# JAVA面试初级篇

|  |  |  |  |
| --- | --- | --- | --- |
| 作者 | 时间 | 版本 | 备注 |
| 张行 | 20160626 | 1.0 | 创建 |
|  |  |  |  |

# Java基本概念

## Java 语言的优点?

**简单、高效**

Java 语言与 C++ 类似，如果用户了解 C++ 和面向对象的概念，就可以很快编写出 Java 程序；此外，Java 又不同于诸如 C++ 语言提供的各种各样的方法，它只提供了基本的方法，这样就减少了编程的复杂性，例如去掉了头文件、指针变量、结构、运算符重载、多重继承等复杂特性。Java 语言虽然简单，却很高效，它可以用面向对象的方法来描述用户的每一个动作。

**面向对象**

Java 语言是一种新的面向对象的程序设计语言，它除了几种基本的数据类型外，大都是类似 C++ 中的对象和方法，程序代码大多体现了类机制，以类的形式组织，由类来定义对象的各种行为。Java 同样支持类继承，这样也减少了程序设计的复杂性。

**平台无关性**

所谓一处编译处处运行。Java 语言经编译后生成与计算机硬件结构无关的字节代码（ Bytecode ），这些字节代码被定义为不依赖任何硬件平台和操作系统。当 Java 程序在运行时，需要由一个解释程序对生成的字节代码解释执行。这体现了Java语言的与平台无关性，使得 Java 程序可以在任何平台上运行，如 MS-DOS , Windows，Unix 等，因此具有很强的移植性。

**交互式特性**

Java 是面向对象的网络编程语言，由于它支持 TCP／IP 协议，使得用户可以通过浏览器访问到 Internet 上的各种动态对象。并且在网络上用户可以交互式地进行各种动作，而多线程技术的引入使得这种交互式操作更为容易。

**多线程机制**

Java 语言支持多线程机制，多线程机制使得 Java 程序能够并行处理多项任务。Java 程序可以设计成具有多个线程，例如让一个线程负责数据的检索、查寻，另一个线程与用户进行交互，这样，两个线程得以并行执行。多线程机制可以很容易地实现网络上的交互式操作。

**动态的内存管理机制**

Java 语言采用了自动垃圾回收机制进行内存的管理。在 C++ 语言中，程序员在编写程序时要仔细地处理内存的使用，例如当某个内存快使用完毕时，要及时释放，以供其它程序使用，一旦内存管理不当，就有可能造成内存空间浪费或程序运行故障。 在 Java 系统中包括了一个自动垃圾回收程序，它可以自动、安全地回收不再使用的内存块，这样，程序员在编程时就无需担心内存的管理问题，从而使 Java 程序的编写变得简单，同时也减少了内存管理方面出错的可能性。

**安全性**

在类似 Internet 的这种分布式网络环境中，安全性是个不容忽视的问题。Java 语言在安全性方面引入了实时内存分配及布局来防止程序员直接修改物理内存布局；通过字节代码验证器对字节代码的检验，以防止网络病毒及其它非法代码侵入。此外，Java 语言还采用了许多面向对象的异常处理机制，负责对一些异常事件进行处理，如内存空间不够，程序异常中止等的处理

Java 是解释型的 运行 Java 程序需要一个解释器。Java 程序编译成 Java 虚拟机编码，这种编码称为字节码。字节码是独立于计算机的，他能在所有具有 Java 解释器的机器上运行，Java 解释器也是 JVM 的一部分。

**Java 的效率**

早期 Sun 发现 JVM 的用户注意到了 Java 运行很慢，但是新的 Java 虚拟机显著加快。新的虚拟机使用了一种称为实时编译的技术，他把字节码编译并储存成本机的机器码。当执行字节码时，重调本地码。

## 面向对象的特征有哪些方面？

答：面向对象的特征主要有以下几个方面：

- 抽象：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属性和行为，并不关注这些行为的细节是什么。

- 继承：继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类（超类、基类）；得到继承信息的类被称为子类（派生类）。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段（如果不能理解请阅读阎宏博士的《Java与模式》或《设计模式精解》中关于桥梁模式的部分）。

- 封装：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。可以说，封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口（可以想想普通洗衣机和全自动洗衣机的差别，明显全自动洗衣机封装更好因此操作起来更简单；我们现在使用的智能手机也是封装得足够好的，因为几个按键就搞定了所有的事情）。

- 多态性：多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态性可以解释为：当A系统访问B系统提供的服务时，B系统有多种提供服务的方式，但一切对A系统来说都是透明的（就像电动剃须刀是A系统，它的供电系统是B系统，B系统可以使用电池供电或者用交流电，甚至还有可能是太阳能，A系统只会通过B类对象调用供电的方法，但并不知道供电系统的底层实现是什么，究竟通过何种方式获得了动力）。方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：1). 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；2). 对象造型（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

## 什么是 Java 虚拟机？为什么 Java 被称作是“平台无关的编程语言”？

Java 虚拟机是一个可以执行 Java 字节码的虚拟机进程。Java 源文件被编译成能被 Java 虚拟机执行的字节码文件。

Java 被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台单独重写或者是重新编译。Java 虚拟机让这个变为可能，因为它知道底层硬件平台的指令长度和其他特性。

## 指针

Java 语言让编程者无法找到指针来直接访问内存无指针，并且增添了自动的内存管理功能，从而有效地防止了 C/C++ 语言中指针操作失误，如野指针所造成的系统崩溃。但也不是说 Java 没有指针，虚拟机内部还是使用了指针，只是外人不得使用而已。这有利于 Java 程序的安全

## Java 支持的数据类型有哪些？什么是自动拆装箱？

Java 语言支持的 8 中基本数据类型是：

byte

short

int

long

float

double

boolean

char

自动装箱是 Java 编译器在基本数据类型和对应的对象包装类型之间做的一个转化。比如：把 int 转化成 Integer ，double 转化成 Double，等等。反之就是自动拆箱。

## 什么是值传递和引用传递？

对象被值传递，意味着传递了对象的一个副本。因此，就算是改变了对象副本，也不会影响源对象的值。

对象被引用传递，意味着传递的并不是实际的对象，而是对象的引用。因此，外部对引用对象所做的改变会反映到所有的对象上。

## 一个 ".java" 源文件中是否可以包括多个类（不是内部类）？有什么限制？

可以有多个类，但只能有一个 public 的类，并且 public 的类名必须与文件名相一致。

## 静态变量和实例变量的区别？

在语法定义上的区别：静态变量前要加 static 关键字，而实例变量前则不加。

在程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量。静态变量不属于某个实例对象，而是属于类，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

## 不通过构造函数也能创建对象吗?

A. 是

B. 否

答案：A

解析：Java 创建对象的几种方式（重要）：

(1) 用 new 语句创建对象，这是最常见的创建对象的方法。 (2) 运用反射手段,调用 java.lang.Class 或者 java.lang.reflect.Constructor 类的 newInstance() 实例方法。 (3) 调用对象的 clone() 方法。 (4) 运用反序列化手段，调用 java.io.ObjectInputStream 对象的 readObject() 方法。

(1)和(2)都会明确的显式的调用构造函数 ；(3)是在内存上对已有对象的影印，所以不会调用构造函数 ；(4)是从文件中还原类的对象，也不会调用构造函数。

## 是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？

答：不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，因此在调用静态方法时可能对象并没有被初始化。

## 如何实现对象克隆？

答：有两种方式：

1.实现 Cloneable 接口并重写 Object 类中的 clone() 方法；

2.实现 Serializable 接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆。

## Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？

答：可以继承其他类或实现其他接口，在Swing编程中常用此方式来实现事件监听和回调。

## 内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制？

答：一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员

静态嵌套类(Static Nested Class)和内部类（Inner Class）的不同？   
答：Static Nested Class是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化

## 列出自己常用的 jdk 包.

答：JDK 常用的 package

java.lang：这个是系统的基础类，比如 String 等都是这里面的，这个 package 是唯一一个可以不用 import 就可以使用的 Packag

java.io: 这里面是所有输入输出有关的类，比如文件操作等

java.net: 这里面是与网络有关的类，比如 URL,URLConnection 等。

java.util: 这个是系统辅助类，特别是集合类 Collection,List,Map 等。

java.sql: 这个是数据库操作的类，Connection, Statememt，ResultSet 等

## JDK, JRE 和 JVM 的区别？

JDK, JRE 和 JVM 是 Java 编程语言的核心概念。尽管它们看起来差不多，作为程序员我们也不怎么关心这些概念，但是它们是不同的针对特定目的的产品。这是一道常见的 Java 面试题，而本文则会一一解释这些概念并给出它们之间的区别。

Java 开发工具包 (JDK)

Java 开发工具包是 Java 环境的核心组件，并提供编译、调试和运行一个 Java 程序所需的所有工具，可执行文件和二进制文件。JDK 是一个平台特定的软件，有针对 Windows，Mac 和 Unix 系统的不同的安装包。可以说 JDK 是 JRE 的超集，它包含了 JRE 的 Java 编译器，调试器和核心类。目前 JDK 的版本号是 1.7，也被称为 Java 7。

Java 虚拟机(JVM)

JVM 是 Java 编程语言的核心。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 也是平台特定的，并提供核心的 Java 方法，例如内存管理、垃圾回收和安全机制等。JVM 是可定制化的，我们可以通过 Java 选项(java options)定制它，比如配置 JVM 内存的上下界。JVM 之所以被称为虚拟的是因为它提供了一个不依赖于底层操作系统和机器硬件的接口。这种独立于硬件和操作系统的特性正是 Java 程序可以一次编写多处执行的原因。

Java 运行时环境(JRE)

JRE 是 JVM 的实施实现，它提供了运行 Java 程序的平台。JRE 包含了 JVM、Java 二进制文件和其它成功执行程序的类文件。JRE 不包含任何像 Java 编译器、调试器之类的开发工具。如果你只是想要执行 Java 程序，你只需安装 JRE 即可，没有安装 JDK 的必要。

JDK, JRE 和 JVM 的区别

JDK 是用于开发的而 JRE 是用于运行 Java 程序的。

JDK 和 JRE 都包含了 JVM，从而使得我们可以运行 Java 程序。

JVM 是 Java 编程语言的核心并且具有平台独立性。

# 面向对象编程

## Java 中的方法覆盖 (Overriding) 和方法重载 (Overloading) 是什么意思？

Java 中的方法重载发生在同一个类里面两个或者是多个方法的方法名相同但是参数不同的情况。与此相对，方法覆盖是说子类重新定义了父类的方法。方法覆盖必须有相同的方法名，参数列表和返回类型。覆盖者可能不会限制它所覆盖的方法的访问。

## Overload 和 Override 的区别？ Overloaded 的方法是否可以改变返回值的类型?

Overload 是重载的意思，Override 是覆盖的意思，也就是重写。

重载 Overload 表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

重写 Override 表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。子类方法的访问权限只能比父类的更大，不能更小。如果父类的方法是 private 类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。

至于 Overloaded 的方法是否可以改变返回值的类型这个问题，要看你倒底想问什么呢？这个题目很模糊。如果几个 Overloaded 的方法的参数列表不一样，它们的返回者类型当然也可以不一样。但我估计你想问的问题是：如果两个方法的参数列表完全一样，是否可以让它们的返回值不同来实现重载 Overload。这是不行的，我们可以用反证法来说明这个问题，因为我们有时候调用一个方法时也可以不定义返回结果变量，即不要关心其返回结果，例如，我们调用 map.remove(key) 方法时，虽然 remove 方法有返回值，但是我们通常都不会定义接收返回结果的变量，这时候假设该类中有两个名称和参数列表完全相同的方法，仅仅是返回类型不同，Java 就无法确定编程者倒底是想调用哪个方法了，因为它无法通过返回结果类型来判断。

Override 可以翻译为覆盖，从字面就可以知道，它是覆盖了一个方法并且对其重写，以求达到不同的作用。对我们来说最熟悉的覆盖就是对接口方法的实现，在接口中一般只是对方法进行了声明，而我们在实现时，就需要实现接口声明的所有方法。除了这个典型的用法以外，我们在继承中也可能会在子类覆盖父类中的方法。在覆盖要注意以下的几点：

