# JAVA基础

## 1.JAVA语法概念

### （★）Java中的方法覆盖（Overriding）和方法重载（Overloading）是什么意思？ 重写和重载的区别?

答: 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。

**重载（Overloading）**

（1） 方法重载是让类以统一的方式处理不同类型数据的一种手段。多个同名函数同时存在，具有不同的参数个数/类型。重载Overloading是一个类中多态性的一种表现。

（2） Java的方法重载，就是在类中可以创建多个方法，它们具有相同的名字，但具有不同的参数和不同的定义。调用方法时通过传递给它们的不同参数个数和参数类型来决定具体使用哪个方法, 这就是多态性。

（3） 重载的时候，方法名要一样，但是参数类型和个数不一样，返回值类型可以相同也可以不相同。无法以返回型别作为重载函数的区分标准。 **重写（Overriding）**

（1） 父类与子类之间的多态性，对父类的函数进行重新定义。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 （Overriding）。在Java中，子类可继承父类中的方法，而不需要重新编写相同的方法。但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。方法重写又称方法覆盖。

（2）若子类中的方法与父类中的某一方法具有相同的方法名、返回类型和参数表，则新方法将覆盖原有的方法。如需父类中原有的方法，可使用super关键字，该关键字引用了当前类的父类。

（3）子类函数的访问修饰权限不能少于父类的；

### （★）Java中，什么是构造函数？什么是构造函数重载？什么是复制构造函数？

答: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。

Java不支持像C++中那样的复制构造函数，这个不同点是因为如果你不自己写构造函数的情况下，Java不会创建默认的复制构造函数。

### （★）java支持多继承么？

答: Java中类不支持多继承，只支持单继承（即一个类只有一个父类）。 但是java中的接口支持多继承，，即一个子接口可以有多个父接口。（接口的作用是用来扩展对象的功能，一个子接口继承多个父接口，说明子接口扩展了多个功能，当类实现接口时，类就扩展了相应的功能）。

### （★）接口和抽象类的区别是什么？

答:接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为public abstract类型，接口中的成员变量类型默认为public static final （这里需要说一点,既然一个变量被final修饰了,那么这个变量就是一个常量!!!!!变量必须初始化成常量!!!!!）

**下面比较一下两者的语法区别：**

❶抽象类可以有构造方法，接口中不能有构造方法。

❷抽象类中可以有普通成员变量，接口中没有普通成员变量!!!!!!!（注意重点在 普通 即 非静态  和  变量!!!!）

❸抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法。

❹抽象类中的抽象方法的访问类型可以是public，protected和（默认类型,虽然eclipse下不报错，但应该也不行），但接口中的抽象方法只能是public类型的，并且默认即为public abstract类型。

❺抽象类中可以包含静态方法（static），接口中不能包含静态方法.

❻抽象类和接口中都可以包含静态成员变量（static），抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是public static final类型，并且默认即为public static final类型。

❼一个类可以实现多个接口，但只能继承一个抽象类。

### （★）何时用接口,何时用抽象类?

子类继承抽象类就不能再继承其他类（java的单继承原则）,所以有抽象方法时优先使用接口定义.

如果所有方法都没有实现,优先使用接口.

一些方法已经可以实现，而另一些方法无法或不想实现时,且继承（实现）关系很简单时,考虑用抽象类定义，留给后代去实现。jdk8之后,也可以在接口中定义default方法或static方法来实现.

### （★）什么是值传递和引用传递？

答:值传递是对基本型变量而言的,传递的是该变量的一个副本,改变副本不影响原变量.

引用传递一般是对于对象型变量而言的,传递的是该对象地址的一个副本, 并不是原对象本身。所以对引用对象进行操作会同时改变原对象。一般认为,java内的传递都是值传递.

### （★）JDK和JRE的区别是什么？

答: Java运行时环境（JRE）是将要执行Java程序的Java虚拟机。它同时也包含了执行applet需要的浏览器插件。Java开发工具包（JDK）是完整的Java软件开发包，包含了JRE，编译器和其他的工具（比如：JavaDoc，Java调试器），可以让开发者开发、编译、执行Java应用程序。

### （★）"=="操作符与equals的区别?

答:对于值类型：“==”比较数值是否相等，值类型没有equals方法

对于引用类型：

如果该类型没有重载equals方法，则“==”和“equals”方法返回的结果一致，即比较两个变量引用的地址是否相等

如果重载了equals方法，则equals方法的行为需要看重载方法的具体定义，“==”依然比较的是两个变量的引用地址

如果则“==”返回true则equals应该也会返回true（正确实现了equals），反之则不一定

### （★）java里final finally finalize区别?

答: final 用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。

finally 是异常处理语句结构的一部分，表示总是执行。

finalize 是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等

final—修饰符（关键字）如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承。因此一个类不能既被声明为 abstract的，又被声明为final的。将变量或方法声明为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，不能重载。  
 finally—在异常处理时提供 finally 块来执行任何清除操作。如果抛出一个异常，那么相匹配的 catch 子句就会执行，然后控制就会进入 finally 块（如果有的话）。  
 finalize—方法名。Java 技术允许使用 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确定这个对象没有被引用时对这个对象调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize（） 方法以整理系统资源或者执行其他清理工作。finalize（） 方法是在垃圾收集器删除对象之前对这个对象调用的。

### （★）什么是多态 多态的使用范围

答:多态的定义：指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）多态存在的三个必要条件

一、要有继承；

二、要有重写；

三、父类引用指向子类对象。Java中多态的实现方式：接口实现，继承父类进行方法重写，同一个类中进行方法重载。

### （★）java中 public/private/protected和default（默认）的区别

答:

public：

具有最大的访问权限，可以访问任何一个在classpath下的类、接口、异常等。它往往用于对外的情况，也就是对象或类对外的一种接口的形式。

protected：

主要的作用就是用来保护子类的。它的含义在于子类可以用它修饰的成员，其他的不可以，它相当于传递给子类的一种继承的东西

default：

有时候也称为friendly，它是针对本包访问而设计的，任何处于本包下的类、接口、异常等，都可以相互访问，即使是父类没有用protected修饰的成员也可以。

private：

访问权限仅限于类的内部，是一种封装的体现，例如，大多数成员变量都是修饰符为private的，它们不希望被其他任何外部的类访问。

### （★）String、StringBuffer、StringBuilder有什么区别？

答:

1. String 字符串常量

2. StringBuffer 字符串变量（线程安全）

3.StringBuilder 字符串变量（非线程安全）

1.String类型是Java内的对象，是个不可变的对象，当每次对String进行改变时都需要生成一个新的String对象，然后将指针指向一个新的对象，如果在一个循环里面，不断的改变一个对象，就要不断的生成新的对象，而且对象多了，Java的垃圾自动回收机制会开始工作，所以效率很低，建议在不断更改String对象的地方不要使用String类型

2. StringBuffer 是个可变的对象，就是每次操作都是对对象本身进行操作，而不用生成新的对象，这样效率肯定就会有有很大的提高，在大部分情况下StringBuffer的效率要比String类型要高

3. StringBuilder 与StringBuffer一样是个可变的字符序列，提供与StringBuffer兼容的API，但是不能保证同步，用在字符串缓冲区被当个线程使用的情况，在单机非多线程的情况下使用StringBuilder会有比较好的效率，因为StringBuilder没有处理同步（Synchronized）问题。StringBuffer则会处理同步问题，如果StringBuilder会在多线程下被操作，则要改用StringBuffer，让对象自行管理同步问题。

注意点：

在使用StringBuffer时指定其容量，会比不指定其容量要快40%到50%左右，甚至比不指定容量的StringBuilder要快。所以在使用时最好指定其容量。这是一个良好的编程习惯，对于性能有很大的提升。

### （★）说出一些常用的类，包，接口，请各举5个

答:

常用的类：

BufferedReader BufferedWriter FileReader FileWirter String Integer

常用的包：

java.lang java.awt java.io java.util java.sql

常用的接口：

Remote List Map Document NodeList

### （★）解释一下static关键字 被static修饰的方法 调用 静态/非静态函数的方法

答: 1. 静态方法

通常，在一个类中定义一个方法为static，那就是说，无需本类的对象即可调用此方法

声明为static的方法有以下几条限制：

它们仅能调用其他的static 方法。

它们只能访问static数据。

它们不能以任何方式引用this 或super。

2. 静态变量

声明为static的变量实质上就是全局变量。当声明一个对象时，并不产生static变量的拷贝，而是该类所有的实例变量共用同一个static变量。静态变量与静态方法类似。所有此类实例共享此静态变量，也就是说在类装载时，只分配一块存储空间，所有此类的对象都可以操控此块存储空间，当然对于final则另当别论了

3. static代码块

static关键字还有一个比较关键的作用就是 用来形成静态代码块以优化程序性能。static块可以置于类中的任何地方，类中可以有多个static块。在类初次被加载的时候，会按照static块的顺序来执行每个static块，并且只会执行一次。

### （★）堆区和栈区的区别

答:

栈区：先进后出 存放局部变量 连续的存储空间

堆区：存放new出来的对象，不连续的存储空间

### （★）&和&&的区别

答:&是位运算符，表示按位与运算，&&是逻辑运算符，表示逻辑与（and）

### （★）Java支持的数据类型有哪些？什么是自动拆装箱？

答:Java语言支持的8种基本数据类型是：

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。

### （★）什么是多态

答:面向对象的三大特性：封装、继承、多态。从一定角度来看，封装和继承几乎都是为多态而准备的。这是我们最后一个概念，也是最重要的知识点。

多态的定义：指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）

实现多态的技术称为：动态绑定（dynamic binding），是指在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。

多态的作用：消除类型之间的耦合关系。

现实中，关于多态的例子不胜枚举。比方说按下 F1 键这个动作，如果当前在 Flash 界面下弹出的就是 AS 3 的帮助文档；如果当前在 Word 下弹出的就是 Word 帮助；在 Windows 下弹出的就是 Windows 帮助和支持。同一个事件发生在不同的对象上会产生不同的结果。

下面是多态存在的三个必要条件，要求大家做梦时都能背出来！

多态存在的三个必要条件  
 一、要有继承；  
 二、要有重写；  
 三、父类引用指向子类对象。

### （★）什么是继承，继承的特点？

答：子类继承父类的特征和行为，使得子类具有父类的各种属性和方法。或子类从父类继承方法，使得子类具有父类相同的行为。

特点：在继承关系中，父类更通用、子类更具体。父类具有更一般的特征和行为，而子类除了具有父类的特征和行为，还具有一些自己特殊的特征和行为。

在继承关系中。父类和子类需要满足is-a的关系。子类是父类。

表示父类和子类的术语：父类和子类、超类和子类、基类和派生类，他们表示的是同一个意思。

### （★）为什么需要继承？什么时候应该继承？

答：使用继承可以有效实现代码复用，避免重复代码的出现。

当两个类具有相同的特征（属性）和行为（方法）时，可以将相同的部分抽取出来放到一个类中作为父类，其它两个类继承这个父类。

继承实现了面向对象的原则：write once,only once（编写一次、且编写一次）

### （★）如何实现继承？

答：在[Java](http://lib.csdn.net/base/javase)语言中，用extends（扩展）关键字来表示一个类继承了另一个类。

在父类中只定义一些通用的属性和方法。

子类自动继承父类的属性和方法，子类中可以定义特定的属性和方法。或子类重新定义父类的属性、重写父类的方法可以获得与父类不同的功能。

### （★）封装有什么好处？

答: 一.是用private把类的细节与外界隔离起来，从而实现数据项和方法的隐藏，而要访问这些数据项和方法唯一的途径就是通过类本身，类才有资格调用它所拥有的资源（方法，数据项属性等等）。所以第一个好处就是数据的安全性提高了。

二.是通过隐藏隔离，只允许外部对类做有限的访问，开发者可以自由的改变类的内部实现，而无需修改使用该类的那些程序。只要那些在类外部就能被调用的方法保持其外部特征不变，内部代码就可以自由改变，各取所需，利于分工。

三.就是提高了代码的重用性，封装成工具类以后能够减少很多繁琐的步骤。

（★）**23.作用域public,private,protected,以及不写时的区别。**

**答：**

（1）、public：public表明该数据成员、成员函数是对所有用户开放的，所有用户都可以直接进行调用

（2）、private：private表示私有，私有的意思就是除了class自己之外，任何人都不可以直接使用，私有财产神圣不可侵犯嘛，即便是子女，朋友，都不可以使用。

（3）、protected：protected对于子女、朋友来说，就是public的，可以自由使用，没有任何限制，而对于其他的外部class，protected就变成private。

作用域  当前类 同一package 子孙类 其他package

public   √  √  √  √

protected √  √  √  ×

friendly  √  √  ×  ×

private  √  ×  ×  ×

不写时默认为friendly

### （★）面向对象的特征有哪些方面? 【基础】

答：面向对象的特征主要有以下几个方面：

1）抽象：抽象就是忽略一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。抽象并不打算了解全部问题，而只是选择其中的一部分，暂时不用部分细节。抽象包括两个方面，一是过程抽象,二是数据抽象。

2）继承：继承是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量，并且类可以修改或增加新的方法使之更适合特殊的需要。

3）封装：封装是把过程和数据包围起来，对数据的访问只能通过已定义的界面。面向对象计算始于这个基本概念，即现实世界可以被描绘成一系列完全自治、封装的对象,这些对象通过一个受保护的接口访问其他对象。

4）多态性：多态性是指允许不同类的对象对同一消息作出响应。多态性包括参数化多态性和包含多态性。多态性语言具有灵活、抽象、行为共享、代码共享的优势，很好的解决了应用程序函数同名问题。

### （★）String 是最基本的数据类型吗? 【基础】

答：不是。

### （★）float 型float f=3.4是否正确? 【基础】

答:不正确；精度不准确,应该用强制类型转换，如下所示：float f=（float）3.4 。

### （★）语句float f=1.3；编译能否通过？【基础】

答:不能；应该用强制类型转换，如下所示：float f=（float）1.3； 。

### （★）short s1 = 1; s1 = s1 + 1;有什么错?

**short s1 = 1; s1 += 1;有什么错? 【基础】**

答：

short s1 = 1;

s1 = s1 + 1;

s1+1运算结果是int 型，需要强制转换类型；short s1 = 1; s1 += 1;可以正确编译,自动类型提升。

### （★）Java 有没有goto? 【基础】

答：

goto 是java 中的保留字，现在没有在java 中使用。

### （★）int 和Integer 有什么区别? 【基础】

答：

Java 提供两种不同的类型：

引用类型和原始类型（或内置类型）；

int 是java 的原始数据类型，Integer 是java 为int 提供的封装类。

Java 为每个原始类型提供了封装类：

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

封装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double引用类型和原始类型的行为完全不同，并且它们具有不同的语义。引用类型和原始类型具有不同的特征和用法，它们包括：大小和速度问题，这种类型以哪种类型的数据结构存储，当引用类型和原始类型用作某个类的实例数据时所指定的缺省值。对象引用实例变量的缺省值为null，而原始类型实例变量的缺省值与它们的类型有关。

### （★）&和&&的区别？

答：&是位运算符，表示按位与运算，&&是逻辑运算符，表示逻辑与（and）。

### （★）简述逻辑操作（&,|,^）与条件操作（&&,||）的区别？

答：区别主要有两点：a.条件操作只能操作布尔型的,而逻辑操作不仅可以操作布尔型,而且可以操作数值型b.逻辑操作不会产生短路。

### （★）heap 和stack 有什么区别？

答：栈是一种线形集合，其添加和删除元素的操作应在同一段完成，栈按照后进先出的方式进行处理；堆是栈的一个组成元素。

### （★）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，参数加1/2 后求其floor。

### （★）swtich 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：switch（expr1）中，expr1 是一个整数表达式。因此传递给switch 和case语句的参数应该是int、short、char 或者byte。long,string 都不能作用于swtich。

### （★）编程题: 用最有效率的方法算出2 乘以8 等於几?

答： 2 << 3。

### （★）有没有length（）这个方法? String 有没有length（）这个方法？

答：数组没有length（）这个方法，有length 的属性。String 有length（）这个方法。

### （★）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label 标识,然后用break:label 方法即可跳出多重循环。

### （★）构造器Constructor 是否可被override?

答：构造器Constructor 不能被继承，因此不能重写Overriding，但可以被重载Overloading。

### （★）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，有相同的hash code。

### （★）是否可以继承String 类?

答：String 类是final 类，故不可以继承。

### （★）以下二条语句返回值为true 的有：

A：“beijing”==“beijing”；

B：“beijing”.equalsIgnoreCase（new String（“beijing”））；

答：A 和B 。

### （★）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 编程语言只有值传递参数。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。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的。

### （★★）我们在web 应用开发过程中经常遇到输出某种编码的字符，如iso8859-1等，如何输出一个某种编码的字符串？

答：

public String translate（String str）{

String tempStr = "";

try{

tempStr = new String（str.getBytes（"ISO-8859-1"）, "GBK"）;

tempStr = tempStr.trim（）;

}catch （Exception e）{

System.err.println（e.getMessage（））;

}

return tempStr;

}

### （★）定义类A 和类B 如下：【基础】

class A {

int a=1;

double d=2.0;

void show（）{

System.out.println（"Class A: a="+a +"\td="+d）;

}

}

class B extends A{

float a=3.0f;

String d="Java program.";

void show（）{

super.show（ ）;

System.out.println（"Class B: a="+a +"\td="+d）;

}

}

（1） 若在应用程序的main 方法中有以下语句：

A a=new A（）;

a.show（）;

则输出的结果如何？

（2） 若在应用程序的main 方法中定义类B 的对象b：

A b=new B（）;

b.show（）;

则输出的结果如何？

答：输出结果为：

1）Class A: a=1 d=2.0 ；

2）Class A: a=1 d=2.0

Class B: a=3.0 d=Java program。

### （★★）描述一下JVM 加载class 文件的原理机制? 【基础】

答：JVM 中类的装载是由ClassLoader 和它的子类来实现的,Java ClassLoader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的类。

### （★）char 型变量中能不能存贮一个中文汉字?为什么?

答：能够定义成为一个中文的，因为java 中以unicode 编码，一个char 占16个字节，所以放一个中文是没问题的。

### （★★）abstract class 和interface 有什么区别? 【基础】

答：声明方法的存在而不去实现它的类被叫做抽象类（abstract class），。然而可以创建一个变量，其类型是一个抽象类，它用于要创建一个体现某些基本行为的类，并为该类声明方法，但不能在该类中实现该类的情况。不能创建abstract 类的实例并让它指向具体子类的一个实例。不能有抽象构造函数或抽象静态方法。Abstract 类的子类为它们父类中的所有抽象方法提供实现，否则它们也是抽象类为。取而代之，在子类中实现该方法。知道其行为的其它类可以在类中实现这些方法。接口（interface）是抽象类的变体。新型多继承性可通过实现这样的接口而获得。接口中的所有方法都是抽象的，所有成员变量都是publicstatic final 的。一个类可以实现多个接口，当类实现特殊接口时，它定义（即

