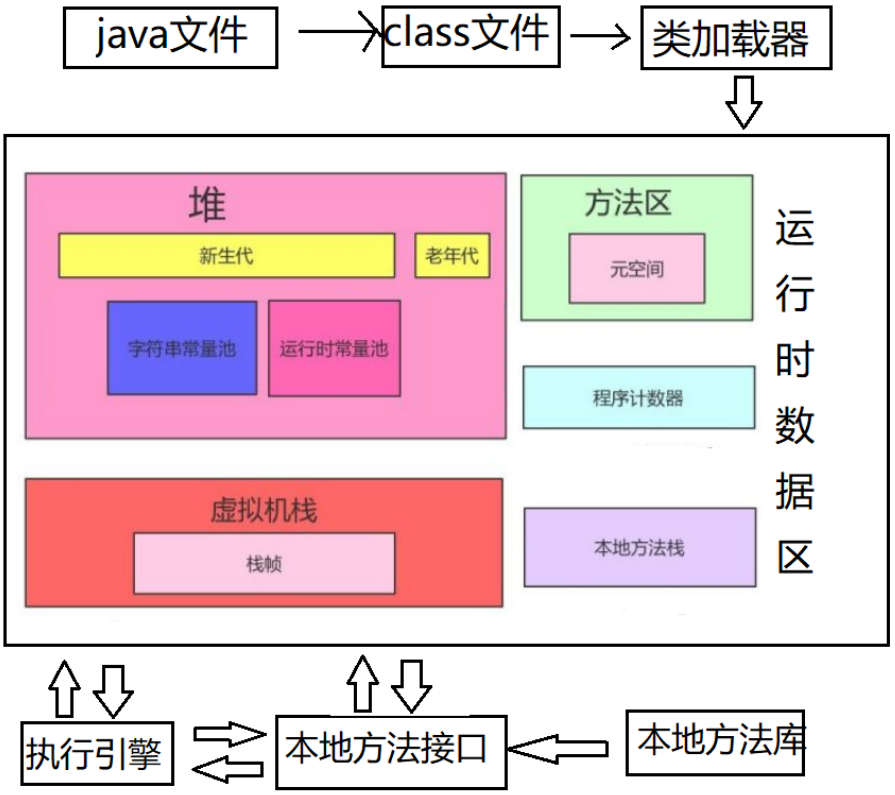
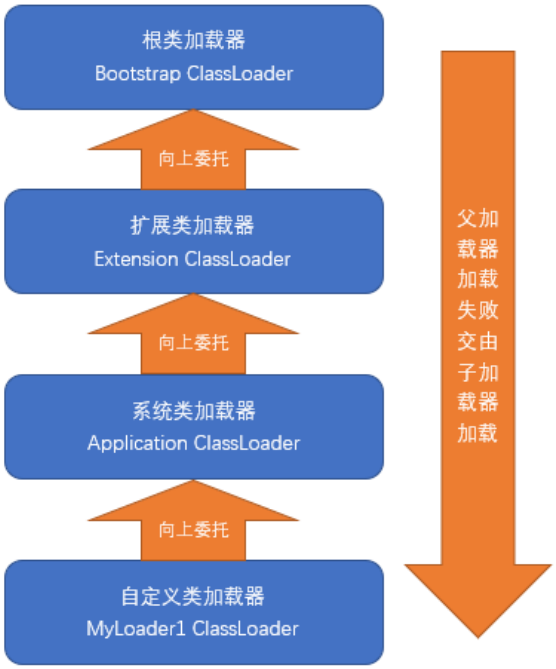
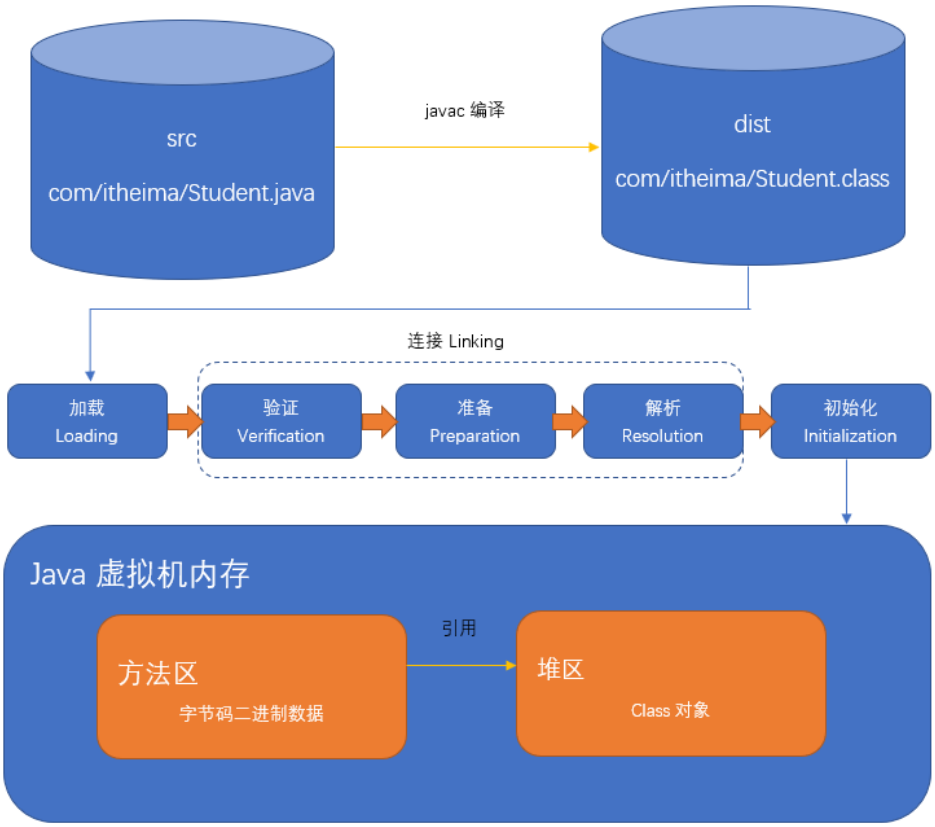
一.类加载过程



