/\*\*

\* java类加载器

\* 类加载器负责加载与连接，这个过程是在运行时进行的，这种机制为java提供极大的灵活性

\*

\* 类的生命周期

\* class文件-->加载-->验证-->准备-->解析-->初始化-->使用-->卸载

\* 上述这种顺序只代表各个阶段的开始顺序，并不是说一个阶段完成后另外一个阶段才开始，如：加载还没有彻底完成的时候，验证已经开始

\*

\* 连接包括验证-->准备-->解析这三个步骤

\*

\* 初始化触发条件

\* 1、new 类的实例

\* 2、访问静态字段或是方法（子类会导致父类相关字段的初始化，但是父类不会触发子类的初始化，但是会造成子类的加载，但是子接口并不会触发父接口的相关的操作）

\* 3、通过Reflect实例化对象会触发初始化操作

\* 4、Main方法

\*

\* 加载时机：可以由具体的虚拟机自己把握，规范中并没有规定

\*

\* 加载阶段的责任

\* 1、通过类的全限定名称来获取定义此类的二进制字节流(这个定义很灵活，好多应用就是基于这个诞生的)

\* 2、把这个字节流所描述的静态存储结构转化为方法区的运行时数据结构（方法区中的数据结构可由实现着自行定义）

\* 3、在堆中生成一个代表这个类的java.lang.Class对象，作为访问方法区数据的入口

\*

\* 验证

\* 这一阶段确保class文件中包含的内容符合虚拟机的要求且不会危害虚拟机自身的安全

\* 但是并没有规定验证什么，怎么验证，只是规定了验证出错的时候抛出java.lang.VerifyError或是其子类

\* 一般虚拟机都会包括以下验证：

\* 1、文件格式的验证

\* 2、元数据验证

\* 3、字节码验证

\* 4、符号引用的验证

\* 验证对于类加载器并不是必须的，如果自己写的或是第三方已经过验证的代码可以选择关闭这项功能-Xverify:none

\*

\* 准备

\* 这一阶段为类在方法区中分配内存及设置默认值（静态变量会在方法区分配）如static int i=123;会在方法区为i分配空间，且值为0，并不是123

\*

\* 解析

\* 把符号引用转变为直接引用，虚拟机规范中并没有规定什么时候解析符号引用，也没有规定解析一次还是重复的解析

\*

\* 符号引用于直接引用的区别：

\* 符号引用：用一组符号来描述所引用的对象，符号引用可以是任意形式的字面量，只要能够无歧义的定位到目标即可，与具体的内存布局无关，引用的目标并不一定记载道内存中

\* 直接引用：可以是直接指向目标的指针、间接定位到目标的句柄或是偏移量均可。与内存布局相关，同一个class文件在不同的虚拟机中直接引用的形式一般不同，但是有了知觉引用后引用的目标在内存中一定已经存在

\*

\* 初始化

\* 这里的初始化也就是运行<clinit>方法，这个方法是由编译器收集类的字段复制语句和静态块构成的，静态块不可以引用静态块后面定义的字段，但是可以为其赋值

\*

\* 类加载器

\* 用于加载类文件，java设计者孤立开发者自定义类加载器，用于实现各种基于类加载器的功能，比如代码加密、热部署、从网络中加载class文件、osgi等等

\* 从JVM的角度看，类加载器只有两种，启动类加载器（c++写的）及其他类加载器（java实现）

\* 从java开发者的角度看，类加载器可以划分的更新,BootStrapClassLoader、SystemClassLoader、ClassPathClassLoader及自定义的类加载器

\*

\* 加载机制：

\* 1、委托代理加载机制

\* 2、上下文类加载机制（广泛的用语Service Provider机制中）

\*

\*/