将程序体给予）所有这种接口的方法。然后，它可以在实现了该接口的类的任何对象上调用接口的方法。由于有抽象类，它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接口类型转换，instanceof 运算符可以用来决定某对象的类是否实现了接口。

### （★★）Static Nested Class 和Inner Class 的不同？【基础】

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。

### （★★）java 中会存在内存泄漏吗，请简单描述。【基础】

答：会；存在无用但可达的对象，这些对象不能被GC 回收，导致耗费内存资源。

### （★★）abstract 的method 是否可同时是static,是否可同时是native，是否可同时是synchronized? 【基础】

答：都不能。

### （★★）静态变量和实例变量的区别？【基础】

答：静态变量也称为类变量，归全类共有，它不依赖于某个对象，可通过类名直接访问；而实例变量必须依存于某一实例，只能通过对象才能访问到它。

### （★★）是否可以从一个static 方法内部发出对非static 方法的调用？【基础】

答：不可以,如果其中包含对象的method（），不能保证对象初始化。

### （★★）写clone（）方法时，通常都有一行代码，是什么？【基础】

答：Clone 有缺省行为：super.clone（），他负责产生正确大小的空间，并逐位复制。

### （★★）GC 是什么? 为什么要有GC? 【基础】

答：GC 是垃圾收集的意思（Gabage Collection）,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（） 。

### （★★）垃圾回收的优点和原理。并考虑2 种回收机制。【基础】

答：Java 语言中一个显著的特点就是引入了垃圾回收机制，使c++程序员最头疼的内存管理的问题迎刃而解，它使得Java 程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制，Java 中的对象不再有“作用域”的概念，只有对象的引用才有“作用域”。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级别的线程运行，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楚和回收，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。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

### （★★）垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？【基础】

答：对于GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆（heap）中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当GC 确定一些对象为"不可达"时，GC 就有责任回收这些内存空间。可以。程序员可以手动执行System.gc（），通知GC 运行，但是Java 语言规范并不保证GC 一定会执行。

### （★★）String s=new String（“xyz”）;创建了几个String Object? 【基础】

答：两个对象，一个是"xyx",一个是指向"xyx"的引用对象s。

### （★）接口是否可继承接口? 抽象类是否可实现（implements）接口? 抽象类是否可继承实体类（concrete class）? 【基础】

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

### （★★）Java 的接口和C++的虚类的相同和不同处。【基础】

答：由于Java 不支持多继承，而有可能某个类或对象要使用分别在几个类或对象里面的方法或属性，现有的单继承机制就不能满足要求。与继承相比，接口有更高的灵活性，因为接口中没有任何实现代码。当一个类实现了接口以后，该类要实现接口里面所有的方法和属性，并且接口里面的属性在默认状态下面都是public static,所有方法默认情况下是public.一个类可以实现多个接口。

### （★）一个“.java”源文件中是否可以包含多个类（不是内部类）？有什么限制？【基础】

答：可以；必须只有一个类名与文件名相同。

### （★）说出一些常用的类，包，接口，请各举5 个。【基础】

答：常用的类：BufferedReader BufferedWriter FileReader FileWirter String Integer；

常用的包：java.lang java.awt java.io java.util java.sql；

常用的接口：Remote List Map Document NodeList

### （★★）Anonymous Inner Class （匿名内部类） 是否可以extends（继承）其它类？是否可以implements（实现）interface（接口）? 【基础】

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。

### （★）内部类可以引用他包含类的成员吗？有没有什么限制？【基础】

答：一个内部类对象可以访问创建它的外部类对象的内容。

### （★）java 中实现多态的机制是什么？【基础】

答：方法的覆盖Overriding 和重载Overloading 是java 多态性的不同表现；覆盖Overriding 是父类与子类之间多态性的一种表现，重载Overloading 是一个类中多态性的一种表现。

### （★）在java 中一个类被声明为final 类型，表示了什么意思？【基础】

答：表示该类不能被继承，是顶级类。

### （★★）下面哪些类可以被继承? 【基础】

1）java.lang.Thread （T）

2）java.lang.Number （T）

3）java.lang.Double （F）

4）java.lang.Math （F）

5）java.lang.Void （F）

6）java.lang.Class （F）

7）java.lang.ClassLoader （T）

答：1、2、7 可以被继承。

### （★★）指出下面程序的运行结果: 【基础】

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[] ars）{

A ab = new B（）; //执行到此处,结果: 1a2b

ab = new B（）; //执行到此处,结果: 1a2b2b

}

}

答：输出结果为1a2b2b；类的static 代码段,可以看作是类首次加载（虚拟机加载）执行的代码,而对于类加载,首先要执行其基类的构造,再执行其本身的构造。

### （★★）继承时候类的执行顺序问题,一般都是选择题,问你将会打印出什么?【基础】

父类：

package test;

public class FatherClass {

public FatherClass（） {

System.out.println（"FatherClass Create"）;

}

}

子类:

package test;

import test.FatherClass;

public class ChildClass extends FatherClass {

public ChildClass（） {

System.out.println（"ChildClass Create"）;

}

public static void main（String[] args） {

FatherClass fc = new FatherClass（）;

ChildClass cc = new ChildClass（）;

}

}

答：输出结果为：

FatherClass Create

FatherClass Create

ChildClass Create

### （★★）内部类的实现方式? 【基础】

答：示例代码如下：

package test;

public class OuterClass {

private class InterClass {

public InterClass（） {

System.out.println（"InterClass Create"）;

}

}

public OuterClass（） {

InterClass ic = new InterClass（）;

System.out.println（"OuterClass Create"）;

}

public static void main（String[] args） {

OuterClass oc = new OuterClass（）;

}

}

输出结果为:

InterClass Create

OuterClass Create

### （★★）关于内部类： 【基础】

public class OuterClass {

private double d1 = 1.0;

//insert code here

}

You need to insert an inner class declaration at line 3，Which two

inner class declarations are valid?（Choose two.）

A. class InnerOne{

public static double methoda（） {return d1;}

}

B. public class InnerOne{

static double methoda（） {return d1;}

}

C. private class InnerOne{

double methoda（） {return d1;}

}

D. static class InnerOne{

protected double methoda（） {return d1;}

}

E. abstract class InnerOne{

public abstract double methoda（）;

}

答：答案为C、E；说明如下：

1）静态内部类可以有静态成员，而非静态内部类则不能有静态成员；故A、B错；

2）静态内部类的非静态成员可以访问外部类的静态变量，而不可访问外部类的非静态变量；故D 错；

3）非静态内部类的非静态成员可以访问外部类的非静态变量；故C 正确。

### （★★）数据类型之间的转换:

1）如何将数值型字符转换为数字？

2）如何将数字转换为字符？

3）如何取小数点前两位并四舍五入? 【基础】

答：1）调用数值类型相应包装类中的方法parse\*\*\*（String）或valueOf（String）即可返回相应基本类型或包装类型数值；

2）将数字与空字符串相加即可获得其所对应的字符串;另外对于基本类型数字还可调用String 类中的valueOf（…）方法返回相应字符串,而对于包装类型数字则可调用其toString（）方法获得相应字符串；

3）可用该数字构造一java.math.BigDecimal 对象,再利用其round（）方法进行四舍五入到保留小数点后两位,再将其转换为字符串截取最后两位。

### （★★）字符串操作：如何实现字符串的反转及替换？【基础】

答：可用字符串构造一StringBuffer 对象,然后调用StringBuffer 中的reverse方法即可实现字符串的反转,调用replace 方法即可实现字符串的替换。

### （★★）编码转换：怎样将GB2312 编码的字符串转换为ISO-8859-1 编码的字符串？【基础】

答：示例代码如下:

String s1 = "你好";

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

### （★★）写一个函数，要求输入一个字符串和一个字符长度，对该字符串进行分隔。【基础】

答：函数代码如下：

public String[] split（String str, int chars）{

int n = （str.length（）+ chars - 1）/chars;

String ret[] = new String[n];

for（int i=0; i<n; i++）{

if（i < n-1）{

ret[i] = str.substring（i\*chars , （i+1）\*chars）;

}else{

ret[i] = str.substring（i\*chars）;

}

}

return ret;

}

### （★★）写一个函数，2 个参数，1 个字符串，1 个字节数，返回截取的字符串，要求字符串中的中文不能出现乱码：如（“我ABC”，4）应该截为“我AB”，输入（“我ABC 汉DEF”，6）应该输出为“我ABC”而不是“我ABC+汉的半个”。【基础】

答：代码如下：

public String subString（String str, int subBytes） {

int bytes = 0; // 用来存储字符串的总字节数

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

if （bytes == subBytes） {

return str.substring（0, i）;

}

char c = str.charAt（i）;

if （c < 256） {

bytes += 1; // 英文字符的字节数看作1

} else {

bytes += 2; // 中文字符的字节数看作2

if（bytes - subBytes == 1）{

return str.substring（0, i）;

}

}

}

return str;

}

### （★）日期和时间：

1）如何取得年月日、小时分秒？

2）如何取得从1970 年到现在的毫秒数？

3）如何取得某个日期是当月的最后一天？

4）如何格式化日期？【基础】

答：1）创建java.util.Calendar 实例（Calendar.getInstance（））,调用其get（）方法传入不同的参数即可获得参数所对应的值,如：calendar.get（Calendar.YEAR）;//获得年

2）以下方法均可获得该毫秒数:Calendar.getInstance（）.getTimeInMillis（）;System.currentTimeMillis（）;

3）示例代码如下:

Calendar time = Calendar.getInstance（）;

time.set（Calendar.DAY\_OF\_MONTH,

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

4）利用java.text.DataFormat 类中的format（）方法可将日期格式化。

### （★）Java 编程,打印昨天的当前时刻。【基础】

答：public class YesterdayCurrent{

public static void main（String[] args）{

Calendar cal = Calendar.getInstance（）;

cal.add（Calendar.DATE, -1）;

System.out.println（cal.getTime（））;

}

}

### （★）java 和javasciprt 的区别。【基础】

答：JavaScript 与Java 是两个公司开发的不同的两个产品。Java 是SUN 公司推出的新一代面向对象的程序设计语言，特别适合于Internet 应用程序开发；而JavaScript 是Netscape 公司的产品，其目的是为了扩展Netscape Navigator功能,而开发的一种可以嵌入Web 页面中的基于对象和事件驱动的解释性语言,它的前身是Live Script；而Java 的前身是Oak 语言。下面对两种语言间的异同作如下比较：

1）基于对象和面向对象：Java 是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象；JavaScript 是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object Based）和事件驱动（Event Driver）的编程语言。因而它本身提供了非常丰富的内部对象供设计人员使用；

2）解释和编译：Java 的源代码在执行之前，必须经过编译；JavaScript 是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行；

3）强类型变量和类型弱变量：Java 采用强类型变量检查，即所有变量在编译之前必须作声明；JavaScript 中变量声明，采用其弱类型。即变量在使用前不需作声明，而是解释器在运行时检查其数据类型；

4）代码格式不一样。

### （★★★）什么时候用assert？

答：assertion（断言）在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后， assertion 检查通常是关闭的。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为true；如果表达式计算为false，那么系统会报告一个Assertionerror。

断言用于调试目的：

assert（a > 0）; // throws an Assertionerror if a <= 0

断言可以有两种形式：

assert Expression1 ;

assert Expression1 : Expression2 ;

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的String 消息。

断言在默认情况下是禁用的，要在编译时启用断言，需使用source 1.4 标记：

javac -source 1.4 Test.java

要在运行时启用断言，可使用-enableassertions 或者-ea 标记。

要在运行时选择禁用断言，可使用-da 或者-disableassertions 标记。

要在系统类中启用断言，可使用-esa 或者-dsa 标记。还可以在包的基础上启用或者禁用断言。可以在预计正常情况下不会到达的任何位置上放置断言。断言可以用于验证传递给私有方法的参数。不过，断言不应该用于验证传递给公有方法的参数，因为不管是否启用了断言，公有方法都必须检查其参数。不过，既可以在公有方法中，也可以在非公有方法中利用断言测试后置条件。另外，断言不应该以任何方式改变程序的状态。

### （★）正则表达式中\*？+的作用分别是什么

答:

\* ？ + 都有用来匹配数量的，\* 表示 0 或多个，？表示 0 个或 1 个，+ 表示 1 个或多个。

### （★）简单说明什么是递归？什么情况会使用？并使用java实现一个简单的递归程序。

答:

一、递归函数，通俗的说就是函数本身自己调用自己...

如:n!=n（n-1）!

你定义函数f（n）=nf（n-1）

而f（n-1）又是这个定义的函数。。这就是递归

二、为什么要用递归：递归的目的是简化程序设计，使程序易读

三、递归的弊端：虽然非递归函数效率高，但较难编程，可读性较差。递归函数的缺点是增加了系统开销，也就是说，每递归一次，栈内存就多占用一截

四、递归的条件：需有完成任务的语句，需满足递归的要求（减小而不是发散）

### （★）线性表如果要频繁的执行插入和删除操作，该线性表采取的存储结构应该是（）

A.散列 B.顺序 C.链式 D.索引

答:

C

### （★）什么是构造函数,构造代码块,静态代码块?分别的作用是什么?三者的顺序

答:

构造函数是类用来创建对象使用的一种函数可以有有参构造函数也可以有无参构造函数

构造代码块的作用和构造函数类似可以完成类中的成员变量进行初始化也可以调用成员方法

静态代码块属于类的,静态代码块可以随着类的加载而加载可以初始化中的静态成员变量,也

可以在连接JBCD时用于读取文件中的连接信息

先执行静态代码块 在执行构造代码块 最后执行构造方法

### （★）double和Double的区别?

答:

double是基本数据类型,值会存储在栈中并且初始值是0.0,不能存在集合中

Doubl是包装类即double的引用类型,值会存储在堆中初始值是null,可以替代double存储集合中

### （★）判断字符串为空时如何避免空指针异常?

答:

String str=null;

if（str == null || str.isEmpty（））来进行判断

### （★）什么是重载和重写?

答:

重载: 是面向对象中多态特性的一种体现,即方法名相同形参列表不同（参数,个数,顺序）满足其中一个条件即可

重写: 是面向对象中继承特性的一种体现,即父类中所提供的方法无法满足子类需求时,子类可以实现和父类一样的方法即为重写,但是fianl,private,static修饰的方法无法重写

### （★）final修饰的方法可以被继承/重载/重写吗?

答:

final修饰的方法可以继承和重载,但是不能重写

### （★）什么是反射?如何通过反射访问私有字段?

答:

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

要想解剖一个类,必须先要获取到该类的字节码文件对象。而解剖使用的就是Class类中的方法.所以先要获取到每一个字节码文件对应的Class类型的对象.

可以用getDeclaredField方法并且需要开启权限setAccessible（true）

### （★）不使用循环条和件语句实现1+2+3+...n

答:

public static int sum（int n） {

int sum = n;

boolean isContinue = （n > 0） && （sum += sum（--n）） > 0;

return sum;

}

public static void main（String[] args） {

int result = sum（5）;

System.out.println（result）;

}

### （★） hashcode，hashMap，list，hashSet，equals（结构原理），A extends B（类的加载顺序）

1.父类静态代码块；

2.子类静态代码块；

3.父类非静态代码块；

4.父类构造函数；

5.子类非静态代码块；

6.子类构造函数；

### （★） String 是最基本的数据类型吗？

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）。

### （★） 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;。

### （★） 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。

### （★） 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）

### （★★） 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了基本数据类型，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Java 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

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

}

}

最近还遇到一个面试题，也是和自动装箱和拆箱有点关系的，代码如下所示：

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。

提醒：越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的功力。

### （★） &和&&的区别？

答：&运算符有两种用法：（1）按位与；（2）逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true整个表达式的值才是true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是null而且不是空字符串，应当写为：username != null &&!username.equals（“”），二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的equals比较，否则会产生NullPointerException异常。注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

### （★★） 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。

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

答：Math.round（11.5）的返回值是12，Math.round（-11.5）的返回值是-11。四舍五入的原理是在参数上加0.5然后进行下取整。

### （★） 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。

### （★） 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）

### （★） 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（16比特），所以放一个中文是没问题的。

### （★） 一个”.java”源文件中是否可以包含多个类（不是内部类）？有什么限制？

答：可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

### （★） 什么样的标识符是合法的？

a） 由字母、数字、\_和$组成，长度不限。其中字母可以是大写或小写的英文字母，数

字为0到9。

b） 标识符的第一个字符不能是数字。

c） 标识符区分大小写。

d） 标识符不能包含空格。

### （★） Java有几种基本数据类型？

a） byte，short，int，long，char，boolean，float，double

### （★） 什么是隐式类型转换？什么是显示类型转换？

a） 当将占位数少的类型赋值给占位数多的类型时，Java自动使用隐式类型转换。 b） 当把在级别高的变量的值赋给级别底变量时，必须使用显示类型转换运算。

### （★） &&和&区别，||和|区别？

a） &&和||是短路与，短路或，当左边的表达式能判断当前结果，则不判断右边的表

达式。

b） 而& 和|则将两边的表达式都运算完毕后，再算结果。

### （★） break，continue区别？

a） break结束最近的一个循环，continue结束当次循环，进入下次循环。

### （★） 类的命名规则是什么？

a） 如果类名使用拉丁字母，那么名字的首写字母使用大写字母。

b） 类名最好见名得意，当类名由几个单词复合而成时，每个单词的首写字母使用大写。

### （★） java关键字区分大小写吗？

a） java关键字一律小写。所以无所谓区分大小写，大写的不是关键字。

### （★） java采用什么字符集？该字符集有多少字符

a） Java 使用 unicode 字符集,所以常量共有 65535 个

### （★） 列举算术运算符？

a） +，-，\*，/，%

### （★） 算术混合运算结果精度如何确定？

a） Java按照运算符两边的操作元的最高精度保留结果的精度。

### （★） &是位运算符，与运算的规则是什么？

a） 全1则1，否则为0

### （★） |是位运算符，或运算的规则是什么？

a） 全0则0，否则是1

### （★） ^是位运算符，异或运算的规则是什么？

a） 相同0，不同1

### （★） ~是位运算符，非运算的规则是什么？

a） 遇1则0，遇0则1

### （★） if语句后边有个括号，该括号里表达式为什么类型？

a） boolean类型

### （★） switch语句后括号里的表达式是什么类型？case后类型如何确定？

a） char，int，byte，short，枚举类型。

b） case后面是一个常量，该常量类型由switch后括号内表达式来确定。

### （★） switch语句后，default关键字起什么作用？break起什么作用？

a） default：当用户指定的表达式与任何case都不匹配的时候，执行default后的语句。 b） break：退出当前case。

### （★） for循环后括号里三个表达式分别起什么作用？

a） 1：循环初始化的时候执行，只执行一次。

b） 2：循环成立的条件。

c） 3：循环每次执行都会调用该表达式，一般做变量自增。

### （★） while 和do...while区别？

a） while先判断条件，再执行。

b） do……while先执行，再判断条件

c） do……while至少执行一次。

### （★） 什么是编译执行的语言？什么是解释执行的语言？

a） 编译方式：Compilation:针对当前的机器处理器芯片,将源程序全部翻译成机器指令,

称做目标程序,再将目标程序交给计算机执行.

b） 解释方式：Interpretation:这种方式不产生整个的目标程序,而是根据当前的机器处

理器芯片,边翻译边执行,翻译一句执行一句.

