[**深入理解JVM(八)——类加载的时机**](https://blog.csdn.net/u010425776/article/details/51251430)

<https://blog.csdn.net/u010425776/article/details/51251430>

## 类的生命周期

一个类从加载进内存到卸载出内存为止，一共经历7个阶段：   
加载——>验证——>准备——>解析——>初始化——>使用——>卸载

其中，类加载包括5个阶段：   
加载——>验证——>准备——>解析——>初始化

在类加载的过程中，以下3个过程称为连接：   
验证——>准备——>解析

因此，JVM的类加载过程也可以概括为3个过程：   
加载——>连接——>初始化

C/C++在运行前需要完成预处理、编译、汇编、链接；而在Java中，类加载(加载、连接、初始化)是在程序运行期间完成的。   
在程序运行期间进行类加载会稍微增加程序的开销，但随之会带来更大的好处——提高程序的灵活性。Java语言的灵活性体现在它可以在运行期间**动态扩展**，所谓动态扩展就是在运行期间**动态加载**和**动态连接**。 

## 类加载的时机

### 1. 类加载过程中每个步骤的顺序

我们已经知道，类加载的过程包括：加载、连接、初始化，连接又分为：验证、准备、解析，所以说类加载一共分为5步：加载、验证、准备、解析、初始化。

其中加载、验证、准备、初始化的**开始**顺序是依次进行的，这些步骤开始之后的过程可能会有重叠。   
而解析过程会发生在初始化过程中。 

### 2. 类加载过程中“初始化”开始的时机

JVM规范中只定义了类加载过程中初始化过程开始的时机，加载、连接过程都应该在初始化之前开始(解析除外)，这些过程具体在何时开始，JVM规范并没有定义，不同的虚拟机可以根据具体的需求自定义。

**初始化开始的时机：**

1. 在运行过程中遇到如下字节码指令时，如果类尚未初始化，那就要进行初始化：new、getstatic、putstatic、invokestatic。这四个指令对应的Java代码场景是：   
   * 通过new创建对象；
   * 读取、设置一个类的静态成员变量(不包括final修饰的静态变量)；
   * 调用一个类的静态成员函数。
2. 使用java.lang.reflect进行反射调用的时候，如果类没有初始化，那就需要初始化；
3. 当初始化一个类的时候，若其父类尚未初始化，那就先要让其父类初始化，然后再初始化本类；
4. 当虚拟机启动时，虚拟机会首先初始化带有main方法的类，即主类；

### 3. 主动引用 与 被动引用

JVM规范中要求在程序运行过程中，“当且仅当”出现上述4个条件之一的情况才会初始化一个类。如果间接满足上述初始化条件是不会初始化类的。   
其中，直接满足上述初始化条件的情况叫做**主动引用**；间接满足上述初始化过程的情况叫做**被动引用**。   
那么，只有当程序在运行过程中满足主动引用的时候才会初始化一个类，若满足被动引用就不会初始化一个类。 

### 4. 被动引用的场景示例

* 示例一

public class Fu{

public static String name = "柴毛毛";

static{

System.out.println("父类被初始化！");

}

}

public class Zi{

static{

System.out.println("子类被初始化！");

}

}

public static void main(String[] args){

System.out.println(Zi.name);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

**输出结果：**   
父类被初始化！   
柴毛毛   
**原因分析：**   
本示例看似满足初始化时机的第一条：当要获取某一个类的静态成员变量的时候如果该类尚未初始化，则对该类进行初始化。   
但由于这个静态成员变量属于Fu类，Zi类只是间接调用Fu类中的静态成员变量，因此Zi类调用name属性属于间接引用，而Fu类调用name属性属于直接引用，由于JVM只初始化直接引用的类，因此只有Fu类被初始化。 

* 示例二

public class A{

public static void main(String[] args){

Fu[] arr = new Fu[10];

}

}

* 1
* 2
* 3
* 4
* 5

**输出结果：**   
并没有输出“父类被初始化！”   
**原因分析：**   
这个过程看似满足初始化时机的第一条：遇到new创建对象时若类没被初始化，则初始化该类。   
但现在通过new要创建的是一个数组对象，而非Fu类对象，因此也属于间接引用，不会初始化Fu类。 

* 示例三

public class Fu{

public static final String name = "柴毛毛";

static{

System.out.println("父类被初始化！");

}

}

public class A{

public static void main(String[] args){

System.out.println(Fu.name);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

**输出结果：**   
柴毛毛   
**原因分析：**   
本示例看似满足类初始化时机的第一个条件：获取一个类静态成员变量的时候若类尚未初始化则初始化类。   
但是，Fu类的静态成员变量被final修饰，它已经是一个常量。被final修饰的常量在Java代码编译的过程中就会被放入它被引用的class文件的常量池中(这里是A的常量池)。所以程序在运行期间如果需要调用这个常量，直接去当前类的常量池中取，而不需要初始化这个类。 

### 5. 接口的初始化

接口和类都需要初始化，接口和类的初始化过程基本一样，不同点在于：类初始化时，如果发现父类尚未被初始化，则先要初始化父类，然后再初始化自己；但接口初始化时，并不要求父接口已经全部初始化，只有程序在运行过程中用到当父接口中的东西时才初始化父接口。

