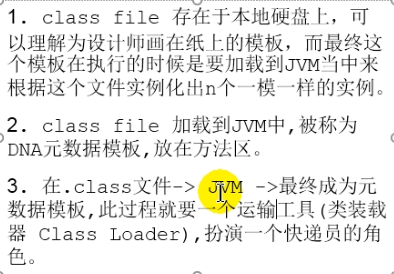
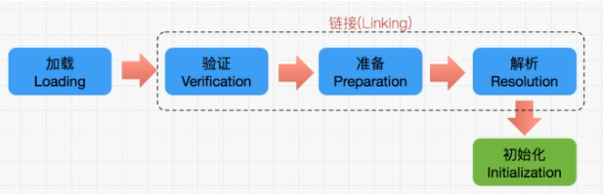
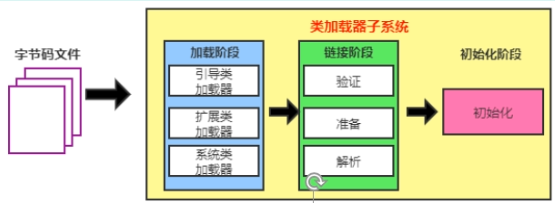
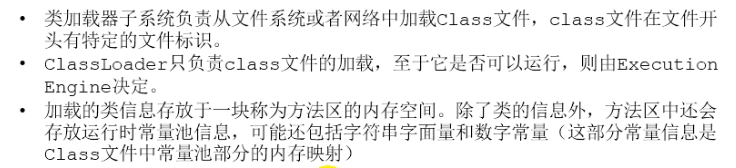
# 概述和概览



