1、Java 中 JDK、JRE、JVM 是什么？

1）**JDK**：英文全称 Java Development Kit，是 **Java 的开发工具包**， JDK 是提供给 Java 开发人员使用的，其中包含了 **Java 的开发工具**（**编译**工具（javac.exe）、**打包**工具（jar.exe）等）和 **JRE**。通俗的说就是**开发用的**。

2）**JRE**：英文全称 Java Runtime Environment，是 **Java 运行环境**，JRE 包括 **Java 虚拟机**(**JVM** Java Virtual Machine)和 **Java 程序所需的核心类库**等，如果想要**运行一个开发好的 Java 程序**，计算机中**只需要安装 JRE** 即可。通俗的说就是**运行用的**。

3）**JVM**：英文全称 Java Virtual Machine，是 **java 虚拟机**。它是整个 java 实现**跨平台的**最核心的部分，负责**解释执行**字节码文件，是可以**运行 java 字节码文件**的虚拟计算机。**不同平台上的 JVM** 向编译器提供**相同的接口**，而**编译器**只需要**面向 JVM 虚拟机**，**生成**虚拟机能识别的代码（即与平台无关的**.class 字节码文件**），然后由虚拟机来解释执行，**编译后的字节码文件**就可以**在该平台上运行**。

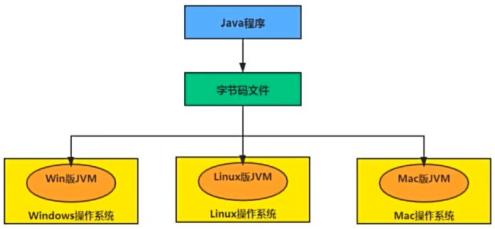
关系：

JDK=JRE +开发工具集(如 Javac 编译工具等)

JRE = JVM + Java SE 标准类库

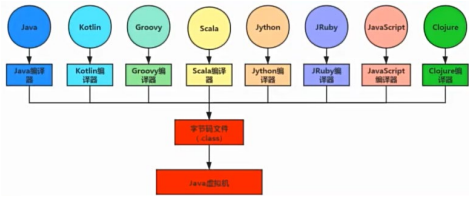
2、如何理解 java 是跨平台语言？

当 **java 源代码**成功编译成**字节码**后，如果想在**不同平台上**运行，则**无须**再次编译，**只要**在不同的操作系统上**装不同操作系统的 JVM**，即可实现一个 java 程序在不同 os 上运行。



3、如何理解 JVM 是跨语言的平台？

Java 虚拟机根本**不关心**运行在其内部的程序到底是**使用哪种编程语言编写的**，它**只关心字节码文件**，只要各种语言自己的编译器能够**遵循** java 虚拟机规范**生成** java 虚拟机能够识别的字节码文件即可在 java 虚拟机上运行。



4、JVM 举例：

1）SUN 公司的 HotSpot VM

2）BEA 的 JRockit

3）IBM 的 J9

4）TaobaoJVM

5、JVM 的生命周期？

1）虚拟机的**启动**：java 虚拟机的启动是通过**引导类加载器**创建一个**初始类**来完成的，这个类是由虚拟机的具体实现指定的。

2）虚拟机的**退出有如下几种情况**：

1> 某线程**调用** Runtime 类或 System 类的 **exit 方法**或者 Runtime 类的 **halt 方法**，并且 **java 安全管理器**也**允许**这次 exit 或 halt 操作；

2> 程序**正常执行结束**；

3> 程序在执行过程中遇到了**异常或错误**而**异常终止**；

4> 由于**操作系统出现错误**而导致 java 虚拟机**进程终止**。

6、字节码文件是跨平台的嘛？

是的，java 虚拟机不和包括 java 在内的任何语言绑定，它**只与 class 文件**这种特定的二进制文件格式**所关联**。无论使用哪种语言进行软件开发，只要能将**源文件编译**为正确的 **class 文件**，那么这种语言就可以在 java 虚拟机上执行。即，统一而强大的 **class 文件结构**是 **java 虚拟机**的**基石、桥梁**。

