首先是JDK

JDK(Java Development Kit) 是 Java 语言的软件开发工具包([SDK](http://baike.baidu.com/view/429424.htm))。**在JDK的安装目录下有一个jre目录**，里面有两个文件夹bin和lib，在这里可以认为**bin里的就是jvm，lib中则是jvm工作所需要的类库，而jvm和 lib合起来就称为jre。**

然后我们来看JRE

JRE（Java Runtime Environment，Java运行环境），**包含JVM标准实现及Java核心类库。**JRE是Java运行环境，并不是一个开发环境，所以没有包含任何开发工具（如编译器和调试器）

最后JVM也一目了然了

JVM是Java Virtual Machine（Java[虚拟机](http://baike.baidu.com/view/1132.htm)）的缩写，JVM是一种用于计算设备的规范，它是**一个虚构出来的计算机**，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。



由上图一目了然：

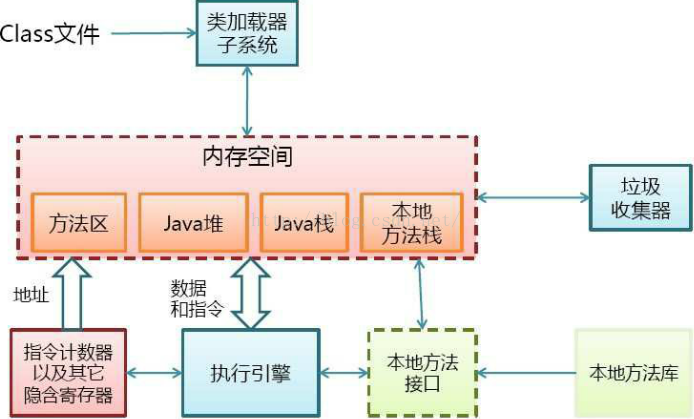
JDK是整个Java的核心，包括了Java运行环境JRE、Java工具和Java基础类库。JRE是运行JAVA程序所必须的环境的集合，包含JVM标准实现及Java核心类库。JVM是整个java实现跨平台的最核心的部分，能够运行以Java语言写的程序。

思考：Java是如何通过JVM实现跨平台的？

源程序编译的结果不是生成机器码，而是生成字节码，字节码不能直接运行，必须通过JVM翻译成机器码才能运行。不同平台下编译生成的字节码是一样的，但是由JVM翻译成的机器码却不一样。跨平台的是Java程序，不是JVM。JVM是用C/C++开发的，与各个平台是相关的。JVM编译后的机器码是不能跨平台的。

JVM体系结构与运行原理：

Java语言写的源程序通过Java编译器，编译成与平台无关的‘字节码程序’(.class文件，也就是0，1二进制程序)，然后在OS之上的Java解释器中解释执行。



**JVM整个类加载过程的步骤：**

**1.**       **装载**

装载过程负责找到二进制字节码并加载至JVM中，JVM通过类名、类所在的包名通过ClassLoader来完成类的加载，同样，也采用以上三个元素来标识一个被加载了的类：类名+包名+ClassLoader实例ID。

**2.**       **链接**

链接过程负责对二进制字节码的格式进行校验、初始化装载类中的静态变量以及解析类中调用的接口、类。

完成校验后，JVM初始化类中的静态变量，并将其值赋为默认值。

最后对类中的所有属性、方法进行验证，以确保其需要调用的属性、方法存在，以及具备应的权限（例如public、private域权限等），会造成NoSuchMethodError、NoSuchFieldError等错误信息。

**3.**       **初始化**

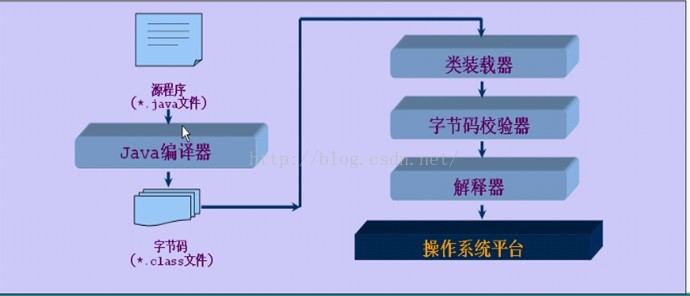
初始化过程即为执行类中的静态初始化代码、构造器代码以及静态属性的初始化，在四种情况下初始化过程会被触发执行：

调用了new；

反射调用了类中的方法；

子类调用了初始化；

JVM启动过程中指定的初始化类。



**ClassLoader抽象类的几个关键方法：**

（1）       loadClass

此方法负责加载指定名字的类，ClassLoader的实现方法为先从已经加载的类中寻找，如没有则继续从parent ClassLoader中寻找，如仍然没找到，则从System ClassLoader中寻找，最后再调用findClass方法来寻找，如要改变类的加载顺序，则可覆盖此方法

（2）       findLoadedClass

此方法负责从当前ClassLoader实例对象的缓存中寻找已加载的类，调用的为native的方法。

（3）       findClass

此方法直接抛出ClassNotFoundException，因此需要通过覆盖loadClass或此方法来以自定义的方式加载相应的类。

（4）       findSystemClass

此方法负责从System ClassLoader中寻找类，如未找到，则继续从Bootstrap ClassLoader中寻找，如仍然为找到，则返回null。

（5）       defineClass

此方法负责将二进制的字节码转换为Class对象

（6）       resolveClass

此方法负责完成Class对象的链接，如已链接过，则会直接返回。

**JVM运行时数据区：**

**第一块：PC寄存器**

PC寄存器是用于存储每个线程下一步将执行的JVM指令，如该方法为native的，则PC寄存器中不存储任何信息。

**第二块：JVM栈**

JVM栈是线程私有的，每个线程创建的同时都会创建JVM栈，JVM栈中存放的为当前线程中局部基本类型的变量（java中定义的八种基本类型：boolean、char、byte、short、int、long、float、double）、部分的返回结果以及Stack Frame，非基本类型的对象在JVM栈上仅存放一个指向堆上的地址

**第三块：堆（Heap）**

它是JVM用来存储对象实例以及数组值的区域，可以认为Java中所有通过new创建的对象的内存都在此分配，Heap中的对象的内存需要等待GC进行回收。

**第四块：方法区域（Method Area）**

（1）在Sun JDK中这块区域对应的为PermanetGeneration，又称为持久代。

（2）方法区域存放了所加载的类的信息（名称、修饰符等）、类中的静态变量、类中定义为final类型的常量、类中的Field信息、类中的方法信息，当开发人员在程序中通过Class对象中的getName、isInterface等方法来获取信息时，这些数据都来源于方法区域，同时方法区域也是全局共享的，在一定的条件下它也会被GC，当方法区域需要使用的内存超过其允许的大小时，会抛出OutOfMemory的错误信息。

**第五块：运行时常量池（Runtime Constant Pool）**

存放的为类中的固定的常量信息、方法和Field的引用信息等，其空间从方法区域中分配。

**第六块：本地方法堆栈（Native Method Stacks）**

JVM采用本地方法堆栈来支持native方法的执行，此区域用于存储每个native方法调用的状态。

思考：

String s1 = “hello”;

String s2 = “hello”;

在这个过程中共创建了几个对象？

String s3 = new String(“hello girl”);

String s4 = new String(“hello girl”);

在这个过程中共创建了几个对象？

类加载器是将什么什么加载到JVM的哪个内存区？