《面试必问之 jvm 与性能优化》

1. 描述一下 JVM 加载 Class 文件的原理机制?

在面试 java 工程师的时候，这道题经常被问到，故需特别注意。

Java 中的所有类，都需要由类加载器装载到 JVM 中才能运行。类加载器本身也是一个类，而它的工作就是把 class 文件从硬盘读取到内存中。在写程序的时候，我们几乎不需要关心类的加载，因为这些都是隐式装载的，除非我们有特殊的用法，像是反射，就需要显式的加载所需要的类。

Java 类的加载是动态的，它并不会一次性将所有类全部加载后再运行，而是保证程序运行的基础类(像是基类)完全加载到 jvm 中，至于其他类，则在需要的时候才加载。这当然就是为了节省内存开销。

Java 的类加载器有三个，对应 Java 的三种类:

```
Bootstrap Loader // 负责加载系统类（指的是内置类，像是String，对应于C#中的System类和C/C++标准库中的类）
      |
    - - ExtClassLoader  // 负责加载扩展类(就是继承类和实现类)
            |
          - - AppClassLoader  // 负责加载应用类(程序员自定义的类)
```

三个加载器各自完成自己的工作，但它们是如何协调工作呢？哪一个类该由哪个类加载器完成呢？为了解决这个问题，Java 采用了委托模型机制。

委托模型机制的工作原理很简单：当类加载器需要加载类的时候，先请示其 Parent(即上一层加载器)在其搜索路径载入，如果找不到，才在自己的搜索路径搜索该类。这样的顺序其实就是加载器层次上自顶而下的搜索，因为加载器必须保证基础类的加载。之所以是这种机制，还有一个安全上的考虑：如果某人将一个恶意的基础类加载到 jvm，委托模型机制会搜索其父类加载器，显然是不可能找到的，自然就不会将该类加载进来。

我们可以通过这样的代码来获取类加载器:

```
ClassLoader loader = ClassName.class.getClassLoader();
ClassLoader ParentLoader = loader.getParent();
```

注意一个很重要的问题，就是 Java 在逻辑上并不存在 BootstrapKLoader 的实体！因为它是用 C++编写的，所以打印其内容将会得到 null。

前面是对类加载器的简单介绍，它的原理机制非常简单，就是下面几个步骤:

1.装载:查找和导入 class 文件;

2.连接:

```
(1)检查:检查载入的class文件数据的正确性;

(2)准备:为类的静态变量分配存储空间;

(3)解析:将符号引用转换成直接引用(这一步是可选的)
```

3. 初始化:初始化静态变量，静态代码块。

```
这样的过程在程序调用类的静态成员的时候开始执行，所以静态方法main()才会成为一般程序的入口方法。类的构造器也会引发该动作。
```

来源：https://www.cnblogs.com/wenjiang/archive/2013/04/26/3044132.html

2. 什么是类加载器?

类加载器是一个用来加载类文件的类。Java 源代码通过 javac 编译器编译成类文件。然后 JVM 来执行类文件中的字节码来执行程序。类加载器负责加载文件系统、网络或其他来源的类文件。

3. 类加载器有哪些?

有三种默认使用的类加载器： Bootstrap 类加载器、 Extension 类加载器和 Application 类加载器。每种类加载器都有设定好从哪里加载类。

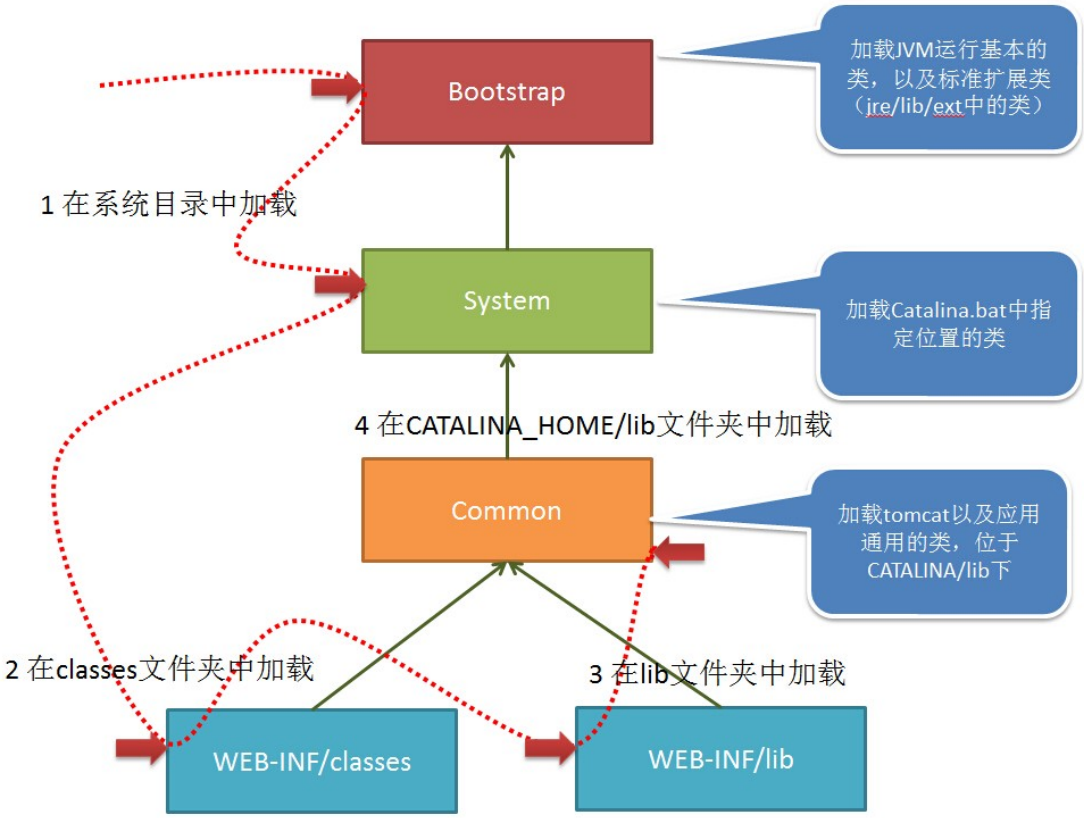
Bootstrap 类加载器负责加载 rt.jar 中的 JDK 类文件，它是所有类加载器的父加载器。 Bootstrap 类加载器没有任何父类加载器，如果你调用 String.class.getClassLoader()，会返回 null，任何基于此的代码会抛出 NullPointerException 异常。Bootstrap 加载器被称为初始类加载器。