1. 我们所编写的*.java 文件，在经历 javac 命令后会变成*.class 文件，然后所有的信息都会封装在 class 文件里的静态常量池中，以二进制字节码的方式存储，这个时候 class 文件是在磁盘中的，而不是在内存中，class 文件包括魔数与 CLass 文件的版本、常量池、访问标志、类索引父类索引与接口索引集合、字段表集合、方法表集合、属性表集合，常量池只是其中的一部分而已。
2. 然后就是把磁盘中的文件加载到内存中，这个装载过程是用的类加载器，通过完全限定名（包名+类名）查找到二进制字节码文件，一般加载方式分为显式加载和隐式加载，显式加载是指在 java 代码中通过调用 ClassLoader 加载 class 对象，比如 Class.forName(String name)；this.getClass().getClassLoader().loadClass()加载类。隐式加载指不需要在 java 代码中明确调用加载的代码，而是通过虚拟机自动加载到内存中。比如在加载某个 class 时，该 class 引用了另外一个类的对象，那么这个对象的字节码文件就会被虚拟机自动加载到内存中，一般是通过 new 等关键字创建。
3. 然后就是不管是显式加载和隐式加载，都会用到双亲委派机制来加载类，即所要用到的类，不管是显式的强制加载还是隐式的需要就加载，都会利用双亲委派规则来依次加载到 JVM 的内存中，



其目的是（1）防止重复加载同一个.class。通过委托去向上面问一问，加载过了，就不用再加载一遍。保证数据安全。（2）保证核心.class 不能被篡改。通过委托方式，不会去篡改核心.class，即使篡改也不会去加载，即使加载也不会是同一个.class 对象了。不同的加载器加载同一个.class 也不是同一个 Class 对象。这样保证了 Class 执行安全。（因为刚开始基础类已经加载到 JVM 中，而且不同的加载器只处理特定目录下类，而且不同的加载器对类的处理方式也不同），关于加载后在内存的分布问题，类加载过程如下：



