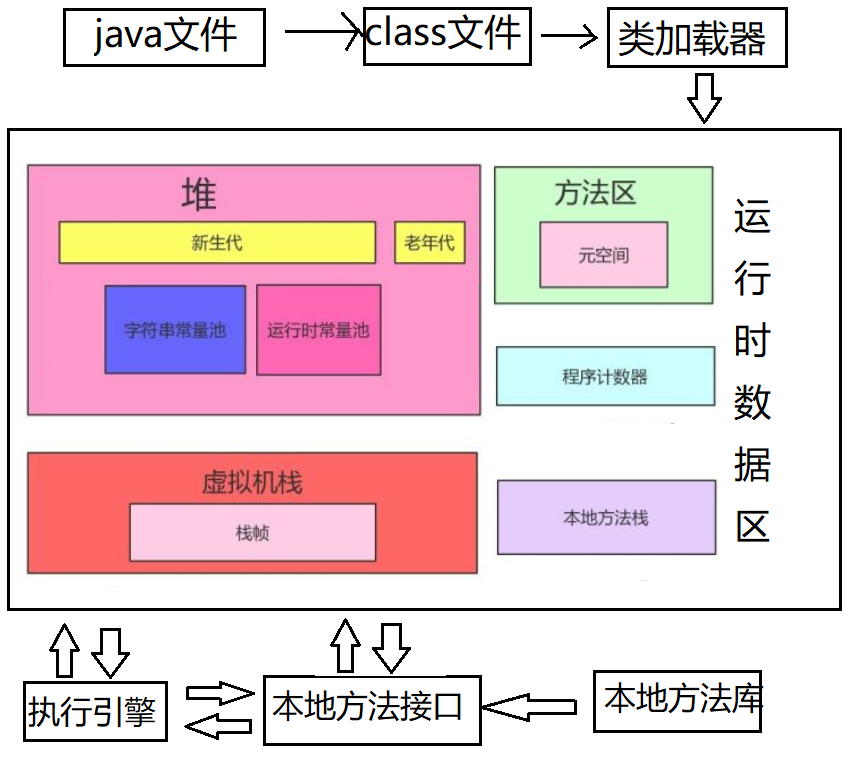
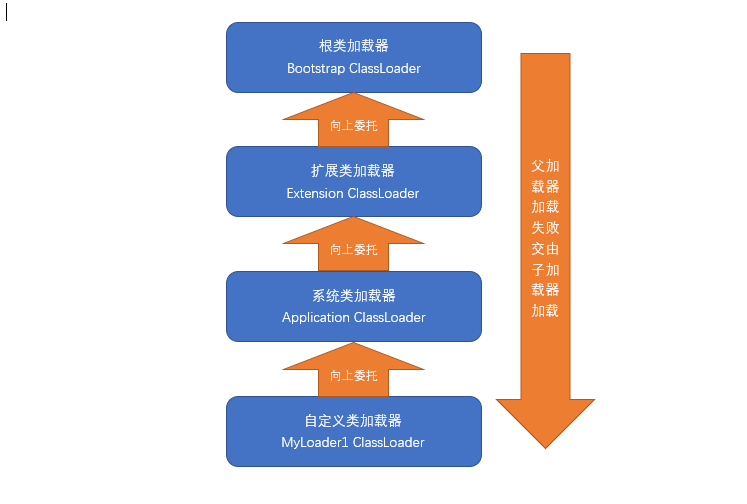
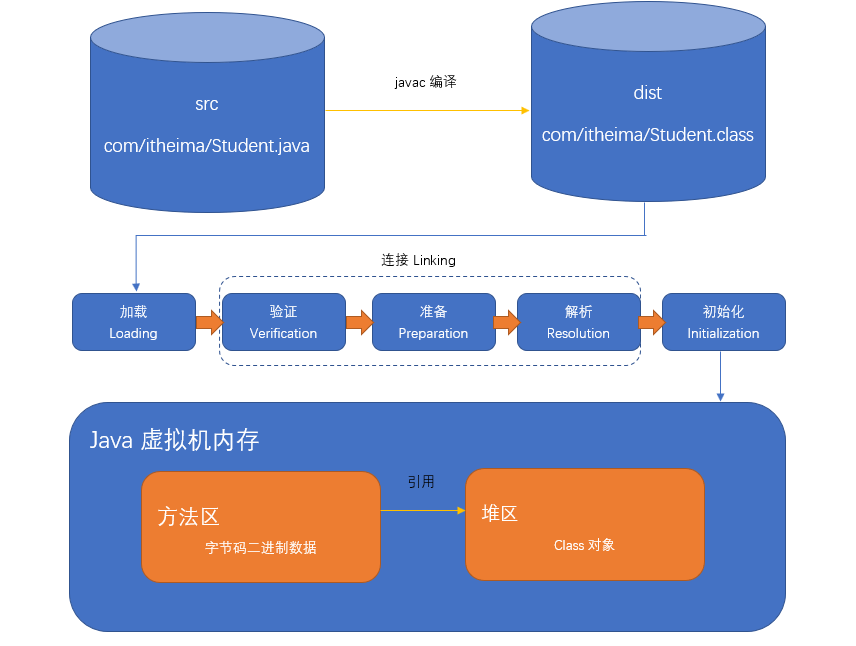
**一.类加载过程**