而 Extension 将加载类的请求先委托给它的父加载器，也就是 Bootstrap，如果没有成功加载的话，再从 jre/lib/ext 目录下或者 java.ext.dirs 系统属性定义的目录下加载类。Extension 加载器由 sun.misc.Launcher$ExtClassLoader 实现。

第三种默认的加载器就是 Application 类加载器了。它负责从 classpath 环境变量中加载某些应用相关的类，classpath 环境变量通常由-classpath 或-cp 命令行选项来定义，或者是 JAR 中的 Manifest 的 classpath 属性。Application 类加载器是 Extension 类加载器的子加载器。通过 sun.misc.Launcher$AppClassLoader 实现。

4. 什么是 tomcat 类加载机制?

在 tomcat 中类的加载稍有不同，如下图：



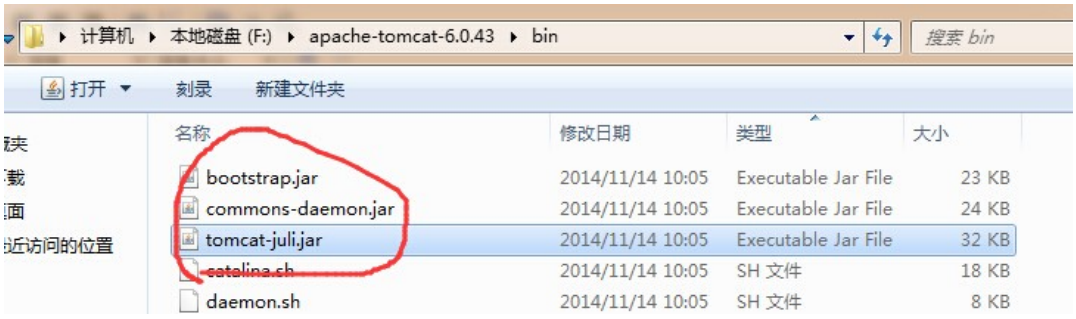
当 tomcat 启动时，会创建几种类加载器：

1 Bootstrap 引导类加载器

加载 JVM 启动所需的类，以及标准扩展类（位于 jre/lib/ext 下）

2 System 系统类加载器

加载 tomcat 启动的类，比如 bootstrap.jar，通常在 catalina.bat 或者 catalina.sh 中指定。位于 CATALINA_HOME/bin 下。



3 Common 通用类加载器

16. 深拷贝和浅拷贝？

浅拷贝被复制对象的所有变量都含有与原来的对象相同的值，而所有的对其他对象的引用仍然指向原来的对象。即对象的浅拷贝会对“主”对象进行拷贝，但不会复制主对象里面的对象。”里面的对象“会在原来的对象和它的副本之间共享。
简而言之，浅拷贝仅仅复制所考虑的对象，而不复制它所引用的对象。

深拷贝深拷贝是一个整个独立的对象拷贝，深拷贝会拷贝所有的属性,并拷贝属性指向的动态分配的内存。当对象和它所引用的对象一起拷贝时即发生深拷贝。深拷贝相比于浅拷贝速度较慢并且花销较大。
简而言之，深拷贝把要复制的对象所引用的对象都复制了一遍。

17. 什么是分布式垃圾回收（DGC）？它是如何工作的？

RMI 子系统实现基于引用计数的“分布式垃圾回收”(DGC)，以便为远程服务器对象提供自动内存管理设施。
当客户机创建（序列化）远程引用时，会在服务器端 DGC 上调用 dirty()。当客户机完成远程引用后，它会调用对应的 clean() 方法。
针对远程对象的引用由持有该引用的客户机租用一段时间。租期从收到 dirty() 调用开始。在此类租约到期之前，客户机必须通过对远程引用额外调用 dirty() 来更新租约。如果客户机不在租约到期前进行续签，那么分布式垃圾收集器会假设客户机不再引用远程对象。

18. 在 Java 中，对象什么时候可以被垃圾回收？

19. 简述 Minor GC 和 Major GC？

20. Java 中垃圾收集的方法有哪些？

21. 讲讲你理解的性能评价及测试指标？

22. 常用的性能优化方式有哪些？

23. 说说分布式缓存和一致性哈希？

24. 什么是 GC 调优？