http://www.cnblogs.com/lanxuezaipiao/p/4138511.html

http://wiki.jikexueyuan.com/project/java-vm/java-debug.html

预定义类加载器和双亲委派机制

1. JVM预定义的三种类型类加载器：
    - **启动（Bootstrap）类加载器**：是用本地代码实现的类装入器，它负责将 `<Java_Runtime_Home>/lib`下面的类库加载到内存中（比如`rt.jar`）。由于引导类加载器涉及到虚拟机本地实现细节，开发者无法直接获取到启动类加载器的引用，所以不允许直接通过引用进行操作。
    - **标准扩展（Extension）类加载器**：是由 Sun 的`ExtClassLoader（sun.misc.Launcher$ExtClassLoader`）实现的。它负责将`< Java_Runtime_Home >/lib/ext`或者由系统变量 `java.ext.dir`指定位置中的类库加载到内存中。开发者可以直接使用标准扩展类加载器。
    - **系统（System）类加载器**：是由 Sun 的`AppClassLoader（sun.misc.Launcher$AppClassLoader`）实现的。它负责将系统类路径（`CLASSPATH`）中指定的类库加载到内存中。开发者可以直接使用系统类加载器。

除了以上列举的三种类加载器，还有一种比较特殊的类型 — 线程上下文类加载器。

2. 双亲委派机制描述

某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父类加载器，**依次递归**，如果父类加载器可以完成类加载任务，就成功返回；只有父类加载器无法完成此加载任务时，才自己去加载。

几点思考

1. Java虚拟机的第一个类加载器是Bootstrap，这个加载器很特殊，**它不是Java类，因此它不需要被别人加载，它嵌套在Java虚拟机内核里面，也就是JVM启动的时候Bootstrap就已经启动，它是用C++写的二进制代码（不是字节码）**，它可以去加载别的类。

这也是我们在测试时为什么发现`System.class.getClassLoader()`结果为null的原因，这并不表示System这个类没有类加载器，而是它的加载

器比较特殊，是`BootstrapClassLoader`，由于它不是Java类，因此获得它的引用肯定返回null。

2. 委托机制具体含义

当Java虚拟机要加载一个类时，到底派出哪个类加载器去加载呢？

- 首先当前线程的类加载器去加载线程中的第一个类（假设为类A）。

注：当前线程的类加载器可以通过Thread类的getContextClassLoader()获得，也可以通过setContextClassLoader()自己设置类加载器。

- 如果类A中引用了类B，Java虚拟机将使用加载类A的类加载器去加载类B。
- 还可以直接调用`ClassLoader.loadClass()`方法来指定某个类加载器去加载某个类。

3. 委托机制的意义 — 防止内存中出现多份同样的字节码

比如两个类A和类B都要加载System类：

- 如果不用委托而是自己加载自己的，那么类A就会加载一份System字节码，然后类B又会加载一份System字节码，**这样内存中就出现了两份System字节码。**
- 如果使用委托机制，会递归的向父类查找，也就是**首选用Bootstrap尝试加载**，如果找不到再向下。这里的System就能在Bootstrap中找到然后加载，如果此时类B也要加载System，也从Bootstrap开始，此时**Bootstrap发现已经加载过了System那么直接返回内存中的System即可而不需要重新加载**，这样内存中就只有一份System的字节码了。

一道面试题

- 能不能自己写个类叫`java.lang.System`？

答案：通常不可以，但可以采取另类方法达到这个需求。

解释：为了不让我们写System类，类加载采用委托机制，这样可以保证爸爸们优先，爸爸们能找到的类，儿子就没有机会加载。而System类是Bootstrap加载器加载的，就算自己重写，也总是使用Java系统提供的System，**自己写的System类根本没有机会得到加载。**

但是，我们可以**自己定义一个类加载器来达到这个目的**，为了避免双亲委托机制，这个类加载器也必须是特殊的。由于系统自带的三个类加载器都加载特定目录下的类，如果我们自己的类加载器放在一个特殊的目录，那么系统的加载器就无法加载，也就是最终还是由我们自己的加载器加

载。