7、class 文件里是什么？

源代码经过**编译器编译**后便会**生成一个字节码文件**，字节码文件是一种**二进制的类文件**，它的内容是 **JVM 的指令**，而**不像 C、C++**经由编译器直接**生成机器码**。

8、能介绍一下生成 class 文件的编译器嘛？

前端编译器主要是负责将符合 java 语言规范的 **java 代码**转换为符合 JVM 规范的**字节码文件**。**javac** 是一种能够将 java 源码编译为字节码的前端编译器，**配置在 path 环境变量**中。

【注意】java 是**半编译半解释型**语言是指**后端**编译器(JIT)可以寻找热点代码**及时编译存在缓存中**供后面使用，**提高效率**。

9、哪些类型对应有 Class 对象？

1）class：外部类，成员（成员内部类/静态内部类），局部内部类，匿名内部类

2）interface：接口

3）[]：数组

4）enum：枚举

5）annotation：注解@interface

6）primitive type：基本数据类型

7）void

```
int[] a = new int[10];
int[] b = new int[100];
Class c10 = a.getClass();
Class c11 = b.getClass();
// 只要元素类型与维度一样，就是同一个Class
System.out.println(c10 == c11);
```

【注意】只要元素**类型**和**维度**一样，就是同一个 Class。

10、包装类对象的缓存问题

包装类	缓存对象
Byte	-128~127
Short	-128~127
Integer	-128~127
Long	-128~127
Float	没有
Double	没有
Character	0~127
Boolean	true 和 fasle

11、class 文件结构有哪些部分？

1）**魔数**：确定这个文件**是否为**一个**能被虚拟机**接受的**有效合法 class 文件**；

2）**class 文件版本**（向下兼容）

3）**常量池**（class 文件的**基石**）：**class 文件的资源仓库**，是 class 文件结构中**关联最多的数据类型**，也是占用 class 文件最大的数据项目之一。（字面量和符号引用）

4）访问标识（或标志）：表示**该 class 的属性和访问类型**，比如该 class 是类还是接口，访问类型是否为 public，类型是否被标记为 final；

5）**类索引，父类索引，接口索引集合**：class 文件靠类索引、父类索引和接口索引这三个数据项数据来**确定这个类的继承**；

6）**字段表集合**：用于描述**接口或类中声明的变量**。比如变量的作用域、是否为静态变量、数据类型等；

7）**方法表集合**：用于描述方法的类型和作用于等；

8）**属性表集合**：用于描述**某些场景专有信息**，如字段表中特殊的属性、方法表中特殊的属性等。

12、魔数是什么？

class 文件的标志，每个 class 文件**开头的 4 个字节**的**无符号整数**称为魔数。它的唯一作用是确定这个文件**是否为**一个**能被虚拟机**接受的**有效合法 class 文件**。

【注意】**魔数值**固定为 0xCAFEBABE **不会变**；使用魔数而**不是扩展名**来进行识别主要是基于安全方面考虑，因为**扩展名可以随意改动**。

13、为什么需要常量池计数器？

常量池数量不固定，时长时短，所以需要放置**两个字节**来表示**常量池容量计数值**。

【注意】常量池计数值：**从 1 开始计数**，表示常量池中**有多少项常量**。例如其值为 0x0016（也就是 22），实际上只有 21 项常量，索引范围是 1-21。**原因**：它**把第 0 项常量空出来**了，这是为了满足后面某些指向常量池的索引值的数据在特定情况下需要表达“**不引用任何一个常量池项**”的含义，这种情况可**用索引值 0 来表示**。

