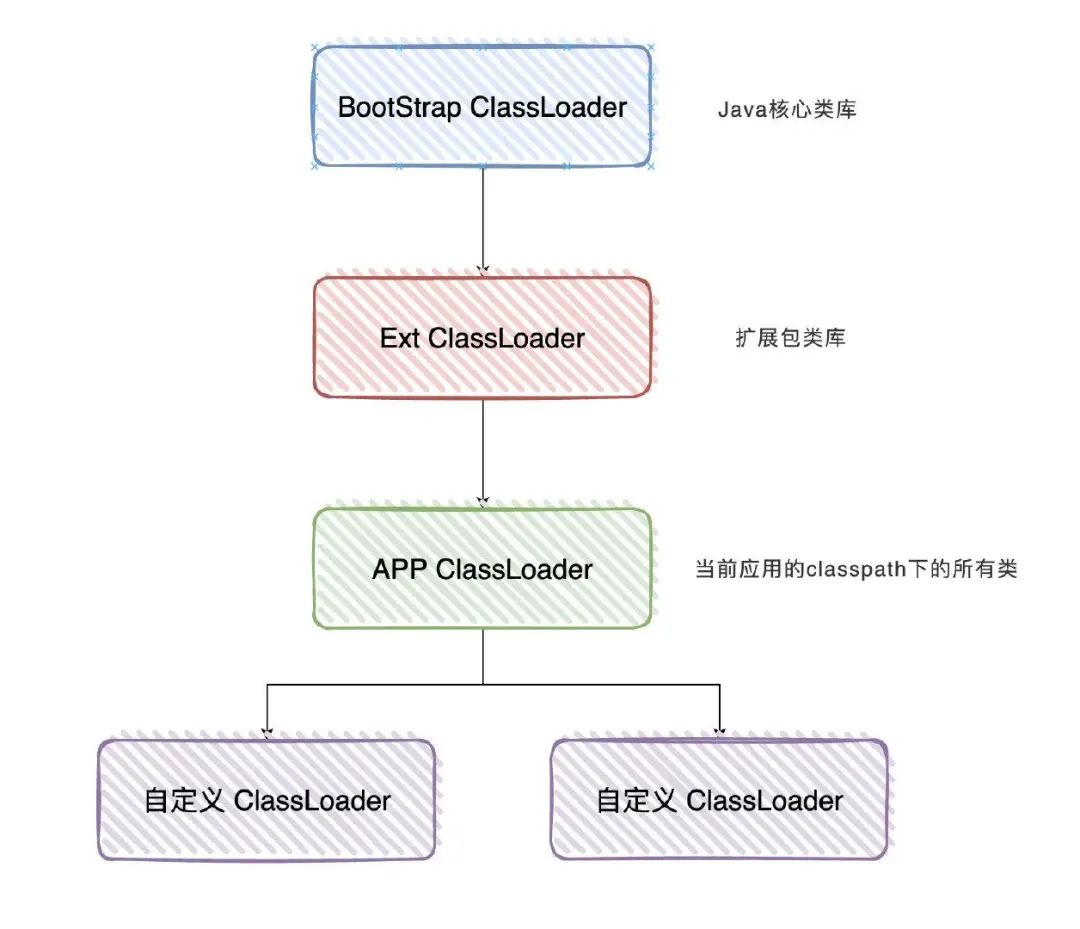
**为什么 Tomcat 会破坏双亲委派机制？**

<https://mp.weixin.qq.com/s/8Yk0R1iHtaPBL6sa_MDgig>

class文件是通过「类加载器」装载至JVM中的

为了防止内存中存在多份同样的字节码，使用了双亲委派机制（它不会自己去尝试加载类，而是把请求委托给父加载器去完成，依次向上）

JDK 中的本地方法类一般由根加载器（Bootstrp loader）装载，JDK 中内部实现的扩展类一般由扩展加载器（ExtClassLoader ）实现装载，而程序中的类文件则由系统加载器（AppClassLoader ）实现装载。



**打破双亲委派机制是什么意思？**

只要我加载类的时候，不是从APPClassLoader->Ext ClassLoader->BootStrap ClassLoader 这个顺序找，那就算是打破了。