大体上分为：加载、连接、初始化三步，其中连接又分为验证、准备、解析，以某个类举例：

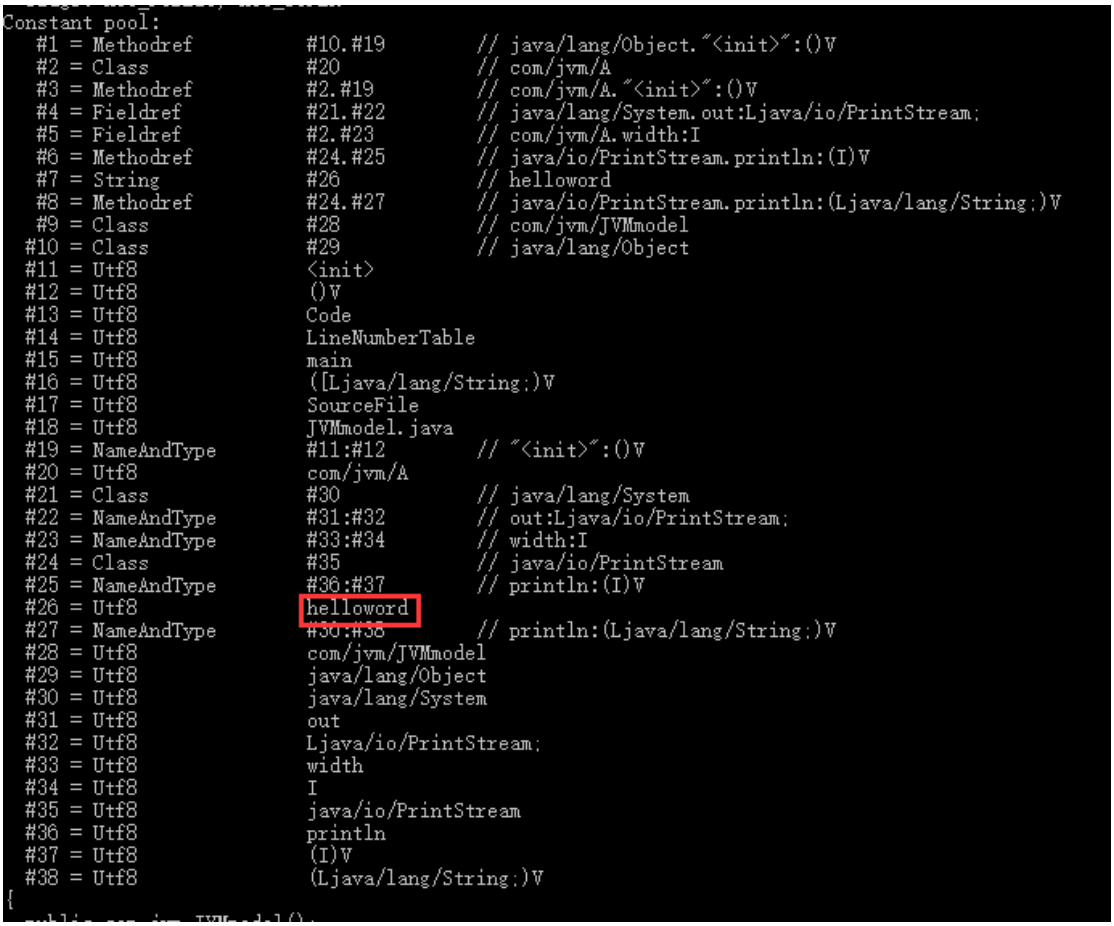
```
package com.jvm;

public class JVMmodel {
	public static void main(String[] args) {
		A a = new A();
		System.out.println(a.width);
		System.out.println(a.str);
	}
}
class A{
	public static final String str = "helloword";
	public static int width = 100;
	static {
		System.out.println("静态初始化类A");
		width = 300;
	}
	public A(){
		System.out.println("创建A类的对象");
	}
}
```

图中涉及到两个类，即使处在一个文件里面，每个类仍会编译成一个 class 文件，每个文件会有一个常量池
其中 JVMmodel 类对应的常量池为

字面量（其实就是一些变量的具体值）	文本字符串	Helloword（因为在 A 类中是用 final 限定的，所以编译阶段直接进入 JVMmodel 的常量池）
	被声明为 final 的常量值	
	基本数据类型	无
符号引用量（其实就是名字 :类名，方法名，变量名）	类和结构的完全限定名	包名+类名　如上图中 import 之后 JVMmodel、A、a、`System`
	字段名称	Width、str
	方法名称	Out、`println`
	描述符	Public、static、

