**Java跨平台原理**

Java程序不是直接在电脑上运行的，**是在虚拟机上运行的，JVM也是一个软件**，不同的平台有不同的版本。JVM就是负责将字节码文件翻译成特定平台下的机器码然后运行。Java代码首先被编译成字节码文件，再由JVM将字节码文件翻译成机器语言，**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。即使将Java程序打包成可执行文件（例如 .exe），仍然需要JVM的支持。注意：**跨平台的是Java程序，不是JVM**。JVM是用C/C++开发的，是编译后的机器码，不能跨平台，不同平台下需要安装不同版本的JVM。

**JVM：**当我们运行一个程序时，JVM负责将字节码转换为特定机器代码，JVM提供了内存管理/垃圾回收和安全机制等。这种独立于硬件和操作系统，正是Java程序可以一次编写多处执行的原因。

**在JVM内存里面主要分布有以下三个区域：**

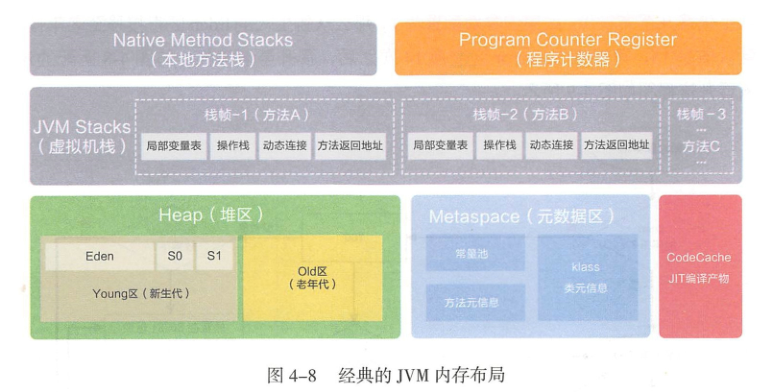
**栈**：存放基础数据和自定义对象的引用

**堆**：主要存储创建的对象，即new出来的对象。

**方法区**：加载存放class文件（字节码文件）

**JVM内存分布**

内存是硬盘和CPU的中间仓库及桥梁，承载着操作系统和应用程序的实时运行。JVM内存布局规定了Java在运行过程中内存申请、分配、管理的策略，保证了JVM的高效稳定运行。不同的JVM对于内存的划分方式和管理机制存在着部分差异。



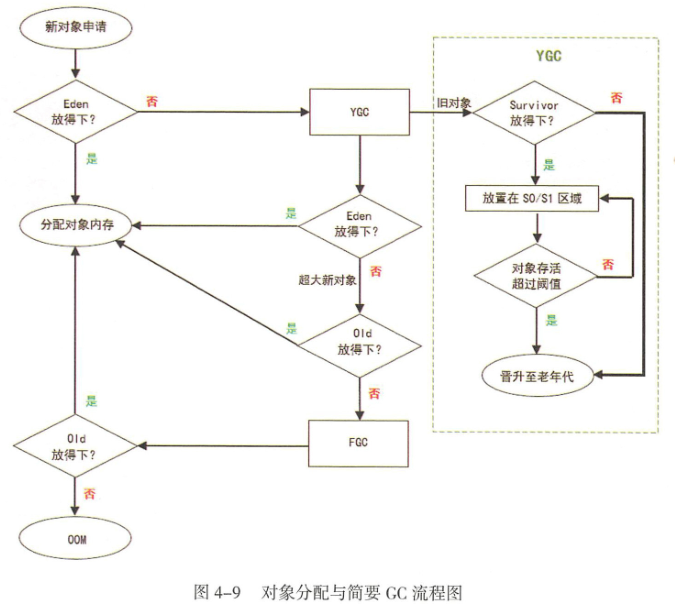
**堆区（Heap）**

**堆区是OOM故障最主要的发源地，它存储着几乎所有的实例对象，堆由垃圾收集器自动回收，堆区由各子线程共享使用。**通常情况下，它占用的空间是所有内存区域中最大的，但如果无节制的创建大量对象，也容易消耗完所有空间。**堆的内存空间可以固定大小，也可以在运行时动态的调整。**可以通过下列参数设定初始值和最大值：**-Xms256M –Xms1024M，其中-X表示它是JVM运行参数，ms是memory start的简称，mx是memory max的简称，分别代表最小堆容量和最大堆容量**。但在通常情况下，服务器在运行过程中，堆空间不断地扩容与收缩，势必造成不必要的系统压力，所以在线上生产环境中，JVM的**Xms和Xmx设置成一样大小**，避免在GC后调整堆大小时带来的额外压力。