### （★） 简述一个java程序执行的过程?

a） 首先编写java源文件（扩展名为.java的文本文档）。

b） 用javac命令把源文件编译成字节码文件（.class文件）

c） 用java命令执行字节码文件。

### （★） 成员变量的作用范围？局部变量的作用范围？

a） 成员变量在整个类内都有效

b） 局部变量只在定义它的类内有效。

### （★） 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）。

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

答：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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。

### （★） 构造器（constructor）是否可被重写（override）？

答：构造器不能被继承，因此不能被重写，但可以被重载。

### （★） 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。

using System;

namespace CS01 {

class Program {

public static void swap（ref int x, ref int y） {

int temp = x;

x = y;

y = temp;

}

public static void Main （string[] args） {

int a = 5, b = 10;

swap （ref a, ref b）;

// a = 10, b = 5;

Console.WriteLine （"a = {0}, b = {1}", a, b）;

}

}

}

说明：Java中没有传引用实在是非常的不方便，这一点在Java 8中仍然没有得到改进，正是如此在Java编写的代码中才会出现大量的Wrapper类（将需要通过方法调用修改的引用置于一个Wrapper类中，再将Wrapper对象传入方法），这样的做法只会让代码变得臃肿，尤其是让从C和C++转型为Java程序员的开发者无法容忍。

### （★★） 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/StringBuilder类表示的字符串对象可以直接进行修改。StringBuilder是Java 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要高。

面试题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？

面试题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。

### （★） 重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分？

答：方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。重载发生在一个类中，同名的方法如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；重写发生在子类与父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常（里氏代换原则）。重载对返回类型没有特殊的要求。

### （★） 抽象类（abstract class）和接口（interface）有什么异同？

答：抽象类和接口都不能够实例化，但可以定义抽象类和接口类型的引用。一个类如果继承了某个抽象类或者实现了某个接口都需要对其中的抽象方法全部进行实现，否则该类仍然需要被声明为抽象类。接口比抽象类更加抽象，因为抽象类中可以定义构造器，可以有抽象方法和具体方法，而接口中不能定义构造器而且其中的方法全部都是抽象方法。抽象类中的成员可以是private、默认、protected、public的，而接口中的成员全都是public的。抽象类中可以定义成员变量，而接口中定义的成员变量实际上都是常量。有抽象方法的类必须被声明为抽象类，而抽象类未必要有抽象方法。

### （★） 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，其语法看起来挺诡异的，如下所示。

/\*\*

\* 扑克类（一副扑克）

\* @author 骆昊

\*

\*/

public class Poker {

private static String[] suites = {"黑桃", "红桃", "草花", "方块"};

private static int[] faces = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};

private Card[] cards;

/\*\*

\* 构造器

\*

\*/

public Poker（） {

cards = new Card[52];

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

for（int j = 0; j < faces.length; j++） {

cards[i \* 13 + j] = new Card（suites[i], faces[j]）;

}

}

}

/\*\*

\* 洗牌 （随机乱序）

\*

\*/

public void shuffle（） {

for（int i = 0, len = cards.length; i < len; i++） {

int index = （int） （Math.random（） \* len）;

Card temp = cards[index];

cards[index] = cards[i];

cards[i] = temp;

}

}

/\*\*

\* 发牌

\* @param index 发牌的位置

\*

\*/

public Card deal（int index） {

return cards[index];

}

/\*\*

\* 卡片类（一张扑克）

\* [内部类]

\* @author 骆昊

\*

\*/

public class Card {

private String suite; // 花色

private int face; // 点数

public Card（String suite, int face） {

this.suite = suite;

this.face = face;

}

@Override

public String toString（） {

String faceStr = "";

switch（face） {

case 1: faceStr = "A"; break;

case 11: faceStr = "J"; break;

case 12: faceStr = "Q"; break;

case 13: faceStr = "K"; break;

default: faceStr = String.valueOf（face）;

}

return suite + faceStr;

}

}

}

测试代码：

class PokerTest {

public static void main（String[] args） {

Poker poker = new Poker（）;

poker.shuffle（）; // 洗牌

Poker.Card c1 = poker.deal（0）; // 发第一张牌

// 对于非静态内部类Card

// 只有通过其外部类Poker对象才能创建Card对象

Poker.Card c2 = poker.new Card（"红心", 1）; // 自己创建一张牌

System.out.println（c1）; // 洗牌后的第一张

System.out.println（c2）; // 打印: 红心A

}

}

面试题 – 下面的代码哪些地方会产生编译错误？

class Outer {

class Inner {}

public static void foo（） { new Inner（）; }

public void bar（） { new Inner（）; }

public static void main（String[] args） {

new Inner（）;

}

}

注意：Java中非静态内部类对象的创建要依赖其外部类对象，上面的面试题中foo和main方法都是静态方法，静态方法中没有this，也就是说没有所谓的外部类对象，因此无法创建内部类对象，如果要在静态方法中创建内部类对象，可以这样做：

new Outer（）.new Inner（）;

### （★） 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。

### （★） 阐述静态变量和实例变量的区别。

答：静态变量是被static修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；实例变量必须依存于某一实例，需要先创建对象然后通过对象才能访问到它。静态变量可以实现让多个对象共享内存。

补充：在Java开发中，上下文类和工具类中通常会有大量的静态成员。

### （★） 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。

### （★★★） 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，代码如下。

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

public class MyUtil {

private MyUtil（） {

throw new AssertionError（）;

}

public static <T> T clone（T obj） throws Exception {

ByteArrayOutputStream bout = new ByteArrayOutputStream（）;

ObjectOutputStream oos = new ObjectOutputStream（bout）;

oos.writeObject（obj）;

ByteArrayInputStream bin = new ByteArrayInputStream（bout.toByteArray（））;

ObjectInputStream ois = new ObjectInputStream（bin）;

return （T） ois.readObject（）;

// 说明：调用ByteArrayInputStream或ByteArrayOutputStream对象的close方法没有任何意义

// 这两个基于内存的流只要垃圾回收器清理对象就能够释放资源，这一点不同于对外部资源（如文件流）的释放

}

}

下面是测试代码：

import java.io.Serializable;

/\*\*

\* 人类

\* @author 骆昊

\*

\*/

class Person implements Serializable {

private static final long serialVersionUID = -9102017020286042305L;

private String name; // 姓名

private int age; // 年龄

private Car car; // 座驾

public Person（String name, int age, Car car） {

this.name = name;

this.age = age;

this.car = car;

}

public String getName（） {

return name;

}

public void setName（String name） {

this.name = name;

}

public int getAge（） {

return age;

}

public void setAge（int age） {

this.age = age;

}

public Car getCar（） {

return car;

}

public void setCar（Car car） {

this.car = car;

}

@Override

public String toString（） {

return "Person [name=" + name + ", age=" + age + ", car=" + car + "]";

}

}

/\*\*

\* 小汽车类

\* @author 骆昊

\*

\*/

class Car implements Serializable {

private static final long serialVersionUID = -5713945027627603702L;

private String brand; // 品牌

private int maxSpeed; // 最高时速

public Car（String brand, int maxSpeed） {

this.brand = brand;

this.maxSpeed = maxSpeed;

}

public String getBrand（） {

return brand;

}

public void setBrand（String brand） {

this.brand = brand;

}

public int getMaxSpeed（） {

return maxSpeed;

}

public void setMaxSpeed（int maxSpeed） {

this.maxSpeed = maxSpeed;

}

@Override

public String toString（） {

return "Car [brand=" + brand + ", maxSpeed=" + maxSpeed + "]";

}

}

class CloneTest {

public static void main（String[] args） {

try {

Person p1 = new Person（"Hao LUO", 33, new Car（"Benz", 300））;

Person p2 = MyUtil.clone（p1）; // 深度克隆

p2.getCar（）.setBrand（"BYD"）;

// 修改克隆的Person对象p2关联的汽车对象的品牌属性

// 原来的Person对象p1关联的汽车不会受到任何影响

// 因为在克隆Person对象时其关联的汽车对象也被克隆了

System.out.println（p1）;

} catch （Exception e） {

e.printStackTrace（）;

}

}

}

注意：基于序列化和反序列化实现的克隆不仅仅是深度克隆，更重要的是通过泛型限定，可以检查出要克隆的对象是否支持序列化，这项检查是编译器完成的，不是在运行时抛出异常，这种是方案明显优于使用Object类的clone方法克隆对象。让问题在编译的时候暴露出来总是优于把问题留到运行时。

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

答：两个对象，一个是静态区的”xyz”，一个是用new创建在堆上的对象。

### （★） 简述this关键字用法。

a） this关键字使用在实例方法中，代表调用该方法的当前对象。（★） 如何确定方法的返回类型？

### （★） 返回值为void的方法，可否有return？

a） 可以。但return后没有任何值。

### （★） 解释什么是类方法，什么是实例方法？

a） static修饰的方法是类方法，无static修饰的方法是实例方法。

### （★） 什么是方法重载？

a） 方法重载是指一个类中可以有多个方法具有相同的名字，但这些方法的参数必须不同，即或者是参数的个数不同，或者是参数的类型不同。

### （★） 什么是构造方法？

a） 构造方法是一种特殊方法，它的名字必须与它所在的类的名字完全相同，并且不返

回任何数据类型。

### （★） 如何创建一个对象？

a） 使用new 运算符和类的构造方法为对象分配内存，如果类中没有构造方法，系统

会调用默认的构造方法。

### （★） 系统什么情况下会为类提供构造方法，提供什么样的构造方法？

a） 如果类中没有构造方法，系统会提供一个默认的构造方法，默认的构造方法是无参

的。

### （★） 对象如何调用自己的成员变量和方法？

a） 使用运算符” . ”来调用自己的成员变量和方法。

### （★） 为什么可以直接用类名来访问类成员变量和类方法？

a） 因为当类被加载到虚拟机的时候，类成员变量就被分配内存，类方法被分配入口地

址，所以不用创建对象，可以直接通过类名调用。

### （★） 类变量有什么特点？

a） 一个类的所有对象共享同一个类变量。

### （★） 类方法有什么特点？

a） 类方法只能调用类变量和类方法。（同一类中）

### （★） package关键字有什么作用，使用中注意什么问题？

a） package指定一个类所在的包，该语句为源代码第一行。

### （★）import关键字有什么作用？

a） 引入程序中所用到的类。

### （★） 类有几种访问权限？变量和方法有几种访问权限？分别是什么？

a） 类有两种访问权限：public，友好的。

b） 方法和变量的访问权限：public，protected，友好的，private。

### （★） 简述java的访问权限。

a） public：公有的，任何类都可以访问。

b） protected：受保护的，同一个包的类可以访问。不同包的子类可以访问。 c） 友好的：同一个包的类可以访问。

d） private：私有的，在同一个类中才能访问。

### （★） 子类能继承父类的哪些变量和方法？

a） 如果子类和父类在同一个包中，那么，子类自然地继承了其父类中不是private 的

成员变量作为自己的成员变量，并且也自然地继承了父类中不是private 的方法作为自己的方法。

b） 如果子类和父类不在同一个包中，那么，子类继承了父类的protected，public 成

员变量做为子类的成员变量，并且继承了父类的protected，public 方法为子类的方法。

### （★） 子类重写父类的方法，可否降低访问权限？

a） 不可以降低。

### （★） final关键字可以用来修饰什么？分别起什么作用？

a） final可以修饰类，这样的类不能被继承。

b） final可以修饰方法，这样的方法不能被重写。

c） final可以修饰变量，这样的变量的值不能被修改，是常量。

### （★） 简述super关键字作用？

a） 使用super调用父类的构造方法。

b） 使用super操作被隐藏的成员变量和方法。

### （★） 简述什么是对象上转型？

答：a） 假设，A 类是B 类的父类，当我们用子类创建一个对象，并把这个对象的引用放

到父类的对象中时，我们称这个父类对象是子类对象的上转型对象。

### （★） 上转型对象可以操作什么？不可以操作什么？

答：a） 上转对象不能操作子类新增的成员变量，失掉了这部分属性，不能使用子类新增的

方法，失掉了一些功能 。

b） 上转型对象可以操作子类继承或重写的成员变量，也可以使用子类继承的或重写的

方法。

### （★） 什么是抽象类？什么是抽象方法？有什么特点？

答：a） 用关键字abstract修饰类称为抽象类，abstract类不能用new运算创建对象，必须

产生其子类，由子类创建对象。

b） 用关键字abstract修饰方法称为抽象方法，abstract方法，只允许声明，而不允许

实现。

### （★） 一个类声明实现一个接口，那么这个类需要做什么工作？

答：a） 实现接口里所有的方法，并且这些方法的访问权限必须是public。

### （★） 构造方法能否被重写？为什么？

答：a） 不能，因为构造方法不能被继承，所以不能重写。

### （★） 构造方法有什么作用？

答：a） 在创建对象的时候，java虚拟机会调用类的构造方法来创建对象。一般对象的初始

化工作可以放在构造方法里。

### （★） 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。

### （★） 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编程和Android开发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。

### （★） 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）。

### （★） 用Java写一个单例类。

答：线程安全的懒汉式：

public class Singleton {

private static Singleton instance;

private Singleton （）{

}

public static synchronized Singleton getInstance（）{

if（instance==null）{

instance=new SingletonDemo（）;

}

return instance;

}

}

饿汉式单例：

public class Singleton {

private static Singleton instance = new Singleton（）;

private Singleton（）{

}

public static Singleton getInstance（）{

return instance;

}

}

### （★） 类体的内容由两部分构成，哪两部分？

答：a） 一部分是变量的定义，用来刻画属性。

b） 另一部分是方法的定义，用来刻画功能。

### （★） 解释什么是类的成员变量，局部变量，实例成员变量，类成员变量？

答：a） 变量定义部分所定义的变量被称为类的成员变量。

b） 在方法体中定义的变量和方法的参数被称为局部变量。

c） 成员变量又分为实例成员变量和类成员变量（static修饰）。

### （★）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。

### （★）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）。

**补充**：关于equals和hashCode方法，很多Java程序都知道，但很多人也就是仅仅知道而已，在Joshua Bloch的大作《Effective Java》（很多软件公司，《Effective Java》、《Java编程思想》以及《重构：改善既有代码质量》是Java程序员必看书籍，如果你还没看过，那就赶紧去亚马逊买一本吧）中是这样介绍equals方法的：首先equals方法必须满足自反性（x.equals（x）必须返回true）、对称性（x.equals（y）返回true时，y.equals（x）也必须返回true）、传递性（x.equals（y）和y.equals（z）都返回true时，x.equals（z）也必须返回true）和一致性（当x和y引用的对象信息没有被修改时，多次调用x.equals（y）应该得到同样的返回值），而且对于任何非null值的引用x，x.equals（null）必须返回false。实现高质量的equals方法的诀窍包括：1. 使用==操作符检查”参数是否为这个对象的引用”；2. 使用instanceof操作符检查”参数是否为正确的类型”；3. 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；4. 编写完equals方法后，问自己它是否满足对称性、传递性、一致性；5. 重写equals时总是要重写hashCode；6. 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。

### （★）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关系（Has-A）和依赖关系（Use-A）而不是继承关系（Is-A）。

### （★）如何将字符串转换为基本数据类型？

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。

### （★）如何将基本数据类型转换为字符串？

答：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。

### （★）如何实现字符串的反转及替换？

答：方法很多，可以自己写实现也可以使用String或StringBuffer/StringBuilder中的方法。有一道很常见的面试题是用递归实现字符串反转，代码如下所示：

public static String reverse（String originStr） {

if（originStr == null || originStr.length（） <= 1）

return originStr;

return reverse（originStr.substring（1）） + originStr.charAt（0）;

}

### （★）怎样将GB2312编码的字符串转换为ISO-8859-1编码的字符串？

答：代码如下所示：

String s1 = "你好";

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

### （★）打印昨天的当前时刻。

答：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）;      }  } |

### （★）阐述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。

### （★）简述什么是数组？

答：a） 数组是相同类型的数据按顺序组成的一种复合数据类型。通过数组名加数组下标，

来使用数组中的数据。下标从 0 开始排序。

### （★）创建数组是否需要指定数组长度，如何求数组长度？

答：a） 必须指定数组长度，数组调用 .length来获取数组长度。

### （★） equals equalsIgnoreCase startsWith endsWith

答：a） 字符串比较 。

b） 忽略大小写比较。

c） 判断是否以指定字符串开头、结尾。

### （★） regionMatches

答：a） 判断一个字符串的子串是否和另一个字符串的子串匹配。

### （★） compareTo compareToIgnoreCase indexOf

答：a） 按字典顺序比较字符串。

b） 忽略大小写的按字典顺序比较字符串。

c） 字符串检索。

### （★） substring replace trim（）

答：a） 字符串截取

b） 字符串替换

c） 去掉字符串前后空格。

### （★）字符串如何转化为int型，double型？

答：a） Integer.parseInt（“1”）;

b） Double.parseDouble（“25。45”）;

### （★）简述StringTokenizer作用？

答：a） 按照指定的分隔符，将字符串解析成若干语言符号。

### （★）如何判断一个字符是不是数字，是不是大写？

答：a） Character.isDigit（‘a’）

b） Character.isUpperCase（‘U’）

### （★）已知一个java.util.Date对象，如何格式化成如下格式YYYY-MM-dd hh:mm:ss

答：a） SimpleDateFormat formate = new SimpleDateFormat（"yyyy-MM-dd HH:mm:ss"）;

formate.format（new Date（）））;

### （★）如何生成一个0-100的随即整数？

答：a） （int）（Math.random（）\*100）

### （★★）什么时候用断言（assert）？

答：断言在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，断言用于保证程序最基本、关键的正确性。断言检查通常在开发和测试时开启。为了保证程序的执行效率，在软件发布后断言检查通常是关闭的。断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为true；如果表达式的值为false，那么系统会报告一个AssertionError。断言的使用如下面的代码所示：

assert（a > 0）; // throws an AssertionError if a <= 0

断言可以有两种形式：

assert Expression1;

assert Expression1 : Expression2 ;

