# 深入理解JVM

## 类的加载

在Java代码中，类型的加载，连接与初始化过程都是在程序运行期间完成的

提供了更大的灵活性，增加了更多的可能性

## 类的加载深入剖析

Java虚拟机与程序的生命周期

在以下几种情况下，Java虚拟机将结束生命周期

1. 执行了System.exit()方法
2. 程序正常执行结束
3. 程序在执行中过程中遇到了异常或错误而异常终止
4. 由于操作系统出现错误而导致Java虚拟机进程终止

## 类的加载、连接与初始化

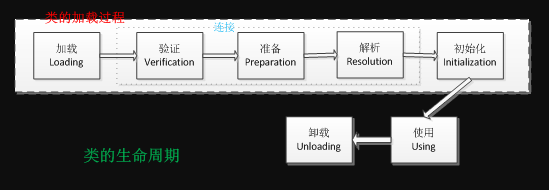
1. 加载：查找并加载类的二进制数据
2. 连接

验证：确保被加载类的正确性

准备：为类的静态变量分配内存，并将其初始化为默认值

解析：把类的符合引用转换为直接引用

1. 初始化：为类的静态变量赋予正确的初始值
2. 使用：
3. 卸载：



Java程序对类的使用方式可分为两种

* + 主动使用
  + 被动使用

所有的Java虚拟机实现必须在每个类或接口被Java程序”首次主动使用“时才初始化他们

主动使用（七种）：

1. 创建类的实例
2. 访问某个类或接口的静态变量，或者对静态变量赋值
3. 调用类的静态方法
4. 反射
5. 初始化一个类的子类
6. Java虚拟机启动时被表明为启动类的类（Java Test）
7. JDK1.7开始提供的动态语言支持：Java.lang.invoke.MethodHandle实例的解析结果REF\_getStatic，REF\_puStatic，REF\_invokeStatic句柄对应的类没有初始化，则初始化

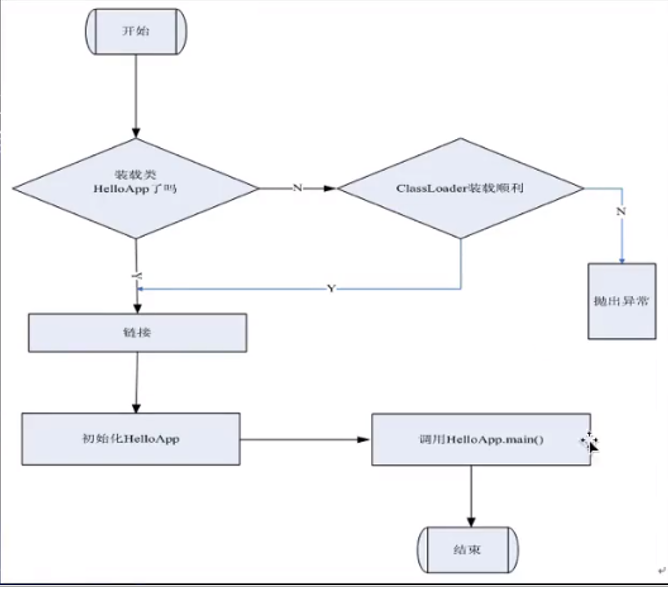
除了以上七种情况，其他使用Java类的方式都被看作是对类的被动使用，都不会导致类的初始化

## 类的加载

类的加载值的是将类的.class文件中的二进制数据读入到内存中，将其放在运行时方法区内(jdk1.8后为Meatspace)，然后在内存中创建一个java.lang.Class的对象（JVM规范并未说明Class对象位于哪里，HotSpot虚拟机将其放在了方法区了）用来封装类在方区内的数据结构

加载.class文件的方式:

1. 从本地系统中直接加载
2. 通过网络下载.class文件
3. 从zip，jar等归档文件中加载.class文件
4. 从专用数据库中提取.class文件
5. 将Java源文件动态编译为.class文件



## 类的加载

类的加载的最终产品是位于内存中的Class对象

Class对象封装了类在方法区内的数据并且向Java程序员提供了访问方法区内的数据结构的接口

## 类的加载

有两种类型的类加载器

Java虚拟机自带的加载器

根类加载器（Bootstrap）

扩展类加载器（Extension）

系统（应用）类加载器（System App）

用户自定义的类加载器

java.lang.ClassLoader的子类

用户可以定制类的加载方式

## 类的加载

类加载器并不需要等到某个类被”首次主动使用“时再加载它

## 类的加载

JVM规范允许类加载器在预料某个类将要被使用时就预先加载它，

如果在预先加载的过程中遇到.class文件缺失或存在错误，类加载器必须在程序首次主动使用该类时才报告错误（LinkageError错误）

如果这个类一直没有被程序主动使用，那么类加载器就不会报告错误

## 类的验证

类被加载后，就进入连接阶段。连接就是将已经读入到内存的类的二进制数据合并到虚拟机的运行时环境中去。

## 类的验证

类的验证的内容

类文件的结构检查

语义检查

字节码验证

二进制兼容性的验证

## 类的准备

在准备阶段，Java虚拟机为类的静态变量分配内存，并设置默认的初始值。例如对于以下Sample类，在准备阶段，将为int类型的静态变量a分配4字节的内存空间，并且赋予默认值0，为long类的静态变量b分配8个字节的内容空间，并且赋予默认值0.

