapdump，然后用mat分析





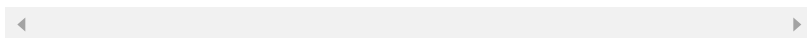
WL

2019-07-04

想问一下老师tomcat为什么采用"首先自己尝试去加载某个类，如果找不到再代理给父类加载"，这种方式呢，我不是很理解. 还有在Tomcat的类加载器的loadClass()方法，会先调用ExtClassLoader加载类，然后才调用findClass(name)，这是不是与上面的"首先自己尝试去...

展开 ∨

作者回复: 不矛盾的，本质还是自己先加载，这里先委托给Ext加载器纯粹为了避免类覆盖



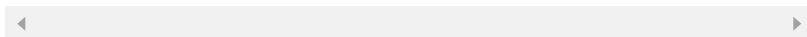
despacito

2019-07-04

loadClass方法分析的六个步骤中的第二步“再看看系统类加载器是否加载”，但是下面总结的时候说用"ExtClassLoader而不先用系统类加载器"，这是不是自相矛盾了？

展开 ∨

作者回复: 不矛盾的，ExtClassLoader是系统类加载器的父加载器。





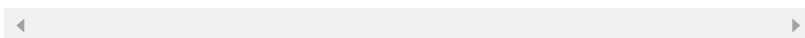
**despacito**

2019-07-04

是不是只要自己写的类或是引用的jar里面的类只要有类名和bootstrap 加载路径下相同的类名，都不会加载成功，但是如果不是bootstrap加载路径下的类，比如appclass loader是可以加载成功的？

展开 ∨

作者回复: 对的

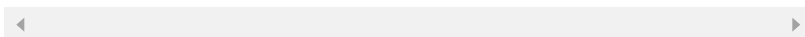


**despacito**

2019-07-04

自己写的Object 包名会不一样，加载的时候不会根据类的全路径名而只是通过简单的类名加载吗？

作者回复: 全路径名



**林子恒#Ralegh**

2019-07-04

老师您好，请教下，tomcat的loadclass方法里，本地和系统的cache是什么时候更新的呢？先加载cache的好处是？

作者回复: cache的作用都是为了性能和效率

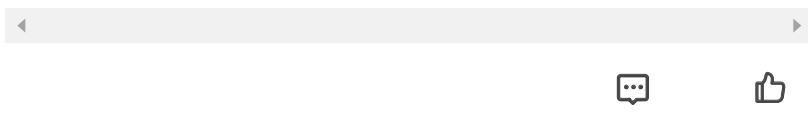


**despacito**

2019-07-04

自己写的Object的类包名称不一样，委托时根据名称展开 ∨

作者回复: 全路径名



**强哥**

2019-07-04

每篇文章最后的总结，若能概括出这么做的意图及优点，这样对读者来说收益更大。

作者回复: Servlet规范建议，全路径类名与系统类同名的话，优先加载web应用自己定义的类。