**新生代和老年代**堆分为两大块：新生代和老年代。**对象产生之初在新生代，步入暮年时进入老年代，但是老年代也接纳在新生代无法容纳的超大对象。新生代=1个Eden区+2个Survivor区。**绝大部分对象在Eden区产生，当Eden区装填满的时候，会触发Young Garbage Collection，即YGC。垃圾回收的时候，在Eden区实现清除策略，没有被引用的对象则直接回收。依然存活的对象会被移送到Survivor区。

**Survivor区分为S0和S1两块内存空间**，送到哪块空间呢？每次YGC的时候，将存活的对象复制到未使用的那块空间，然后将当前正在使用的空间完全清除，交换两块空间的使用状态。如果YGC要移送的对象大于Survivor区容量的上限，或者超大对象的阈值超过上限，则直接移交给老年代。如果老年代也无法放下，则会触发FGC（Full Garbage Collection）。如果依然无法放下，则抛出OOM。堆内存出现OOM的概率是所有内存耗尽异常中最高的。出错时的堆内信息对解决问题非常有帮助所以给JVM设置运行参数：**-XX:+HeapDumpOnOutOfMemoryError**，让JVM遇到OOM异常时能输出堆内信息，特别是对相隔数月才出现的OOM异常尤为重要。假如一些没有进取心的对象以为可以一直在新生代的Survivor区交换来交去，那就错了。每个对象都有一个计数器，每次YGC都会加1。

**-XX:MaxTenuringThreshold**参数能配置计数器的值到达某个阈值的时候，对象从新生代晋升至老年代。如果该参数配置为1，那么从新生代的Eden区直接移至老年代。默认值15，可以在Survivor区交换14次之后晋升至老年代。在不同的JVM实现及不同的回收机制中，堆内存的划分方式是不一样的。



**元空间（Metaspace）**

JDK8之后，元空间的前身Perm区已经被淘汰，译为永久代，在启动时固定大小，很难进行调优，并且FGC时会移动类元信息。区别于永久代，元空间在本地内存中分配。在JDK8中，Perm区的所有内容中字符串常量移至堆内存，其他内容包括类元信息、字段、静态属性、方法、常量等都移至元空间内。

**虚拟机栈（JVM Stack）**

**栈是一个先进后出的数据结构**，相当于基于寄存器的运行环境来说，JVM是基于栈结构的运行环境。栈结构移植性更好，可控性更强。JVM中的虚拟机栈是描述Java方法执行的内存区域，它是线程私有的。栈中的元素用于支持虚拟机进行方法调用，**每个方法从开始调用到执行完成的过程，就是栈帧从入栈到出栈的过程**。在活动线程中，只有位于栈顶的帧才是有效的，称为当前栈帧。正在执行的方法称为当前方法，栈帧是方法运行的基本结构。在执行引擎运行时，所有指令都只能针对当前栈帧进行操作。而StackOverflowError表示请求的栈溢出，导致内存耗尽，通常出现在递归方法中。

虚拟机通过压栈和出栈的方式，对每个方法对应的活动栈帧进行运算处理，方法正常执行结束，肯定会跳转到另一个栈帧上。在执行的过程中，如果出现异常，会进行异常回溯，返回地址通过异常处理表确定。栈帧在整个JVM的地位很高，包括局部变量表、操作栈、动态连接、方法返回地址等。

**局部变量表**

局部变量表是存放方法参数和局部变量的区域。相对于类属性变量的准备阶段和初始化阶段来说，局部变量没有准备阶段，必须显式初始化。如果是非静态方法，则在index[0]位置上存储的是方法所属对象的实例引用，随后存储的是参数和局部变量。字节码指令中的STORE指令就是将操作栈中计算完成的局部变量写回局部变量表的存储空间内。