1、覆盖的方法的标志必须要和被覆盖的方法的标志完全匹配，才能达到覆盖的效果；

2、覆盖的方法的返回值必须和被覆盖的方法的返回一致；

3、覆盖的方法所抛出的异常必须和被覆盖方法的所抛出的异常一致，或者是其子类；

4、被覆盖的方法不能为 private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。

Overload 对我们来说可能比较熟悉，可以翻译为重载，它是指我们可以定义一些名称相同的方法，通过定义不同的输入参数来区分这些方法，然后再调用时，VM 就会根据不同的参数样式，来选择合适的方法执行。在使用重载要注意以下的几点：

1、在使用重载时只能通过不同的参数样式。例如，不同的参数类型，不同的参数个数，不同的参数顺序（当然，同一方法内的几个参数类型必须不一样，例如可以是 fun(int,float)，但是不能 fun(int,int)）； 2、不能通过访问权限、返回类型、抛出的异常进行重载；

3、方法的异常类型和数目不会对重载造成影响；

4、对于继承来说，如果某一方法在父类中是访问权限是 priavte，那么就不能在子类对其进行重载，如果定义的话，也只是定义了一个新方法，而不会达到重载的效果。

## 为什么不能根据返回类型来区分重载?

public int a(){

}

public double a(){

}

a();

## Java 中，什么是构造函数？什么是构造函数重载？

当新对象被创建的时候，构造函数会被调用。每一个类都有构造函数。在程序员没有给类提供构造函数的情况下，Java 编译器会为这个类创建一个默认的构造函数。

Java 中构造函数重载和方法重载很相似。可以为一个类创建多个构造函数。每一个构造函数必须有它自己唯一的参数列表。

## 构造器 Constructor 是否可被 Override?

构造器 Constructor 不能被继承，因此不能重写 Override，但可以被重载 Overload。

## Java 支持多继承么？

不支持，Java 不支持多继承。每个类都只能继承一个类，但是可以实现多个接口。

## 接口和抽象类的区别是什么？

Java 提供和支持创建抽象类和接口。它们的实现有共同点，不同点在于：

接口中所有的方法隐含的都是抽象的。而抽象类则可以同时包含抽象和非抽象的方法。

类可以实现很多个接口，但是只能继承一个抽象类

类如果要实现一个接口，它必须要实现接口声明的所有方法。但是，类可以不实现抽象类声明的所有方法，当然，在这种情况下，类也必须得声明成是抽象的。

抽象类可以在不提供接口方法实现的情况下实现接口。

Java 接口中声明的变量默认都是 final 的。抽象类可以包含非 final 的变量。

Java 接口中的成员函数默认是 public 的。抽象类的成员函数可以是 private，protected 或者是 public 。

接口是绝对抽象的，不可以被实例化。抽象类也不可以被实例化，但是，如果它包含 main 方法的话是可以被调用的。

## 抽象的（abstract）方法是否可同时是静态的（static）,是否可同时是本地方法（native），是否可同时被synchronized修饰？

答：都不能。抽象方法需要子类重写，而静态的方法是无法被重写的，因此二者是矛盾的。本地方法是由本地代码（如C代码）实现的方法，而抽象方法是没有实现的，也是矛盾的。synchronized和方法的实现细节有关，抽象方法不涉及实现细节，因此也是相互矛盾的。

下列说法正确的有（）

A．class 中的 constructor 不可省略

B．constructor 必须与 class 同名，但方法不能与class同名

C．constructor在一个对象被 new 时执行

D．一个 class 只能定义一个 constructor

答案：C

解析：这里可能会有误区，其实普通的类方法是可以和类名同名的，和构造方法唯一的区分就是，构造方法没有返回值。

8. Java 接口的修饰符可以为?

A. private

B. protected

C. final

D. abstract

答案：CD

解析：接口很重要，为了说明情况，这里稍微啰嗦点：

（1）接口用于描述系统对外提供的所有服务,因此接口中的成员常量和方法都必须是公开(public)类型的,确保外部使用者能访问它们；

（2）接口仅仅描述系统能做什么,但不指明如何去做,所以接口中的方法都是抽象(abstract)方法

（3）接口不涉及和任何具体实例相关的细节,因此接口没有构造方法,不能被实例化,没有实例变量，只有静态（static）变量；

（4）接口的中的变量是所有实现类共有的，既然共有，肯定是不变的东西，因为变化的东西也不能够算共有。所以变量是不可变(final)类型，也就是常量了。

（5） 接口中不可以定义变量？如果接口可以定义变量，但是接口中的方法又都是抽象的，在接口中无法通过行为来修改属性。有的人会说了，没有关系，可以通过 实现接口的对象的行为来修改接口中的属性。这当然没有问题，但是考虑这样的情况。如果接口 A 中有一个public 访问权限的静态变量 a。按照 Java 的语义，我们可以不通过实现接口的对象来访问变量 a，通过 A.a = xxx; 就可以改变接口中的变量 a 的值了。正如抽象类中是可以这样做的，那么实现接口 A 的所有对象也都会自动拥有这一改变后的 a 的值了，也就是说一个地方改变了 a，所有这些对象中 a 的值也都跟着变了。这和抽象类有什么区别呢，怎么体现接口更高的抽象级别呢，怎么体现接口提供的统一的协议呢，那还要接口这种抽象来做什么呢？所以接口中 不能出现变量，如果有变量，就和接口提供的统一的抽象这种思想是抵触的。所以接口中的属性必然是常量，只能读不能改，这样才能为实现接口的对象提供一个统 一的属性。

通俗的讲，你认为是要变化的东西，就放在你自己的实现中，不能放在接口中去，接口只是对一类事物的属性和行为更高层次的抽象。对修改关闭，对扩展（不同的实现 implements）开放，接口是对开闭原则的一种体现。

所以：

接口的方法默认是 public abstract；

接口中不可以定义变量即只能定义常量(加上final修饰就会变成常量)。所以接口的属性默认是 public static final 常量，且必须赋初值。

注意：final 和 abstract 不能同时出现。

下面是 People 和 Child 类的定义和构造方法，每个构造方法都输出编号。在执行 new Child("mike") 的时候都有哪些构造方法被顺序调用？请选择输出结果

class People {

String name;

public People() {

System.out.print(1);

}

public People(String name) {

System.out.print(2);

this.name = name;

}

}

class Child extends People {

People father;

public Child(String name) {

System.out.print(3);

this.name = name;

father = new People(name + ":F");

}

public Child() {

System.out.print(4);

}

}

A. 312

B. 32

C. 432

D. 132

答案：D

解析：考察的又是父类与子类的构造函数调用次序。在 Java 中，子类的构造过程中必须调用其父类的构造函数，是因为有继承关系存在时，子类要把父类的内容继承下来。但如果父类有多个构造函数时，该如何选择调用呢？

第一个规则：子类的构造过程中，必须调用其父类的构造方法。一个类，如果我们不写构造方法，那么编译器会帮我们加上一个默认的构造方法（就是没有参数的构造方法），但是如果你自己写了构造方法，那么编译器就不会给你添加了，所以有时候当你 new 一个子类对象的时候，肯定调用了子类的构造方法，但是如果在子类构造方法中我们并没有显示的调用基类的构造方法，如：super(); 这样就会调用父类没有参数的构造方法。

第二个规则：如果子类的构造方法中既没有显示的调用基类构造方法，而基类中又没有无参的构造方法，则编译出错，所以，通常我们需要显示的：super(参数列表)，来调用父类有参数的构造函数，此时无参的构造函数就不会被调用。

总之，一句话：子类没有显示调用父类构造函数，不管子类构造函数是否带参数都默认调用父类无参的构造函数，若父类没有则编译出错。

## 两个对象值相同(x.equals(y) == true)，但却可有不同的 hash code，这句话对不对？

答：不对，如果两个对象 x 和 y 满足 x.equals(y) == true，它们的哈希码（hash code）应当相同。Java 对于 eqauls 方法和 hashCode 方法是这样规定的：(1)如果两个对象相同（equals 方法返回 true ），那么它们的 hashCode 值一定要相同；(2)如果两个对象的 hashCode 相同，它们并不一定相同。当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在 Set 集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

实现高质量的 equals 方法的诀窍包括：

使用 == 操作符检查“参数是否为这个对象的引用”；

使用 instanceof 操作符检查“参数是否为正确的类型”；

对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；

编写完 equals 方法后，问自己它是否满足对称性、传递性、一致性；

重写 equals 时总是要重写 hashCode；

不要将 equals 方法参数中的 Object 对象替换为其他的类型，在重写时不要忘掉 @Override 注解。

## 接口是否可继承（extends）接口? 抽象类是否可实现（implements）接口? 抽象类是否可继承具体类（concrete class）?

答：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承具体类，但前提是具体类必须有明确的构造函数。

指出下面程序的运行结果:

class A{

static{

System.out.print("1");

}

public A(){

System.out.print("2");

}

}

class B extends A{

static{

System.out.print("a");

}

public B(){

System.out.print("b");

}

}

public class Hello{

public static void main(String[] args){

A ab = new B();

ab = new B();

}

}

答：执行结果：1a2b2b。创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

## 当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递？

答：是值传递。Java语言的方法调用只支持参数的值传递。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的属性可以在被调用过程中被改变，但对对象引用的改变是不会影响到调用者的。C++和C#中可以通过传引用或传输出参数来改变传入的参数的值。在C#中可以编写如下所示的代码，但是在Java中却做不到。

## 是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？

答：不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，在调用静态方法时可能对象并没有被初始化。

## 接口是否可继承（extends）接口？抽象类是否可实现（implements）接口？抽象类是否可继承具体类（concrete class）？

答：接口可以继承接口，而且支持多重继承。抽象类可以实现(implements)接口，抽象类可继承具体类也可以继承抽象类。

**指出下面程序的运行结果。**

class A {

static {

System.out.print("1");

}

public A() {

System.out.print("2");

}

}

class B extends A{

static {

System.out.print("a");

}

public B() {

System.out.print("b");

}

}

public class Hello {

public static void main(String[] args) {

A ab = new B();

ab = new B();

}

}

答：执行结果：1a2b2b。创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

# 关键字

## ”static” 关键字是什么意思？Java 中是否可以覆盖(override) 一个 private 或者是 static 的方法？

“static” 关键字表明一个成员变量或者是成员方法可以在没有所属的类的实例变量的情况下被访问。

Java 中 static 方法不能被覆盖，因为方法覆盖是基于运行时动态绑定的，而 static 方法是编译时静态绑定的。static 方法跟类的任何实例都不相关，所以概念上不适用。

不可以覆盖一个private的方法，因为private方法不能被继承。

## 是否可以在 static 环境中访问非 static 变量？

static 变量在 Java 中是属于类的，它在所有的实例中的值是一样的。当类被Java虚拟机载入的时候，会对 static 变量进行初始化。如果你的代码尝试不用实例来访问非 static 的变量，编译器会报错，因为这些变量还没有被创建出来，还没有跟任何实例关联上。

## 访问修饰符 public, private, protected, 以及不写（默认）时的区别？

修饰符 当前类 同 包 子 类 其他包

public √ √ √ √

protected √ √ √ ×

default √ √ × ×

private √ × × ×

类的成员不写访问修饰时默认为 default 。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。Java 中，外部类的修饰符只能是 public 或默认，类的成员（包括内部类）的修饰符可以是以上四种。

## volatile关键字是否能保证线程安全？

答案：不能

解析：volatile 关键字用在多线程同步中，可保证读取的可见性，JVM只是保证从主内存加载到线程工作内存的值是最新的读取值，而非 cache 中。但多个线程对 volatile 的写操作，无法保证线程安全。例如假如线程 1，线程 2 在进行 read,load 操作中，发现主内存中 count 的值都是 5，那么都会加载这个最新的值，在线程 1 堆 count 进行修改之后，会 write 到主内存中，主内存中的 count 变量就会变为 6；线程 2 由于已经进行 read,load 操作，在进行运算之后，也会更新主内存 count 的变量值为 6；导致两个线程及时用 volatile 关键字修改之后，还是会存在并发的情况。

## Java 有没有 goto?

答：goto 是 Java 中的保留字，在目前版本的 Java 中没有使用。（根据James Gosling（Java 之父）编写的《The Java Programming Language》一书的附录中给出了一个 Java 关键字列表，其中有 goto 和 const，但是这两个是目前无法使用的关键字，因此有些地方将其称之为保留字，其实保留字这个词应该有更广泛的意义，因为熟悉 C 语言的程序员都知道，在系统类库中使用过的有特殊意义的单词或单词的组合都被视为保留字）