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的字符串消息。

要在运行时启用断言，可以在启动JVM时使用-enableassertions或者-ea标记。要在运行时选择禁用断言，可以在启动JVM时使用-da或者-disableassertions标记。要在系统类中启用或禁用断言，可使用-esa或-dsa标记。还可以在包的基础上启用或者禁用断言。

注意：断言不应该以任何方式改变程序的状态。简单的说，如果希望在不满足某些条件时阻止代码的执行，就可以考虑用断言来阻止它。

### （★）如何查看一个文件的大小，绝对路径，是否可读？（c,d）

a） File file = new File（“e://a.txt”）;

b） file.length（）;

c） file.getAbsolutePath（）;

d） file.canRead（）;

### （★）下列描述正确的是（AC）

A类只能继承一个父类,但是可以是实现多个接口

B抽象类自身可以定义成员而接口不可以

C抽象类和接口都不能被实例化（忽略匿名内部类）

D一个类可以有多个父类和多个基接口

说明：

A、java为单继承，多实现。可以实现多个接口。

B、接口允许定义成员，但必须是常量。

C、抽象类和接口类的无法实例化，任何编译器中直接使用new会报错。

D、同A，单继承，多实现。

### （★）以下关于final关键字说法正确的是（BD）

A final是java中的修饰符,可以修饰类,接口,抽象类,方法和属性

B final修饰的类肯定不能被继承

C final修饰的方法不能被重载

D final修饰的变量不允许被再次赋值

说明：

A、修饰类（class）。

1、该类不能被继承。

2、类中的方法不会被覆盖，因此默认都是final的。

3、用途：设计类时，如果该类不需要有子类，不必要被扩展，类的实现细节不允许被改变，那么就设计成final类

B、修饰方法（method）

1、该方法可以被继承，但是不能被覆盖。

2、用途：一个类不允许子类覆盖该方法，则用final来修饰

3、好处：可以防止继承它的子类修改该方法的意义和实现；更为高效，编译器在遇到调用fianal方法转入内嵌机制，提高了执行效率。

4、注意：父类中的private成员方法不能被子类覆盖，因此，private方法默认是final型的（可以查看编译后的class文件）

C、修饰变量（variable）

1、用final修饰后变为常量。包括静态变量、实例变量和局部变量这三种。

2、特点：可以先声明，不给初值，这种叫做final空白。但是使用前必须被初始化。一旦被赋值，将不能再被改变。

D、修饰参数（arguments）

1、用final修饰参数时，可以读取该参数，但是不能对其作出修改

### （★★★）java描述spring or struts体系结构?对应各个部分的开发工作主要包括哪些?

答：Struts 是MVC的一种实现，它将 Servlet和 JSP 标记（属于 J2EE 规范）用作实现的一部分。Struts继承了MVC的各项特性，并根据J2EE的特点，做了相应的变化与扩展。

1）模型（Model）

　　在Struts的体系结构中，模型分为两个部分：系统的内部状态和可以改变状态的操作（事务逻辑）。内部状态通常由一组Actinform Bean表示。根据设计或应用程序复杂度的不同，这些Bean可以是自包含的并具有持续的状态，或只在需要时才获得数据（从某个数据库）。大型应用程序通常在方法内部封装事务逻辑（操作），这些方法可以被拥有状态信息的bean调用。比如购物车bean，它拥有用户购买商品的信息，可能还有checkOut（）方法用来检查用户的信用卡，并向仓库发定货信息。小型程序中，操作可能会被内嵌在Action类，它是struts框架中控制器角色的一部分。当逻辑简单时这个方法很适合。建议用户将事务逻辑（要做什么）与Action类所扮演的角色（决定做什么）分开。

　　2）视图（View）

　　视图主要由JSP建立，struts包含扩展自定义标签库（TagLib），可以简化创建完全国际化用户界面的过程。目前的标签库包括：Bean Tags、HTML tags、Logic Tags、Nested Tags 以及Template Tags等。

　　3）控制器（Controller）

　　在struts中，基本的控制器组件是ActionServlet类中的实例servelt，实际使用的servlet在配置文件中由一组映射（由ActionMapping类进行描述）进行定义。对于业务逻辑的操作则主要由Action、ActionMapping、ActionForward这几个组件协调完成的，其中Action扮演了真正的业务逻辑的实现者，ActionMapping与ActionForward则指定了不同业务逻辑或流程的运行方向。struts-config.xml 文件配置控制器。

### （★★★）JSP有哪些内置对象和动作?它们的作用分别是什么?

答：1、request对象

　　request 对象是 javax.servlet.httpServletRequest类型的对象。 该对象代表了客户端的请求信息，主要用于接受通过HTTP协议传送到服务器的数据。（包括头信息、系统信息、请求方式以及请求参数等）。request对象的作用域为一次请求。

　　2、response对象

　　response 代表的是对客户端的响应，主要是将JSP容器处理过的对象传回到客户端。response对象也具有作用域，只在JSP页面内有效。

　　3、session对象

　　session 对象是由服务器自动创建的与用户请求相关的对象。服务器为每个用户都生成一个session对象，用于保存该用户的信息，跟踪用户的操作状态。session对象内部使用Map类来保存数据，因此保存数据的格式为 “Key/value”。 session对象的value可以使复杂的对象类型，而不仅仅局限于字符串类型。

　　4、application对象

　　application 对象可将信息保存在服务器中，直到服务器关闭，否则application对象中保存的信息会在整个应用中都有效。与session对象相比，application对象生命周期更长，类似于系统的“全局变量”。

　　5、out 对象

　　out 对象用于在Web浏览器内输出信息，并且管理应用服务器上的输出缓冲区。在使用 out 对象输出数据时，可以对数据缓冲区进行操作，及时清除缓冲区中的残余数据，为其他的输出让出缓冲空间。待数据输出完毕后，要及时关闭输出流。

　　6、pageContext 对象

　　pageContext 对象的作用是取得任何范围的参数，可以获取 JSP页面的out、request、reponse、session、application 等对象。pageContext对象的创建和初始化都是由容器来完成的，在JSP页面中可以直接使用 pageContext对象。

　　7、config 对象

　　config 对象的主要作用是取得服务器的配置信息。通过 pageConext对象的 getServletConfig（） 方法可以获取一个config对象。当一个Servlet 初始化时，容器把某些信息通过 config对象传递给这个 Servlet。 开发者可以在web.xml 文件中为应用程序环境中的Servlet程序和JSP页面提供初始化参数。

　　8、page 对象

　　page 对象代表JSP本身，只有在JSP页面内才是合法的。 page隐含对象本质上包含当前 Servlet接口引用的变量，类似于Java编程中的 this 指针。

　　9、exception 对象

　　exception 对象的作用是显示异常信息，只有在包含 isErrorPage="true" 的页面中才可以被使用，在一般的JSP页面中使用该对象将无法编译JSP文件。

### （★）Java中，如何实例化List接口？请用代码写出你知道的一种方式：

答：List<String> itemList=new ArrayList<String>（）;

### （★）有如下字符串“1，as，bcc，31,0z”，将此字符串按照逗号分割成数组，请写出代码：

答：

String str1="1,as,bcc,31,0z";

String[] split = str1.split（","）;

for （String item:split） {

System.out.println（item）;

}

### （★）有如下类定义：

Public class Person{

Public void getName（）{

System.out.println（“I am Name”）;

}

Public static void getAge（）{

System.out.println（“I am”+20+“years old”）;

}

}

以下1-5是否正确，不正确请写出正确代码

Person per =new Person（）;1

Per.getName（）;2

Person.getName（）;3

Per.getAge（）;4

Person.getAge（）;5

答：

3->per.getName（）;

### （★）请用JAVA写出一个线程安全的单例（Singleton）考虑下实现是否可以更优。

答：线程安全的懒汉式：

public class Singleton {

private static Singleton instance;

private Singleton （）{

}

public static synchronized Singleton getInstance（）{

if（instance==null）{

instance=new SingletonDemo（）;

}

return instance;

}

}

饿汉式单例：

public class Singleton {

private static Singleton instance = new Singleton（）;

private Singleton（）{

}

public static Singleton getInstance（）{

return instance;

}

}

### （★）下列关于修饰符混用的说法， 错误的是（ D）

A. abstract 不能与final并列修饰同一个类

B. abstract 类中可以有private 的成员

C. abstract 方法必须在abstract类中

D. static 方法中能处理非static的属性

### （★）Java 语言中，String 类的IndexOf）方法返回的类型是?（C）

A.Int16 B. Int32 C. int D. long

### （★）下面哪项说法正确（C）

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

### （★）关于Java编译，下面哪一个正确（B）

A、Java程序经编译后会产生machine code

B、Java程序经编译后会产生byte code

C、Java程序经编译后会产生DLL

D、以上都不正确

### （★）重载（Overload）和重写（Override）的区别，重载的方法能否根据返回类型进行区分？

答：方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。重载发生在一个类中，同名的方法如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；重写发生在子类与父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常（里氏代换原则）。重载对返回类型没有特殊的要求。

### （★★★）抽象类（abstract class）和接口（interface）有什么异同？

答：抽象类：包含抽象方法的类就是抽象类

接口：指抽象方法的集合，方法不包含方法体

相同点：

1.都不能被实例化

2. 接口的实现类或者抽象类的子类都必须实现了接口或抽象类中的方法后才可以被实例化

不同点:

1.接口只有定义，方法不能在接口中实现，实现接口的类要实现接口中所有的方法；抽象类可以有定义与实现，方法可以在抽象类中实现

2.接口要实现，抽象类要继承，一个类可以实现多个接口，但只能继承一个抽象类

3.接口强调设计理念为“has -a”的关系，抽象类强调“is -a”关系

4.接口中定义变量默认为public static final,且要赋初值，方法必须是public,static，且只能是这两个；抽象类中可以有自己的数据成员变量，也可以有非抽象的成员方法，而且成员变量默认为default。这些成员变量可以在子类中被重新定义，也可以重新赋值，抽象方法（有abstract修饰）不能用peivate,static,synchronized,native等访问修饰符修饰，同时方法以分号结尾，并且不带花括号

5.接口被运用于比较常用的功能，便于日后的维护或者添加删除方法；而抽象类更倾向于充当公共类的角色，不适用于对里面的代码进行修改

接口是一种特殊形式的抽象类，使用接口完全有可能实现与抽象类相同的操作。当子类和父类之间存在有逻辑上的层次结构时，推荐使用抽象类；当用于不同类之间，定义不同类之间的通信规则，希望支持差别较大的两个或者更多对象之间的特定交互行为时，应该使用接口。

此外，接口可以继承接口，抽象类可以实现接口，抽象类也可以继承具体类。抽象类也可以有静态的main方法。

### （★）下面哪一项不是Java类访问控制关键字：C

A. public B.protected C.this D.private

### （★）什么是多态？多态是如何实现的？对于成员变量，成员函数和静态函数，那些具有多态性？

事物在运行过程中存在不同的状态，简单的说父类的引用可以指向子类的对象，多态的存在3个前提1.要有继承关系2.子类要重写父类的方法，3.父类引用指向子类对象。

成员变量：编译看父类，运行看父类，成员方法：编译看父类，运行看子类，静态方法：编译看父类，运行看父类

### （★）JAVA语言如何进行异常处理，关键字：throws,throw,try,catch,finally分别代表什么意思，如果try{}里有一个return语句，那么紧跟在这个try后的finally{}里的code什么时候执行，在return前还是后？

Throws是获取异常 throw是抛出异常 try是将会发生的异常的语句括起来，从而进行异常处理 catch是如果有异常就会执行他括号里的语句 finally不论是否有异常都会进行执行语句。

在return还没有返回结果的时候，执行finally{}里面的代码。

### （★）下列不可作为java语言修饰符的是（C）

A a1 B$1 C\_1 D11

以下代码段将创创建几个对象（D）

String s1 = “bc”

String s2 = “bc”

A 2 B 3 C 0 D 1

### （★）有以下方法的定义，请选择该方法的返回类型（B）

ReturnType methof[byte x,double y]

return （short）x/y\*2;

A byte B short C int Ddouble

### （★）执行下面代码下面描述正确的是（D）

public class Person{

static int arr[] = new int[10]

public static void main（String a[]）{

System.out.println（arr[1]）

}

A 产生编译错误 B输出空 C编译正确，运行错误 D输出0

### （★）执行下列语句后i和n的值为（B）

int i = 10

int n = i++%5;

A 10,0 B 11,0 C 10,1 D 11,1

### （★）下列的哪些选项可以正确用以表示八进制8（D）

A、0x8 B、0x10 C、08 D、010

### （★）下面说法正确的有（B）

1. class中的constructor不可忽略
2. constructor必须与class同名，但方法不能与class同名
3. constructor在一个对象被new是执行
4. 一个class只能定义一个constructor

### （★）0.6332的数据类型是（B）

A、float B、double C、Float D、Double

### （★）Java接口的修饰符可以为（CD）

1. private B、protected C、final D、abstract

## 2. 异常部分

### （★）Java中的两种异常类型是什么？他们有什么区别？\*

答: Java中有两种异常：受检查的（checked）异常和不受检查的（unchecked）异常。不受检查的异常不需要在方法或者是构造函数上声明，就算方法或者是构造函数的执行可能会抛出这样的异常，并且不受检查的异常可以传播到方法或者是构造函数的外面。相反，受检查的异常必须要用throws语句在方法或者是构造函数上声明。这里有Java异常处理的一些小建议。

### （★）Java中Exception和Error有什么区别？\*

答: Exception和Error都是Throwable的子类。Exception用于用户程序可以捕获的异常情况。Error定义了不期望被用户程序捕获的异常。

### （★）throw和throws有什么区别？\*

答: throw关键字用来在程序中明确的抛出异常，相反，throws语句用来表明方法不能处理的异常。每一个方法都必须要指定哪些异常不能处理，所以方法的调用者才能够确保处理可能发生的异常，多个异常是用逗号分隔的。

### （★）Java 中的异常处理机制的简单原理和应用？【基础】\*\*

答：当JAVA 程序违反了JAVA 的语义规则时，JAVA 虚拟机就会将发生的错误表示为一个异常。违反语义规则包括2 种情况。一种是JAVA 类库内置的语义检查。例如数组下标越界,会引发IndexOutOfBoundsException;访问null 的对象时会引发NullPointerException。另一种情况就是JAVA 允许程序员扩展这种语义检查，程序员可以创建自己的异常，并自由选择在何时用throw 关键字引发异常。所有的异常都是java.lang.Thowable 的子类。

### （★）error 和exception 有什么区别? 【基础】\*

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。

### （★）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? 【基础】\*

答：会执行，在return 前执行。

### （★）JAVA 语言如何进行异常处理，关键字：throws,throw,try,catch,finally分别代表什么意义？在try 块中可以抛出异常吗？【基础】

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 类或其它子类的实例。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（throws）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”的框架就放到堆栈上面，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，堆栈就会展开，直到遇到有处理这种“异常”的try 语句。

### （★）运行时异常与一般异常有何异同？【基础】

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。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

### （★★）给我一个你最常见到的runtime exception？【基础】

答：ArithmeticException, ArrayStoreException, BufferOverflowException,

BufferUnderflowException, CannotRedoException, CannotUndoException,

ClassCastException, CMMException, ConcurrentModificationException,

DOMException, EmptyStackException, IllegalArgumentException,

IllegalMonitorStateException, IllegalPathStateException,

IllegalStateException, ImagingOpException, IndexOutOfBoundsException,

MissingResourceException, NegativeArraySizeException,

NoSuchElementException, NullPointerException, ProfileDataException,

ProviderException, RasterFormatException, SecurityException,

SystemException, UndeclaredThrowableException,

UnmodifiableSetException, UnsupportedOperationException

### （★）final, finally, finalize 的区别? 【基础】

答：

final：修饰符（关键字）；如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承，因此一个类不能既被声明为abstract的，又被声明为final 的；将变量或方法声明为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，不能重载。

Finally：再异常处理时提供finally 块来执行任何清除操作；如果抛出一个异常，那么相匹配的catch 子句就会执行，然后控制就会进入finally 块（如果有的话）。finalize：方法名；Java 技术允许使用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确定这个对象没有被引用时对这个对象调用的。它是在Object 类中定义的，因此所有的类都继承了它。子类覆盖finalize（） 方法以整理系统资源或者执行其他清理工作。

finalize（） 方法是在垃圾收集器删除对象之前对这个对象调用的。

69、类Example A 继承Exception，类ExampleB 继承Example A；【基础】

有如下代码片断：

try{

throw new ExampleB（“b”）；

}catch（ExampleA e）{

System.out.printfln（“ExampleA”）；

}catch（Exception e）{

System.out.printfln（“Exception”）；

}

输出的内容应该是：

A：ExampleA B：Exception C：b D：无

答：输出为A。

### （★）Error和Exception有什么区别？

Error是系统中的错误，程序员是不能够改变和处理的，是在编译时出现的错误，只能通过修改程序才能修正。一般指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢出。这类错误导致的应用程序中断，仅靠程序本身无法恢复。

Exception程序可以处理的异常，可以捕获可能恢复。遇到这类异常，应该尽可能处理异常，使程序恢复运行，而不应该随意终止异常。

### （★）运行时异常与受检异常有何异同？

-本质区别是运行时异常是运行时候才会发生的异常，而受检异常是编译时异常，编译器会分析哪些异常会在执行一个方法或者构造函数的时候抛出

运行时异常（RuntimeException）也称作未检测的异常（unchecked exception），这表示这种异常不需要编译器来检测。RuntimeException是所有可以在运行时抛出的异常的父类。一个方法除要捕获异常外，如果它执行的时候可能会抛出RuntimeException的子类，那么它就不需要用throw语句来声明抛出的异常

受检查异常（checked exception）都是编译器在编译时进行校验的，通过throws语句或者try{}cathch{} 语句块来处理检测异常。编译器会分析哪些异常会在执行一个方法或者构造函数的时候抛出。

### （★）列出一些你常见的运行时异常？

NullPointerException 空指针异常，ArrayIndexOutOfBoundsException数组越界，IllegalArgumentException传递非法参数异常，IndexOutOfBoundsException。NumberFormatException 数字格式异常