**操作栈**

操作栈是一个初始状态为空的桶式结构栈。在方法执行过程中，会有各种指令往栈中写入和提取信息。JVM的执行引擎是基于栈的执行引擎，其中的栈指的就是操作栈。字节码指令集的定义都是基于栈类型的，栈的深度在方法元信息的stack属性中。栈是一个很深的竖桶，任何时候只能对桶口元素进行操作，所以数据只能在栈顶进行存取。

**动态连接**

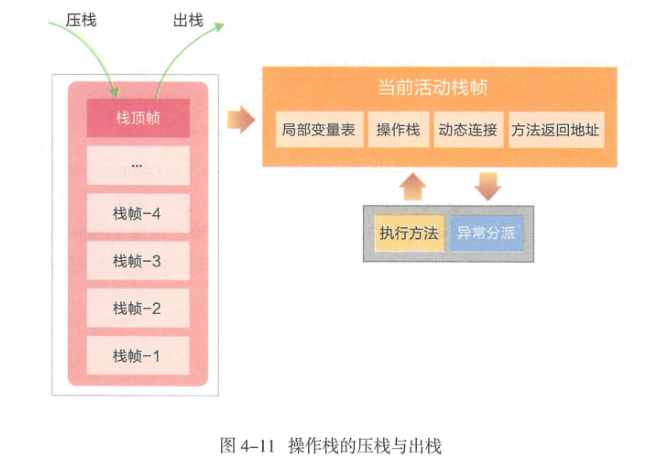
每个栈帧中包含一个在常量池中对当前方法的引用，目的是支持方法调用过程的动态连接。

**方法返回地址**

方法执行时有两种退出情况：第一，正常退出，即正常执行到任何方法的返回字节码指令RETURN、IRETURN ARETURN等；第二，异常退出。无论何种退出情况，都将返回至方法当前被调用的位置。方法退出的过程相当于弹出当前栈帧。

**退出可能有三种方式**

**返回值压入上层调用栈帧。异常信息抛给能够处理的栈帧。PC计数器指向方法调用后的下一条指令。**



**本地方法栈（Native Method Stacks）**

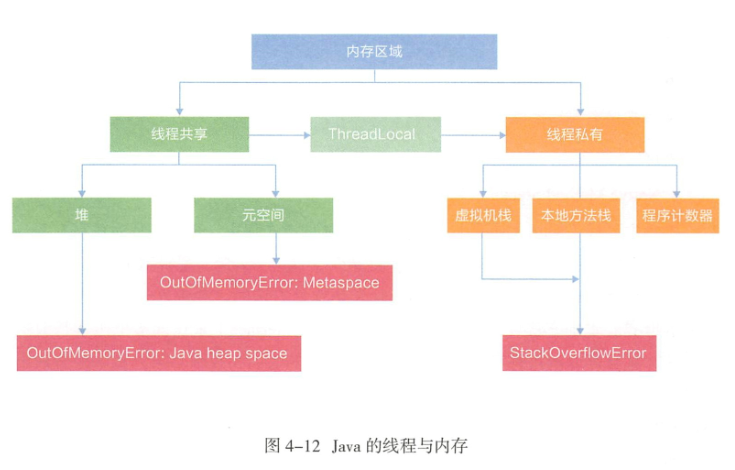
本地方法栈在JVM内存布局中，也是线程对象私有的，但是虚拟机栈“主内 ”，而本地方法栈“主外”。这个“内外”是针对JVM来说的，本地方法栈为Native方法服务。线程开始调用本地方法时，会进入一个不再受JVM约束的世界。本地方法可以通过JNI ( Java Native Interface）来访问虚拟机运行时的数据区，甚至可以调用寄存器，具有和JVM相同的能力和权限。当大量本地方法出现时，势必会削弱JVM对系统的控制力，因为它的出错信息都比较黑盒。对于内存不足的情况 本地方法枝还是会抛出 native heap OutOfMemory。