## Java 中的 final关键字有哪些用法？

答：

(1)修饰类：表示该类不能被继承；

(2)修饰方法：表示方法不能被重写；

(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。

## final, finally, finalize 的区别?

答：

final：修饰符（关键字）有三种用法：如果一个类被声明为final，意味着它不能再派生出新的子类，即不能被继承，因此它和 abstract 是反义词。将变量声明为 final，可以保证它们在使用中不被改变，被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取不可修改。被声明为 final 的方法也同样只能使用，不能在子类中被重写。

finally：通常放在 try…catch 的后面构造总是执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要 JVM 不关闭都能执行，可以将释放外部资源的代码写在 finally 块中。

finalize：Object 类中定义的方法，Java 中允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写finalize() 方法可以整理系统资源或者执行其他清理工作。

# 基本类型与运算

## 说说 & 和 && 的区别?

& 和 && 都可以用作逻辑与的运算符，表示逻辑与（and），当运算符两边的表达式的结果都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，则结果为 false。

&& 还具有短路的功能，即如果第一个表达式为 false，则不再计算第二个表达式，例如，对于 if(str != null&& !str.equals(“”)) 表达式，当 str 为 null 时，后面的表达式不会执行，所以不会出现 NullPointerException 如果将 && 改为 & ，则会抛出NullPointerException 异常。If(x==33 & ++y>0) y 会增长， If(x==33 && ++y>0) 不会增长。

& 还可以用作位运算符，当 & 操作符两边的表达式不是 boolean 类型时，& 表示按位与操作，我们通常使用 0x0f 来与一个整数进行 & 运算，来获取该整数的最低 4 个 bit 位，例如，0x31 & 0x0f 的结果为 0x01。

备注：这道题先说两者的共同点，再说出 && 和 & 的特殊之处，并列举一些经典的例子来表明自己理解透彻深入、实际经验丰富。

## 用最有效率的方法算出 2 乘以 8 等於几?

2 << 3，因为将一个数左移 n 位，就相当于乘以了 2 的 n 次方，那么，一个数乘以 8 只要将其左移 3 位即可，而位运算 cpu 直接支持的，效率最高，所以，2 乘以 8 等於几的最效率的方法是 2 << 3 。

## 存在使 i + 1 < i的数吗?

答案：存在

解析：如果 i 为 int 型，那么当 i 为 int 能表示的最大整数时，i+1 就溢出变成负数了，此时不就 <i 了吗。

扩展：存在使 i > j || i <= j 不成立的数吗?

答案：存在

解析：比如 Double.NaN 或 Float.NaN 。

0.6332 的数据类型是（）

A. float

B. double

C. Float

D. Double

答案：B

解析：默认为 double 型，如果为 float 型需要加上f显示说明，即 0.6332f

System.out.println("5" + 2);的输出结果应该是（）。

A. 52

B. 7

C. 2

D. 5

答案：A

解析：没啥好说的，Java 会自动将 2 转换为字符串。

下面的方法，当输入为 2 的时候返回值是多少?

public static int getValue(int i) {

int result = 0;

switch (i) {

case 1:

result = result + i;

case 2:

result = result + i \* 2;

case 3:

result = result + i \* 3;

}

return result;

}

A. 0

B. 2

C. 4

D. 10

答案：D

解析：注意这里 case 后面没有加 break，所以从case 2开始一直往下运行。

## float f=3.4;是否正确?

答:不正确。3.4 是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换float f =(float)3.4; 或者写成 float f =3.4F;。

## int 和 Integer 有什么区别?

答：Java 是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入不是对象的基本数据类型，但是为了能够将这些基本数据类型当成对象操作，Java 为每一个基本数据类型都引入了对应的包装类型（wrapper class），int 的包装类就是 Integer，从 JDK 1.5 开始引入了自动装箱/拆箱机制，使得二者可以相互转换。

Java 为每个原始类型提供了包装类型：

原始类型: boolean，char，byte，short，int，long，float，double

包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

package com.lovo;

public class AutoUnboxingTest {

public static void main(String[] args) {

Integer a = new Integer(3);

Integer b = 3; // 将3自动装箱成Integer类型

int c = 3;

System.out.println(a == b); // false 两个引用没有引用同一对象

System.out.println(a == c); // true a自动拆箱成int类型再和c比较

}

}

## char 型变量中能不能存贮一个中文汉字?为什么?

答：char 类型可以存储一个中文汉字，因为 Java 中使用的编码是 Unicode（不选择任何特定的编码，直接使用字符在字符集中的编号，这是统一的唯一方法），一个 char 类型占 2 个字节（16bit），所以放一个中文是没问题的。

补充：使用 Unicode 意味着字符在 JVM 内部和外部有不同的表现形式，在 JVM 内部都是 Unicode，当这个字符被从 JVM 内部转移到外部时（例如存入文件系统中），需要进行编码转换。所以 Java 中有字节流和字符流，以及在字符流和字节流之间进行转换的转换流，如 InputStreamReader 和 OutputStreamReader，这两个类是字节流和字符流之间的适配器类，承担了编码转换的任务。

## Math.round(11.5) 等于多少? Math.round(-11.5)等于多少?

答：Math.round(11.5)==12 Math.round(-11.5)==-11 round 方法返回与参数 最接近的长整数。

## short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？

答：对于short s1 = 1; s1 = s1 + 1;由于1是int类型，因此s1+1运算结果也是int 型，需要强制转换类型才能赋值给short型。而short s1 = 1; s1 += 1;可以正确编译，因为s1+= 1;相当于s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

## 自动装箱和拆箱

class AutoUnboxingTest {

public static void main(String[] args) {

Integer a = new Integer(3);

Integer b = 3; // 将3自动装箱成Integer类型

int c = 3;

System.out.println(a == b); // false 两个引用没有引用同一对象

System.out.println(a == c); // true a自动拆箱成int类型再和c比较

}

}

public class Test03 {

public static void main(String[] args) {

Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;

System.out.println(f1 == f2);

System.out.println(f3 == f4);

}

}

如果不明就里很容易认为两个输出要么都是true要么都是false。首先需要注意的是f1、f2、f3、f4四个变量都是Integer对象引用，所以下面的==运算比较的不是值而是引用。装箱的本质是什么呢？当我们给一个Integer对象赋一个int值的时候，会调用Integer类的静态方法valueOf，如果看看valueOf的源代码就知道发生了什么。

public static Integer valueOf(int i) {

if (i >= IntegerCache.low && i <= IntegerCache.high)

return IntegerCache.cache[i + (-IntegerCache.low)];

return new Integer(i);

}

IntegerCache是Integer的内部类，其代码如下所示：

/\*\*

\* Cache to support the object identity semantics of autoboxing for values between

\* -128 and 127 (inclusive) as required by JLS.

\*

\* The cache is initialized on first usage. The size of the cache

\* may be controlled by the {@code -XX:AutoBoxCacheMax=<size>} option.

\* During VM initialization, java.lang.Integer.IntegerCache.high property

\* may be set and saved in the private system properties in the

\* sun.misc.VM class.

\*/

private static class IntegerCache {

static final int low = -128;

static final int high;

static final Integer cache[];

static {

// high value may be configured by property

int h = 127;

String integerCacheHighPropValue =

sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");

if (integerCacheHighPropValue != null) {

try {

int i = parseInt(integerCacheHighPropValue);

i = Math.max(i, 127);

// Maximum array size is Integer.MAX\_VALUE

h = Math.min(i, Integer.MAX\_VALUE - (-low) -1);

} catch( NumberFormatException nfe) {

// If the property cannot be parsed into an int, ignore it.

}

}

high = h;

cache = new Integer[(high - low) + 1];

int j = low;

for(int k = 0; k < cache.length; k++)

cache[k] = new Integer(j++);

// range [-128, 127] must be interned (JLS7 5.1.7)

assert IntegerCache.high >= 127;

}

private IntegerCache() {}

}

简单的说，如果整型字面量的值在-128到127之间，那么不会new新的Integer对象，而是直接引用常量池中的Integer对象，所以上面的面试题中f1==f2的结果是true，而f3==f4的结果是false。

## switch 是否能作用在byte 上，是否能作用在long 上，是否能作用在String上？

答：在Java 5以前，switch(expr)中，expr只能是byte、short、char、int。从Java 5开始，Java中引入了枚举类型，expr也可以是enum类型，从Java 7开始，expr还可以是字符串（String），但是长整型（long）在目前所有的版本中都是不可以的。

## 数据类型之间的转换

- 如何将字符串转换为基本数据类型？

- 如何将基本数据类型转换为字符串？

答：

- 调用基本数据类型对应的包装类中的方法parseXXX(String)或valueOf(String)即可返回相应基本类型；

- 一种方法是将基本数据类型与空字符串（""）连接（+）即可获得其所对应的字符串；另一种方法是调用String 类中的valueOf()方法返回相应字符串

# Java字符串String常见的面试问题

**1. 在Java中String是什么？String是一种数据类型？**

String是在java和在java.lang包中定义的一个类，不像基本数据类型int和long。 String类代表字符的字符串。String几乎用在所有的Java应用程序中。String是不可变的，final的，在Java和JVM使用字符串池来存储所有的String对象。 我们可以使用双引号和“+”操作符的重载串连实例化一个String对象的方式。

1. **有什么不同的方法来创建String对象的？**

我们可以像任何普通的Java类一样使用new运算符来创建String对象，或者我们可以使用双引号来创建一个String对象。String类提供了从char数组，字节数组的StringBuffer和StringBuilder String对象的几个构造函数。

String str = new String(“abc”);

String str1 = “abc”;

当我们使用双引号创建一个字符串时，JVM会先在字符串池中查找是否其他任何字符串存储具有相同的值。如果找到了，返回的它引用，否则创建一个新的String对象，并将其值存储在字符串池中。

当我们使用new运算符时创建String对象时，但JVM不会将其存储到String Pool中。除非我们使用intern（）方法将String对象存储到字符串池中，值如果已经存在String 则返回引用。

1. **写一个方法来检查输入的字符串是否是回文（对称）？**

private static boolean isPalindrome(String str) {  
 if (str == null)  
 return false;  
 StringBuilder strBuilder = new StringBuilder(str);  
 strBuilder.reverse();  
 return strBuilder.toString().equals(str);  
}

1. **写一个方法从字符串中删除给定的字符？**

private static String removeChar(String str, char c) {  
 if (str == null)  
 return null;  
 return str.replaceAll(Character.toString(c), “”);  
}

1. **怎样才能转换字符串大写或小写？**

我们可以使用String类的toUpperCase和toLowerCase方法来转换大小写。

1. **如何在java程序比较两个字符串？**

Java的String实现Comparable接口，使用对象的compaareTo 方法比较两个字符串的大小。

equals方法比较两个字符串的值是否相等

==号比较引用是否相等

1. **我们可否在switch 条件中使用字符串？**

这是一个用来检查你的当前Java发展的知识一个棘手的问题。 在Java 7是可以的，而早期的Java版本不支持这个。

下面程序的运行结果是（）

String str1 = "hello";

String str2 = "he" + new String("llo");

System.err.println(str1 == str2);

答案：false

解析：因为 str2 中的 llo 是新申请的内存块，而 == 判断的是对象的地址而非值，所以不一样。如果是String str2 = str1，那么就是 true 了。

1. **下面代码的运行结果为?**

import java.io.\*;

import java.util.\*;

public class foo{

public static void main (String[] args){

String s;

System.out.println("s=" + s);

}

}

A. 代码得到编译，并输出 “s=”

B. 代码得到编译，并输出 “s=null”

C. 由于 String s 没有初始化，代码不能编译通过

D. 代码得到编译，但捕获到 NullPointException异常

答案：C

解析：开始以为会输出 null 什么的，运行后才发现 Java 中所有定义的基本类型或对象都必须初始化才能输出值。

1. **数组有没有 length() 方法? String 有没有 length() 方法？**

答：数组没有 length()方法，有 length 的属性。String 有 length()方法。JavaScript 中，获得字符串的长度是通过 length 属性得到的，这一点容易和 Java 混淆。