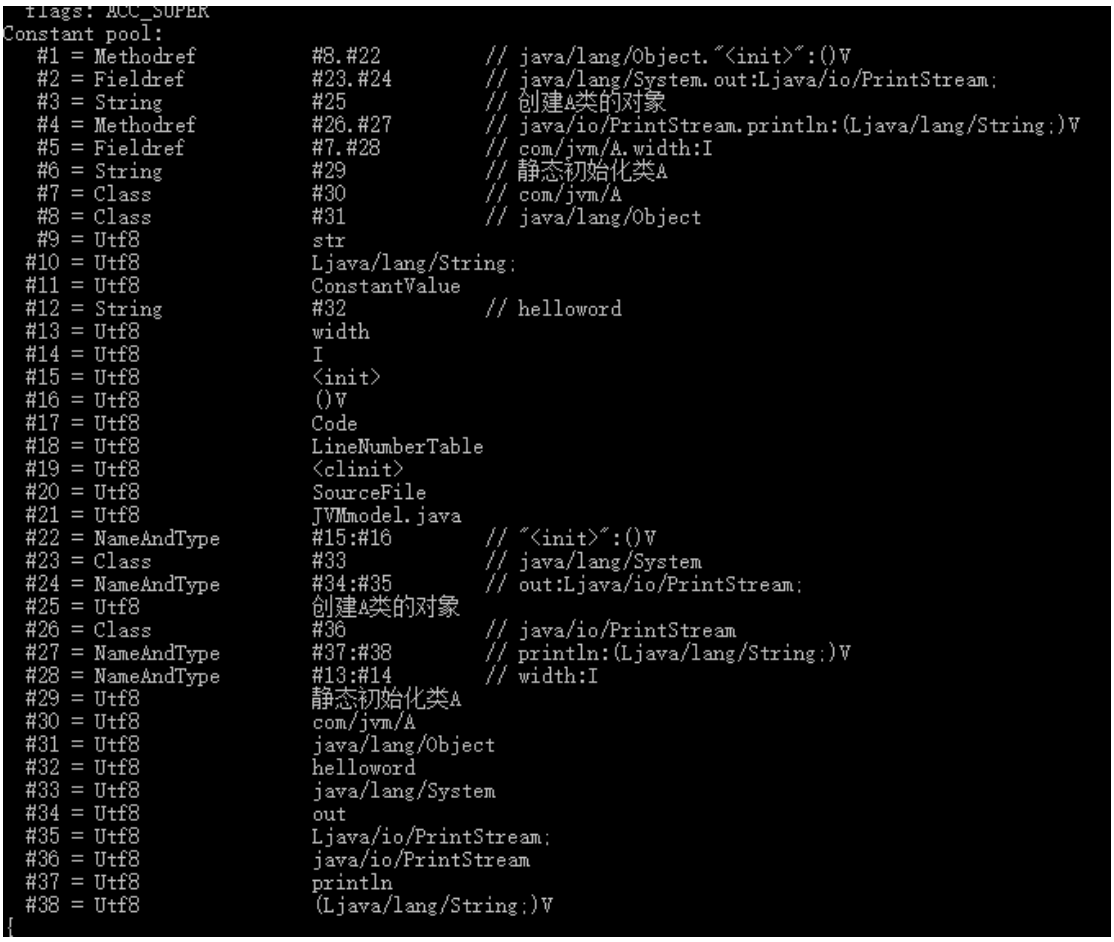
注意 :Helloword 因为是用 final 限定的，所以在编译阶段就会进入到 JVMmodel 的常量池中