重点说一下JNI类本地方法，最著名的本地方法应该是System.currentTimeMillis() , JNI使Java深度使用操作系统的特性功能，复用非Java代码。但是在项目过程中，如果大量使用其他语言来实现JNI就会丧失跨平台特性，威胁到程序运行的稳定性。假如需要与本地代码交互，就可以用中间标准框架进行解辑，这样即使本地方法崩溃也不至于影响到JVM的稳定。当然，如果要求极高的执行效率、偏底层的跨进程操作等，可以考虑设计为JNI调用方式。

**程序计数寄存器（Program Counter Register）**

在程序计数寄存器中，Register的命名源于CPU的寄存器，CPU只有把数据装载到寄存器才能运行。寄存器存储指令相关的现场信息，由于CPU时间片轮限制，众多线程在并发执行过程中，任何一个确定的时刻，一个处理器或者多核处理器中的一个内核，只会执行某个线程中的一条指令。这样必会导致经常中断或恢复。要想保持分毫不差，在每个线程创建后，都会产生自己的程序计数器和栈帧，程序计数器用来存放执行指令的偏移量和行号指示器等，线程执行或恢复都要依赖程序计数器。程序计数器在各个线程之间互不影响，此区域也不会发生内存溢出异常。

**从线程共享的角度来看，堆和元空间是所有线程共享的，而虚拟机栈、本地方法栈、程序计数器是线程内部私有的。**

****

**Java的加载与执行**

对于Java程序员来说，主要做的事情就是编写以.Java结尾的文件，我们称之为源文件，把在源文件里面所编写的代码叫做源代码。但是这个.Java结尾的文件是不能直接运行的，需要使用Javac.exe命令编译成以.class结尾的文件，这个文件夹叫做字节码文件。要想运行这个字节码文件的话，需要使用Java.exe命令，之后会启动JVM，JVM中的类装载器会去电脑硬盘里面寻找这个class文件，然后将它翻译成2进制文件交由操作系统去运行。

**JDK**：编写Java程序的程序员使用的软件（Java Development Kit）

**JRE**：运行Java程序的用户使用的软件（Java Runtime Environment）

**SE**：用于桌面或简单服务器应用的Java平台（Standard Edition）标准版

**EE**：用于复杂服务器应用的Java平台（Enterprise Edition）企业版

**ME**：用于手机或其他小型设备的Java平台（Micro Edition）小型版

**Java特点**

简单性、面向对象、分布式、健壮性、安全性、体系结构中立、可移植性、解释型、高性能、多线程、动态性

**Java和C++的区别**

都是面向对象的语言，都支持封装、继承和多态

Java 不提供指针来直接访问内存，程序内存更加安全

Java 的类是单继承的，C++ 支持多重继承；虽然 Java 的类不可以多继承，但是接口可以多继承。

Java 有自动内存管理机制，不需要程序员手动释放无用内存

**Java和C区别**

C语言是面向过程的语言，执行效率高；Java是面向对象的语言，执行效率比C语言低；

C语言的安全性不如Java，C语言没有Java的垃圾回收机制，申请的空间要手动释放。

Java的通用性好，可以跨平台直接移植，只要有安装Java虚拟机（JVM）就可以了。

C语言的变量可以不进行初始化，但Java的变量要进行初始化

**为什么有数据类型**

Java语言是强类型语言，对于每一种数据都定义了明确的具体数据类型，在内存中分配了不同大小的内存空间,可以合理的分配内存空间。程序员在定义变量的时候，需要声明这个变量的数据类型，这样就会给这个变量开辟一块合理大小的内存区域，从而避免空间浪费。

