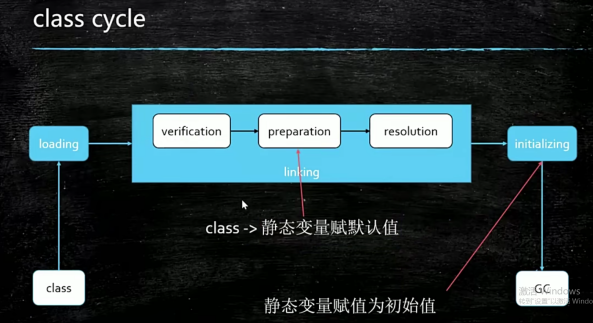
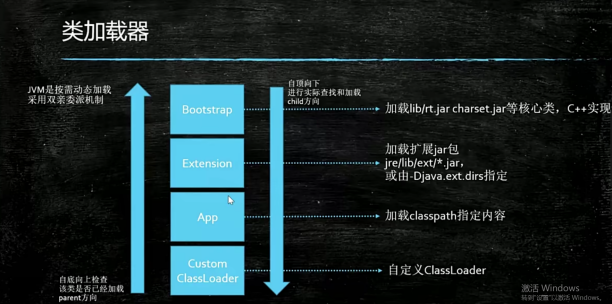
1. 类加载过程



1. Loading

通过类加载器将class字节码文件加载到JVM中，并生成Class对象

（1）类加载器



1. 类加载时机

创建类实例。也就是new的方式

调用某个类的类方法（static）

访问某个类或接口的类变量，或为该类变量赋值（static）

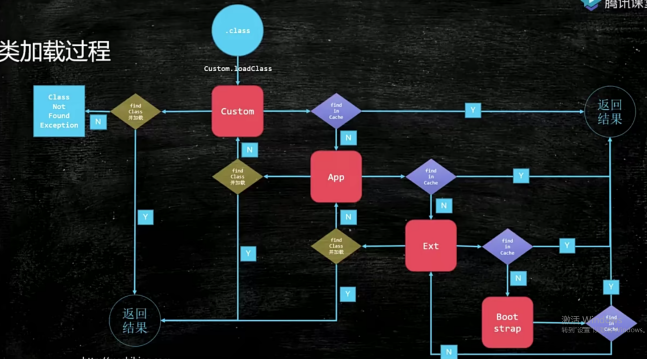
使用反射方式强制创建某个类或接口对应的java.lang.Class对象

初始化某个类的子类，则其父类也会被初始化

直接使用java.exe命令来运行某个主类

1. 类加载过程----双亲委派机制

* 自底向上检查缓存是否已经存在（每个类加载器都有自己的缓存）
* 自顶向下在指定PATH查找class文件并加载



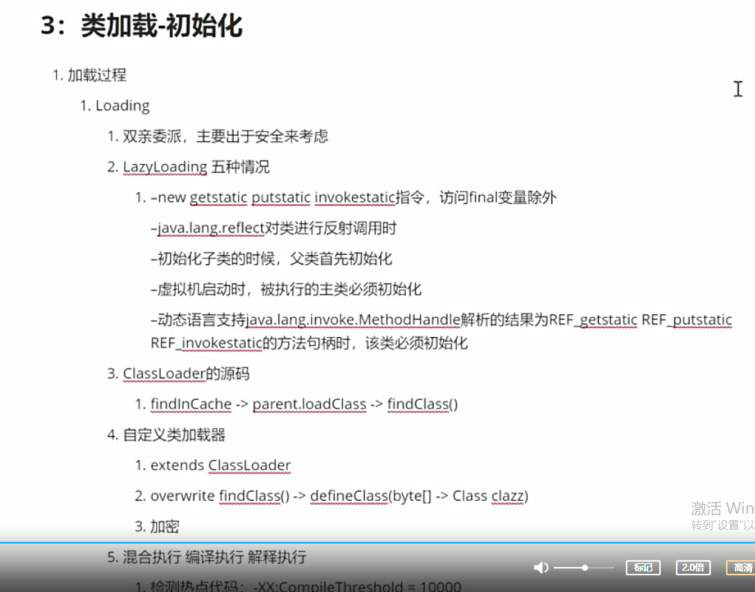
1. 为什么要使用双亲委派？

* 出于安全考虑。保证java核心api中定义类型不会被随意替换，假设通过网络传递一个名为java.lang.Integer的类，通过双亲委托模式传递到启动类加载器，而启动类加载器在核心Java API发现这个名字的类，发现该类已被加载，并不会重新加载网络传递的过来的java.lang.Integer，而直接返回已加载过的Integer.class，这样便可以防止核心API库被随意篡改。
* 避免重复加载。

1. 自定义类加载器

Bootstrap、Ext、App只能加载指定PATH的class，如果想要加载另外路径的class就得自定义一个类加载器去加载它。也可以对class进行加密解密处理。

重写findclass方法，不重写loadclass（双亲委派机制）。



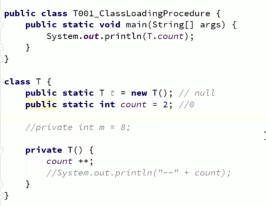


1. Linking、Initialzing



面试题

加载：静态变量，先左边（默认值），后右边（静态变量初始化，静态语句块）



Linking：

1. Verification
2. Preparation：为静态变量赋默认值
3. Resolution：解析

Initialzing：静态变量初始化、执行静态语句块