### （★）简述java异常处理的机制NumberFormatException？

数字格式异常，当我们传入的一个值不能转为int类型的时候就会抱着个错误

### （★★）什么时候会产生OutOfMeoryErrorException？说明常见产生原因？

当对象无限被创建且无法回收时候，java堆区将会出现内存溢出异常

1. 内存中加载的数据量过于庞大，如一次从数据库取出过多数据；

2. 集合类中有对对象的引用，使用完后未清空，使得JVM不能回收；

3. 代码中存在死循环或循环产生过多重复的对象实体；

4. 使用的第三方软件中的BUG；

5. 启动参数内存值设定的过小；

### （★★）java.lang.OutOfMemoryError:PermGen space,请问该错误是于什么原因造成的，如何解决？

答：

内存溢出，解决方法一定是加大内存。

什么会内存溢出：这一部分用于存放Class和Meta的信息,Class在被 Load的时候被放入PermGen space区域，它和和存放Instance的Heap区域不同,GC（Garbage Collection）不会在主程序运行期对PermGen space进行清理，所以如果你的APP会LOAD很多CLASS的话,就很可能出现PermGen space错误。这种错误常见在web服务器对JSP进行pre compile的时候。

### （★）Java中throw和throws有什么区别？

1.用户程序自定义的异常和应用程序特定的异常,必须借助于 throws 和 throw 语句来定义抛出异常。

1.1 throw是语句抛出一个异常。

语法：throw （异常对象）;

throw e;

1.2 throws是方法可能抛出异常的声明。（用在声明方法时，表示该方法可能要抛出异常）

语法：[（修饰符）]（返回值类型）（方法名）（[参数列表]）[throws（异常类）]{......}

public void doA（int a） throws Exception1,Exception3{......}

### （★★）详述java内存溢出两种异常的原因，并未每种原因举例说明。

（1）java.lang.StackOverflowError 当程序递归太深而发生堆栈溢出时，抛出错误

（2）java.lang.OutOfMemoryError：Java heap space 在JVM中假设98%的时间用于GC, 且可用的 Heap size 不足2％的时候将抛出此异常信息。**哪些异常是检查性异常，需要编写程序时声明（C）**

1. NullPointerException
2. ClassCastException
3. FileNotFoundException
4. IndexOutOfBoundException

### （★★）java中的两种异常类型是什么？他们有什么区别？

一般分为Checked异常和Runtime异常，所有RuntimeException类及其子类的实例被称为Runtime异常，不属于该范畴的异常则被称为CheckedException。

①Checked异常

只有java语言提供了Checked异常，Java认为Checked异常都是可以被处理的异常，所以Java程序必须显示处理Checked异常。如果程序没有处理Checked异常，该程序在编译时就会发生错误无法编译。这体现了Java的设计哲学：没有完善错误处理的代码根本没有机会被执行。对Checked异常处理方法有两种

1 当前方法知道如何处理该异常，则用try...catch块来处理该异常。

2 当前方法不知道如何处理，则在定义该方法是声明抛出该异常。

我们比较熟悉的Checked异常有

Java.lang.ClassNotFoundException

Java.lang.NoSuchMetodException

java.io.IOException

②RuntimeException

Runtime如除数是0和数组下标越界等，其产生频繁，处理麻烦，若显示申明或者捕获将会对程序的可读性和运行效率影响很大。所以由系统自动检测并将它们交给缺省的异常处理程序。当然如果你有处理要求也可以显示捕获它们。

我们比较熟悉的RumtimeException类的子类有 Java.lang.ArithmeticException

Java.lang.ArrayStoreExcetpion

Java.lang.ClassCastException

Java.lang.IndexOutOfBoundsException

Java.lang.NullPointerException

## 3.集合部分

### （★）Java集合类框架的基本接口有哪些？

答: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。

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。

### （★）什么是迭代器（Iterator）？

答: 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,但是不可以直接调用集合的remove（Object Obj）删除，可以通过迭代器的remove（）方法删除。

### （★）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？

答:下面列出了Array和ArrayList的不同点：

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

（2）Array大小是固定的，ArrayList的大小是动态变化的。

（3）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？

答: 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。

### （★）HashSet和TreeSet有什么区别？

答: 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）。

### （★）请说出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，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。

### （★★）Set 和 List的区别?

**List：列表。**  
特点：

  1，有序（存储元素的顺序和取出元素的顺序一致）

  2，该集合中的元素都有索引，所以可以通过索引（角标）来访问元素。   
  3，它可以存储重复元素。

 常见子类对象:记住：具体的子类对象，我们要学习应该是该对象的特有的数据结构，以及相关的特点。

 |--Vector:jdk1.0就存在了。底层是数组结构的。可变长度数组，  
      原理：一旦原数组长度不够，会创建新数组，将原数组的元素复制到新数组中，并将新元素添加到新数组中。  
Vector是同步的。  
        |--ArrayList:底层是数组结构，也是支持长度可变数组的。是不同步的。替代了Vector.因为效率高。 查询效率很高。 但是增删的效率很低。  
        |--LinkedList:底层是链接列表结构，简称链表结构。是不同步的。这个中结构的好处：对元素的增删非常效率很高。 查询的效率很低。

**Set：集**

Set中的方法和Collection一致，只要重点注意它的子类对象即可。取出元素只能使用迭代器。  
     特点：  
       1.不包含重复元素。（最大的特点）  
       2.这个集合存入元素的顺序和取出元素的顺序不一定一致。（具体的容器对象数据结构不同，顺序也有不同）  
|--HashSet：底层数据结构是哈希表，不保证顺序，是不同步的。  
          哈希表：提供数组的查询效率而出现的。  
          将要存储的元素先通过哈希算法算出一个哈希值来标识存储的位置，代表着元素。要找元素时，先将该元素通过哈希算法算出哈希值，在通过哈希值到哈希表中去查找。  
      特点：  
          1.不关系元素的顺序。  
          2.提高了查询效率。  
          3.不可能出现重复元素，因为哈希值都不同。即使相同，会再次判断两个元素的equals，内容是否相同。  
            如果内容也相同，不存，如果内容不同，存储。所以哈希表要保证元素的唯一性，必须要依赖于两个方法。  
1,hashCode 2,equals                  
     |--TreeSet：可以给Set集合中的元素进行指定顺序的排序。非同步的。   
        默认情况下，是通过元素的自然顺序排的序。  
        它保证元素唯一性的依据是看比较方法的返回结果是否是0.是0.就视为元素相同。不存。  
        TreeSet排序的方式一：让元素自身具备比较性，需要实现Comparable接口，覆盖compareTo方法。这种比较方式成为自然顺序排序。如果元素自身不具备比较性或者具备的比较性（自然顺序）不是所需要的。  
这时只能用第二种方式 。  
      TreeSet排序的方式二：让容器自身具备比较性。容器一初始化就具备了比较功能。因为容器时在对象构造时完成的。通过查阅，有一个构造方法TreeSet（Comparator）.在容器初始化时可以指定一个比较器。 需要实现Comparator接口，覆盖compare方法即可。所以这种方式成为比较器排序。

当存入的的对象有重复时，用List，没有重复元素时，用Set。

### （★）简单叙述出collections类中的三个常用类

答:对于Collections类的定义可以用一句话来描述"该类只提供了一些静态方法（实际上有三个常量）,通过这些方法可以对集合对象进行操作或返回集合对象。"下面对其所提供方法进行简单介绍。binarySearch/copy

### （★）介绍JAVA 中的Collection FrameWork（及如何写自己的数据结构）【基础】

答：Collection FrameWork 如下：

Collection

├List

│├LinkedList

│├ArrayList

│└Vector

│ └Stack

└Set

Map

├Hashtable

├HashMap

└WeakHashMap

Collection 是最基本的集合接口，一个Collection 代表一组Object，即Collection 的元素（Elements）； Map 提供key 到value 的映射。

### （★）List,Set,Map 是否继承自Collection 接口？【基础】

答：List,Set 是；Map 不是。

### （★）你所知道的集合类都有哪些？主要方法？【基础】

答：最常用的集合类是List 和Map。List 的具体实现包括ArrayList 和Vector，它们是可变大小的列表，比较适合构建、存储和操作任何类型对象的元素列表。List 适用于按数值索引访问元素的情形。Map 提供了一个更通用的元素存储方法。Map 集合类用于存储元素对（称作“键”和“值”），其中每个键映射到一个值。

### （★）说出ArrayList,Vector, LinkedList 的存储性能和特性？【基础】

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，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。

### （★）Collection 和Collections 的区别？【基础】

答：Collection 是java.util 下的接口，它是各种集合的父接口，继承于它的接口主要有Set 和List；Collections 是个java.util 下的类，是针对集合的帮助类，提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

### （★）HashMap 和Hashtable 的区别? 【基础】

答：

二者都实现了Map 接口，是将惟一键映射到特定的值上；

主要区别在于：

1）HashMap 没有排序，允许一个null 键和多个null 值,而Hashtable 不允许；

2）HashMap 把Hashtable 的contains 方法去掉了，改成containsvalue 和containsKey,因为contains 方法容易让人引起误解；

3）Hashtable 继承自Dictionary 类，HashMap 是Java1.2 引进的Map 接口的实现；

4）Hashtable 的方法是Synchronize 的，而HashMap 不是，在多个线程访问Hashtable 时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供外同步。Hashtable 和HashMap 采用的hash/rehash 算法大致一样，所以性能不会有很大的差异。

### （★）Arraylist 与Vector 区别？【基础】

答：就ArrayList 与Vector 主要从二方面来说：

1）同步性：Vector 是线程安全的（同步），而ArrayList 是线程序不安全的；

2）数据增长：当需要增长时,Vector 默认增长一倍，而ArrayList 却是一半。

### （★）List、Map、Set 三个接口，存取元素时，各有什么特点？【基础】

答：List 以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部排序。Map 保存key-value 值，value 可多值。

### （★）Set 里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还是equals（）? 它们有何区别? 【基础】

答：Set 里的元素是不能重复的，用equals （）方法来区分重复与否。覆盖equals（）方法用来判断对象的内容是否相同，而”==”判断地址是否相等,用来决定引用值是否指向同一对象。

### （★★）用程序给出随便大小的10 个数，序号为1-10，按从小到大顺序输出，并输出相应的序号。【基础】

答：代码如下：

package test;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Iterator;

import java.util.List;

import java.util.Random;

public class RandomSort {

public static void printRandomBySort（） {

Random random = new Random（）; // 创建随机数生成器

List list = new ArrayList（）;

// 生成10 个随机数，并放在集合list 中

for （int i = 0; i < 10; i++） {

list.add（random.nextInt（1000））;

}

Collections.sort（list）; // 对集合中的元素进行排序

Iterator it = list.iterator（）;

int count = 0;

while （it.hasNext（）） { // 顺序输出排序后集合中的元素

System.out.println（++count + ": " + it.next（））;

}

}

public static void main（String[] args） {

printRandomBySort（）;

}

}

### （★★）用JAVA 实现一种排序，JAVA 类实现序列化的方法？ 在COLLECTION 框架中，实现比较要实现什么样的接口？【基础】

答：

用插入法进行排序代码如下：

package test;

import java.util.\*;

class InsertSort {

ArrayList al;

public InsertSort（int num,int mod） {

al = new ArrayList（num）;

Random rand = new Random（）;

System.out.println（"The ArrayList Sort Before:"）;

for （int i=0;i<num ;i++ ）{

al.add（new Integer（Math.abs（rand.nextInt（）） % mod +

1））;

System.out.println（"al["+i+"]="+al.get（i））;

}

}

public void SortIt（）{

tempInt;

int MaxSize=1;

for（int i=1;i<al.size（）;i++）{

tempInt = （Integer）al.remove（i）;

if（tempInt.intValue（） >=

（（Integer）al.get（MaxSize-1））.intValue（））{

al.add（MaxSize,tempInt）;

MaxSize++;

System.out.println（al.toString（））;

}else{

for （int j=0;j<MaxSize ;j++ ）{

if （（（Integer）al.get（j））.intValue（）

>=tempInt.intValue（））{

al.add（j,tempInt）;

MaxSize++;

System.out.println（al.toString（））;

break;

}

}

}

}

System.out.println（"The ArrayList Sort After:"）;

for（int i=0;i<al.size（）;i++）{

System.out.println（"al["+i+"]="+al.get（i））;

}

}

public static void main（String[] args）{

InsertSort is = new InsertSort（10,100）;

is.SortIt（）;

}

}

JAVA 类实现序例化的方法是实现java.io.Serializable 接口；Collection 框架中实现比较要实现Comparable 接口和Comparator 接口。

### （★★）创建线程有几种不同的方式？你喜欢哪一种？为什么？

答:有两种方式可以用来创建线程：

（1）继承Thread类

（2）实现Runnable接口

（3）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。

答:

1. 新建（ new ）：新创建了一个线程对象。

2. 可运行（ runnable ）：线程对象创建后，其他线程（比如 main 线程）调用了该对象 的 start （）方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获 取 cpu 的使用权 。

3. 运行（ running ）：可运行状态（ runnable ）的线程获得了 cpu 时间片（ timeslice ） ，执行程序代码。

4. 阻塞（ block ）：阻塞状态是指线程因为某种原因放弃了 cpu 使用权，也即让出了 cpu timeslice ，暂时停止运行。直到线程进入可运行（ runnable ）状态，才有 机会再次获得 cpu timeslice 转到运行（ running ）状态。阻塞的情况分三种：

（一）. 等待阻塞：运行（ running ）的线程执行 o . wait （）方法， JVM 会把该线程放 入等待队列（ waitting queue ）中。

（二）. 同步阻塞：运行（ running ）的线程在获取对象的同步锁时，若该同步锁 被别的线程占用，则 JVM 会把该线程放入锁池（ lock pool ）中。

（三）. 其他阻塞: 运行（ running ）的线程执行 Thread . sleep （ long ms ）或 t . join （）方法，或者发出了 I / O 请求时， JVM 会把该线程置为阻塞状态。当 sleep （）状态超时、 join （）等待线程终止或者超时、或者 I / O 处理完毕时，线程重新转入可运行（ runnable ）状态。

5. 死亡（ dead ）：线程 run （）、 main （） 方法执行结束，或者因异常退出了 run （）方法，则该线程结束生命周期。死亡的线程不可再次复生。

### （★★）同步方法和同步代码块的区别是什么？

答:区别：

1.同步方法默认用this或者当前类class对象作为锁；

2.同步代码块可以选择以什么来加锁，比同步方法要更细颗粒度，我们可以选择只同步会发生同步问题的部分代码而不是整个方法；

### （★★）什么是死锁（deadlock）？如何确保N个线程可以访问N个资源同时又不导致死锁？

答:两个线程或两个以上线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。结果就是这些线程都陷入了无限的等待中.

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。

### （★）多线程的优缺点是什么?

答:**优点：**

（1）多线程技术使程序的响应速度更快 ,因为用户界面可以在进行其它工作的同时一直处于活动状态;

（2）当前没有进行处理的任务时可以将处理器时间让给其它任务;

（3）占用大量处理时间的任务可以定期将处理器时间让给其它任务;

（4）可以随时停止任务;

（5）可以分别设置各个任务的优先级以优化性能。

是否需要创建多个线程取决于各种因素。在以下情况下,最适合采用多线程处理:

（1）耗时或大量占用处理器的任务阻塞用户界面操作;

（2）各个任务必须等待外部资源 （如远程文件或 Internet连接）。

同样的 ,多线程也存在许多缺点 ,在考虑多线程时需要进行充分的考虑。

**多线程的主要缺点包括:**

（1）等候使用共享资源时造成程序的运行速度变慢。这些共享资源主要是独占性的资源 ,如打印机等。

（2）对线程进行管理要求额外的 CPU开销。线程的使用会给系统带来上下文切换的额外负担。当这种负担超过一定程度时,多线程的特点主要表现在其缺点上,比如用独立的线程来更新数组内每个元素。

（3）线程的死锁。即较长时间的等待或资源竞争以及死锁等多线程症状。

（4）对公有变量的同时读或写。当多个线程需要对公有变量进行写操作时,后一个线程往往会修改掉前一个线程存放的数据,从而使前一个线程的参数被修改;另外 ,当公用变量的读写操作是非原子性时,在不同的机器上,中断时间的不确定性,会导致数据在一个线程内的操作产生错误,从而产生莫名其妙的错误,而这种错误是程序员无法预知的。

### （★★）sleep（）和wait（）的区别?

答:

1、这两个方法来自不同的类分别是Thread和Object

2、最主要是sleep方法没有释放锁，而 wait 方法释放了锁，使得其他线程可以使用同步控制块或者方法。

3、wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用（使用范围）

1. sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常

### （★★）什么是线程同步 如何实现线程同步

答:当使用多个线程来访问同一个数据时，非常容易出现线程安全问题（比如多个线程都在操作同一数据导致数据不一致）,所以我们用同步机制来解决这些问题。

实现同步机制有两个方法：

1.同步代码块：

synchronized（同一个数据）{} 同一个数据：就是N条线程同时访问一个数据。

2.同步方法：

public synchronized 数据返回类型 方法名（）{}

就是使用 synchronized 来修饰某个方法，则该方法称为同步方法。对于同步方法而言，无需显示指定同步监视器，同步方法的同步监视器是 this 也就是该对象的本身（这里指的对象本身有点含糊，其实就是调用该同步方法的对象）通过使用同步方法，可非常方便的将某类变成线程安全的类，具有如下特征：

1，该类的对象可以被多个线程安全的访问。

2，每个线程调用该对象的任意方法之后，都将得到正确的结果。

3，每个线程调用该对象的任意方法之后，该对象状态依然保持合理状态。

注：synchronized关键字可以修饰方法，也可以修饰代码块，但不能修饰构造器，属性等。

实现同步机制注意以下几点： 安全性高，性能低，在多线程用。性能高，安全性低，在单线程用。

1. 不要对线程安全类的所有方法都进行同步，只对那些会改变共享资源方法的进行同步。

2，如果可变类有两种运行环境，当线程环境和多线程环境则应该为该可变类提供两种版本：线程安全版本和线程不安全版本（没有同步方法和同步块）。在单线程中环境中，使用线程不安全版本以保证性能，在多线程中使用线程安全版本.