**八种基本数据类型**

一个字节：byte

两个字节：short、char

四个字节：int、float（单精度）

八个字节：long、double（双精度）

**8种基本数据类型除boolean类型之外都可以相互转换**

**byte,short,char做混合运算的时候，各自都先转换成int在做运算**

**多种数据类型做混合运算，先转换成容量最大的那种再做运算**

**五种引用数据类型**

类、接口类型、数组类型、枚举类型、注解类型

**在Java语言里面除了基本数据类型，其他的都是引用数据类型。**

**基本类型与引用类型？**

基本类型值指的是那些保存在栈内存中的简单数据段，即这种值完全保存在内存中的一个位置。

而引用类型值则是指那些保存在堆内存中的对象，意思是变量中保存的实际上只是一个指针，这个指针指向内存中的另一个位置，该位置保存对象。

**参数传递**

基本数据类型的值传递,不改变其值

引用数据类型的值传递,改变其值

String类虽然是引用数据类型,但它当做参数传递时和基本数据类型是一样的

**^：逻辑异或，两边只要是不一致就是true**

**&&和&的区别?**

&&具有短路效果。如果左边结果是false，则右边不执行。

&是无论左边是false还是true,右边都会执行。

**分支语句中，如果只有一条Java语句，大括号可以省略，但是不推荐这样使用，建议加上大括号**

**switch后面的括号中可以填写byte、short、char、int类型，即能够自动转换为int类型的都可以，在jdk7版本之后可以填写String类型。case后面只能是常量，不能是变量，而且，多个case后面的值不能出现相同的**

**while循环本质上和for循环是一样的，可以相互替换，作用相同，格式不同，实际工作中建议使用for循环，因为变量会及早的从内存中消失，可以提高内存的使用效率。**

**break与continue,return结束循环区别**

break：跳出一层循环，可以控制结束嵌套循环

continue：结束一趟循环，也可以像break那样加上循环的名字

return：它的作用不是结束循环的，而是结束方法的

**什么是面向对象**  
面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述某个事物在整个解决问题的步骤中的行为

**面向过程和面向对象的区别**

**面向过程**注重过程，把问题分解成多个不同的步骤，然后把各个步骤变成方法，使用的时候依次调用

**面向对象**注重对象之间的交互，将复杂的事情简单化，把问题分解成各个对象，然后各个对象之间进行交互，每个对象内部都进行了封装

**在工作当中尽量面向抽象编程，不要面向具体编程。**

### 面向过程

**优点：** 性能比面向对象高，类调用时需要实例化，开销比较大，比较消耗资源

**缺点：** 没有面向对象易维护、易复用、易扩展

### 面向对象

**优点：** 易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护

**缺点：** 性能比面向过程低

**类是具有相同属性和方法的一组对象的集合**

**对象是系统中描述客观事物的一个实体**

**方法是指为了完成某个任务，对象所能执行的操作**

**方法的优点就是可以提高代码的复用性**。

**不要在两个方法里面互相调用，程序会报出StackOverflowError错误。**

**main方法是所有程序的入口，这是sun公司规定的，程序都是从main方法开始执行的，如果将一个程序比喻成一座大楼，那么main方法就相当于是这个楼最外面的大门。**

**(String[] args)是形式参数列表，该参数表示是一个String类型的数组，形参的名字是随意的，只不过通常都会将参数名字写成args。**

**什么是递归**

递归就是方法自己调用自己，在编写的时候一定要有一个结束条件，否则将会造成StackOverflowError的错误。

**什么是栈**

栈的英文是stack，特点是先进后出，可以把栈想象成一个桶，向里面放置元素的时候叫做压栈（push），向外取出元素的时候叫做弹栈（pop）。在JVM当中有一块内存空间，就是栈区，专门为方法准备的。方法执行所需要的内存空间都在栈当中分配。

方法调用之后给该方法分配内存，在JVM的“栈区”当中分配空间，

方法调用分配空间的过程就是：压栈（push）的过程。

方法执行结束之后，给这个方法分配的所有空间全部释放，此时发生弹栈（pop）的动作。

如果持续的向栈里面进行压栈，超出栈的存储空间的话，就会出现StackOverflowError的错误。

**成员变量与局部变量的区别**

**成员变量**：写在类里面，方法体的外面，声明时可以不进行初始化值，会自动以类型的默认值而赋值，可以被本类或其他类的方法进行调用，可以被访问控制修饰符及 static 所修饰，存在于堆内存，随着对象的创建而存在  
**局部变量**：写在方法体的里面，声明时必须进行初始化，只能在声明局部变量的方法内进行调用，不能被访问控制修饰符及 static 所修饰，存在于栈内存，随着方法的调用而自动消失