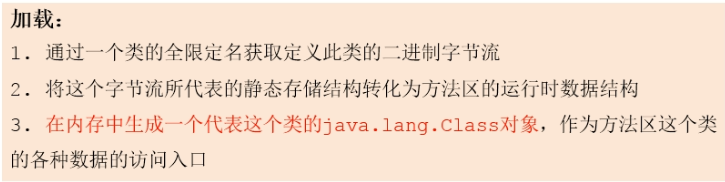


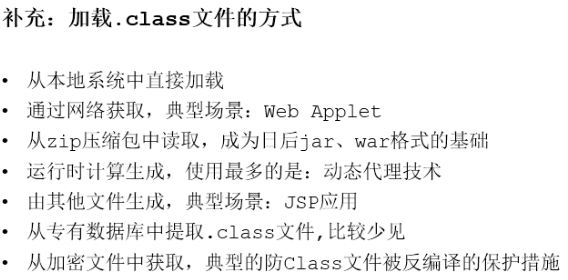
 

# 2、类加载的三个阶段

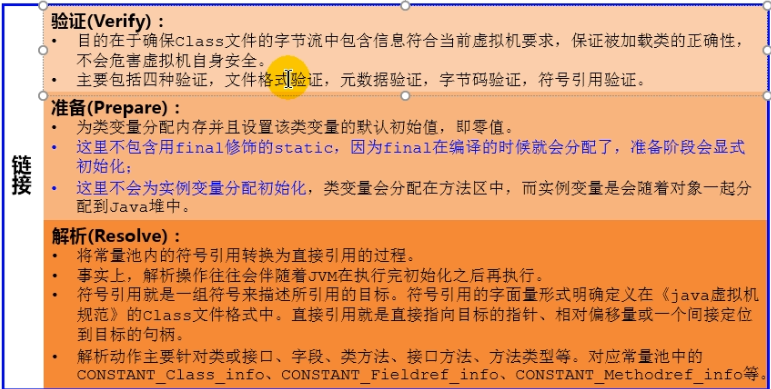
## 1、加载

使用类加载器加载一个类

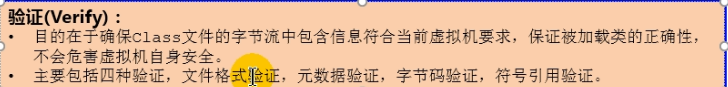


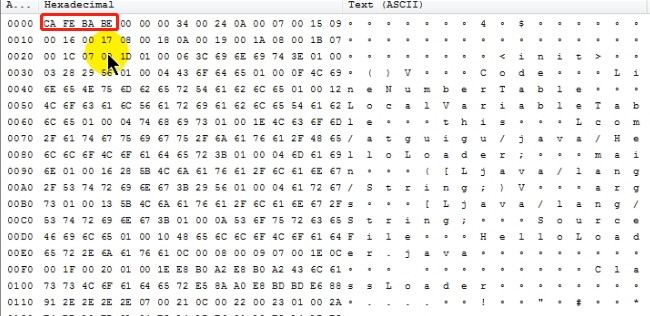


## 2、链接



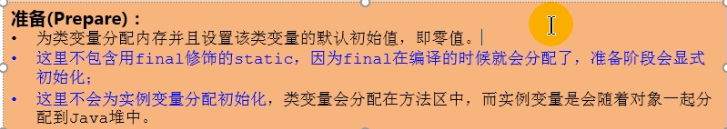
### 1、验证





是以CAFE BABE 开头的

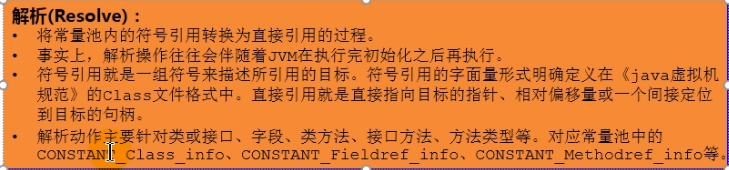
### 准备



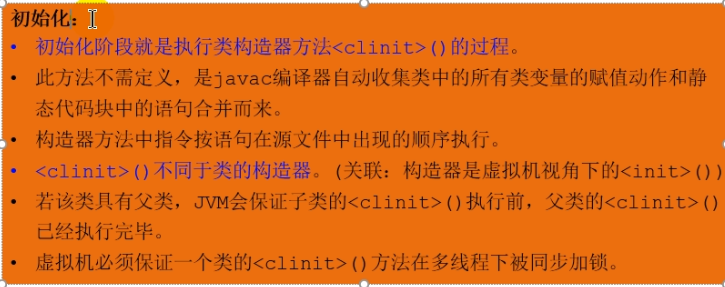
为类变量分配内存并且设置该类变量的初始值，即零值，真正赋值是在初始化阶段

常量是在编译的时候就会显式的初始化

### 解析



## 3、初始化



**就是执行类的构造器方法的过程**

<clinit>() :执行类变量显示赋值和执行静态代码块

<init>() : 构造器

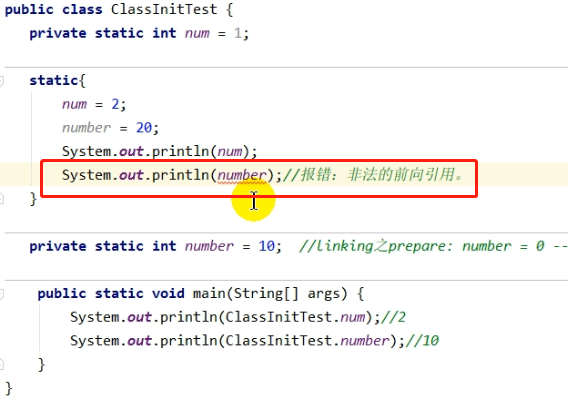
**赋值顺序问题**



number在链接阶段准备阶段已经被加载进了内存了，也做了默认初始化，赋值为0 ,相当于number = 0