### （★★）io流 按照流向分 按照处理内容分? 字节流都有哪些方法?

答:【流的分类详细】

[ 输入流---输出流 ]

按照流的流向，可以将流分为输入流和输出流。

输入流：从其中读取数据

输出流：向其种写入数据

注意：划分输入/输出流是从程序运行时的所在的内存的角度来划分的。

输入流常用的基础类：InputStream和Reader

输出流常用的基础类：OutputStream和Writer

注意：这些积累都是抽象类，无法直接实例化。

[ 字节流---字符流 ]

字节流和字符流的内的方法几乎一样，区别仅仅是其操作的数据单元大小不同，

字节流的数据操作单元： 8位的字节

字符流的数据操作单元： 16位的字节

字节流常用的基础类：InputStream和OutputStream

字符流常用的基础类：Reader和Writer

### （★★★）编写函数扫描制定目录（包含各级目录）下所有文本文件（ \*.txt ），将扫描到的文件全路径以及文件大小记录到指定的文件中，返回扫描到的文件总数。

答: 先上个单线程，留待改进。

/\*\*

\* 总入口

\* @param path

\* @param list\_filo

\*/

static void intScanTextFiles（String path,String list\_filo）{

StringBuffer result = new StringBuffer（）;

int count = scanFiles（path, list\_filo,result）;

writeResult（result, list\_filo,count）;

}

static int scanFiles（String path,String list\_filo,StringBuffer result）{

File file = new File（path）;

File[] list = file.listFiles（）;

int count = 0;

for （File file2 : list） {

if（file2.isDirectory（））{

count += scanFiles（file2.getAbsolutePath（）, list\_filo,result）;

}

if（file2.isFile（）&&isTxt（file2.getName（）））{

result.append（file2.getAbsolutePath（））.append（" 字节数:"）;

result.append（file2.length（））.append（"\n"）;

count ++;

}

}

return count;

}

/\*\*

\* 写入结果

\* @param result）

\* @param list\_filo

\*/

static void writeResult（StringBuffer result,String list\_filo,int count）{

File file = new File（list\_filo）;

File folder = file.getParentFile（）;

if （!folder.exists（）） {// 判断文件夹是否存在,不存在就建立个目录

folder.mkdirs（）;

}

try {

if （!file.exists（）） {

file.createNewFile（）;

}

PrintStream ps = new PrintStream（new FileOutputStream（file））;

ps.println（result.toString（））;// 往文件里写入字符串

ps.print（"扫描总数： "+count）;

ps.close（）;

} catch （FileNotFoundException e1） {

// TODO Auto-generated catch block

} catch （IOException e） {

// TODO Auto-generated catch block

}

}

/\*\*

\* 判断是否为txt文件

\* @param fileName

\* @return

\*/

static boolean isTxt（String fileName）{

//扩展名格式：

if （fileName.lastIndexOf（"."） >= 0） {

String extName = fileName.substring（fileName.lastIndexOf（"."））;

if（".TXT".equals（extName）||".txt".equals（extName））{

return true;

}

}

return false;

}

public static void main（String[] args） {

String path = "C:\\Users\\chen\\Desktop\\面试题";

String result = "C:\\Users\\chen\\Desktop\\result.txt";

ScanFile.intScanTextFiles（path, result）;

}

### （★）List、Set、Map是否继承自Collection接口？

List，Set是，Map不是

Collection是最基本的集合接口，一个Collection代表一组Object，即Collection的元素。一些Collection允许相同的元素而另一些不行。一些能排序而另一些不行。Java JDK不能提供直接继承自Collection的类，Java JDK提供的类都是继承自Collection的"子接口"，如:List和Set。

注意：Map没有继承Collection接口，Map提供key到value的映射。一个Map中不能包含相同key，每个key只能映射一个value。Map接口提供3种集合的视图，Map的内容可以被当做一组key集合，一组value集合，或者一组key-value映射。

### （★）阐述ArrayList、Vector、LinkedList的存储性能和特性。

答：

ArrayList 和Vector他们底层的实现都是一样的，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中的方法由于添加了synchronized修饰，因此Vector是线程安全的容器，但性能上较ArrayList差，因此已经是Java中的遗留容器。

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，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（Java早期的版本中提供的容器，除此之外，Hashtable、Dictionary、BitSet、Stack、Properties都是遗留容器），已经不推荐使用，但是由于ArrayList和LinkedListed都是非线程安全的，如果遇到多个线程操作同一个容器的场景，则可以通过工具类Collections中的synchronizedList方法将其转换成线程安全的容器后再使用（这是对装潢模式的应用，将已有对象传入另一个类的构造器中创建新的对象来增强实现）。

### （★★）Collection和Collections的区别？

1、java.util.Collection 是一个集合接口（集合类的一个顶级接口）。它提供了对集合对象进行基本操作的通用接口方法。Collection接口在Java 类库中有很多具体的实现。Collection接口的意义是为各种具体的集合提供了最大化的统一操作方式，其直接继承接口有List与Set。

2、Collections则是集合类的一个工具类/帮助类，其中提供了一系列静态方法，用于对集合中元素进行排序、搜索以及线程安全等各种操作。

详见https://www.cnblogs.com/cathyqq/p/5279859.html

### （★★）List、Map、Set三个接口存取元素时，各有什么特点？

存放时：

1.List以特定的索引（有顺序的存放）来存放元素,可以有重复的元素

2.Set存放元素是无序的,而且不可重复

3.Map保存键值对的映射,映射关系可以是一对一（键值）或者多对一,需要注意到的是：键无序不可重复,值可以重复

取出时:

（1）List取出元素for循环，foreach循环，Iterator迭代器迭代

（2）Set取出元素foreach循环，Iterator迭代器迭代

（3）Map取出元素需转换为Set，然后进行Iterator迭代器迭代，或转换为Entry对象进行Iterator迭代器迭代

### （★★）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？

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。

详见：https://www.cnblogs.com/bahcelor/p/6717839.html

### （★）LinkedList和ArrayList区别？

区别：

　 1.ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。

2.对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。

3.对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据。

详见：https://www.cnblogs.com/weiren123/p/6489963.html

### （★）ArrayLsit和LinkList的描述,下面说法错误的是?D

A LinkedList和ArrayList都实现的List接口

B ArrayList是可改变大小的数组,而LinkedList是双向链接串列

C LinkedLIst不支持高效的随机元素访问

D 在LinkedList的中间插入或删除一个元素意味着这个列表中生于的元素都会被移动;而ArrayList的中间插入或删除一个元素的开销是固定的

### （★★）ConcurrentHashMap即保证了线程安全，又兼顾了性能，请问它做了哪些方面的工作？

背景：

线程不安全的HashMap

因为多线程环境下，使用Hashmap进行put操作会引起死循环，导致CPU利用率接近100%，所以在并发情况下不能使用HashMap。

效率低下的HashTable容器

HashTable容器使用synchronized来保证线程安全，但在线程竞争激烈的情况下HashTable的效率非常低下。因为当一个线程访问HashTable的同步方法时，其他线程访问HashTable的同步方法时，可能会进入阻塞或轮询状态。如线程1使用put进行添加元素，线程2不但不能使用put方法添加元素，并且也不能使用get方法来获取元素，所以竞争越激烈效率越低。

锁分段技术

HashTable容器在竞争激烈的并发环境下表现出效率低下的原因，是因为所有访问HashTable的线程都必须竞争同一把锁，那假如容器里有多把锁，每一把锁用于锁容器其中一部分数据，那么当多线程访问容器里不同数据段的数据时，线程间就不会存在锁竞争，从而可以有效的提高并发访问效率，这就是ConcurrentHashMap所使用的锁分段技术，首先将数据分成一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据的时候，其他段的数据也能被其他线程访问。有些方法需要跨段，比如size（）和containsValue（），它们可能需要锁定整个表而而不仅仅是某个段，这需要按顺序锁定所有段，操作完毕后，又按顺序释放所有段的锁。这里“按顺序”是很重要的，否则极有可能出现死锁，在ConcurrentHashMap内部，段数组是final的，并且其成员变量实际上也是final的，但是，仅仅是将数组声明为final的并不保证数组成员也是final的，这需要实现上的保证。这可以确保不会出现死锁，因为获得锁的顺序是固定的。

ConcurrentHashMap是由Segment数组结构和HashEntry数组结构组成。Segment是一种可重入锁ReentrantLock，在ConcurrentHashMap里扮演锁的角色，HashEntry则用于存储键值对数据。一个ConcurrentHashMap里包含一个Segment数组，Segment的结构和HashMap类似，是一种数组和链表结构， 一个Segment里包含一个HashEntry数组，每个HashEntry是一个链表结构的元素， 每个Segment守护者一个HashEntry数组里的元素,当对HashEntry数组的数据进行修改时，必须首先获得它对应的Segment锁

### （★）下列说法正确的是（C）

A.LinkedList 继承自List

B.Abstractset继承自Set

C.HashSet继承自Set

D.WeakMap 维承自HashMap

备注

a选项linkedlist类是实现了List接口，而不是继承

b选项AbstractSet类实现Set接口

c选项HashSet继承 AbstractSet类，同时也实现set

d选项WeakMap是js里面的玩意儿吧，，不太懂

### （★★）HashMap 和Hashtable的区别，并谈谈HashMap的put方法

HashMap几乎可以等价于Hashtable，除了HashMap是非synchronized的，并可以接受null（HashMap可以接受为null的键值（key）和值（value），而Hashtable则不行）。

HashMap是非synchronized，而Hashtable是synchronized，这意味着Hashtable是线程安全的，多个线程可以共享一个Hashtable；而如果没有正确的同步的话，多个线程是不能共享HashMap的。Java 5提供了ConcurrentHashMap，它是HashTable的替代，比HashTable的扩展性更好。

另一个区别是HashMap的迭代器（Iterator）是fail-fast迭代器，而Hashtable的enumerator迭代器不是fail-fast的。所以当有其它线程改变了HashMap的结构（增加或者移除元素），将会抛出ConcurrentModificationException，但迭代器本身的remove（）方法移除元素则不会抛出ConcurrentModificationException异常。但这并不是一个一定发生的行为，要看JVM。这条同样也是Enumeration和Iterator的区别。

由于Hashtable是线程安全的也是synchronized，所以在单线程环境下它比HashMap要慢。如果你不需要同步，只需要单一线程，那么使用HashMap性能要好过Hashtable。

HashMap不能保证随着时间的推移Map中的元素次序是不变的。

介绍一下jdk1.8中的hashmap.put方法,

我们在使用put方法的时候会传进key和value参数

在我们将这两个参数传入后，

第一步，我们的put方法会去判断这个hashmap是否为null 或者长度是否为0，

若为null或者长度为0 则新建一个（这也是平时在编程过程中需要经常注意的细节），

第二步，就用到了我们这个key值啦，put方法会根据这个key计算hash码来得到数组的位置，

（这里需要解释一下，我们的hashmap默认是由一个数组加链表组成的）

得到位置后当然是继续判断这个数组下标的值是否为null，

为null 自然是直接插入我们的value值，else

第三步，判断key是否为null，当key！=null我们就可以覆盖value值，else if

第四步，判断数组后面跟着的这个链是否为树（TreeNode），是树呢，我们传入的值就会按照key，value的格式存入了，else

第五步，不是树就是链表，那么put方法就会遍历这个链表，

第六步，在遍历的时候呢我们会判断这个链表的长度是否大于8，大于呢就会将这个链表转换为树，再按照key，value的格式存入

第七步，小于则会判断链表中的key！=null，若kay！=null则覆盖，key==null我们的value就会插入

最后一步为判断扩容，当数组容量超过最大容量时就会扩容一倍（即二进制的进位）

下面po上源码

public V put（K key, V value） {

return putVal（hash（key）, key, value, false, true）;

}

final V putVal（int hash, K key, V value, boolean onlyIfAbsent,

boolean evict） {

Node<K,V>[] tab; Node<K,V> p; int n, i;

if （（tab = table） == null || （n = tab.length） == 0）

n = （tab = resize（））.length;

if （（p = tab[i = （n - 1） & hash]） == null）

tab[i] = newNode（hash, key, value, null）;

else {

Node<K,V> e; K k;

if （p.hash == hash &&

（（k = p.key） == key || （key != null && key.equals（k））））

e = p;

else if （p instanceof TreeNode）

e = （（TreeNode<K,V>）p）.putTreeVal（this, tab, hash, key, value）;

else {

for （int binCount = 0; ; ++binCount） {

if （（e = p.next） == null） {

p.next = newNode（hash, key, value, null）;

if （binCount >= TREEIFY\_THRESHOLD - 1） // -1 for 1st

treeifyBin（tab, hash）;

break;

}

if （e.hash == hash &&

（（k = e.key） == key || （key != null && key.equals（k））））

break;

p = e;

}

}

if （e != null） { // existing mapping for key

V oldValue = e.value;

if （!onlyIfAbsent || oldValue == null）

e.value = value;

afterNodeAccess（e）;

return oldValue;

}

}

++modCount;

if （++size > threshold）

resize（）;

afterNodeInsertion（evict）;

return null;

}

### （★★）写一段代码遍历ArrayList时移除一个元素？

public Text{

public static void main（String[] args） {

ArrayList<String> aList = new ArrayList<String>（）;

aList.add（"a"）;

aList.add（"ab"）;

aList.add（"abc"）;

aList.add（"abcr"）;

aList.add（"abc"）;

aList.add（"abcf"）;

aList.add（"abc"）;

aList.add（"abdc"）;

for（int i = 0;i < aList.size（）;i++）{

if（aList.get（i）.equals（"abc"））{

aList.remove（i）;

}

}

System.out.println（aList）;

}

}

### （★★）阐述ArrayList、Vector、LinkedList的存储性能和特性。

ArrayList 和Vector他们底层的实现都是一样的，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中的方法由于添加了synchronized修饰，因此Vector是线程安全的容器，但性能上较ArrayList差，因此已经是Java中的遗留容器。

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，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（Java早期的版本中提供的容器，除此之外，Hashtable、Dictionary、BitSet、Stack、Properties都是遗留容器），已经不推荐使用，但是由于ArrayList和LinkedListed都是非线程安全的，如果遇到多个线程操作同一个容器的场景，则可以通过工具类Collections中的synchronizedList方法将其转换成线程安全的容器后再使用（这是对装潢模式的应用，将已有对象传入另一个类的构造器中创建新的对象来增强实现）。

### （★★）下面哪个Map最适合用来实现LRU Cache（A）

A Hashtable B TreeMap C HashMap D dentityHashMap

### （★★）java集合类框架的基本接口有哪些（A）

A、Collection B、ArrayList C、HashMap D、Array

### （★★）ArrayList list = new ArrayList（20）中的list扩充几次（）

ArrayList list=new ArrayList（）; 这种是默认创建大小为10的数组，每次扩容大小为1.5倍

ArrayList list=new ArrayList（20）; 这种是指定数组大小的创建，创建时直接分配其大小，没有扩充。

所以，扩充为0次

### （★★）索引的作用？和它的优缺点是什么？

索引就一种特殊的查询表，数据库的搜索引擎可以利用它加速对数据的检索。它很类似与现实生活中书的目录，不需要查询整本书内容就可以找到想要的数据。索引可以是唯一的，创建索引允许指定单个列或者是多个列。缺点是它减慢了数据录入的速度，同时也增加了数据库的尺寸大小。

## 4. 线程部分

### （★）创建线程有几种不同的方式？你喜欢哪一种？为什么？

答:有两种方式可以用来创建线程：

（1）继承Thread类

（2）实现Runnable接口

（3）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。

答:

1. 新建（ new ）：新创建了一个线程对象。

2. 可运行（ runnable ）：线程对象创建后，其他线程（比如 main 线程）调用了该对象 的 start （）方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获 取 cpu 的使用权 。

3. 运行（ running ）：可运行状态（ runnable ）的线程获得了 cpu 时间片（ timeslice ） ，执行程序代码。

4. 阻塞（ block ）：阻塞状态是指线程因为某种原因放弃了 cpu 使用权，也即让出了 cpu timeslice ，暂时停止运行。直到线程进入可运行（ runnable ）状态，才有 机会再次获得 cpu timeslice 转到运行（ running ）状态。阻塞的情况分三种：

（一）. 等待阻塞：运行（ running ）的线程执行 o . wait （）方法， JVM 会把该线程放 入等待队列（ waitting queue ）中。

（二）. 同步阻塞：运行（ running ）的线程在获取对象的同步锁时，若该同步锁 被别的线程占用，则 JVM 会把该线程放入锁池（ lock pool ）中。

（三）. 其他阻塞: 运行（ running ）的线程执行 Thread . sleep （ long ms ）或 t . join （）方法，或者发出了 I / O 请求时， JVM 会把该线程置为阻塞状态。当 sleep （）状态超时、 join （）等待线程终止或者超时、或者 I / O 处理完毕时，线程重新转入可运行（ runnable ）状态。

5. 死亡（ dead ）：线程 run （）、 main （） 方法执行结束，或者因异常退出了 run （）方法，则该线程结束生命周期。死亡的线程不可再次复生。

### （★）同步方法和同步代码块的区别是什么？

答:区别：

1.同步方法默认用this或者当前类class对象作为锁；

2.同步代码块可以选择以什么来加锁，比同步方法要更细颗粒度，我们可以选择只同步会发生同步问题的部分代码而不是整个方法；

### （★★）什么是死锁（deadlock）？如何确保N个线程可以访问N个资源同时又不导致死锁

答:两个线程或两个以上线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。结果就是这些线程都陷入了无限的等待中.

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。

### （★★）多线程的优缺点是什么?

答:**优点：**

（1）多线程技术使程序的响应速度更快 ,因为用户界面可以在进行其它工作的同时一直处于活动状态;

（2）当前没有进行处理的任务时可以将处理器时间让给其它任务;

（3）占用大量处理时间的任务可以定期将处理器时间让给其它任务;

（4）可以随时停止任务;

（5）可以分别设置各个任务的优先级以优化性能。

是否需要创建多个线程取决于各种因素。在以下情况下,最适合采用多线程处理:

（1）耗时或大量占用处理器的任务阻塞用户界面操作;

（2）各个任务必须等待外部资源 （如远程文件或 Internet连接）。

同样的 ,多线程也存在许多缺点 ,在考虑多线程时需要进行充分的考虑。

**多线程的主要缺点包括:**

（1）等候使用共享资源时造成程序的运行速度变慢。这些共享资源主要是独占性的资源 ,如打印机等。

（2）对线程进行管理要求额外的 CPU开销。线程的使用会给系统带来上下文切换的额外负担。当这种负担超过一定程度时,多线程的特点主要表现在其缺点上,比如用独立的线程来更新数组内每个元素。

（3）线程的死锁。即较长时间的等待或资源竞争以及死锁等多线程症状。

（4）对公有变量的同时读或写。当多个线程需要对公有变量进行写操作时,后一个线程往往会修改掉前一个线程存放的数据,从而使前一个线程的参数被修改;另外 ,当公用变量的读写操作是非原子性时,在不同的机器上,中断时间的不确定性,会导致数据在一个线程内的操作产生错误,从而产生莫名其妙的错误,而这种错误是程序员无法预知的。

### （★）sleep（）和wait（）的区别?

答:

1、这两个方法来自不同的类分别是Thread和Object

2、最主要是sleep方法没有释放锁，而 wait 方法释放了锁，使得其他线程可以使用同步控制块或者方法。

3、wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用（使用范围）

1. sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常

### （★★）什么是线程同步 如何实现线程同步

答:当使用多个线程来访问同一个数据时，非常容易出现线程安全问题（比如多个线程都在操作同一数据导致数据不一致）,所以我们用同步机制来解决这些问题。

实现同步机制有两个方法：

1.同步代码块：

synchronized（同一个数据）{} 同一个数据：就是N条线程同时访问一个数据。

2.同步方法：

public synchronized 数据返回类型 方法名（）{}

就是使用 synchronized 来修饰某个方法，则该方法称为同步方法。对于同步方法而言，无需显示指定同步监视器，同步方法的同步监视器是 this 也就是该对象的本身（这里指的对象本身有点含糊，其实就是调用该同步方法的对象）通过使用同步方法，可非常方便的将某类变成线程安全的类，具有如下特征：

1，该类的对象可以被多个线程安全的访问。

2，每个线程调用该对象的任意方法之后，都将得到正确的结果。

3，每个线程调用该对象的任意方法之后，该对象状态依然保持合理状态。

注：synchronized关键字可以修饰方法，也可以修饰代码块，但不能修饰构造器，属性等。

实现同步机制注意以下几点： 安全性高，性能低，在多线程用。性能高，安全性低，在单线程用。

1. 不要对线程安全类的所有方法都进行同步，只对那些会改变共享资源方法的进行同步。

2，如果可变类有两种运行环境，当线程环境和多线程环境则应该为该可变类提供两种版本：线程安全版本和线程不安全版本（没有同步方法和同步块）。在单线程中环境中，使用线程不安全版本以保证性能，在多线程中使用线程安全版本.

