Java的类型加载、连接、和初始化都是在程序运行区间完成的。

类从被加载到虚拟机内存中开始，到卸载出内存为止，它的整个生命周期包括：加载（Loading）、验证（Verification）、准备（Preparation）、解析（Resolution）、初始化（Initialization）、使用（Using）和卸载（Unloading）7个阶段。其中验证、准备、解析3个部分统称为连接（Linking）



对于静态字段，只有直接定义这个字段的类才会被初始化，因此通过其子类来引用父类中定义的静态字段，只会触发父类的初始化而不会触发子类的初始化。

SuperClass[]sca=new SuperClass[10]；

加载顺序

**加载：**

1）通过一个类的全限定名来获取定义此类的二进制字节流。

2）将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构。

3）在内存中生成一个代表这个类的java.lang.Class对象，作为方法区这个类的各种数据的访问入口。

对于数组类而言，情况就有所不同，数组类本身不通过类加载器创建，它是由Java虚拟机直接创建的。

**验证：**

验证是连接阶段的第一步，验证是虚拟机对自身保护的一项重要工作

1.文件格式验证

2.元数据验证（对字节码描述的信息进行语义分析，以保证其描述的信息符合Java语言规范的要求）

3.字节码验证 （通过数据流和控制流分析，确定程序语义是合法的、符合逻辑的）

4.符号引用验证（符号引用中通过字符串描述的全限定名是否能找到对应的类及其方法字段）

**准备：**

准备阶段是正式为**类变量**（非对象变量--static）分配内存并设置类变量初始值的阶段

public static int value=123； 那变量value在准备阶段过后的初始值为0而不是123

**解析：**

解析阶段是虚拟机将常量池内的**符号引用替换为直接引用**的过程

符号引用（Symbolic References）：符号引用以一组符号来描述所引用的目标，符号可以是任何形式的字面量，只要使用时能无歧义地定位到目标即可。

直接引用（Direct References）：直接引用可以是直接指向目标的指针、相对偏移量或是一个能间接定位到目标的句柄。

**虚拟机规范之中并未规定解析阶段发生的具体时间**

**初始化：**

到了初始化阶段，才真正开始执行类中定义的Java程序代码（或者说是字节码）。在准备阶段，变量已经赋过一次系统要求的初始值，初始化阶段是执行类构造器＜clinit＞（）方法的过程

＜clinit＞（）方法是由编译器自动收集类中的所有类变量的赋值动作（不包括方法与构造函数中）和静态语句块（static{}块）中的语句合并产生的

**类加载器：**通过一个类的全限定名来获取描述此类的二进制字节流”这个动作放**到Java虚拟机外部**去实现，以便让应用程序自己决定如何去获取所需要的类。实现这个动作的代码模块称为“类加载器”。（外部是指不通过jvm自身的操作，可以是lang包里的程序，可以自己编写。不要过度理解）

示例：

protected synchronized Class＜?＞loadClass（String name,boolean resolve）throws ClassNotFoundException

{

//首先，检查请求的类是否已经被加载过了

Class c=findLoadedClass（name）；

if（c==null）{

try{

if（parent！=null）{

c=parent.loadClass（name,false）；

}else{

c=findBootstrapClassOrNull（name）；

}

}catch（ClassNotFoundException e）{

//如果父类加载器抛出ClassNotFoundException

//说明父类加载器无法完成加载请求

}

if（c==null）{

//在父类加载器无法加载的时候

//再调用本身的findClass方法来进行类加载

c=findClass（name）；

}

}

if（resolve）{

resolveClass（c）；

}

return c；

}