下图中使用的是 javap –v 命令　如　javap –v　xxx.class



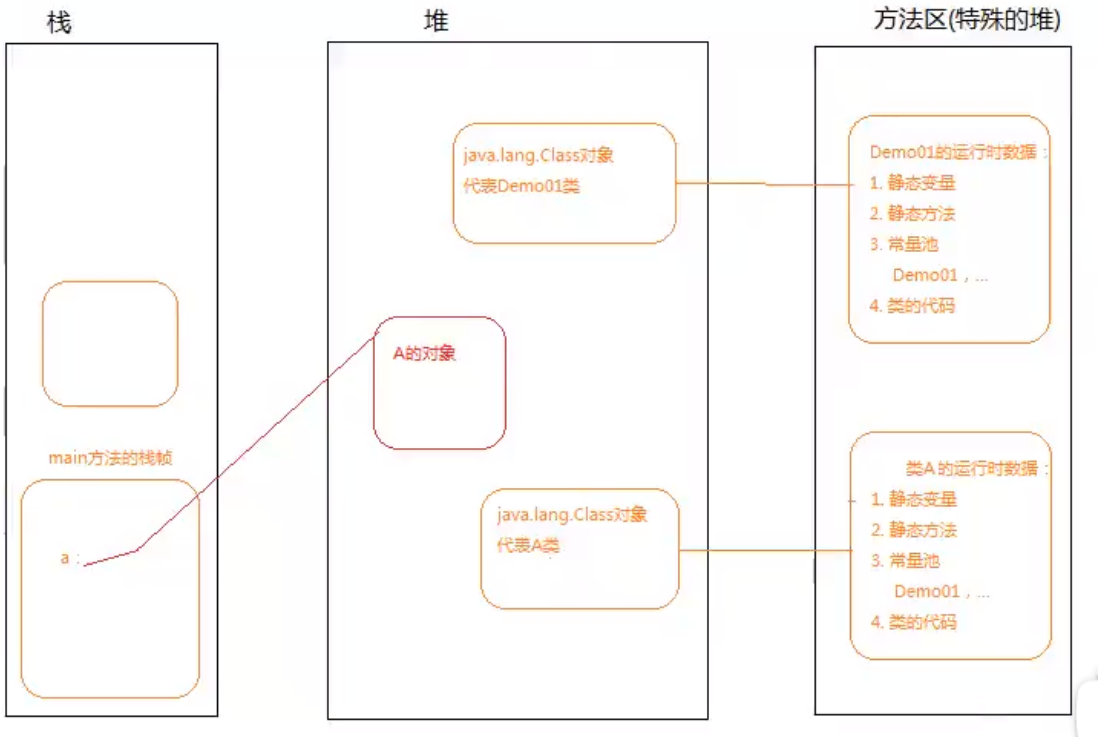
同理，A 的常量池为

字面量（其实就是一些变量的具体值）	文本字符串	helloword	静态初始化类 A、创建 A 类的对象
	被声明为 final 的常量值		
	基本数据类型	100、300	
符号引用量（其实就是名字：类名，方法名，变量名）	类和结构的完全限定名	包名+类名　如上图中 import 之后 A、`System`	
	字段名称	Width、str	
	方法名称	Out、`println`	
	描述符	Public、static、final	



编译成 class 文件后，就会利用类加载器进行加载，通过上面的不管是显式加载和隐式加载，利用双亲委派机制到了内存里面方法区中，这是 JVM 动态的分配的一块内存，他们在内存上的情况是这样的：

[1] Main 方法为入口方法，那么程序就会加载 JVMmodel 这个类，会将 class 文件加载到内存中，静态常量池也变成了动态常量池，存在方法区中（JDK1.8 搬到了堆中），并在堆中生成 JVMmodel 这个类的对象，作为访问常量池中 JVMmodel 的相关信息的入口，这也是反射操作数据的入口，同时会在栈中压入 main 方法的栈帧，并在栈中生成 A 类的引用，如果 A 类还没有加载，所以方法区会加载 A 类的信息到常量池中，在堆中生成 A 类的入口，也是为了方便访问 A 类的常量池信息，并在堆中生成 new a 的对象，指向方法区 A 的信息，同时栈中会压入 A 的构造函数，对类的所有变量通过访问堆中 A 的引用，从常量池中进行加载。如果有基本变量，也是通过这个 A 类的引用从常量池中加载。



[2] 栈中堆中的变量都是从常量池里获取的，因为栈和堆中的变量会被定时清理，所以常量池的存在可以使变量不必要的被重复创建。