**什么是封装？**  
封装是指隐藏对象的属性和实现细节，仅对外提供公共访问方式

**封装的优点：**  
不用关心具体实现，提高安全性，提高了代码的复用性，减少耦合。

**封装的使用**

将成员变量用private修饰

提供对应的getXxx()和setXxx()方法

private仅仅是封装的一种体现形式,封装不是私有

### 什么是继承？

继承是子类自动共享父类数据结构和方法的机制，是类之间的一种关系。在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并加入若干新的内容。在**Java语言里面只支持单继承**，支持**多层继承，**子类可以继承父类中的非private修饰的成员方法和成员变量，构造方法不能被继承。**如果一个类没有显示的继承其他类，那么这个类会默认继承Object类，Object是SUN公司提供的Java中的根类**。

**继承的优点**

提高了代码的复用性

提高了代码的维护性

让类与类之间产生了关系，是多态的前提

**继承的缺点**  
增强了类之间的耦合。  
软件开发的一个原则是**高内聚，低耦合。**

**内聚是一个模块内各个元素彼此结合的紧密程度  
耦合是一个软件里面不同模块之间相互连接的数量**

**什么是多态？**

一种行为，多种状态。同一个接口，不同的实例执行不同的操作，在面向对象语言中，接口的多种不同的实现方式即为多态。

**如何实现多态**

有继承

有方法重写

有父类引用指向子类对象

**静态绑定：**在程序执行前已经被绑定，也就是说在编译过程中就已经知道这个方法是哪个类的方法，此时由编译器获取其他连接程序实现。

**动态绑定：**在运行根据具体对象的类型进行绑定。

子类向父类型进行转换，是**自动类型转换**，也叫做向上转型。还有一种情况是父类向子类型转换，是**强制类型转换**，也叫向下转型。

**多态的优点**

提高了程序的扩展性

降低了代码之间的耦合

**方法重载**

发生在同一个类中.

方法名相同

参数列表不同(类型，个数，顺序)

方法重载和返回值类型、修饰符无关

### 方法重写

重写，也叫做覆盖，当父类中的方法无法满足子类需求时，子类可以将父类的方法进行重写编写来满足需求。

**方法重写的条件**

两个类必须是继承关系

必须具有相同的方法名，相同的返回值类型，相同的参数列表.

重写的方法不能比被重写的方法拥有更低的访问权限，抛出更宽泛的异常。

私有方法、构造方法不能被重写。

静态的方法不存在重写。

重写指的是成员方法，和成员变量无关。

**构造方法的作用**

给对象数据进行初始化。

**构造方法格式特点**

方法名与类名相同

没有返回值类型

没有void修饰

没有具体的返回值return;

如果一个类没有提供任何构造方法，系统默认提供无参数构造方法

如果一个类已经手动的提供了构造方法，那么系统不会再提供任何构造方法。

在创建对象的时候会默认执行**无参构造方法**。构造方法不能手动调用。

如果我们还想使用无参构造方法，就必须自己写出。**建议自己写出无参构造方法**

**This**

**代表着对当前对象的引用，**在堆内存中的每个Java对象上都有一个this指向自己

**This的作用？**

可以使用this.方法名的方式调用当前对象的方法

可以区分成员变量和局部变量。

如果要是使用this调用构造方法的话，那么this**必须出现在构造方法的第一行**。

在Java里面，局部变量和成员变量的标识符是可以相同的，如果有相同的标识符，则可以通过this调用成员变量

### Super

super代表的是当前子类对象中的父类型特征。

**Super的作用**  
子类和父类中都有某个数据，例如，子类和父类中都有name这个属性。如果要再子类中访问父类中的name属性，需要使用super。

子类重写了父类的某个方法（假设这个方法名叫m1），如果在子类中需要调用父类中的m1方法时，需要使用super。

子类调用父类中的构造方法时，需要使用super，**一个构造方法第一行如果没有this(…);也没有显示的去调用super(…);系统会默认调用super();如果已经有this了，那么就不会调用super了**，super(…);的调用只能放在构造方法的第一行，只是调用了父类中的构造方法，但是并不会创建父类的对象。