因为加载class核心的方法在ClassLoader类的loadClass方法上（双亲委派机制的核心实现）。

那只要我自定义个ClassLoader，重写loadClass方法（不依照往上开始寻找类加载器），那就算是打破双亲委派机制了。

**那你知道有哪个场景破坏了双亲委派机制吗？**

最明显的就Tomcat啊

在初学时部署项目，我们是把war包放到tomcat的webapp下，这意味着一个tomcat可以运行多个Web应用程序。

那假设我现在有两个Web应用程序，它们都有一个类，叫做User，并且它们的类全限定名都一样，比如都是com.yyy.User。但是他们的具体实现是不一样的。

那么Tomcat是如何保证它们是不会冲突的呢？

答案就是，Tomcat给每个 Web 应用创建一个类加载器实例（WebAppClassLoader），该加载器重写了loadClass方法，优先加载当前应用目录下的类，如果当前找不到了，才一层一层往上找。

那这样就做到了Web应用层级的隔离



**Tomcat还有别的类加载器吗？**

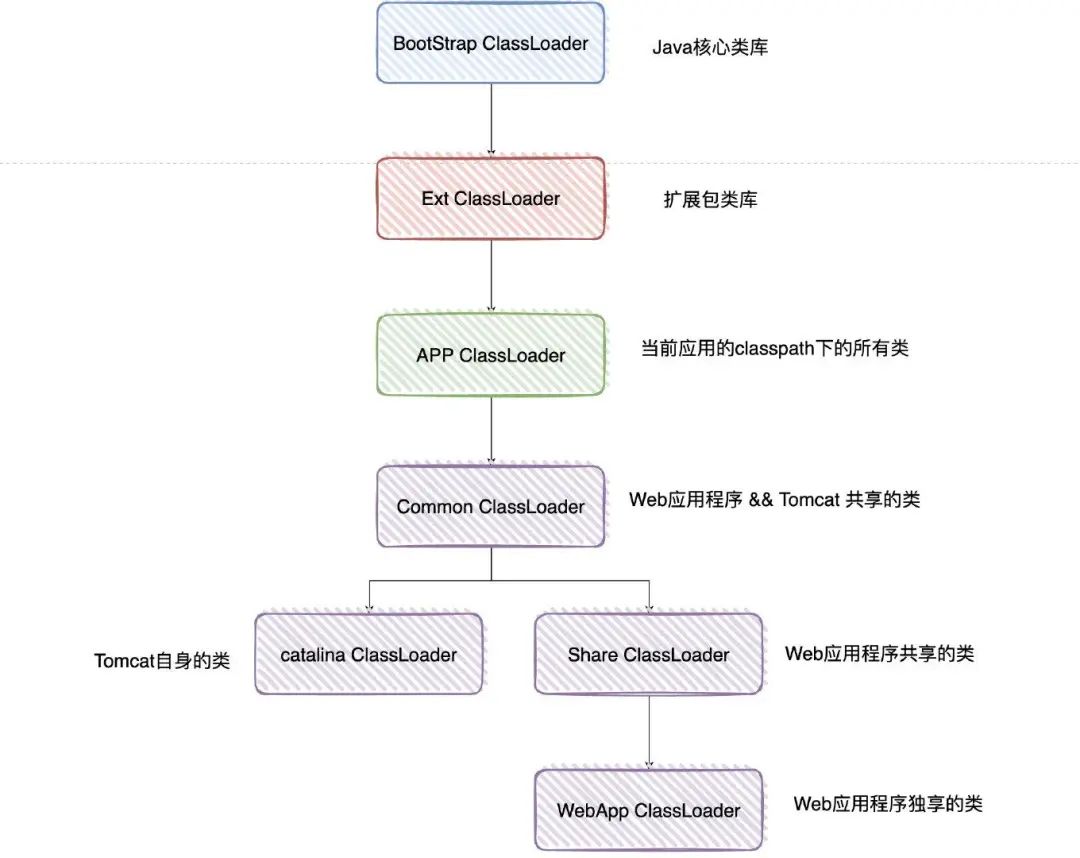
并不是Web应用程序下的所有依赖都需要隔离的，比如Redis就可以Web应用程序之间共享（如果有需要的话），因为如果版本相同，没必要每个Web应用程序都独自加载一份啊。

做法也很简单，Tomcat就在WebAppClassLoader上加了个父类加载器（SharedClassLoader），如果WebAppClassLoader自身没有加载到某个类，那就委托SharedClassLoader去加载。