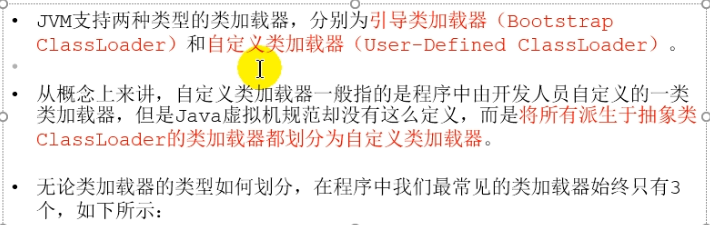
所以在初始化阶段中，number按照顺序 先被赋值为 20 后被赋值为10

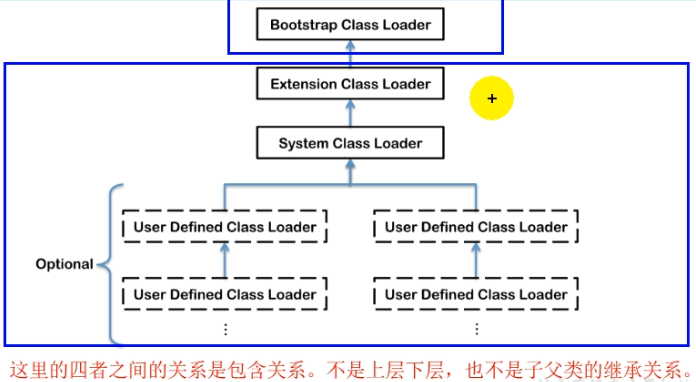
**调用问题**



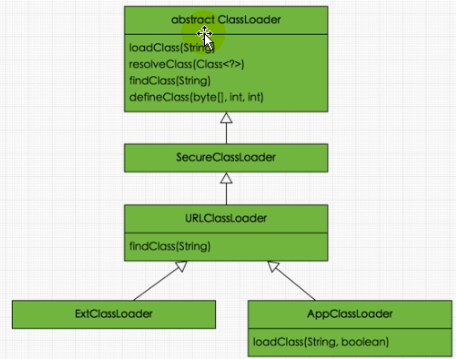
可以赋值但是不可以调用

# 类加载器的分类

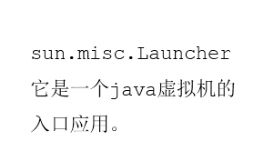




他们之间的关系相当于目录的关系



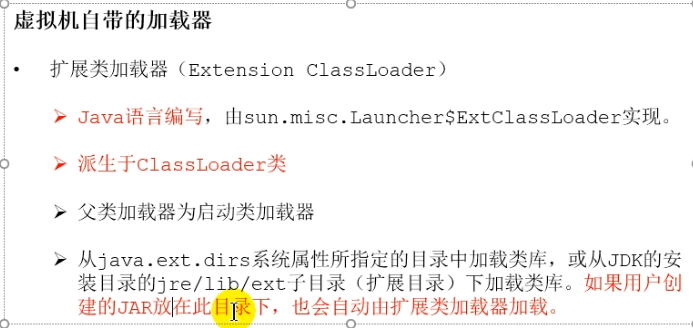
除了引导类加载器 其他都是继承自ClassLoader



## 1、引导类加载器



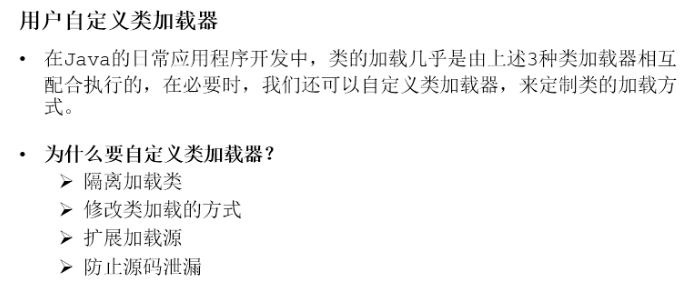
## 2、扩展类加载器



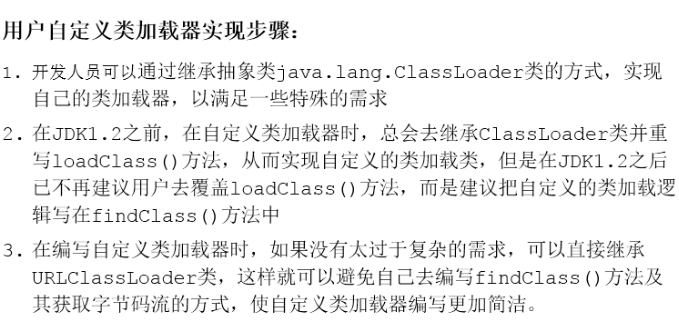
## 3、系统类加载器



# 为什么要自定义类加载器



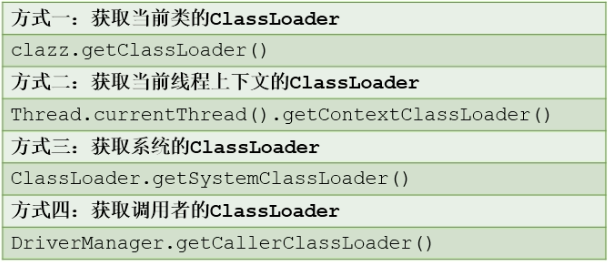
# 6、自定义类加载器的实现步骤



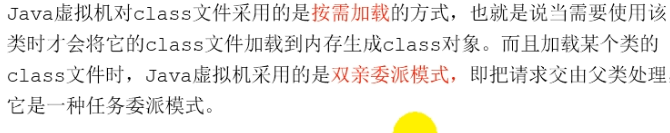
# 7、Classsloader的api

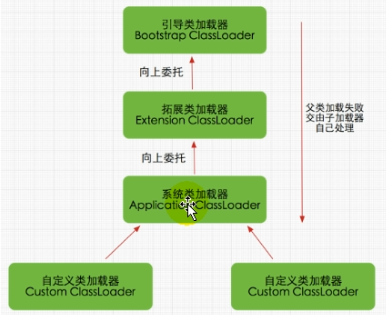
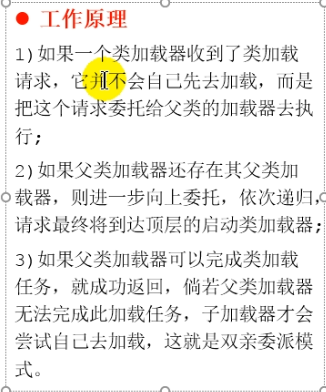