### （★）sleep（） 和wait（） 有什么区别? 【基础】

答：sleep 是线程类（Thread）的方法，导致此线程暂停执行指定时间，给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。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，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运行状态。

### （★）当一个线程进入一个对象的一个synchronized 方法后，其它线程是否可进入此对象的其它方法? 【基础】

答：其它线程只能访问该对象的其它非同步方法，同步方法则不能进入。

### （★）请说出你所知道的线程同步的方法。【基础】

答：wait（）:使一个线程处于等待状态，并且释放所持有的对象的lock；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，注意的是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，而且不是按优先级；

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。

### （★）多线程有几种实现方法,都是什么?同步有几种实现方法,都是什么? 【基础】

答：多线程有两种实现方法，分别是继承Thread 类与实现Runnable 接口,同步的实现方面有两种，分别是synchronized,wait 与notify。

### （★）同步和异步有何异同，在什么情况下分别使用他们？举例说明。【基础】

答：如果数据将在线程间共享。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，那么这些数据就是共享数据，必须进行同步存取。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。

### （★）启动一个线程是用run（）还是start（）?【基础】

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（）方法可以产生必须退出的标志来停止一个线程。

### （★）线程的基本概念、线程的基本状态以及状态之间的关系？【基础】

答：线程指在程序执行过程中，能够执行程序代码的一个执行单位，每个程序至少都有一个线程，也就是程序本身；Java 中的线程有四种状态分别是：运行、就绪、挂起、结束。

### （★★）简述synchronized 和java.util.concurrent.locks.Lock 的异同【中等难度】

答：主要相同点：Lock 能完成synchronized 所实现的所有功能；主要不同点：Lock 有比synchronized 更精确的线程语义和更好的性能。synchronized 会自动释放锁，而Lock 一定要求程序员手工释放，并且必须在finally 从句中释放。

### （★★）java 中有几种方法可以实现一个线程？用什么关键字修饰同步方法?stop（）和suspend（）方法为何不推荐使用？【中等难度】

答：有两种实现方法，分别是继承Thread 类与实现Runnable 接口；用synchronized 关键字修饰同步方法；反对使用stop（），是因为它不安全。它会解除由线程获取的所有锁定，而且如果对象处于一种不连贯状态，那么其他线程能在那种状态下检查和修改它们。结果很难检查出真正的问题所在；suspend（）方法容易发生死锁。调用suspend（）的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被“挂起”的线程恢复运行。对任何线程来说，如果它们想恢复目标线程，同时又试图使用任何一个锁定的资源，就会造成死锁。故不应该使用suspend（），而应在自己的Thread 类中置入一个标志，指出线程应该活动还是挂起。若标志指出线程应该挂起，便用wait（）命其进入等待状态。若标志指出线程应当恢复，则用一个notify（）重新启动线程。

### （★★）设计4 个线程，其中两个线程每次对j 增加1，另两个线程对j 每次减少1；写出程序。【中等难度】

答：以下程序使用内部类实现线程，对j 增减的时候没有考虑顺序问题：

public class TestThread {

private int j;

public TestThread（int j） {

this.j = j;

}

private synchronized void inc（）{

j++;

System.out.println（j + "--Inc--" +

Thread.currentThread（）.getName（））;

}

private synchronized void dec（）{

j--;

System.out.println（j + "--Dec--" +

Thread.currentThread（）.getName（））;

}

public void run（） {

（new Dec（））.start（）;

new Thread（new Inc（））.start（）;

（new Dec（））.start（）;

new Thread（new Inc（））.start（）;

}

class Dec extends Thread {

public void run（） {

for（int i=0; i<100; i++）{

dec（）;

}

}

}

class Inc implements Runnable {

public void run（） {

for（int i=0; i<100; i++）{

inc（）;

}

}

}

public static void main（String[] args） {

（new TestThread（5））.run（）;

}

}

### （★★）Thread类的sleep（）方法和对象的wait（）方法都可以让线程暂停执行，它们有什么区别?

sleep（）方法（休眠）是线程类（Thread）的静态方法，调用此方法会让当前线程暂停执行指定的时间，

将执行机会（CPU）让给其他线程，但是对象的锁依然保持，因此休眠时间结束后会自动恢复

wait（）是Object类的方法，调用对象的wait（）方法导致当前线程放弃对象的锁（线程暂停执行），进入对象的等待池（wait pool），只有调用对象的notify（）方法（或notifyAll（）方法）时才能唤醒等待池中的线程进入等锁池（lockpool），如果线程重新获得对象的锁就可以进入就绪状态

### （★★）线程的sleep（）方法和yield（）方法有什么区别？

线程睡眠（sleep）

让当前正在执行的线程暂停一段时间，并进入阻塞状态，则可以通过调用Thread类的静态sleep（）方法来实现。当前线程调用sleep（）方法进入阻塞状态后，在其睡眠时间内，该线程不会获得执行的机会，而其它任何优先级的线程都可以得到执行的机会，即使系统中没有其它可执行的线程，处于sleep（）的线程也不会执行，sleep（）是用来暂停线程的执行。

线程让步（yield）

yield（）方法是一个和sleep（）方法有点相似的方法，它也是Thread类提供的一个静态方法。可以让当前正在执行的线程暂停，但它不会阻塞该线程，只是将该线程转入就绪状态。yeild（）只是让当前线程暂停一下，让系统的线程调度器重新调度一次，完全可能的情况是：当某个线程调用了yield（）线程暂停之后，线程调度器又将其调度出来重新执行。

当某个线程调用了yield（）方法暂停之后，只有优先级与当前线程相同，或者优先级比当前线程更高的处于就绪状态的线程才会获得执行机会。

sleep（）方法的声明抛出了InterruptedException异常，所以调用sleep（）方法时要么捕捉异常，要么抛出该异常。

sleep（）方法比yield（）方法具有更好的可移动性，所以建议不要使用yield（）方法来控制并发线程的执行

### （★★）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B？

不能，一个对象的一个synchronized方法只能由一个线程访问。

详见：https://blog.csdn.net/dabing69221/article/details/12502027

### （★★）请说出与线程同步以及线程调度相关的方法。

-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，调用此方法要处理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（）：唤醒所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态；

### （★★）编写多线程程序有几种实现方式？

一种是继承Thread类；另一种是实现Runnable接口。

两种方式都要通过重写run（）方法来定义线程的行为，推荐使用后者，因为Java中的继承是单继承，一个类有一个父类，如果继承了Thread类就无法再继承其他类了，显然使用Runnable接口更为灵活

### （★★）synchronized关键字的用法？

synchronized 关键字，它包括两种用法：synchronized 方法和 synchronized 块。

详见https://www.cnblogs.com/yaobolove/p/6237556.html

### （★★）举例说明同步和异步。

同步:发送一个请求,等待返回,然后再发送下一个请求

异步:发送一个请求,不等待返回,随时可以再发送下一个请求

比如发短信，就是一个异步例子。发起者不关心接收者的状态。不需要等待接收者的返回信息，则可以进行下一次发送。

电话，就是一个同步例子。发起者需要等待接收者，接通电话后，通信才开始。需要等待接收者的返回信息

### （★★）启动一个线程是调用run（）还是start（）方法？

start（）方法来启动线程，真正实现了多线程运行，这时无需等待run方法体代码执行完毕而直接继续执行下面的代码

run（）方法当作普通方法的方式调用，程序还是要顺序执行，还是要等待run方法体执行完毕后才可继续执行下面的代码

### （★★）什么是线程池（thread pool）？

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。

详见https://www.jianshu.com/p/916ea29ff3b9

### （★★）线程的基本状态以及状态之间的关系？

基本状态：就绪、阻塞和运行三种基本状态。

就绪状态，指线程具备运行的所有条件，逻辑上可以运行，在等待处理机；

运行状态，指线程占有处理机正在运行；

阻塞状态，指线程在等待一个事件（如信号量），逻辑上不可执行。

### （★★）简述synchronized 和java.util.concurrent.locks.Lock的异同？

1.Lock能完成几乎所有synchronized的功能，并有一些后者不具备的功能，如锁投票、定时锁等候、可中断锁等候等

2.synchronized 是Java 语言层面的，是内置的关键字；Lock 则是JDK 5中出现的一个包，在使用时，synchronized 同步的代码块可以由JVM自动释放；Lock 需要程序员在finally块中手工释放，如果不释放，可能会引起难以预料的后果（在多线程环境中）。

详见：https://blog.csdn.net/hintcnuie/article/details/11022049

### （★★）如何通过反射调用对象的方法？

1.通过Class.forName （“包名+方法的类名”）拿到方法的对象

如：Class<?> clazz = Class.forName（"ReflectionTest"）;

也可以通过实例，获取方法的对象类，

如：String str = "test , reflection";

Class<?> clazz = str.getClass（）;

2.明确反射方法名称 ;

3.明确方法的参数类型，就可以拿到对象的方法。

如：Method method = clazz.getMethod（"test",String.class,int.class）；

### （★★）简述一下面向对象的”六原则一法则”。

1、单一职责原则：一个类只做它该做的事情。（单一职责原则想表达的就是"高内聚"，写代码最终极的原则只有六个字"高内聚、低耦合"）

2、开闭原则：软件实体应当对扩展开放，对修改关闭。（在理想的状态下，当我们需要为一个软件系统增加新功能时，只需要从原来的系统派生出一些新类就可以，不需要修改原来的任何一行代码。要做到开闭有两个要点：①抽象是关键，一个系统中如果没有抽象类或接口系统就没有扩展点;②封装可变性，将系统中的各种可变因素封装到一个继承结构中，如果多个可变因素混杂在一起，系统将变得复杂而换乱）

3、依赖倒转原则：面向接口编程。（该原则说得直白和具体一些就是声明方法的参数类型、方法的返回类型、变量的引用类型时，尽可能使用抽象类型而不用具体类型，因为抽象类型可以被它的任何一个子类型所替代）

4、里氏替换原则：任何时候都可以用子类型替换掉父类型。（子类一定是增加父类的能力而不是减少父类的能力，因为子类比父类的能力更多，把能力多的对象当成能力少的对象来用当然没有任何问题。）

5、接口隔离原则：接口要小而专，绝不能大而全。（臃肿的接口是对接口的污染，既然接口表示能力，那么一个接口只应该描述一种能力，接口也应该是高度内聚的。Java中的接口代表能力、代表约定、代表角色，能否正确的使用接口一定是编程水平高低的重要标识。）

6、合成聚合复用原则：优先使用聚合或合成关系复用代码。（通过继承来复用代码是面向对象程序设计中被滥用得最多的东西，记住：任何时候都不要继承工具类，工具是可以拥有并可以使用的，而不是拿来继承的。）

迪米特法则：迪米特法则又叫最少知识原则，一个对象应当对其他对象有尽可能少的了解。（迪米特法则简单的说就是如何做到"低耦合"，门面模式和调停者模式就是对迪米特法则的践行。Java Web开发中作为前端控制器的Servlet或Filter不就是一个门面吗，浏览器对服务器的运作方式一无所知，但是通过前端控制器就能够根据你的请求得到相应的服务。调停者模式也可以举一个简单的例子来说明，例如一台计算机，CPU、内存、硬盘、显卡、声卡各种设备需要相互配合才能很好的工作。迪米特法则用通俗的话来将就是不要和陌生人打交道，如果真的需要，找一个自己的朋友，让他替你和陌生人打交道。）

### （★）下列哪些情况可以终止当前线程的运行?

A 当一个优先级高的线程进入就绪状态时

B 当线程调用sleep（）方法时

C 当创建一个新线程时

D 抛出一个异常时

A是抢占,不是终止

B是终止

C是暂停,不是终止

D和A的意思差不多

### （★）下列关于JAVA多线程的叙述正确的是（）

A调用start（）方法和run（）都可以启动一个线程

B CyclicBarrier和CountDownLatch都可以让一组线程等待其他线程

C Callable类的call（）方法可以返回值和抛出异常

D 新建的线程调用start（）方法就能立即进行运行状态

A.start（）方法来启动线程，真正实现了多线程运行，调用了run（）方法；run（）方法当作普通方法的方式调用。

B.CyclicBarrier让一组线程等待其他线程；CountDownLatch让一组线程等待某个事件发生。

C Callable能够抛出checked exception。

D.start（）方法让thread进去可运行状态（runnable），等待获取CPU的使用权。

### （★）实现线程有几种方式？

答：http://www.cnblogs.com/felixzh/p/6036074.html

### （★）多线程轮流顺序打印，a，b，c三个线程按顺序打印数字1-96，例

a 1

b 2

c 3

a 4

b 5

c 6

### （★★）请写出多线程代码,使用Thread或者runnable,并说出两种的区别

答：

**Thread ：**

public static void main（String[] args） {

//线程使用示例一：

new Thread（） {

public void run（） {

while （true） {

try {

System.out.println（"线程输出"）;

//休眠两秒

Thread.sleep（2 \* 1000）;

} catch （InterruptedException e） {

e.printStackTrace（）;

}

}

};

}.start（）;

}

**Runnable：**

Thread t1 = new Thread（new Runnable（）{

public void run（）{

System.out.println（"Mythread 线程t1"）;

while（true）{

Singleton3 single3 = Singleton3.getInstance（null）;

System.out.println（"线程t1 >> " + single3.toString（））;

try {

Thread.sleep（2000）;

} catch （InterruptedException e） {

// TODO Auto-generated catch block

e.printStackTrace（）;

}

}

}

}）;

t1.start（）;

区别:

Runnable 是接口。

Thread 是类，且实现了Runnable接口。

在程序开发中只要是多线程肯定永远以实现Runnable接口为主。

实现Runnable接口相比继承Thread类有如下好处：

1、避免继承的局限，一个类可以继承多个接口。

2、适合于资源的共享。

### （★）下面哪些是Thread类的方法（ABD）

1. start（） B.run（） C.exit（） D.getPriority（）

### （★★）谈读并发编程中的锁。数据共享以及并发模型。

并发编程中的锁：

1.内置锁

　　synchronized，这个不用说了，作用就是实现互斥之外，还有内存可见性，复合操作原子性等作用；一个线程拥有某个对象的synchronized锁，该线程调用wait方法后，会释放对象的锁；

2.ReentrantLock

　　ReentrantLock和synchronized在并发性和内存语义方面都相同，但是又多了锁投票，定时锁等候和中断锁等候

　　线程A和B都想获得对象O的锁，A获得锁后，如果是synchronized，线程B会一直等待直到获得锁为止；

　　而ReentrantLock会等待一段时间后停止等待；

　　ReentrantLock获得锁的三种方式

　　lock（）方法：如果获得了锁，立即返回，否则处于休眠状态，直到获得锁为止；

　　tryLock（）方法：如果获得了锁，返回true，否则，返回false；

　　tryLock（long timeout, TimeUnit unit）: 等待timeout时间后还没有获得锁，则返回false，否则，返回true；

　　lockInterruptibly（）：如果获得了锁，立即返回；如果没有获得锁，该线程处于休眠状态直到获得锁，或者该线程被中断；

　　newCondition（）：作用和synchronized里的wait/notify方法很像；

　　Condition里面的await（）方法和synchronized的wait（）方法一样，Condition的signal（）和notify（）方法类似；

　　ReentrantLock和synchronized的区别：synchronized是在JVM层面实现的，当运行出错时，JVM会自动终止程序运行；而ReentrantLock是在JDK层面上实现的，需要手动释放锁，最后要在finally里手动释放；

　　结论：synchronized在竞争不是很激烈的时候适合使用，性能比ReentrantLock好一点点；但是竞争很激烈时，synchronized性能下降很厉害，而ReentrantLock能保持不变。

3.自旋锁

　　3.1为什么会出现自旋锁？

　　操作系统CPU进行线程的上下文替换需要时间比较长，所以我们想办法减少线程的上下文切换次数；自旋锁一般用于多处理器，线程A、B在不同的处理器上被执行，且都需要对象O的锁，现在O的锁被B占着，A请求O的锁，A不会进入阻塞状态，而是执行空循环等待B释放O的锁。

　　3.1使用场景？

可知，如果B占用O很短，那么自旋锁效率很高，如果B会占用很长时间，或者该锁竞争激烈，效果不好。

### （★★）请举例说明乐观锁和悲观锁的使用场景

定义：

悲观锁（Pessimistic Lock）：

每次获取数据的时候，都会担心数据被修改，所以每次获取数据的时候都会进行加锁，确保在自己使用的过程中数据不会被别人修改，使用完成后进行数据解锁。由于数据进行加锁，期间对该数据进行读写的其他线程都会进行等待。

乐观锁（Optimistic Lock）：

每次获取数据的时候，都不会担心数据被修改，所以每次获取数据的时候都不会进行加锁，但是在更新数据的时候需要判断该数据是否被别人修改过。如果数据被其他线程修改，则不进行数据更新，如果数据没有被其他线程修改，则进行数据更新。由于数据没有进行加锁，期间该数据可以被其他线程进行读写操作。

适用场景：

悲观锁：比较适合写入操作比较频繁的场景，如果出现大量的读取操作，每次读取的时候都会进行加锁，这样会增加大量的锁的开销，降低了系统的吞吐量。

乐观锁：比较适合读取操作比较频繁的场景，如果出现大量的写入操作，数据发生冲突的可能性就会增大，为了保证数据的一致性，应用层需要不断的重新获取数据，这样会增加大量的查询操作，降低了系统的吞吐量。

总结：两种所各有优缺点，读取频繁使用乐观锁，写入频繁使用悲观锁。

像乐观锁适用于写比较少的情况下，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但如果经常产生冲突，上层应用会不断的进行retry，这样反倒是降低了性能，所以这种情况下用悲观锁就比较合适,之所以用悲观锁就是因为两个用户更新同一条数据的概率高，也就是冲突比较严重的情况下，所以才用悲观锁.