1. **是否可以继承 String 类?**

答：String 类是 final 类，不可以被继承。

补充：继承 String 本身就是一个错误的行为，对 String 类型最好的重用方式是关联（HAS-A）而不是继承（IS-A）。

1. **String 和StringBuilder、StringBuffer 的区别?**

答：Java 平台提供了两种类型的字符串：String 和StringBuffer / StringBuilder，它们可以储存和操作字符串。其中 String 是只读字符串，也就意味着 String 引用的字符串内容是不能被改变的。而 StringBuffer 和 StringBuilder 类表示的字符串对象可以直接进行修改。StringBuilder 是 JDK 1.5 中引入的，它和 StringBuffer 的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被 synchronized 修饰，因此它的效率也比 StringBuffer 略高。

有一个面试题问：有没有哪种情况用 + 做字符串连接比调用 StringBuffer / StringBuilder 对象的 append 方法性能更好？如果连接后得到的字符串在静态存储区中是早已存在的，那么用+做字符串连接是优于 StringBuffer / StringBuilder 的 append 方法的。

1. **String s=new String(“xyz”);创建了几个字符串对象？**

答：两个对象，一个是静态存储区的"xyz",一个是用new创建在堆上的对象。

1. **将字符 “12345” 转换成 long 型**

解答： String s=”12345″;

long num=Long.valueOf(s).longValue();

1. **为了显示 myStr = 23 这样的结果，写出在控制台输入的命令**

public class MyClass {

public static void main(String args[]) {

String s1 = args[0];

String s2 = args[1];

String myStr = args[2];

System.out.printin(“myStr =” + s2 + myStr);

}

}

答：java MyClass 1 2 3 4

1. **String s = "Hello";s = s + " world!”; 这两行代码执行后，原始的 String 对象中的内容到底变了没有？**

没有。因为 String 被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。在这段代码中，s 原先指向一个 String 对象，内容是 "Hello"，然后我们对 s 进行了+操作，那么 s 所指向的那个对象是否发生了改变呢？答案是没有。这时， s 不指向原来那个对象了，而指向了另一个 String 对象，内容为 "Hello world!"，原来那个对象还存在于内存之中，只是 s 这个引用变量不再指向它了。

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用 String 来代表字符串的话会引起很大的内存开销。因为 String 对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个 String 对象来表示。这时，应该考虑使用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。

同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都 new 一个 String。例如我们要在构造器中对一个名叫 s 的 String 引用变量进行初始化，把它设置为初始值，应当这样做：

public class Demo {

private String s;

...

public Demo {

s = "Initial Value";

}

...

}

而非 s = new String("Initial Value”);

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为 String 对象不可改变，所以对于内容相同的字符串，只要一个 String 对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的 String 类型属性 s 都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java 认为它们代表同一个 String 对象。而用关键字 new 调用构造器，总是会创建一个新的对象，无论内容是否相同。

至于为什么要把 String 类设计成不可变类，是它的用途决定的。其实不只 String，很多 Java 标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以 Java 标准类库还提供了一个可变版本，即 StringBuffer。

1. **如何把一段逗号分割的字符串转换成一个数组?**

如果不查 jdk api，我很难写出来！我可以说说我的思路：

1 用正则表达式，代码大概为：String [] result = orgStr.split(“,”);

2 用 StingTokenizer ,代码为：

StringTokenizer tokener = StringTokenizer(orgStr,”,”);

String [] result =new String[tokener .countTokens()];

Int i=0;

while(tokener.hasNext(){result[i++]=toker.nextToken();}

1. **下面这条语句一共创建了多少个对象：String s=“a”+”b”+”c”+”d”;**

答：对于如下代码：

String s1 = "a";

String s2 = s1 + "b";

String s3 = "a" + "b";

System.out.println(s2 == "ab");

System.out.println(s3 == "ab");

第一条语句打印的结果为 false ，第二条语句打印的结果为 true，这说明 Javac 编译可以对字符串常量直接相加的表达式进行优化，不必要等到运行期去进行加法运算处理，而是在编译时去掉其中的加号，直接将其编译成一个这些常量相连的结果。

题目中的第一行代码被编译器在编译时优化后，相当于直接定义了一个”abcd”的字符串，所以，上面的代码应该只创建了一个 String 对象。写如下两行代码，

String s ="a" + "b" + "c" + "d";

System.out.println(s== "abcd");

最终打印的结果应该为 true。

1. **String 和 StringBuffer 的区别?**

答：JAVA 平台提供了两个类：String 和 StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符数据。这个 String 类提供了数值不可改变的字符串。而这个 StringBuffer 类提供的字符串进行修改。当你知道字符数据要改变 的时候你就可以使用 StringBuffer。典型地，你可以使用 StringBuffers 来动态构造字符数据。

1. **String 是最基本的数据类型吗？**

答：

不是。Java中的基本数据类型只有8个：byte、short、int、long、float、double、char、boolean；除了基本类型（primitive type）和枚举类型（enumeration type），剩下的都是引用类型（reference type）。

1. **什么情况下用+运算符进行字符串连接比调用StringBuffer/StringBuilder对象的append方法连接字符串性能更好？**
2. **请说出下面程序的输出。**

class StringEqualTest {

public static void main(String[] args) {

String s1 = "Programming";

String s2 = new String("Programming");

String s3 = "Program" + "ming";

System.out.println(s1 == s2);

System.out.println(s1 == s3);

System.out.println(s1 == s1.intern());

}

}

补充：String对象的intern方法会得到字符串对象在常量池中对应的版本的引用（如果常量池中有一个字符串与String对象的equals结果是true），如果常量池中没有对应的字符串，则该字符串将被添加到常量池中，然后返回常量池中字符串的引用。

1. **怎样将GB2312编码的字符串转换为ISO-8859-1编码的字符串？**

String s1 = "你好";

String s2 = new String(s1.getBytes("GB2312"), "ISO-8859-1");

# 日期与时间

## 如何取得年月日、小时分钟秒？

答：

问题1：创建java.util.Calendar 实例，调用其get()方法传入不同的参数即可获得参数所对应的值。Java 8中可以使用java.time.LocalDateTimel来获取，代码如下所示。

public class DateTimeTest {

public static void main(String[] args) {

Calendar cal = Calendar.getInstance();

System.out.println(cal.get(Calendar.YEAR));

System.out.println(cal.get(Calendar.MONTH)); // 0 - 11

System.out.println(cal.get(Calendar.DATE));

System.out.println(cal.get(Calendar.HOUR\_OF\_DAY));

System.out.println(cal.get(Calendar.MINUTE));

System.out.println(cal.get(Calendar.SECOND));

// Java 8

LocalDateTime dt = LocalDateTime.now();

System.out.println(dt.getYear());

System.out.println(dt.getMonthValue()); // 1 - 12

System.out.println(dt.getDayOfMonth());

System.out.println(dt.getHour());

System.out.println(dt.getMinute());

System.out.println(dt.getSecond());

}

}

## **如何取得从1970年1月1日0时0分0秒到现在的毫秒数？**

Calendar.getInstance().getTimeInMillis();

System.currentTimeMillis();

Clock.systemDefaultZone().millis(); // Java 8

## **如何取得某月的最后一天？**

Calendar time = Calendar.getInstance();

time.getActualMaximum(Calendar.DAY\_OF\_MONTH);

## **如何格式化日期？**

利用java.text.DataFormat 的子类（如SimpleDateFormat类）中的format(Date)方法可将日期格式化。Java 8中可以用java.time.format.DateTimeFormatter来格式化时间日期，代码如下所示。

import java.text.SimpleDateFormat;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.util.Date;

class DateFormatTest {

public static void main(String[] args) {

SimpleDateFormat oldFormatter = new SimpleDateFormat("yyyy/MM/dd");

Date date1 = new Date();

System.out.println(oldFormatter.format(date1));

// Java 8

DateTimeFormatter newFormatter = DateTimeFormatter.ofPattern("yyyy/MM/dd");

LocalDate date2 = LocalDate.now();

System.out.println(date2.format(newFormatter));

}

}

## 打印昨天的当前时刻。

import java.util.Calendar;

class YesterdayCurrent {

public static void main(String[] args){

Calendar cal = Calendar.getInstance();

cal.add(Calendar.DATE, -1);

System.out.println(cal.getTime());

}

}

在Java 8中，可以用下面的代码实现相同的功能。

import java.time.LocalDateTime;

class YesterdayCurrent {

public static void main(String[] args) {

LocalDateTime today = LocalDateTime.now();

LocalDateTime yesterday = today.minusDays(1);

System.out.println(yesterday);

}

}

# 正则表达式

## 简述正则表达式及其用途

答：在编写处理字符串的程序时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。

说明：计算机诞生初期处理的信息几乎都是数值，但是时过境迁，今天我们使用计算机处理的信息更多的时候不是数值而是字符串，正则表达式就是在进行字符串匹配和处理的时候最为强大的工具，绝大多数语言都提供了对正则表达式的支持。

## Java中是如何支持正则表达式操作的

答：Java中的String类提供了支持正则表达式操作的方法，包括：matches()、replaceAll()、replaceFirst()、split()。此外，Java中可以用Pattern类表示正则表达式对象，它提供了丰富的API进行各种正则表达式操作。

# 循环控制语句

## 在Java中，如何跳出当前的多重嵌套循环？

答：在最外层循环前加一个标记如A，然后用break A;可以跳出多重循环。（Java中支持带标签的break和continue语句，作用有点类似于C和C++中的goto语句，但是就像要避免使用goto一样，应该避免使用带标签的break和continue，因为它不会让你的程序变得更优雅，很多时候甚至有相反的作用，所以这种语法其实不知道更好）

## 用Java写一个冒泡排序

答：冒泡排序几乎是个程序员都写得出来，但是面试的时候如何写一个逼格高的冒泡排序却不是每个人都能做到，下面提供一个参考代码：

publi cclass bubbleSort {

public bubbleSort(){

int a[]={1,54,6,3,78,34,12,45};

int temp=0;

for(int i=0;i<a.length;i++){

for(int j=i+1;j<a.length;j++){

if(a[i]>a[j]){

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

for(int i=0;i<a.length;i++)

System.out.println(a[i]);

}

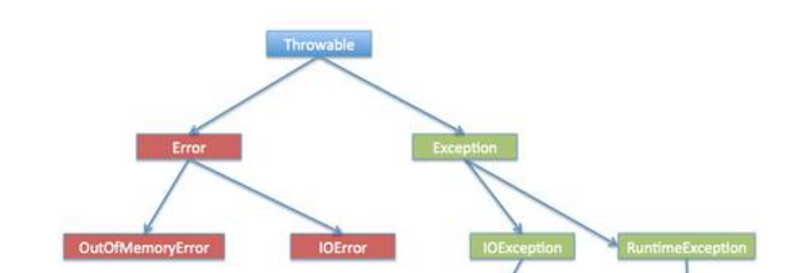
}

# Java异常机制

1、 异常是程序运行过程过程出现的错误，在Java中用类来描述，用对象来表示具体的异常。Java将其区分为Error与Exception，Error是程序无力处理的错误，Exception是程序可以处理的错误。异常处理是为了程序的健壮性。  
2、 Java异常类来自于Java API定义和用户扩展。通过继承Java API异常类可以实现异常的转译。  
3、 异常能处理就处理，不能处理就抛出，最终没有处理的异常JVM会进行处理。  
4、 异常可以传播，也可以相互转译，但应该根据需要选择合理的异常转译的方向。

5、 对于一个应用系统，设计一套良好的异常处理体系很重要。这一点在系统设计的时候就应该考虑到。

## Java异常体系



Thorwable类所有异常和错误的超类，有两个子类Error和Exception，分别表示错误和异常。其中异常类Exception又分为 运行时异常(RuntimeException)和非运行时异常，这两种异常有很大的区别，也称之为不检查异常（Unchecked Exception）和检查异常（Checked Exception）

Error与Exception    Error是程序无法处理的错误，比如OutOfMemoryError、ThreadDeath等。这些异常发生时，Java虚拟机（JVM）一般会选择线程终止。   
    Exception是程序本身可以处理的异常，这种异常分两大类运行时异常和非运行时异常。程序中应当尽可能去处理这些异常。

## 运行时异常和非运行时异常

运行时异常都是RuntimeException类及其子类异常，如NullPointerException、 IndexOutOfBoundsException等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引 起的，程序应该从逻辑角度尽可能避免这类异常的发生。