### super和this的区别

**调用成员变量**

this.成员变量： 调用本类的成员变量

super.成员变量： 调用父类的成员变量

**调用构造方法**

this(…) ：调用本类的构造方法

super(…)：调用父类的构造方法

**调用成员方法**

this.成员方法：调用本类的成员方法

super.成员方法：调用父类的成员方法

### static的作用

static可以修饰变量，被static修饰的变量叫做静态变量，程序运行时静态变量存放在方法区里面。

static可以修饰方法，被static修饰的方法叫做静态方法，**不用创建对象就能直接访问该方法**。静态方法不能直接访问非静态的数据，静态方法不能使用this和super。

static可以定义静态语句块，**静态语句块在类加载阶段执行，并且只执行一次，并且是自上而下的顺序执行，在构造方法之前执行**

static修饰的变量、方法、代码块都是隶属于**类(class)**级别的,跟对象无关。某一类物体如果可以被多个其他物体所共享，那么可以将这类物体使用static修饰。可以通过类名.静态方法名的方式访问，工作中使用这种写法

### Final的作用

final修饰的类无法被继承。

final修饰的方法无法被重写

final修饰的局部变量，一旦赋值，不可再改变。

final修饰的成员变量必须初始化值。

final修饰的引用类型，该引用不可再重新指向其他的Java对象。但是fianl修饰的引用，地址值不能被改变，该引用指向的对象的属性值是可以修改的。

### 什么是代码块？

使用{}括起来的代码被称为代码块

**局部代码块**，在方法中出现，限定变量生命周期，及早释放，提高内存利用率

**构造代码块**，在类中方法外出现；多个构造方法方法中相同的代码存放到一起，每次调用构造都执行，并且在构造方法前执行

**静态代码块**，在类中方法外出现，并加上static修饰；用于给类进行初始化，在加载的时候就执行，并且只执行一次。一般用于加载驱动。

**同步代码块**（多线程）

方法执行顺序：  
1.**静态代码块**，随着类加载而加载,且只执行一次  
2.**构造代码块**，每创建一个对象就会执行一次，优先于构造方法执行  
3.**构造方法**，每创建一个对象就会执行一次

**抽象类的特点**

抽象类无法被实例化，无法创建抽象类的对象，但是抽象类也有构造方法，该构造方法是给子类创建对象用的。

抽象类中不一定有抽象方法，但抽象方法必须出现在抽象类中。

抽象类中的子类可以是抽象类，如果不是抽象类的话必须对抽象类中的抽象方法进行重写。

抽象类和抽象方法不能被final修饰

**接口的特点**

在Java语言里面使用interface来声明一个接口，接口其实是一个特殊的抽象类，在接口里面的方法全部都是抽象的。

接口中只能出现常量和抽象方法

接口里面没有构造方法，无法创建接口的对象

接口和接口之间支持多继承，即一个接口可以有多个父接口

一个类可以实现多个接口，即一个类可以有多个父接口

一个类如果实现了接口，那么这个类需要重写接口中所有的抽象方法（建议），如果不重写则这个类需要声明为抽象类（不建议）

接口中方法默认用public abstract

接口中常量必须用public static final修饰，可以省略

### 接口的作用

可以使项目分层，都面向接口开发，提高开发效率

降低了代码之间的耦合度，提高了代码的可插拔性

开发中尽量使用接口，少用抽象类，一个类可以实现多个接口，却只能继承一个父类

**接口和抽象类的区别是什么**

**抽象类**

抽象类不能被实例化，只有抽象类的非抽象子类可以创建对象。

抽象类中不一定包含抽象方法，但是有抽象方法的类必定是抽象类。

抽象类可以提供某些方法的部分实现

构造方法，类方法（用static修饰的方法）不能声明为抽象方法。

抽象类的子类必须给出抽象类中的抽象方法的具体实现，除非该子类也是抽象类。

抽象是对类的抽象，是一种模板设计

**接口**

接口不能用于实例化对象，但可以声明，但是必须引用一个实现该接口的对象。

接口没有构造方法。

接口中所有的方法必须是抽象方法。

接口不能包含成员变量，除了常量。

接口不是被类继承了，而是要被类实现。

接口是行为的抽象，是一种行为的规范。

**API是Application Programming Interface的缩写,中文意思是应用程序编程接口。sun公司提供了一些预先定义好的类和方法以供Java开发者使用，有了API，开发者可以在不查看源代码的情况下了解某个类或者某个方法的使用方式。**

