# 1、tomcat 原理

1. 打破双亲委派

以Tomcat类加载为例，Tomcat 如果使用默认的双亲委派类加载机制行不行？

我们思考一下：Tomcat是个web容器， 那么它要解决什么问题：

1. 一个web容器可能需要部署两个应用程序，不同的应用程序可能会依赖同一个第三方类库的不同版本，不能要求同一个类库在同一个服务器只有一份，因此要保证每个应用程序的类库都是独立的，保证相互隔离。

2. 部署在同一个web容器中相同的类库相同的版本可以共享。否则，如果服务器有10个应用程序，那么要有10份相同的类库加载进虚拟机。

3. web容器也有自己依赖的类库，不能与应用程序的类库混淆。基于安全考虑，应该让容器的类库和程序的类库隔离开来。

4. web容器要支持jsp的修改，我们知道，jsp 文件最终也是要编译成class文件才能在虚拟机中运行，但程序运行后修改jsp已经是司空见惯的事情， web容器需要支持 jsp 修改后不用重启。

再看看我们的问题：Tomcat 如果使用默认的双亲委派类加载机制行不行？

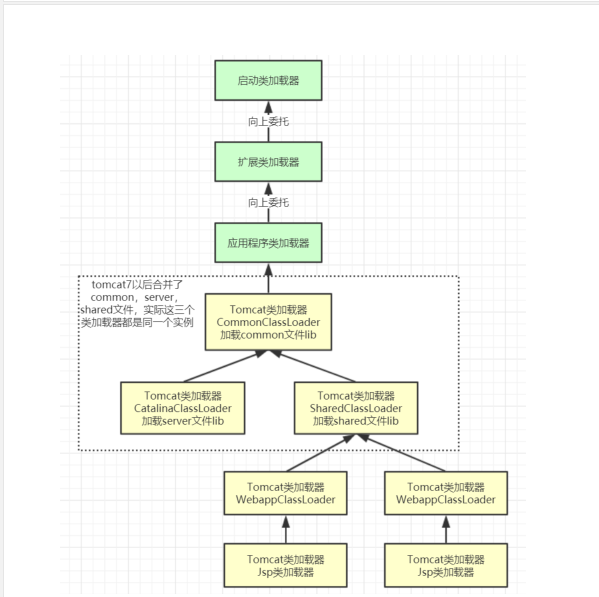
答案是不行的。为什么？

第一个问题，如果使用默认的类加载器机制，那么是无法加载两个相同类库的不同版本的，默认的类加器是不管你是什么版本的，只在乎你的全限定类名，并且只有一份。

第二个问题，默认的类加载器是能够实现的，因为他的职责就是保证唯一性。

第三个问题和第一个问题一样。

我们再看第四个问题，我们想我们要怎么实现jsp文件的热加载，jsp 文件其实也就是class文件，那么如果修改了，但类名还是一样，类加载器会直接取方法区中已经存在的，修改后的jsp是不会重新加载的。那么怎么办呢？我们可以直接卸载掉这jsp文件的类加载器，所以你应该想到了，每个jsp文件对应一个唯一的类加载器，当一个jsp文件修改了，就直接卸载这个jsp类加载器。重新创建类加载器，重新加载jsp文件。



tomcat的几个主要类加载器：

commonLoader：Tomcat最基本的类加载器，加载路径中的class可以被

Tomcat容器本身以及各个Webapp访问；

catalinaLoader：Tomcat容器私有的类加载器，加载路径中的class对于

Webapp不可见；

sharedLoader：各个Webapp共享的类加载器，加载路径中的class对于所有

Webapp可见，但是对于Tomcat容器不可见；

WebappClassLoader：各个Webapp私有的类加载器，加载路径中的class只对

当前Webapp可见；

从图中的委派关系中可以看出：

**CommonClassLoader**能加载的类都可以被CatalinaClassLoader和SharedClassLoader使用，从而实现了公有类库的共用，而CatalinaClassLoader和SharedClassLoader自己能加载的类则与对方相互隔离。

**WebAppClassLoader**可以使用SharedClassLoader加载到的类，但各个WebAppClassLoader实例之间相互隔离。

**JasperLoader**的加载范围仅仅是这个JSP文件所编译出来的那一个.Class文件，它出现的目的就是为了被丢弃：当Web容器检测到JSP文件被修改时，会替换掉目前的JasperLoader的实例，并通过再建立一个新的Jsp类加载器来实现JSP文件的热加载功能。

tomcat 这种类加载机制违背了java 推荐的双亲委派模型了吗？答案是：违背了。

我们前面说过，双亲委派机制要求除了顶层的启动类加载器之外，其余的类加载器都应当由自己的父类加载器加载。很显然，tomcat 不是这样实现，tomcat 为了实现隔离性，没有遵守这个约定，每个webappClassLoader加载自己的目录下的class文件，不会传递给父类加载器，打破了双亲委派机制。

# 2、tomcat使用