非运行时异常是RuntimeException以外的异常，类型上都属于Exception类及其子类。从程序语法角度讲是必须进行处理的异常，如果不 处理，程序就不能编译通过。如IOException、SQLException等以及用户自定义的Exception异常，一般情况下不自定义检查异常。

## 异常的处理方式

Java异常处理涉及到五个关键字，分别是：try、catch、finally、throw、throws。下面将骤一介绍，通过认识这五个关键字，掌握基本异常处理知识。

1. 捕获

在java中，异常处理的完整语法是：  
     try{  
      //（尝试运行的）程序代码  
    }catch(异常类型 异常的变量名){  
      //异常处理代码  
    }finally{  
      //异常发生，方法返回之前，总是要执行的代码  
    }

以上语法有三个代码块：

try语句块，表示要尝试运行代码，try语句块中代码受异常监控，其中代码发生异常时，会抛出异常对象。

catch语句块会捕获try代码块中发生的异常并在其代码块中做异常处理，catch语句带一个Throwable类型的参数，表示可捕获异常类型。当 try中出现异常时，catch会捕获到发生的异常，并和自己的异常类型匹配，若匹配，则执行catch块中代码，并将catch块参数指向所抛的异常对 象。catch语句可以有多个，用来匹配多个中的一个异常，一旦匹配上后，就不再尝试匹配别的catch块了。通过异常对象可以获取异常发生时完整的 JVM堆栈信息，以及异常信息和异常发生的原因等。

finally语句块是紧跟catch语句后的语句块，这个语句块总是会在方法返回前执行，而不管是否try语句块是否发生异常。并且这个语句块总是在方 法返回前执行。目的是给程序一个补救的机会。这样做也体现了Java语言的健壮性。

try、catch、finally三个语句块应注意的问题  
    第一、try、catch、finally三个语句块均不能单独使用，三者可以组成 try…catch…finally、try…catch、try…finally三种结构，catch语句可以有一个或多个，finally语句最多一个。  
    第二、try、catch、finally三个代码块中变量的作用域为代码块内部，分别独立而不能相互访问。如果要在三个块中都可以访问，则需要将变量定义到这些块的外面。  
    第三、多个catch块时候，只会匹配其中一个异常类并执行catch块代码，而不会再执行别的catch块，并且匹配catch语句的顺序是由上到下。

1. 抛出

 throw关键字是用于方法体内部，用来抛出一个Throwable类型的异常。如果抛出了检查异常，则还应该在方法头部声明方法可能抛出的异常类型。该方法的调用者也必须检查处理抛出的异常。如果所有方法都层层上抛获取的异常，最终JVM会进行处理，处理也很简单，就是打印异常消息和堆栈信息。如果抛出 的是Error或RuntimeException，则该方法的调用者可选择处理该异常。有关异常的转译会在下面说明。    throws关键字用于方法体外部的方法声明部分，用来声明方法可能会抛出某些异常。仅当抛出了检查异常，该方法的调用者才必须处理或者重新抛出该异常。 当方法的调用者无力处理该异常的时候，应该继续抛出，而不是囫囵吞枣一般在catch块中打印一下堆栈信息做个勉强处理。下面给出一个简单例子，看看如何 使用这两个关键字：

public static void test3() throws Exception{  
      //抛出一个检查异常  
            throw new Exception(“方法test3中的Exception”);  
        }

## Throwable类中的常用方法

 getCause()：返回抛出异常的原因。如果 cause 不存在或未知，则返回 null。  
 getMessage()：返回异常的消息信息。  
 printStackTrace()：对象的堆栈跟踪输出至错误输出流，作为字段 System.err 的值。

## 异常处理的一般原则

  1、 能处理就早处理，抛出不去还不能处理的就想法消化掉或者转换为RuntimeException处理。因为对于一个应用系统来说，抛出大量异常是有问题的，应该从程序开发角度尽可能的控制异常发生的可能。  
   2、 对于检查异常，如果不能行之有效的处理，还不如转换为RuntimeException抛出。这样也让上层的代码有选择的余地――可处理也可不处理。  
   3、 对于一个应用系统来说，应该有自己的一套异常处理框架，这样当异常发生时，也能得到统一的处理风格，将优雅的异常信息反馈给用户。

1. 下面关于java.lang.Exception类的说法正确的是（）

A. 继承自 Throwable

B. Serialable

CD 不记得，反正不正确

答案：A

解析：Java 异常的基类为 java.lang.Throwable，java.lang.Error 和 java.lang.Exception 继承 Throwable，RuntimeException 和其它的 Exception 等继承 Exception，具体的 RuntimeException 继承 RuntimeException。

## 扩展：错误和异常的区别(Error vs Exception)

1) java.lang.Error: Throwable 的子类，用于标记严重错误。合理的应用程序不应该去 try/catch 这种错误。绝大多数的错误都是非正常的，就根本不该出现的。

java.lang.Exception: Throwable 的子类，用于指示一种合理的程序想去 catch 的条件。即它仅仅是一种程序运行条件，而非严重错误，并且鼓励用户程序去 catch 它。

2) Error 和 RuntimeException 及其子类都是未检查的异常（unchecked exceptions），而所有其他的 Exception 类都是检查了的异常（checked exceptions）

checked exceptions: 通常是从一个可以恢复的程序中抛出来的，并且最好能够从这种异常中使用程序恢复。比如 FileNotFoundException, ParseException 等。检查了的异常发生在编译阶段，必须要使用 try…catch（或者 throws ）否则编译不通过。

unchecked exceptions: 通常是如果一切正常的话本不该发生的异常，但是的确发生了。发生在运行期，具有不确定性，主要是由于程序的逻辑问题所引起的。比如 ArrayIndexOutOfBoundException, ClassCastException 等。从语言本身的角度讲，程序不该去 catch 这类异常，虽然能够从诸如 RuntimeException 这样的异常中 catch 并恢复，但是并不鼓励终端程序员这么做，因为完全没要必要。因为这类错误本身就是 bug，应该被修复，出现此类错误时程序就应该立即停止执行。 因此，面对 Errors 和 unchecked exceptions 应该让程序自动终止执行，程序员不该做诸如 try/catch 这样的事情，而是应该查明原因，修改代码逻辑。

RuntimeException：RuntimeException体系包括错误的类型转换、数组越界访问和试图访问空指针等等。处理 RuntimeException 的原则是：如果出现 RuntimeException，那么一定是程序员的错误。例如，可以通过检查数组下标和数组边界来避免数组越界访问异常。其他（IOException等等）checked 异常一般是外部错误，例如试图从文件尾后读取数据等，这并不是程序本身的错误，而是在应用环境中出现的外部错误。

3. getCustomerInfo() 方法如下，try 中可以捕获三种类型的异常，如果在该方法运行中产生了一个 IOException，将会输出什么结果?

public void getCustomerInfo() {

try {

// do something that may cause an Exception

} catch (java.io.FileNotFoundException ex) {

System.out.print("FileNotFoundException!");

} catch (java.io.IOException ex) {

System.out.print("IOException!");

} catch (java.lang.Exception ex) {

System.out.print("Exception!");

}

}

A. IOException!

B. IOException!Exception!

C. FileNotFoundException!IOException!

D. FileNotFoundException!IOException!Exception!

答案：A

解析：考察多个 catch 语句块的执行顺序。当用多个 catch 语句时，catch 语句块在次序上有先后之分。从最前面的 catch 语句块依次先后进行异常类型匹配，这样如果父异常在子异常类之前，那么首先匹配的将是父异常类，子异常类将不会获得匹配的机会，也即子异常类型所在的 catch 语句块将是不可到达的语句。所以，一般将父类异常类即 Exception 老大放在 catch 语句块的最后一个。

## try{} 里有一个 return 语句，那么紧跟在这个 try 后的 finally{} 里的 code 会不会被执行，什么时候被执行，在 return 前还是后?

答：会执行，在方法返回调用者前执行。

Java 允许在 finally 中改变返回值的做法是不好的，因为如果存在 finally 代码块，try 中的 return 语句不会立马返回调用者，而是记录下返回值待 finally 代码块执行完毕之后再向调用者返回其值，然后如果在 finally 中修改了返回值，这会对程序造成很大的困扰，C# 中就从语法上规定不能做这样的事。

## Java 语言如何进行异常处理，关键字：throws、throw、try、catch、finally 分别如何使用？

答：Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在 Java 中，每个异常都是一个对象，它是 Throwable 类或其子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并进行处理。Java 的异常处理是通过 5 个关键词来实现的：try、catch、throw、throws 和 finally。一般情况下是用 try 来执行一段程序，如果出现异常，系统会抛出（throw）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；try 用来指定一块预防所有“异常”的程序；catch 子句紧跟在 try 块后面，用来指定你想要捕捉的“异常”的类型；throw 语句用来明确地抛出一个“异常”；throws用来标明一个成员函数可能抛出的各种“异常”；finally 为确保一段代码不管发生什么“异常”都被执行一段代码；可以在一个成员函数调用的外面写一个 try 语句，在这个成员函数内部写另一个 try 语句保护其他代码。每当遇到一个 try 语句，“异常”的框架就放到栈上面，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种“异常”进行处理，栈就会展开，直到遇到有处理这种“异常”的 try 语句。

## 运行时异常与受检异常有何异同？

答：异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误，只要程序设计得没有问题通常就不会发生。受检异常跟程序运行的上下文环境有关，即使程序设计无误，仍然可能因使用的问题而引发。Java编译器要求方法必须声明抛出可能发生的受检异常，但是并不要求必须声明抛出未被捕获的运行时异常。异常和继承一样，是面向对象程序设计中经常被滥用的东西，神作《Effective Java》中对异常的使用给出了以下指导原则：

不要将异常处理用于正常的控制流（设计良好的API不应该强迫它的调用者为了正常的控制流而使用异常）

对可以恢复的情况使用受检异常，对编程错误使用运行时异常

避免不必要的使用受检异常（可以通过一些状态检测手段来避免异常的发生）

优先使用标准的异常

每个方法抛出的异常都要有文档

保持异常的原子性

不要在 catch 中忽略掉捕获到的异常

## 请写出 5 种常见到的runtime exception。

答：

NullPointerException：当操作一个空引用时会出现此错误。

NumberFormatException：数据格式转换出现问题时出现此异常。

ClassCastException：强制类型转换类型不匹配时出现此异常。

ArrayIndexOutOfBoundsException：数组下标越界，当使用一个不存在的数组下标时出现此异常。

ArithmeticException：数学运行错误时出现此异常

## error 和 exception 有什么区别?

答：

error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况； exception 表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

# JAVA中的集合

## HashSet vs. TreeSet vs. LinkedHashSet 比较

Set集合不包含重复的元素，这是使用Set的主要原因。有三种常见的Set实现——HashSet, TreeSet和LinkedHashSet。什么时候使用它们，使用哪个是个重要的问题。总体而言，如果你需要一个访问快速的Set，你应该使用HashSet；当你需要一个排序的Set，你应该使用TreeSet；当你需要记录下插入时的顺序时，你应该使用LinedHashSet。

HashSet是采用hash表来实现的。其中的元素没有按顺序排列，add()、remove()以及contains()等方法都是复杂度为O(1)的方法。

TreeSet是采用树结构实现(红黑树算法)。元素是按顺序进行排列，但是add()、remove()以及contains()等方法都是复杂度为O(log (n))的方法。它还提供了一些方法来处理排序的set，如first(), last(), headSet(), tailSet()等等。

LinkedHashSet介于HashSet和TreeSet之间。它也是一个hash表，但是同时维护了一个双链表来记录插入的顺序。基本方法的复杂度为O(1)。

1. 下列说法正确的是（）

A. LinkedList继承自List B. AbstractSet继承自Set C. HashSet继承自AbstractSet D. WeakMap继承自HashMap

答案：AC

2. ArrayList list = new ArrayList(20);中的 list 扩充几次?

A. 0

B. 1

C. 2

D. 3

答案：A

解析：这里有点迷惑人，大家都知道默认 ArrayList 的长度是 10 个，所以如果你要往 list 里添加 20 个元素肯定要扩充一次（扩充为原来的 1.5 倍），但是这里显示指明了需要多少空间，所以就一次性为你分配这么多空间，也就是不需要扩充了。