14、常量池（表）constant_pool [ ]

constant_pool 是一种**表结构**，以 **1 ~ constant_pool_count - 1** 为索引。表明了后面有多少个常量项。

常量池主要**存放两大类**常量：**字面量**（如文本字符串，final 常量等）和**符号引用**（类和接口的全限定名、字段的名称和描述符号、方法的名称和描述符）

它包含了 class 文件结构及其子结构中引用的所有字符串常量、类或接口名、字段名和其他常量。常量池中的**每一项都**具备相同的特征。**第 1 个字节**作为**类型标记**，用于**确定**该项的格式，这个字节称为 tag byte （标记字节、标签字节）。

15、谈谈你对符号引用、直接引用的理解？

Java 代码在进行 Javac 编译的时候，并不像 C 和 C++那样有“连接”这一步骤，而是在虚拟机**加载 Class 文件时**才会进行**动态链接**，也就是说，**Class 文件中不会保存**各个方法和字段的**最终内存布局信息**，因此，这些字段和方法的**符号引用**不经过转换是**无法直接被虚拟机使用**的。当**虚拟机运行时**，需要**从常量池中**获得对应的**符号引用**，再在**类加载过程中**的**解析阶段**将其**替换为直接引用**，并**翻译**到**具体的内存地址**中。

【注意】符号引用和直接引用的区别与关联：

符号引用：符号引用**以一组符号**来**描述所引用的目标**，符号可以是**任何形式的字面量**，只要使用时能无歧义地定位到目标即可。符号引用与虚拟机实现的**内存布局无关**，引用的目标**并不一定**已经**加载到了内存**中。

直接引用：直接引用可以是直接指向目标的**指针**、**相对偏移量**或是一个能**间接定位到目标的句柄**。直接引用是与虚拟机实现的**内存布局相关的**，**同一个符号引用**在不同虚拟机实例上**翻译**出来的**直接引用一般不会相同**。如果**有了直接引用**，那说明**引用的目标**必定**已经存在于内存之中**了。

16、方法调用指令

1）**invokevirtual** 指令用于调用对象的**实例方法**，根据对象的实际类型进行分派（虚方法分派），支持多态。这也是 Java 语言中最常见的方法分派方式。

2）**invokeinterface** 指令用于调用**接口方法**，它会在运行时搜索由特定对象所实现的这个接口方法，并找出适合的方法进行调用。

3）**invokespecial** 指令用于调用**一些需要特殊处理的实例方法**，包括**实例初始化方法**（构造器）、**私有方法**和**父类方法**。这些方法**都是静态类型绑定的**，不会在调用时进行动态派发。

4）**invokestatic** 指令用于调用命名类中的**类方法**（static 方法）。这是**静态绑定**的。

5）**invokedynamic**：调用**动态绑定**的方法，这个是 JDK 1.7 后新加入的指令。用于在运行时动态解析出**调用点限定符**所引用的方法，并执行该方法。前面 4 条调用指令的分派逻辑都固化在 java 虚拟机内部，而 invokedynamic 指令的分派逻辑是由**用户所设定的引导方法**决定的。

17、为什么不把基本类型放堆中呢？

1）首先是栈、堆的特点不同，**堆比栈要大**，但是**栈比堆**的**运算速度要快**；

2）将**复杂数据类型**放在**堆中**的目的是为了**不影响栈的效率**，而是**通过引用**的方式**去堆中查找**；（**八大基本类型**创建时候已经**确立大小**，**三大引用类型**创建时候**无法确定大小**）

3）**简单数据类型**比较**稳定**，并且它只**占据很小的内存**，将它放在**空间小、运算速度快**的***栈***中，能够**提高效率**。

18、java 中有指针的概念嘛？

Java 中没有指针的操作，所有的对象都是**通过引用来访问的**，引用是一个**指向对象内存地址的值**，它指向对象在堆上的位置。开发人员可以通过**引用**来**访问对象的属性和方法**，但**不能直接访问**内存地址，使得它们更加**安全**和易于使用。