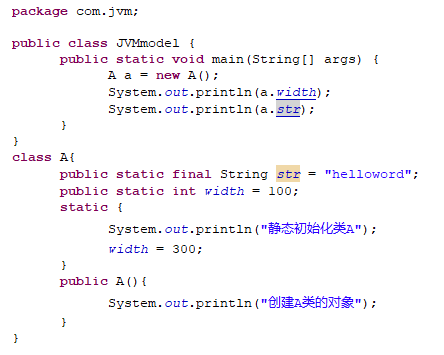
1. 我们所编写的\*.java文件，在经历javac命令后会变成\*.class文件，然后所有的信息都会封装在class文件里的静态常量池中，以二进制字节码的方式存储，这个时候class文件是在磁盘中的，而不是在内存中，class文件包括魔数与CLass文件的版本、常量池、访问标志、类索引父类索引与接口索引集合、字段表集合、方法表集合、属性表集合，常量池只是其中的一部分而已。
2. 然后就是把磁盘中的文件加载到内存中，这个装载过程是用的类加载器，通过完全限定名（包名+类名）查找到二进制字节码文件，一般加载方式分为显式加载和隐式加载，显式加载是指在java代码中通过调用ClassLoader加载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：



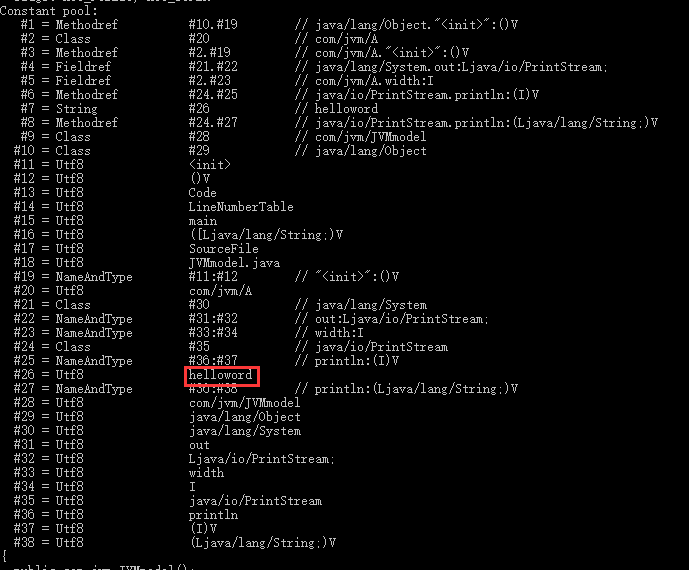
图中涉及到两个类，即使处在一个文件里面，每个类仍会编译成一个class文件，每个文件会有一个常量池

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



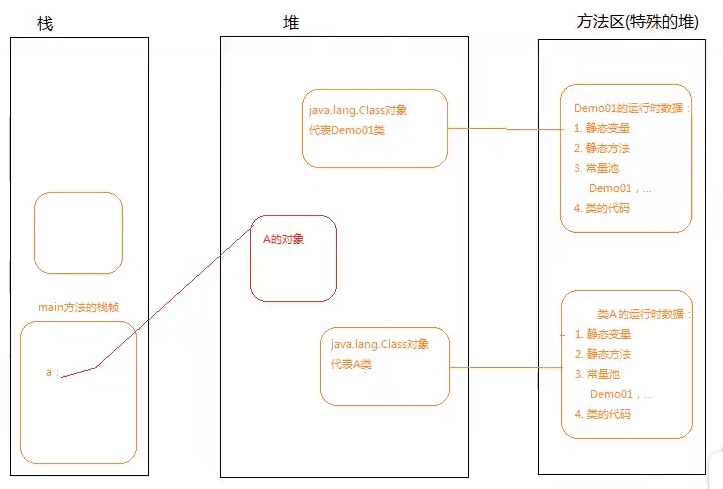
同理，A的常量池为

|  |  |  |  |
| --- | --- | --- | --- |
| 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。



1. 当main方法执行完毕后，会执行GC，将失去引用的变量进行垃圾回收操作。
2. 一个线程对应的是一个栈，但是堆却是多线程共享的，一个类对应一个常量池
3. 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。

**一.关于静态块为什么只加载一次**

* **首先，静态块是在类加载过程的准备阶段给其分配堆中的物理内存，即从磁盘到内存的过程中就开始分配内存了。**
* 而实例变量则会在内存中随着对象的调用而在堆中创建。
* 类只会在JVM中加载一次，所以静态块只会被加载一次，也只会执行一次，而实例变量的调用可以进行无数次。
* 静态块的执行顺序：父类静态构造快 → 子类静态构造块 → 父类构造块 → 父类构造函数 → 子类构造块 → 子类构造函数