## Java集合类框架的基本接口有哪些？

Java 集合类提供了一套设计良好的支持对一组对象进行操作的接口和类。Java集合类里面最基本的接口有：

Collection：代表一组对象，每一个对象都是它的子元素。

Set：不包含重复元素的 Collection。

List：有顺序的 collection，并且可以包含重复元素。

Map：可以把键(key)映射到值(value)的对象，键不能重复。

## 为什么集合类没有实现 Cloneable 和 Serializable 接口？

集合类接口指定了一组叫做元素的对象。集合类接口的每一种具体的实现类都可以选择以它自己的方式对元素进行保存和排序。有的集合类允许重复的键，有些不允许。

## 什么是迭代器(Iterator)？

Iterator 接口提供了很多对集合元素进行迭代的方法。每一个集合类都包含了可以返回迭代器实例的 迭代方法。迭代器可以在迭代的过程中删除底层集合的元素。

克隆(cloning)或者是序列化(serialization)的语义和含义是跟具体的实现相关的。因此，应该由集合类的具体实现来决定如何被克隆或者是序列化。

## Iterator和ListIterator的区别是什么？

下面列出了他们的区别：

Iterator 可用来遍历 Set 和 List 集合，但是 ListIterator 只能用来遍历 List 。

Iterator 对集合只能是前向遍历，ListIterator 既可以前向也可以后向。

ListIterator 实现了 Iterator 接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引，等等。

## 快速失败(fail-fast)和安全失败(fail-safe)的区别是什么？

Iterator 的安全失败是基于对底层集合做拷贝，因此，它不受源集合上修改的影响。java.util 包下面的所有的集合类都是快速失败的，而 java.util.concurrent 包下面的所有的类都是安全失败的。快速失败的迭代器会抛出 ConcurrentModificationException 异常，而安全失败的迭代器永远不会抛出这样的异常。

## Java 中的 HashMap 的工作原理是什么？

Java 中的 HashMap 是以键值对(key-value)的形式存储元素的。HashMap 需要一个hash函数，它使用 hashCode()和 equals()方法来向集合/从集合添加和检索元素。当调用 put()方法的时候，HashMap会计算 key 的 hash 值，然后把键值对存储在集合中合适的索引上。如果 key 已经存在了，value 会被更新成新值。HashMap 的一些重要的特性是它的容量(capacity)，负载因子(load factor)和扩容极限(threshold resizing)。

## hashCode() 和 equals() 方法的重要性体现在什么地方？

Java 中的 HashMap 使用 hashCode() 和 equals() 方法来确定键值对的索引，当根据键获取值的时候也会用到这两个方法。如果没有正确的实现这两个方法，两个不同的键可能会有相同的 hash 值，因此，可能会被集合认为是相等的。而且，这两个方法也用来发现重复元素。所以这两个方法的实现对 HashMap 的精确性和正确性是至关重要的。

## HashMap 和 Hashtable 有什么区别？

HashMap 和 Hashtable 都实现了 Map 接口，因此很多特性非常相似。但是，他们有以下不同点： HashMap 允许键和值是 null，而 Hashtable 不允许键或者值是 null。

Hashtable 是同步的，而 HashMap 不是。因此， HashMap 更适合于单线程环境，而 Hashtable 适合于多线程环境。

HashMap 提供了可供应用迭代的键的集合，因此，HashMap 是快速失败的。另一方面，Hashtable 提供了对键的列举(Enumeration)。

一般认为 Hashtable 是一个遗留的类。

## 数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用 Array 而不是 ArrayList？

下面列出了 Array 和 ArrayList 的不同点：

Array 可以包含基本类型和对象类型，ArrayList 只能包含对象类型。

Array 大小是固定的，ArrayList 的大小是动态变化的。

ArrayList 提供了更多的方法和特性，比如：addAll()，removeAll()，iterator()等等。 对于基本类型数据，集合使用自动装箱来减少编码工作量。但是，当处理固定大小的基本数据类型的时候，这种方式相对比较慢。

## ArrayList 和 LinkedList 有什么区别？

ArrayList 和 LinkedList 都实现了 List 接口，他们有以下的不同点：

ArrayList 是基于索引的数据接口，它的底层是数组。它可以以O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList 是以元素列表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是O(n)。

相对于 ArrayList，LinkedList 的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。

LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

也可以参考 ArrayList vs. LinkedList。

## Comparable 和Comparator 接口是干什么的？列出它们的区别。

Java 提供了只包含一个 compareTo() 方法的 Comparable 接口。这个方法可以个给两个对象排序。具体来说，它返回负数，0，正数来表明输入对象小于，等于，大于已经存在的对象。

Java 提供了包含 compare() 和 equals() 两个方法的 Comparator 接口。compare() 方法用来给两个输入参数排序，返回负数，0，正数表明第一个参数是小于，等于，大于第二个参数。equals() 方法需要一个对象作为参数，它用来决定输入参数是否和 comparator 相等。只有当输入参数也是一个 comparator 并且输入参数和当前 comparator 的排序结果是相同的时候，这个方法才返回 true。

## Java集合类框架的最佳实践有哪些？

根据应用的需要正确选择要使用的集合的类型对性能非常重要，比如：假如元素的大小是固定的，而且能事先知道，我们就应该用 Array 而不是 ArrayList。 有些集合类允许指定初始容量。因此，如果我们能估计出存储的元素的数目，我们可以设置初始容量来避免重新计算 hash 值或者是扩容。

为了类型安全，可读性和健壮性的原因总是要使用泛型。同时，使用泛型还可以避免运行时的 ClassCastException。

使用 JDK 提供的不变类(immutable class)作为Map的键可以避免为我们自己的类实现 hashCode() 和 equals() 方法。

编程的时候接口优于实现。

底层的集合实际上是空的情况下，返回长度是0的集合或者是数组，不要返回 null。

## Enumeration 接口和 Iterator 接口的区别有哪些？

Enumeration 速度是 Iterator 的2倍，同时占用更少的内存。但是，Iterator 远远比 Enumeration 安全，因为其他线程不能够修改正在被 iterator 遍历的集合里面的对象。同时，Iterator 允许调用者删除底层集合里面的元素，这对 Enumeration 来说是不可能的。

## HashSet 和 TreeSet 有什么区别？

HashSet 是由一个 hash 表来实现的，因此，它的元素是无序的。add()，remove()，contains()方法的时间复杂度是 O(1)。

另一方面，TreeSet 是由一个树形的结构来实现的，它里面的元素是有序的。因此，add()，remove()，contains() 方法的时间复杂度是 O(logn)。

## List、Set、Map 是否继承自 Collection 接口？

答：List、Set 是，Map 不是。Map 是键值对映射容器，与 List 和 Set 有明显的区别，而 Set 存储的零散的元素且不允许有重复元素（数学中的集合也是如此），List 是线性结构的容器，适用于按数值索引访问元素的情形。

## 说出 ArrayList、Vector、LinkedList 的存储性能和特性？

答：ArrayList 和 Vector 都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector 由于使用了 synchronized 方法（线程安全），通常性能上较 ArrayList 差，而 LinkedList 使用双向链表实现存储（将内存中零散的内存单元通过附加的引用关联起来，形成一个可以按序号索引的线性结构，这种链式存储方式与数组的连续存储方式相比，其实对内存的利用率更高），按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

Vector 属于遗留容器（早期的 JDK 中使用的容器，除此之外 Hashtable、Dictionary、BitSet、Stack、Properties 都是遗留容器），现在已经不推荐使用，但是由于 ArrayList 和 LinkedListed 都是非线程安全的，如果需要多个线程操作同一个容器，那么可以通过工具类 Collections 中的 synchronizedList 方法将其转换成线程安全的容器后再使用（这其实是装潢模式最好的例子，将已有对象传入另一个类的构造器中创建新的对象来增加新功能）。

## List、Map、Set 三个接口存储元素时各有什么特点？

答：

1）List 是有序的 Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在 List 中的位置，类似于数组下标）来访问 List 中的元素，这类似于 Java 的数组。

2）Set 是一种不包含重复的元素的 Collection，即任意的两个元素 e1 和 e2 都有e1.equals(e2)=false，Set 最多有一个 null 元素。

3）Map 接口 ：请注意，Map 没有继承 Collection 接口，Map 提供 key 到 value 的映射

## 判断下列语句是否正确，如果有错误，请指出错误所在？

List a = new ArrayList();

a.add(5);

答：错误,默认封装 int 类型。

## 你是怎么理解 Java 泛型的？

答： 在 Java SE 1.5之前，没有泛型的情况的下，通过对类型 Object 的引用来实现参数的“任意化”，“任意化”带来的缺点是要做显式的强制类型转换，而这种转换是要求开发者对实际参数类型可以预知的情况下进行的。对于强制类型转换错误的情况，编译器可能不提示错误，在运行的时候才出现异常，这是一个安全隐患。

泛型是 Java SE 1.5的新特性，泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口、泛型方法。

泛型的好处是在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，提高代码的重用率。

## Collection和Collections的区别？

答：Collection是一个接口，它是Set、List等容器的父接口；Collections是个一个工具类，提供了一系列的静态方法来辅助容器操作，这些方法包括对容器的搜索、排序、线程安全化等等。

## TreeMap和TreeSet在排序时如何比较元素？Collections工具类中的sort()方法如何比较元素？

答：TreeSet要求存放的对象所属的类必须实现Comparable接口，该接口提供了比较元素的compareTo()方法，当插入元素时会回调该方法比较元素的大小。TreeMap要求存放的键值对映射的键必须实现Comparable接口从而根据键对元素进行排序。Collections工具类的sort方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象比较实现Comparable接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是Comparator接口的子类型（需要重写compare方法实现元素的比较），相当于一个临时定义的排序规则，其实就是通过接口注入比较元素大小的算法，也是对回调模式的应用（Java中对函数式编程的支持）。

# 输入输出流

## Java中如何实现序列化，有什么意义？

答：序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决对象流读写操作时可能引发的问题（如果不进行序列化可能会存在数据乱序的问题）。

要实现序列化，需要让一个类实现Serializable接口，该接口是一个标识性接口，标注该类对象是可被序列化的，然后使用一个输出流来构造一个对象输出流并通过writeObject(Object)方法就可以将实现对象写出（即保存其状态）；如果需要反序列化则可以用一个输入流建立对象输入流，然后通过readObject方法从流中读取对象。序列化除了能够实现对象的持久化之外，还能够用于对象的深度克隆。

## **Java中有几种类型的流？**

答：字节流和字符流。字节流继承于InputStream、OutputStream，字符流继承于Reader、Writer。在java.io 包中还有许多其他的流，主要是为了提高性能和使用方便。关于Java的I/O需要注意的有两点：一是两种对称性（输入和输出的对称性，字节和字符的对称性）；二是两种设计模式（适配器模式和装潢模式）。另外Java中的流不同于C#的是它只有一个维度一个方向。

## 编程实现文件拷贝。（这个题目在笔试的时候经常出现，下面的代码给出了两种实现方案）

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.nio.ByteBuffer;

import java.nio.channels.FileChannel;

public final class MyUtil {

private MyUtil() {

throw new AssertionError();

}

public static void fileCopy(String source, String target) throws IOException {

try (InputStream in = new FileInputStream(source)) {

try (OutputStream out = new FileOutputStream(target)) {

byte[] buffer = new byte[4096];

int bytesToRead;

while((bytesToRead = in.read(buffer)) != -1) {

out.write(buffer, 0, bytesToRead);

}

}

}

}

public static void fileCopyNIO(String source, String target) throws IOException {

try (FileInputStream in = new FileInputStream(source)) {

try (FileOutputStream out = new FileOutputStream(target)) {

FileChannel inChannel = in.getChannel();

FileChannel outChannel = out.getChannel();

ByteBuffer buffer = ByteBuffer.allocate(4096);

while(inChannel.read(buffer) != -1) {

buffer.flip();

outChannel.write(buffer);

buffer.clear();

}

}

}

}

}

## 写一个方法，输入一个文件名和一个字符串，统计这个字符串在这个文件中出现的次数。

import java.io.BufferedReader;

import java.io.FileReader;

public final class MyUtil {

// 工具类中的方法都是静态方式访问的因此将构造器私有不允许创建对象(绝对好习惯)

private MyUtil() {

throw new AssertionError();

}

/\*\*

\* 统计给定文件中给定字符串的出现次数

\*

\* @param filename 文件名

\* @param word 字符串

\* @return 字符串在文件中出现的次数

\*/

public static int countWordInFile(String filename, String word) {

int counter = 0;

try (FileReader fr = new FileReader(filename)) {

try (BufferedReader br = new BufferedReader(fr)) {

String line = null;

while ((line = br.readLine()) != null) {

int index = -1;

while (line.length() >= word.length() && (index = line.indexOf(word)) >= 0) {

counter++;

line = line.substring(index + word.length());

}

}

}

} catch (Exception ex) {

ex.printStackTrace();

}

return counter;

}

}

## 如何用Java代码列出一个目录下所有的文件？

答：

如果只要求列出当前文件夹下的文件，代码如下所示：

import java.io.File;

class Test12 {

public static void main(String[] args) {

File f = new File("/Users/Hao/Downloads");

for(File temp : f.listFiles()) {

if(temp.isFile()) {

System.out.println(temp.getName());

}

}

}

}

1. 下面哪个流类属于面向字符的输入流?

