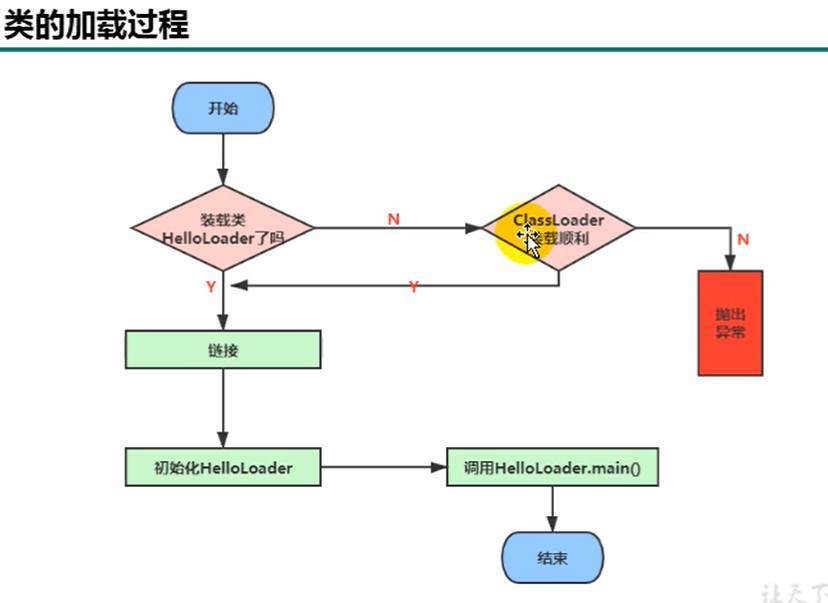
类加载器子系统的功能：从文件中或者网络中加载class文件。Class文件在文件开头有特定文件标识(**这个特定的文件标识就是CAFEBABE，用二进制查看器打开.class就能看到**)。

(JclassLib、PXBinaryViewerSetup)

（从Java到class经过javac前端编译得到静态数据结构，而加载过程将静态转换为运行时数据区中的数据结构。）



注意：

1.类加载器子系统只负责加载class文件，能不能运行取决于执行引擎（execution engine）

2.类加载器子系统加载的类信息存放于一块称为方法区中。方法区中还存放运行时常量池信息、可能还有字面量数字常量（而这部分常量是class文件中的常量池部分的内存映射）。