# 8、获取classloader 的途径



# 双亲委派机制

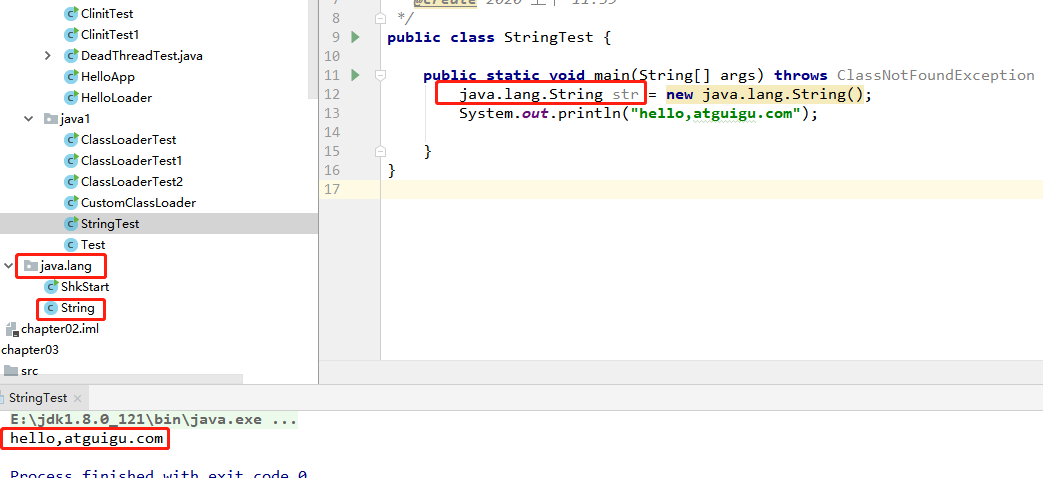




**实例**：在自己的工程里面建一个java.lang.String 类，最后加载出来的是rt.jar 中的java.lang.String类

因为这个类在加载的时候，appclassloader向上委托 extclassloader。 extclassloader 向上委托bootstrapclassloader ，最后由bootstrapclassloader加载了位于 rt.jar中的 String 类

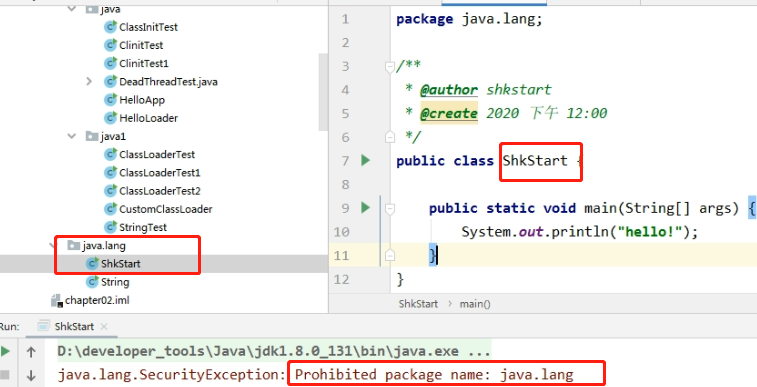
如图：



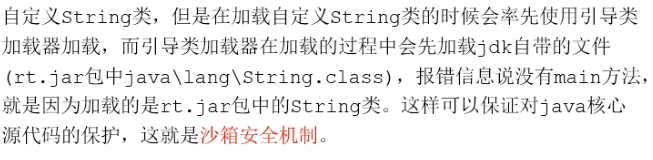


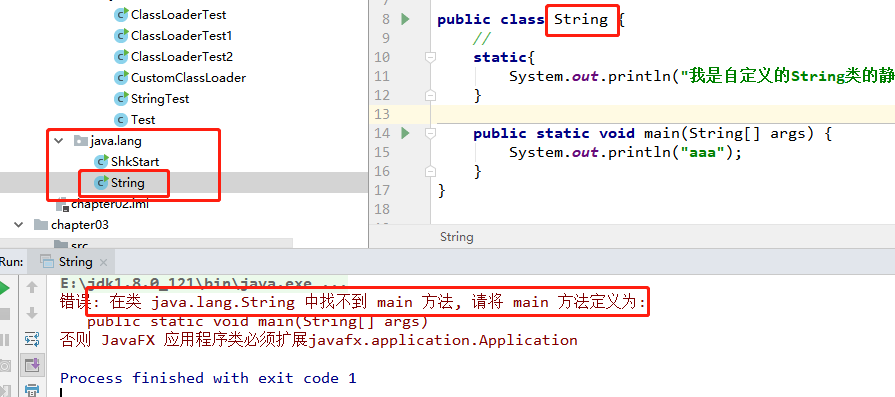
出于安全的考虑

防止自定义类对引导类加载器本身造成破坏



# 沙箱安全机制





# 11、其他