A. BufferedWriter

B. FileInputStream

C. ObjectInputStream

D. InputStreamReader

答案：D

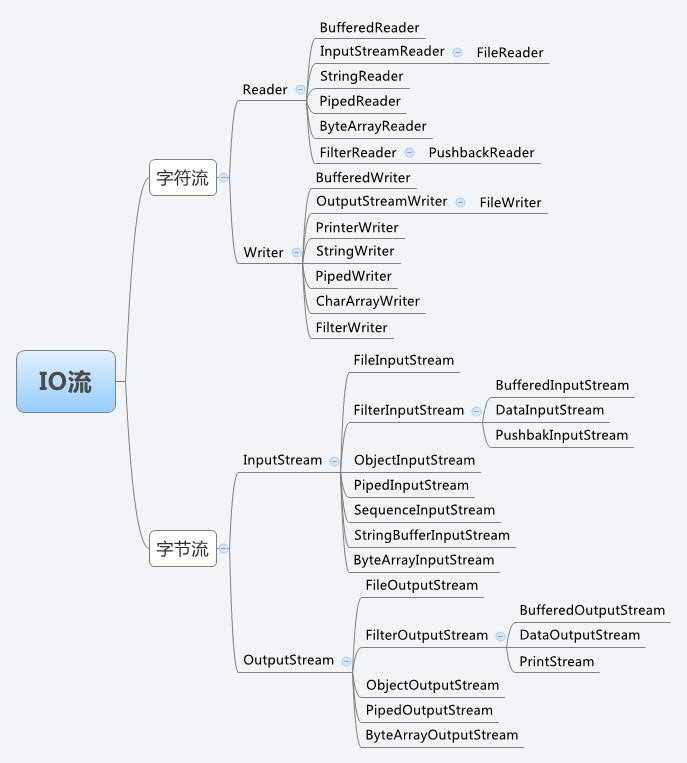
解析：Java 的 IO 操作中有面向字节(Byte)和面向字符(Character)两种方式。

面向字节的操作为以 8 位为单位对二进制的数据进行操作，对数据不进行转换，这些类都是 InputStream 和 OutputStream 的子类。

面向字符的操作为以字符为单位对数据进行操作，在读的时候将二进制数据转为字符，在写的时候将字符转为二进制数据，这些类都是 Reader 和 Writer 的子类。

总结：以 InputStream（输入）/OutputStream（输出）为后缀的是字节流；以Reader（输入）/Writer（输出）为后缀的是字符流。

扩展：Java 流类图结构，一目了然，解决大部分选择题：



2. 阅读 Shape 和 Circle 两个类的定义。在序列化一个 Circle 的对象 circle 到文件时，下面哪个字段会被保存到文件中？

class Shape {

public String name;

}

class Circle extends Shape implements Serializable{

private float radius;

transient int color;

public static String type = "Circle";

}

A. name B. radius C. color D. type

答案：B

3. 什么是 Java 序列化，如何实现 Java 序列化？

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决在对对象流进行读写操作时所引发的问题。

序列化的实现：将需要被序列化的类实现 Serializable 接口，该接口没有需要实现的方法， implements Serializable 只是为了标注该对象是可被序列化的，然后使用一个输出流(如：FileOutputStream)来构造一个ObjectOutputStream(对象流)对象，接着，使用ObjectOutputStream对象的writeObject(Object obj)方法就可以将参数为obj的对象写出(即保存其状态)，要恢复的话则用输入流。

# JAVA与内存

## 堆栈和堆

在java中有两类内存。分别称为stack（堆栈）和heap（堆）。

stack是程序内存空间，因此所有的基本类型和对象的引用是存在stack中。

heap是java虚拟机储存对象的，它是一个巨大的内存，当你创造一个对象，java虚拟机把对象放入heap中，把创造的对象的地址放入stack中。

因此，基本类型、对象的引用储存在stack中；对象储存在heap中。

## java中的垃圾回收机制

当你new一个新的对象，java分配必需的内存。当你用完一个对象时，java的垃圾回收器为你把内存收回。

垃圾回收以线程的形式在后台运行，寻找那些无有用引用（reference）的对象，发现之后便销毁对象，并收回内存。

垃圾回收是在java虚拟机间实现的，它们通常有相同的步骤，首先垃圾回收器获得正在运行的线程和所有已经加载的类的快照，

然后所有线程中涉及到的对象被标记为最近使用的，当可能涉及的对象都被标记的时候，剩下没标记的就被舍弃。

为了帮助虚拟机，我们主动移除一些不在需要的对象是一个不错的做法，可以通过将引用设置为null来实现。

eg：

Test t = new Test();

t.someAction();

//all done

t = null;

## GC线程是否为守护线程？（）

答案：是

解析：线程分为守护线程和非守护线程（即用户线程）。

只要当前JVM实例中尚存在任何一个非守护线程没有结束，守护线程就全部工作；只有当最后一个非守护线程结束时，守护线程随着 JVM 一同结束工作。 守护线程最典型的应用就是 GC (垃圾回收器)

## 解释内存中的栈（stack）、堆(heap)和静态存储区的用法。

答：通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用内存中的栈空间；而通过new关键字和构造器创建的对象放在堆空间；程序中的字面量（literal）如直接书写的100、“hello”和常量都是放在静态存储区中。栈空间操作最快但是也很小，通常大量的对象都是放在堆空间，整个内存包括硬盘上的虚拟内存都可以被当成堆空间来使用。

String str = new String(“hello”);

上面的语句中 str 放在栈上，用 new 创建出来的字符串对象放在堆上，而“hello”这个字面量放在静态存储区。

## Java 中会存在内存泄漏吗，请简单描述。

答：理论上 Java 因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是 Java 被广泛使用于服务器端编程的一个重要原因）；然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被 GC 回收也会发生内存泄露。一个例子就是 Hibernate 的Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象。下面的例子也展示了 Java 中发生内存泄露的情况：

package com.lovo;

import java.util.Arrays;

import java.util.EmptyStackException;

public class MyStack<T> {

private T[] elements;

private int size = 0;

private static final int INIT\_CAPACITY = 16;

public MyStack() {

elements = (T[]) new Object[INIT\_CAPACITY];

}

public void push(T elem) {

ensureCapacity();

elements[size++] = elem;

}

public T pop() {

if(size == 0)

throw new EmptyStackException();

return elements[--size];

}

private void ensureCapacity() {

if(elements.length == size) {

elements = Arrays.copyOf(elements, 2 \* size + 1);

}

}

}

上面的代码实现了一个栈（先进后出（FILO））结构，乍看之下似乎没有什么明显的问题，它甚至可以通过你编写的各种单元测试。然而其中的 pop 却存在内存泄露的问题，当我们用 pop 方法弹出栈中的对象时，该对象不会被当作垃圾回收，即使使用栈的程序不再引用这些对象，因为栈内部维护着对这些对象的过期引用（obsolete reference）。在支持垃圾回收的语言中，内存泄露是很隐蔽的，这种内存泄露其实就是无意识的对象保持。如果一个对象引用被无意识的保留起来了，那么垃圾回收器不会处理这个对象，也不会处理该对象引用的其他对象，即使这样的对象只有少数几个，也可能会导致很多的对象被排除在垃圾回收之外，从而对性能造成重大影响，极端情况下会引发 Disk Paging（物理内存与硬盘的虚拟内存交换数据），甚至造成 OutOfMemoryError。

## GC 是什么？为什么要有 GC？

答：GC 是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或Runtime.getRuntime().gc() ，但 JVM 可以屏蔽掉显示的垃圾回收调用。

垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在 Java 诞生初期，垃圾回收是 Java 最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今 Java 的垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得 iOS 的系统比 Android 系统有更好的用户体验，其中一个深层次的原因就在于 Android 系统中垃圾回收的不可预知性。

第 3 行中生成的 object在第几行执行后成为 garbage collection 的对象？

1.public class MyClass {

2. public StringBuffer aMethod() {

3. StringBuffer sf = new StringBuffer(“Hello”);

4. StringBuffer[] sf\_arr = new StringBuffer[1];

5. sf\_arr[0] = sf;

6. sf = null;

7. sf\_arr[0] = null;

8. return sf;

9. }

10.}

答：第 7 行

## 描述一下 JVM 加载 class 文件的原理机制?

答：JVM 中类的装载是由类加载器（ClassLoader） 和它的子类来实现的，Java 中的类加载器是一个重要的 Java 运行时系统组件，它负责在运行时查找和装入类文件中的类。

由于 Java 的跨平台性，经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接(验证、准备和解析)和初始化。类的加载是指把类的 .class 文件中的数据读入到内存中，通常是创建一个字节数组读入 .class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备(为静态变量分配内存并设置默认的初始值)和解析(将符号引用替换为直接引用)三个步骤。最后 JVM 对类进行初始化，包括：1. 如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2. 如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader的子类）。从JDK 1.2开始，类加载过程采取了父亲委托机制(PDM)。PDM 更好的保证了 Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM 不会向 Java 程序提供对 Bootstrap 的引用。下面是关于几个类加载器的说明：

a)Bootstrap：一般用本地代码实现，负责加载JVM基础核心类库（rt.jar）；

b)Extension：从 java.ext.dirs 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap；

c)System：又叫应用类加载器，其父类是Extension。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

## 解释内存中的栈(stack)、堆(heap)和静态区(static area)的用法。

答：通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用内存中的栈空间；而通过new关键字和构造器创建的对象放在堆空间；程序中的字面量（literal）如直接书写的100、"hello"和常量都是放在静态区中。栈空间操作起来最快但是栈很小，通常大量的对象都是放在堆空间，理论上整个内存没有被其他进程使用的空间甚至硬盘上的虚拟内存都可以被当成堆空间来使用。

String str = new String("hello");

上面的语句中变量str放在栈上，用new创建出来的字符串对象放在堆上，而"hello"这个字面量放在静态区。

补充：较新版本的Java（从Java 6的某个更新开始）中使用了一项叫"逃逸分析"的技术，可以将一些局部对象放在栈上以提升对象的操作性能。

# Java多线程

1. 下面哪些是Thread类的方法?