类加载三个过程：

加载、连接（验证-准备-解析）、初始化

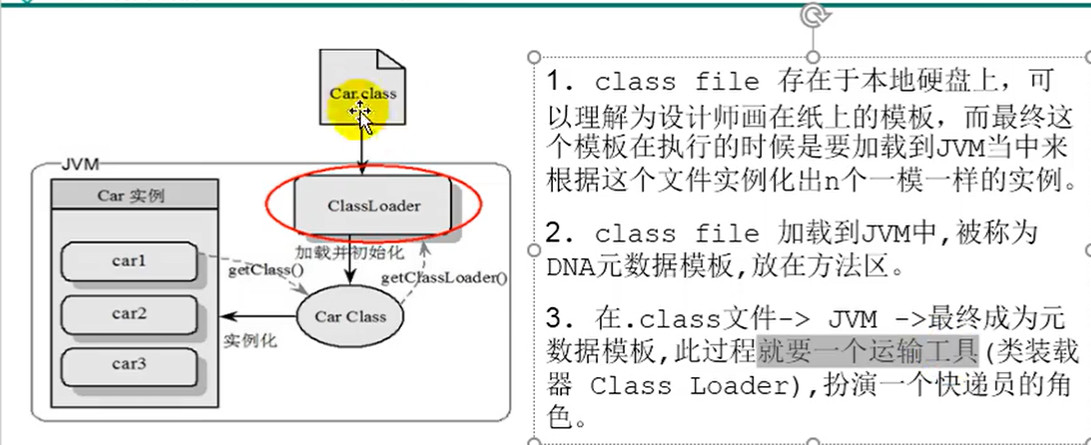
加载阶段：（Loading）

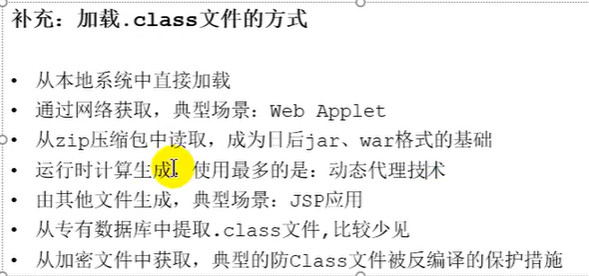
1.通过一个类的全限定名获取定义此类的二进制字节流。

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**，作为方法区这个类中的数据的访问入口。

类加载器：





链接阶段：

**验证（verify）**：

**目的**：确保Class文件中的字节流中包含信息符合当前虚拟机要求，保证加载类的正确性、不会危害虚拟机自身安全。（什么叫做不会危害虚拟机自身安全？虚拟机运行时不就是个虚拟机进程吗，应该是不会让输入虚拟机不可掌控，从而危害计算机。其实如果因为某一个Class文件被恶意修改，而导致整个JVM不能运行这又如何不叫作危害虚拟机自身安全？）

**验证的4种类型（或者叫做四个方面）：文件格式验证、元数据验证、字节码验证、符号引用验证。**

**准备（prepare）**：

1、为类变量分配内存并且设置该类变量的默认初始值，即零值。

（



）

2、这里不包括用final修饰的static，因为final在编译时就进行分配了，准备阶段会进行显式的初始化。

3、这里不会为实例变量分配初始化，类变量会分配在方法区中，而实例变量是会随着对象一起分配到Java堆中。

**解析（resolve）**：

1、将常量池内的符号引用转换为直接引用的过程。

2、事实上，解析操作往往会伴随着JVM在执行完初始化之后再执行。

3、符号引用就是一组符号来描述所引用的目标，符号引用的字面量形式明确定义在《Java虚拟机规范》的Class文件格式中，直接引用就是直接指向目标的指针，相对偏移量或一个间接定位到目标的句柄。（**例子：Hotspot虚拟机中的常量池的引用就是使用：#数字**）

[java 内存分配 final关键字\_chengzhezhijian的专栏-CSDN博客](https://blog.csdn.net/chengzhezhijian/article/details/19846961)

[java -- JVM的符号引用和直接引用 - hello\_史努比 - 博客园 (cnblogs.com)](https://www.cnblogs.com/shinubi/articles/6116993.html)



4、解析动作主要针对类或接口、字段、类方法、接口方法、方法类型等。对应常量池中的**CONSTANT\_Class\_info、CONSTANT\_Fieldref\_info、CONSTANT\_Methodref\_info**等。

初始化阶段：

1、初始化阶段就是执行**类构造器**的<clinit>（）的过程。（通过JClassLib可以进行查看）

2、此方法不需要定义，是javac编译器自动收集类中**所有类变量的赋值动作和静态代码块中的语句合并而来**。（如果代码中没有这样的要求，就不会进行生成，也就是只有类属信息才会被进行初始化。也就是非静态变量还是遵循三种初始化方式及顺序：1.成员变量初始化语句；2.成员变量初始化块；3.构造函数）

3、构造器方法中指令按语句在源文件中出现的顺序执行。（比如下面的结果就是10）



4、<clinint>()不同于类的构造器。（关联：类提供的构造器是虚拟机视角下的<init>()）

5、若该类具有父类，JVM会保证子类的<clinit>()执行前，父类的<clinit>()已经执行完毕。

6、虚拟机必须保证一个类的<clinit>()方法在多线程下被同步加锁。（引发这个的原因是因为，当我的程序中需要用到该类时才会去加载它，而加载只需加载一次就行，所以可能在某时间段会存在多个线程使用到该类而该类话没有加载完成。）