SUN在Object类中设计toString方法的目的：返回Java对象的字符串表示形式。

在现实的开发过程中，Object中的toString方法就是要被重写的。

如果直接打印一个引用数据类型的对象，系统会默认调用其toString方法。

### finalize方法

finalize方法不需要程序员去调用，由**系统自动调用**。Java对象如果没有更多的引用指向它，则该Java对象成为垃圾数据，等待垃圾回收器的回收，垃圾回收器在回收这个Java对象之前会自动调用该对象的finalize方法。finalize方法是该对象马上就要被回收了，例如：需要释放资源，则可以在该方法中释放。程序员只能“建议”垃圾回收器回收垃圾：**System.gc()**;

重写Object中的finalize方法.，不是给程序员调用，垃圾回收器在回收之前会自动调用

**包**

在类名前面使用关键字package加入包名来避免命名冲突问题，因为域名是世界上唯一的，所以建议使用公司倒写的域名来命名包名,通常是**小写**的

**package语句只能出现在.Java源文件的第一行**

**package语句在一个Java文件中只能有一个**

**如果没有package，默认表示无包名**

**Java.lang软件包下所有类不需要手动导入，系统自动导入，Object类，String类都在这个包里面**

**访问级别修饰符**

private 、protected、public和default（默认），可以限定其他类对该类、属性和方法的使用权限。

对类的修饰只有：public和default

**private：本类**

**default：本类、同包类**

**protected：本类、同包类、子类**

**public：所有类**

### 内部类

内部类，顾名思义就是在一个类的内部声明一个类。

**内部类可以直接访问外部类的成员，包括private修饰的变量和方法**

内部类主要分为：

静态内部类

匿名内部类

成员内部类

局部内部类

### 什么是数组？

数组是存储同一种数据类型多个元素的集合。也可以看成是一个容器。

数组里面既可以存储基本数据类型，也可以存储引用数据类型，但是**只能存储同一种数据类型**。

数组是一种简单的数据结构，线性的结构

**数组一旦创建其长度是不可变的**

数组是引用数据类型

### 静态初始化数组

数组初始化就是为数组开辟连续的内存空间，并为每个数组元素赋予值。知道数组的首元素的内存地址，要查找的元素只要知道下标就可以快速的计算出偏移量，通过首元素内存地址加上偏移量可以快速计算出要查找元素的内存地址，通过内存地址快速定位该元素， 所以数组查找元素的效率较高。  
**注意：数组下标是从0开始的**

### 动态初始化数组

动态初始化数组,会先在堆内存中分配这个数组，并且数组中每一个元素都采用默认值。

main方法中的String[] args是专门用来接收命令行参数的。

**equals跟==区别？**

== : 它的作用是判断两个对象的地址是不是相等。即，判断两个对象是不是同一个对象。(基本数据类型比较的是值，引用数据类型比较的是内存地址)

equals() : 它的作用也是判断两个对象是否相等。但它一般有两种使用情况：

情况1：类没有覆盖equals()方法。则通过equals()比较该类的两个对象时，等价于通过“==”比较这两个对象。

情况2：类覆盖了equals()方法。一般，我们都覆盖equals()方法来两个对象的内容相等；若它们的内容相等，则返回true(即，认为这两个对象相等)。

String中的equals方法是被重写过的，因为object的equals方法是比较的对象的内存地址，而String的equals方法比较的是对象的值。

**当创建String类型的对象时，虚拟机会在常量池中查找有没有已经存在的值和要创建的值相同的对象，如果有就把它赋给当前引用。如果没有就在常量池中重新创建一个String对象。**