A start() B run() C exit() D getPriority()

答案：ABD

解析：看 Java API docs吧：http://docs.oracle.com/javase/7/docs/api/， exit() 是 System 类的方法，如System.exit(0)。

2. 下面程序的运行结果?

public static void main(String args[]) {

Thread t = new Thread() {

public void run() {

pong();

}

};

t.run();

System.out.print("ping");

}

static void pong() {

System.out.print("pong");

}

A. pingpong

B. pongping

C. pingpong和pongping都有可能

D. 都不输出

答案：B

解析：这里考的是 Thread 类中 start() 和 run() 方法的区别了。start() 用来启动一个线程，当调用 start 方法后，系统才会开启一个新的线程，进而调用 run() 方法来执行任务，而单独的调用run() 就跟调用普通方法是一样的，已经失去线程的特性了。因此在启动一个线程的时候一定要使用 start() 而不是 run()。

## 进程和线程的区别是什么？

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是操作系统进行资源分配和调度的一个独立单位；

线程是进程的一个实体，是CPU调度和分派的基本单位，是比进程更小的能独立运行的基本单位。线程的划分尺度小于进程，这使得多线程程序的并发性高；进程在执行时通常拥有独立的内存单元，而线程之间可以共享内存。使用多线程的编程通常能够带来更好的性能和用户体验，但是多线程的程序对于其他程序是不友好的，因为它可能占用了更多的CPU资源。当然，也不是线程越多，程序的性能就越好，因为线程之间的调度和切换也会浪费CPU时间。时下很时髦的Node.js就采用了单线程异步I/O的工作模式。

## 创建线程有几种不同的方式？你喜欢哪一种？为什么？

有三种方式可以用来创建线程：

继承 Thread 类

实现 Runnable 接口

应用程序可以使用 Executor 框架来创建线程池

实现 Runnable 接口这种方式更受欢迎，因为这不需要继承 Thread 类。在应用设计中已经继承了别的对象的情况下，这需要多继承（而 Java 不支持多继承），只能实现接口。同时，线程池也是非常高效的，很容易实现和使用。

## 概括的解释下线程的几种可用状态。

线程在执行过程中，可以处于下面几种状态：

就绪(Runnable):线程准备运行，不一定立马就能开始执行。

运行中(Running)：进程正在执行线程的代码。

等待中(Waiting):线程处于阻塞的状态，等待外部的处理结束。

睡眠中(Sleeping)：线程被强制睡眠。

I/O阻塞(Blocked on I/O)：等待I/O操作完成。

同步阻塞(Blocked on Synchronization)：等待获取锁。

死亡(Dead)：线程完成了执行。

## 同步方法和同步代码块的区别是什么？

在 Java 语言中，每一个对象有一把锁。线程可以使用 synchronized 关键字来获取对象上的锁。 synchronized 关键字可应用在方法级别(粗粒度锁)或者是代码块级别(细粒度锁)。

## 在监视器(Monitor)内部，是如何做线程同步的？程序应该做哪种级别的同步？

监视器和锁在 Java 虚拟机中是一块使用的。监视器监视一块同步代码块，确保一次只有一个线程执行同步代码块。每一个监视器都和一个对象引用相关联。线程在获取锁之前不允许执行同步代码。

## 什么是死锁(deadlock)？

两个进程都在等待对方执行完毕才能继续往下执行的时候就发生了死锁。结果就是两个进程都陷入了无限的等待中。

## 如何确保 N 个线程可以访问 N 个资源同时又不导致死锁？

使用多线程的时候，一种非常简单的避免死锁的方式就是：指定获取锁的顺序，并强制线程按照指定的顺序获取锁。因此，如果所有的线程都是以同样的顺序加锁和释放锁，就不会出现死锁了

## sleep() 和 wait() 有什么区别?

答：sleep()方法是线程类（Thread）的静态方法，导致此线程暂停执行指定时间，将执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复（线程回到就绪（ready）状态），因为调用 sleep 不会释放对象锁。wait() 是 Object 类的方法，对此对象调用 wait()方法导致本线程放弃对象锁(线程暂停执行)，进入等待此对象的等待锁定池，只有针对此对象发出 notify 方法（或 notifyAll）后本线程才进入对象锁定池准备获得对象锁进入就绪状态。

补充：这里似乎漏掉了一个作为先决条件的问题，就是什么是进程，什么是线程？为什么需要多线程编程？答案如下所示：

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是操作系统进行资源分配和调度的一个独立单位；线程是进程的一个实体，是 CPU 调度和分派的基本单位，是比进程更小的能独立运行的基本单位。线程的划分尺度小于进程，这使得多线程程序的并发性高；进程在执行时通常拥有独立的内存单元，而线程之间可以共享内存。使用多线程的编程通常能够带来更好的性能和用户体验，但是多线程的程序对于其他程序是不友好的，因为它占用了更多的 CPU 资源。

## sleep() 和 yield() 有什么区别?

答：

① sleep() 方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield() 方法只会给相同优先级或更高优先级的线程以运行的机会；

② 线程执行 sleep() 方法后转入阻塞（blocked）状态，而执行 yield() 方法后转入就绪（ready）状态；

③ sleep() 方法声明抛出InterruptedException，而 yield() 方法没有声明任何异常；

④ sleep() 方法比 yield() 方法（跟操作系统相关）具有更好的可移植性。

## 当一个线程进入一个对象的 synchronized 方法 A 之后，其它线程是否可进入此对象的 synchronized 方法？

答：不能。其它线程只能访问该对象的非同步方法，同步方法则不能进入。

## 请说出与线程同步相关的方法。

答：

wait():使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；

sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉InterruptedException 异常；

notify():唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且与优先级无关；

notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争；

JDK 1.5 通过 Lock 接口提供了显式(explicit)的锁机制，增强了灵活性以及对线程的协调。Lock 接口中定义了加锁（lock()）和解锁(unlock())的方法，同时还提供了 newCondition() 方法来产生用于线程之间通信的 Condition 对象；

JDK 1.5 还提供了信号量(semaphore)机制，信号量可以用来限制对某个共享资源进行访问的线程的数量。在对资源进行访问之前，线程必须得到信号量的许可（调用Semaphore对象的acquire()方法）；在完成对资源的访问后，线程必须向信号量归还许可（调用 Semaphore 对象的 release() 方法）。

## synchronized 关键字的用法？

答：synchronized 关键字可以将对象或者方法标记为同步，以实现对对象和方法的互斥访问，可以用synchronized(对象) { … }定义同步代码块，或者在声明方法时将 synchronized 作为方法的修饰符。在第60题的例子中已经展示了 synchronized 关键字的用法。

## 举例说明同步和异步。

答：如果系统中存在临界资源（资源数量少于竞争资源的线程数量的资源），例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就必须进行同步存取（数据库操作中的悲观锁就是最好的例子）。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。事实上，所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作。

## 启动一个线程是用 run() 还是 start() 方法?

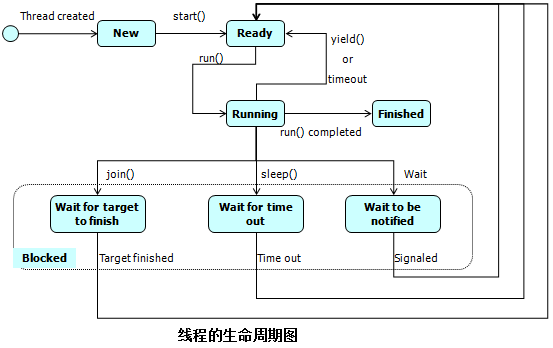
答：启动一个线程是调用 start() 方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由JVM 调度并执行，这并不意味着线程就会立即运行。run()方法是线程启动后要进行回调（callback）的方法。

## 什么是线程池（thread pool）？

答：在面向对象编程中，创建和销毁对象是很费时间的，因为创建一个对象要获取内存资源或者其它更多资源。在 Java 中更是如此，虚拟机将试图跟踪每一个对象，以便能够在对象销毁后进行垃圾回收。所以提高服务程序效率的一个手段就是尽可能减少创建和销毁对象的次数，特别是一些很耗资源的对象创建和销毁，这就是"池化资源"技术产生的原因。线程池顾名思义就是事先创建若干个可执行的线程放入一个池（容器）中，需要的时候从池中获取线程不用自行创建，使用完毕不需要销毁线程而是放回池中，从而减少创建和销毁线程对象的开销。

18. 线程的基本状态以及状态之间的关系？

答：



thread

除去起始（new）状态和结束（finished）状态，线程有三种状态，分别是：就绪（ready）、运行（running）和阻塞（blocked）。其中就绪状态代表线程具备了运行的所有条件，只等待 CPU 调度（万事俱备，只欠东风）；处于运行状态的线程可能因为 CPU 调度（时间片用完了）的原因回到就绪状态，也有可能因为调用了线程的 yield 方法回到就绪状态，此时线程不会释放它占有的资源的锁，坐等 CPU 以继续执行；运行状态的线程可能因为 I/O 中断、线程休眠、调用了对象的 wait 方法而进入阻塞状态（有的地方也称之为等待状态）；而进入阻塞状态的线程会因为休眠结束、调用了对象的 notify 方法或 notifyAll 方法或其他线程执行结束而进入就绪状态。注意：调用 wait 方法会让线程进入等待池中等待被唤醒， notify 方法或 notifyAll 方法会让等待锁中的线程从等待池进入等锁池，在没有得到对象的锁之前，线程仍然无法获得 CPU 的调度和执行。

## 死锁的必要条件？怎么克服？

答：产生死锁的四个必要条件：

互斥条件：一个资源每次只能被一个进程使用。

请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。

循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

## 死锁的解决方法:

a 撤消陷于死锁的全部进程；

b 逐个撤消陷于死锁的进程，直到死锁不存在；

c 从陷于死锁的进程中逐个强迫放弃所占用的资源，直至死锁消失。

d 从另外一些进程那里强行剥夺足够数量的资源分配给死锁进程，以解除死锁状态

# JAVA的反射

## Class.forName（String className）这个方法的作用

答：通过类的全名获得该类的类对象

16. 判断下列语句是否正确，如果有错误，请指出错误所在？

interface A{

int add(final A a);

}

class B implements A{

long add(final A a){

return this.hashCode() + a.hashCode();

}

}

答：返回值不是 long 类型

## 获得一个类的类对象有哪些方式？

答：

- 方法1：类型.class，例如：String.class

- 方法2：对象.getClass()，例如："hello".getClass()

- 方法3：Class.forName()，例如：Class.forName("java.lang.String")

## 如何通过反射创建对象？

答：

- 方法1：通过类对象调用newInstance()方法，例如：String.class.newInstance()

- 方法2：通过类对象的getConstructor()或getDeclaredConstructor()方法获得构造器（Constructor）对象并调用其newInstance()方法创建对象，例如：String.class.getConstructor(String.class).newInstance("Hello");

## 如何通过反射获取和设置对象私有字段的值？

答：可以通过类对象的getDeclaredField()方法字段（Field）对象，然后再通过字段对象的setAccessible(true)将其设置为可以访问，接下来就可以通过get/set方法来获取/设置字段的值了。下面的代码实现了一个反射的工具类，其中的两个静态方法分别用于获取和设置私有字段的值，字段可以是基本类型也可以是对象类型且支持多级对象操作，例如ReflectionUtil.get(dog, "owner.car.engine.id");可以获得dog对象的主人的汽车的引擎的ID号。

import java.lang.reflect.Constructor;

import java.lang.reflect.Field;

import java.lang.reflect.Modifier;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* 反射工具类

\* @author 骆昊

\*

\*/

public class ReflectionUtil {

private ReflectionUtil() {

throw new AssertionError();

}

/\*\*

\* 通过反射取对象指定字段(属性)的值

\* @param target 目标对象

\* @param fieldName 字段的名字

\* @throws 如果取不到对象指定字段的值则抛出异常

\* @return 字段的值

\*/

public static Object getValue(Object target, String fieldName) {

Class<?> clazz = target.getClass();

String[] fs = fieldName.split("\\.");

try {

for(int i = 0; i < fs.length - 1; i++) {

Field f = clazz.getDeclaredField(fs[i]);

f.setAccessible(true);

target = f.get(target);

clazz = target.getClass();

}

Field f = clazz.getDeclaredField(fs[fs.length - 1]);

f.setAccessible(true);

return f.get(target);

}

catch (Exception e) {

throw new RuntimeException(e);

}

}

/\*\*

\* 通过反射给对象的指定字段赋值

\* @param target 目标对象

\* @param fieldName 字段的名称

\* @param value 值

\*/

public static void setValue(Object target, String fieldName, Object value) {

Class<?> clazz = target.getClass();

String[] fs = fieldName.split("\\.");

try {

for(int i = 0; i < fs.length - 1; i++) {

Field f = clazz.getDeclaredField(fs[i]);

f.setAccessible(true);

Object val = f.get(target);

if(val == null) {

Constructor<?> c = f.getType().getDeclaredConstructor();

c.setAccessible(true);

val = c.newInstance();

f.set(target, val);

}

target = val;

clazz = target.getClass();

}

Field f = clazz.getDeclaredField(fs[fs.length - 1]);

f.setAccessible(true);

f.set(target, value);

}

catch (Exception e) {

throw new RuntimeException(e);

}

}

}

## 如何通过反射调用对象的方法？

请看下面的代码：

import java.lang.reflect.Method;

class MethodInvokeTest {

public static void main(String[] args) throws Exception {

String str = "hello";

Method m = str.getClass().getMethod("toUpperCase");

System.out.println(m.invoke(str)); // HELLO

}

}