为了隔绝Web应用程序与Tomcat本身的类，又有类加载器(CatalinaClassLoader)来装载Tomcat本身的依赖。

如果Tomcat本身的依赖和Web应用还需要共享，那么还有类加载器(CommonClassLoader)来装载进而达到共享。

个类加载器的加载目录可以到tomcat的catalina.properties配置文件上查看。



**面试官**：**顺便，我想问下，JDBC你不是知道吗，听说它也是破坏了双亲委派模型的，你怎么理解的。**

**候选者**：Eumm，这个有没有破坏，见仁见智吧。

**候选者**：JDBC定义了接口，具体实现类由各个厂商进行实现嘛(比如MySQL)

**候选者**：类加载有个规则：如果一个类由类加载器A加载，那么这个类的依赖类也是由「相同的类加载器」加载。

**候选者**：我们用JDBC的时候，是使用DriverManager进而获取Connection，DriverManager在java.sql包下，显然是由BootStrap类加载器进行装载

**候选者**：当我们使用DriverManager.getConnection()时，得到的一定是厂商实现的类。

**候选者**：但BootStrap ClassLoader会能加载到各个厂商实现的类吗？

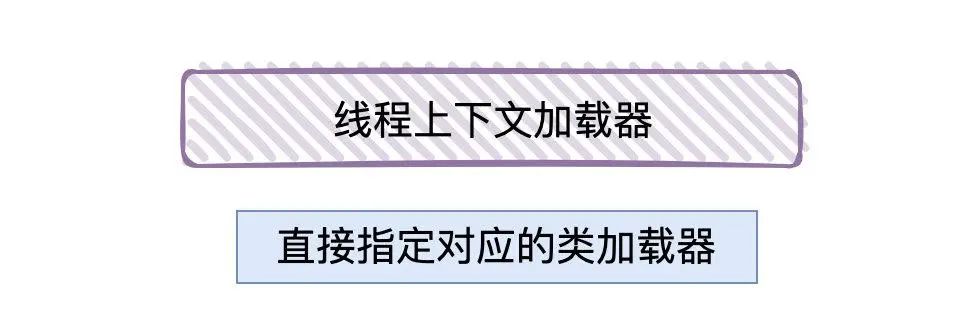
**候选者**：显然不可以啊，这些实现类又没在java包中，怎么可能加载得到呢

**面试官**：嗯..

**候选者**：DriverManager的解决方案就是，在DriverManager初始化的时候，得到「线程上下文加载器」

**候选者**：去获取Connection的时候，是使用「线程上下文加载器」去加载Connection的，而这里的线程上下文加载器实际上还是App ClassLoader

**候选者**：所以在获取Connection的时候，还是先找Ext ClassLoader和BootStrap ClassLoader，只不过这俩加载器肯定是加载不到的，最终会由App ClassLoader进行加载



**面试官**：嗯..

**候选者**：那这种情况，有的人觉得破坏了双亲委派机制，因为本来明明应该是由BootStrap ClassLoader进行加载的，结果你来了一手「线程上下文加载器」，改掉了「类加载器」

**候选者**：有的人觉得没破坏双亲委派机制，只是改成由「线程上下文加载器」进行类加载，但还是遵守着：「依次往上找父类加载器进行加载，都找不到时才由自身加载」。认为”原则”上是没变的。

**面试官**：那我了解了

**本文总结**：

* **前置知识：** JDK中默认类加载器有三个：AppClassLoader、Ext ClassLoader、BootStrap ClassLoader。AppClassLoader的父加载器为Ext ClassLoader、Ext ClassLoader的父加载器为BootStrap ClassLoader。这里的父子关系并不是通过继承实现的，而是组合。
* **什么是双亲委派机制：** 加载器在加载过程中，先把类交由父类加载器进行加载，父类加载器没找到才由自身加载。
* **双亲委派机制目的：** 为了防止内存中存在多份同样的字节码（安全）
* **类加载规则：** 如果一个类由类加载器A加载，那么这个类的依赖类也是由「相同的类加载器」加载。
* **如何打破双亲委派机制：** 自定义ClassLoader，重写loadClass方法（只要不依次往上交给父加载器进行加载，就算是打破双亲委派机制）
* **打破双亲委派机制案例：** Tomcat
  + 为了Web应用程序类之间隔离，为每个应用程序创建WebAppClassLoader类加载器
  + 为了Web应用程序类之间共享，把ShareClassLoader作为WebAppClassLoader的父类加载器，如果WebAppClassLoader加载器找不到，则尝试用ShareClassLoader进行加载
  + 为了Tomcat本身与Web应用程序类隔离，用CatalinaClassLoader类加载器进行隔离，CatalinaClassLoader加载Tomcat本身的类
  + 为了Tomcat与Web应用程序类共享，用CommonClassLoader作为CatalinaClassLoader和ShareClassLoader的父类加载器
  + ShareClassLoader、CatalinaClassLoader、CommonClassLoader的目录可以在Tomcat的catalina.properties进行配置
* **线程上下文加载器：** 由于类加载的规则，很可能导致父加载器加载时依赖子加载器的类，导致无法加载成功（BootStrap ClassLoader无法加载第三方库的类），所以存在「线程上下文加载器」来进行加载。