悲观锁比较适合强一致性的场景，但效率比较低，特别是读的并发低。乐观锁则适用于读多写少，并发冲突少的场景。

### （★★）随机排序0-99数字存入list，不重复，并获取在表中 下标

### （★★）JAVA中多线程的实现方式有几种？分别是什么、怎么样保证线程同步？

两种方法：

1、继承Thread类

2、实现Runnable接口

线程同步的方法：

1.同步方法

2.同步代码块

3.使用特殊域变量（volatile）实现线程同步

4.使用重入锁实现线程同步

5.使用局部变量实现线程同步

### （★★）创建两个线程、每个线程打印出线程名字再睡眠，给其他线程执行的机会，主线程也要打印出线程名字后睡眠，每个线程前后睡眠5次？

### （★★）有关线程的描述哪个是正确的？

A:一旦一个线程别创建，他就立即开始运行

B:当一个线程因为抢先机制而停止运行，它被放在可运行队列的前面

C:使用start（）方法可以使一个线程成为可运行的，但他不一定立即开始运行

D:一个线程可能因为不同的原因停止（cease）并进入就绪状态

B

解析：

一个新创建的线程并不是自动开始运行的，必须调用它的start（ ）方法使之将线程放入可运行态。不同优先级的线程间是抢先式的，而同级线程间是轮转式的。主线程具有默认优先级。Java中线程的优先级从低到高以整数1-10表示，分10级。

### （★★）10个线程和2个线程的同步代码，哪个更容易些？

从写代码的角度来说，两者的复杂度是相同的，因为同步代码与线程数量是相互独立的。但是同步策略的选择依赖于线程的数量，因为越多的线程意味着更大的竞争，所以你需要利用同步技术，如锁分离，这要求更复杂的代码和专业知识。

## 5.IO流及Socket 部分

### （★）io流 按照流向分 按照处理内容分? 字节流都有哪些方法?

答:【流的分类详细】

[ 输入流---输出流 ]

按照流的流向，可以将流分为输入流和输出流。

输入流：从其中读取数据

输出流：向其种写入数据

注意：划分输入/输出流是从程序运行时的所在的内存的角度来划分的。

输入流常用的基础类：InputStream和Reader

输出流常用的基础类：OutputStream和Writer

注意：这些积累都是抽象类，无法直接实例化。

[ 字节流---字符流 ]

字节流和字符流的内的方法几乎一样，区别仅仅是其操作的数据单元大小不同，

字节流的数据操作单元： 8位的字节

字符流的数据操作单元： 16位的字节

字节流常用的基础类：InputStream和OutputStream

字符流常用的基础类：Reader和Writer

### （★）什么是java 序列化，如何实现java 序列化？【基础】

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 中有几种类型的流？JDK 为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类？【基础】

答：字节流，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。

### （★★）文件和目录（IO）操作：

1）如何列出某个目录下的所有文件？

2）如何列出某个目录下的所有子目录？

3）如何判断一个文件或目录是否存在？

4）如何读写文件？【基础】

答：1）示例代码如下:

File file = new File（"e:\\总结"）;

File[] files = file.listFiles（）;

for（int i=0; i<files.length; i++）{

if（files[i].isFile（）） System.out.println（files[i]）;

}

2）示例代码如下:

File file = new File（"e:\\总结"）;

File[] files = file.listFiles（）;

for（int i=0; i<files.length; i++）{

if（files[i].isDirectory（）） System.out.println（files[i]）;

}

3）创建File 对象,调用其exsit（）方法即可返回是否存在,如:

System.out.println（new File（"d:\\t.txt"）.exists（））;

4）示例代码如下:

//读文件:

FileInputStream fin = new FileInputStream（"e:\\tt.txt"）;

byte[] bs = new byte[100];

while（true）{

int len = fin.read（bs）;

if（len <= 0） break;

System.out.print（new String（bs,0,len））;

}

fin.close（）;

//写文件:

FileWriter fw = new FileWriter（"e:\\test.txt"）;

fw.write（"hello world!" + System.getProperty（"line.separator"））;

fw.write（"你好！北京！"）;

fw.close（）;

### （★）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。【基础】

答：代码如下：

public int countWords（String file, String find） throws Exception{

int count = 0;

Reader in = new FileReader（file）;

int c;

while （（c = in.read（）） != -1） {

while （c == find.charAt（0）） {

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

c = in.read（）;

if （c != find.charAt（i））

break;

if （i == find.length（） - 1）

count++;

}

}

}

return count;

}

### （★）Java 的通信编程，编程题（或问答），用JAVA SOCKET 编程，读服务器几个字符，再写入本地显示？【基础】

答：Server 端程序:

package test;

import java.net.\*;

import java.io.\*;

public class Server{

private ServerSocket ss;

private Socket socket;

private BufferedReader in;

private PrintWriter out;

public Server（）{

try {

ss=new ServerSocket（10000）;

while（true）{

socket = ss.accept（）;

String RemoteIP = socket.getInetAddress（）

.getHostAddress（）;

String RemotePort = ":"+socket.getLocalPort（）;

System.out.println（"A client come in!IP:"

+ RemoteIP+RemotePort）;

in = new BufferedReader（new

InputStreamReader（socket.getInputStream（）））;

String line = in.readLine（）;

System.out.println（"Cleint send is :" + line）;

out = new PrintWriter（socket.getOutputStream（）,true）;

out.println（"Your Message Received!"）;

out.close（）;

in.close（）;

socket.close（）;

}

}catch （IOException e）{

out.println（"wrong"）;

}

}

public static void main（String[] args）{

new Server（）;

}

}

Client 端程序:

package test;

import java.io.\*;

import java.net.\*;

public class Client {

Socket socket;

BufferedReader in;

PrintWriter out;

public Client（）{

try {

System.out.println（"Try to Connect to 127.0.0.1:10000"）;

socket = new Socket（"127.0.0.1",10000）;

System.out.println（"The Server Connected!"）;

System.out.println（"Please enter some Character:"）;

BufferedReader line = new BufferedReader（new

InputStreamReader（System.in））;

out = new PrintWriter（socket.getOutputStream（）,true）;

out.println（line.readLine（））;

in = new BufferedReader（

new InputStreamReader（socket.getInputStream（）））;

System.out.println（in.readLine（））;

out.close（）;

in.close（）;

socket.close（）;

}catch（IOException e）{

out.println（"Wrong"）;

}

}

public static void main（String[] args） {

new Client（）;

}

}

### （★★）已知如下代码

File f = new File（"myfile.dat"）;

FileInputStream istream = new FileInputStream（f）; 如何从流中读取数据。

a） byte[] buff = new byte[100];

b） istream.read（buff）;

### （★）Java中有几种类型的流？

java中有两种类型的流：字节流（byte）和字符流（char）。

字节流（byte）:InputStream ,outputStream

字符流（char）:InputStreamReader,outputStream

输出流：outputStream outputStreamWrite

输入流：InputStream InputStreamreader

### （★★）什么是I/O流，有什么作用？

I/O是 input/output的缩写，即输入输出端口。每个设备都会有一个专用的I/O地址，用来处理自己的输入输出信息。

分类：根据操作分为：输入流和输出流

根据类型分为：字节流和字符流

字节流：可以操作任何数据的传入和传出，因此传输的文件可以是：音频，视频，图片，文本

字符流：只能传输文本类型的数据（人可以认识的字体），因此传输的文件只可以是：文本，但其有其存在的价值是：在字节流的基础上融入了编码的操作

字节流的父类（抽象的abstract）：InputStream 字节输入流（读）

OutputStream 字节输出流（写）

字符流的父类（抽象的abstract）：Reader 字符输入流

Writer字符输出流

### （★★）Java中如何实现序列化，有什么意义？

如何实现：

1）让类实现Serializable接口,该接口是一个标志性接口,标注该类对象是可被序列

2）然后使用一个输出流来构造一个对象输出流并通过writeObect（Obejct）方法就可以将实现对象写出

3）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readObeject方法从流中读取对象

意义：

1）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,也可以将流化后的对象传输与网络之间;

2）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）

3）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

## 6.其他

### （★★★）旧的Java内存模型

Java使用的是共享内存的并发模型，在线程之间共享变量。Java语言定义了线程模型规范，通过内存模型控制线程与变量的交互，从而实现Java线程之间的通信。在JDK5之前，Java一直使用的是旧内存模型。如图1所示。变量保存在由所有线程共享的主内存中，主内存中的变量称为mastingcopy。每个线程都有一个工作内存，它保存变量的workingcopy。旧的内存模型定义了若干规则，通过这些规则来保证线程何时将主内存中的mastingcopy传送到线程的工作内存中；以及线程何时将工作内存中的workingcopy传送回主内存。旧内存模型使用8个操作来定义线程可以执行的动作。

* read（读）操作：主内存把mastingcopy传送到线程的工作内存，以供后面的load操作使用。
* load（装载）操作：线程将由read操作从主内存传送过来的值，放入工作内存中。
* use（使用）操作：线程将变量的workingcopy传送到线程执行引擎。
* assign（赋值）操作：线程将变量值从线程执行引擎传送到线程的工作内存中。
* store（存储）操作：线程将变量的workingcopy传送到主内存，供后面的write操作使用。
* write（写）操作：主内存将由store操作传送过来的值，放入主内存中。
* lock（锁定）操作：线程获得指定对象的锁。
* unlock（解锁）操作：线程释放指定对象的锁。

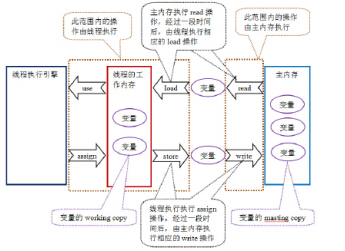


图1 Java旧内存模型

这里的关键是，由于read操作是由主内存执行，而对应的load是由线程执行，read操作和load操作之间是松散耦合的。也就是说，主内存和线程工作内存之间的变量传递是松散耦合的。同样，由于store操作是由线程执行，而对应的write是由主内存执行，store操作和write操作之间是松散耦合的。也就是说，线程工作内存和主内存之间的变量传递是松散耦合的。旧Java内存模型对Java实现如何执行变量的读/写，加锁/解锁，以及volatile变量的读/写，定义了非常严格的规则。这些规则非常复杂，具体详情请参考《JVM规范》，这里就不赘述了。旧Java内存模型通过这些复杂的规则，来保证多线程程序的线程之间，可以可靠地传递共享变量，从而保证多线程程序的正确性。

### （★★★）新的Java内存模型

从JDK5开始，Java使用新的内存模型，新内存模型完全抛弃了旧内存模型的主内存和工作内存的概念，也抛弃了旧内存模型的8个内存操作。也就是说，新内存模型完全是重新设计的。

新内存模型引入了一个新的概念，叫happens-before。happens-before的概念最初由LeslieLamport在其一篇影响深远的论文中提出LeslieLamport使用happens-before来定义分布式系统中，事件之间的一个偏序关系（partialordering）。LeslieLamport在这篇论文中给出了一个分布式算法，该算法可以将该偏序关系扩展为某种全序关系。（《Time,ClocksandtheOrderingofEventsinaDistributedSystem》）

JSR-133使用happens-before的概念来指定两个操作（这里的操作是指程序中对变量的读/写，对锁的加锁和解锁）之间的执行顺序。新内存模型定义了如下的happens-before规则。

·程序顺序规则：一个线程中的每个操作，happensbefore于该线程中的任意后续操作。

·监视器锁规则：对一个锁的解锁，happens-before于随后对这个锁的加锁。

·volatile变量规则：对一个volatile域的写，happensbefore于任意后续对这个volatile域的读。

·传递性：如果Ahappens-beforeB，且BhappensbeforeC，那么Ahappens-beforeC。

·start（）规则：如果线程A执行操作ThreadB.start（）（启动线程B），那么A线程的ThreadB.start（）操作happensbefore于线程B中的任意操作。

·join（）规则：如果线程A执行操作ThreadB.join（）并成功返回，那么线程B中的任意操作happens-before于线程A从ThreadB.join（）操作成功返回。

由于两个操作可以在一个线程之内，也可以是在不同线程之间。因此JMM可以通过happens-before关系向程序员提供跨线程的内存可见性保证（如果A线程的写操作a与B线程的读操作b之间存在happens-before关系，尽管a操作和b操作在不同的线程中执行，

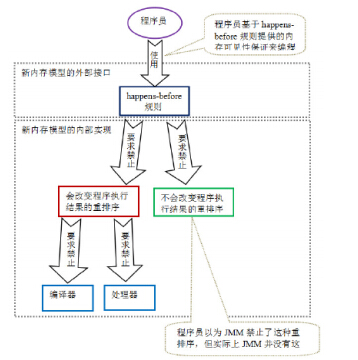


图2新内存模型的设计示意图

但JMM向程序员保证a操作将对b操作可见）。在新内存模型向程序员提供happens-before规则，程序员只需要与happens-before打交道即可，因此Java程序员的学习负担大大降低。同时，新内存模型允许不会改变程序结果的重排序，这可以最大限度地放松对编译器和处理器的束缚，新内存模型的执行性能比旧内存模型要好。

### （★★★）栈

编程语言书籍中经常解释值类型被创建在栈上，引用类型被创建在堆上，但是并没有本质上解释这堆和栈是什么。我仅有高级语言编程经验，没有看过对此更清晰的解释。我的意思是我理解什么是栈，但是它们到底是什么，在哪儿呢（站在实际的计算机物理内存的角度上看）？

1. 在通常情况下由操作系统（OS）和语言的运行时（runtime）控制吗？
2. 它们的作用范围是什么？
3. 它们的大小由什么决定？
4. 哪个更快？

答案一

栈是为执行线程留出的内存空间。当函数被调用的时候，栈顶为局部变量和一些 bookkeeping 数据预留块。当函数执行完毕，块就没有用了，可能在下次的函数调用的时候再被使用。栈通常用后进先出（LIFO）的方式预留空间；因此最近的保留块（reserved block）通常最先被释放。这么做可以使跟踪堆栈变的简单；从栈中释放块（free block）只不过是指针的偏移而已。

堆（heap）是为动态分配预留的内存空间。和栈不一样，从堆上分配和重新分配块没有固定模式；你可以在任何时候分配和释放它。这样使得跟踪哪部分堆已经被分配和被释放变的异常复杂；有许多定制的堆分配策略用来为不同的使用模式下调整堆的性能。

每一个线程都有一个栈，但是每一个应用程序通常都只有一个堆（尽管为不同类型分配内存使用多个堆的情况也是有的）。

直接回答你的问题： 1. 当线程创建的时候，操作系统（OS）为每一个系统级（system-level）的线程分配栈。通常情况下，操作系统通过调用语言的运行时（runtime）去为应用程序分配堆。 2. 栈附属于线程，因此当线程结束时栈被回收。堆通常通过运行时在应用程序启动时被分配，当应用程序（进程）退出时被回收。 3. 当线程被创建的时候，设置栈的大小。在应用程序启动的时候，设置堆的大小，但是可以在需要的时候扩展（分配器向操作系统申请更多的内存）。 4. 栈比堆要快，因为它存取模式使它可以轻松的分配和重新分配内存（指针/整型只是进行简单的递增或者递减运算），然而堆在分配和释放的时候有更多的复杂的 bookkeeping 参与。另外，在栈上的每个字节频繁的被复用也就意味着它可能映射到处理器缓存中，所以很快（译者注：局部性原理）。

答案二

**Stack:**

1. 和堆一样存储在计算机 RAM 中。
2. 在栈上创建变量的时候会扩展，并且会自动回收。
3. 相比堆而言在栈上分配要快的多。
4. 用数据结构中的栈实现。
5. 存储局部数据，返回地址，用做参数传递。
6. 当用栈过多时可导致栈溢出（无穷次（大量的）的递归调用，或者大量的内存分配）。
7. 在栈上的数据可以直接访问（不是非要使用指针访问）。
8. 如果你在编译之前精确的知道你需要分配数据的大小并且不是太大的时候，可以使用栈。
9. 当你程序启动时决定栈的容量上限。

**Heap：**

1. 和栈一样存储在计算机RAM。
2. 在堆上的变量必须要手动释放，不存在作用域的问题。数据可用 delete, delete[] 或者 free 来释放。
3. 相比在栈上分配内存要慢。
4. 通过程序按需分配。
5. 大量的分配和释放可造成内存碎片。
6. 在 C++ 中，在堆上创建数的据使用指针访问，用 new 或者 malloc 分配内存。
7. 如果申请的缓冲区过大的话，可能申请失败。
8. 在运行期间你不知道会需要多大的数据或者你需要分配大量的内存的时候，建议你使用堆。
9. 可能造成内存泄露。

举例：

int foo（）{  
 char \*pBuffer; //<–nothing allocated yet （excluding the pointer itself, wh ich is allocated here on the stack）.  
 bool b = true; // Allocated on the stack.  
 if（b）{  
 //Create 500 bytes on the stack  
 char buffer[500];

//Create 500 bytes on the heap  
pBuffer = new char[500];

}//<– buffer is deallocated here, pBuffer is not  
}//<— oops there’s a memory leak, I should have called delete[] pBuffer;

**答案三**

堆和栈是两种内存分配的两个统称。可能有很多种不同的实现方式，但是实现要符合几个基本的概念:

1.对栈而言，栈中的新加数据项放在其他数据的顶部，移除时你也只能移除最顶部的数据（不能越位获取）T-1。

2.对堆而言，数据项位置没有固定的顺序。你可以以任何顺序插入和删除，因为他们没有“顶部”数据这一概念 T-2。





T-1 T-2

上面上个图片很好的描述了堆和栈分配内存的方式。

**在通常情况下由操作系统（OS）和语言的运行时（runtime）控制吗？**

答：如前所述，堆和栈是一个统称，可以有很多的实现方式。计算机程序通常有一个栈叫做[调用栈](http://en.wikipedia.org/wiki/Call_stack)，用来存储当前函数调用相关的信息（比如：主调函数的地址，局部变量），因为函数调用之后需要返回给主调函数。栈通过扩展和收缩来承载信息。实际上，程序不是由运行时来控制的，它由编程语言、操作系统甚至是系统架构来决定。

堆是在任何内存中动态和随机分配的（内存的）统称；也就是无序的。内存通常由操作系统分配，通过应用程序调用 API 接口去实现分配。在管理动态分配内存上会有一些额外的开销，不过这由操作系统来处理。

**它们的作用范围是什么？**

答：调用栈是一个低层次的概念，就程序而言，它和“作用范围”没什么关系。如果你反汇编一些代码，你就会看到指针引用堆栈部分。就高级语言而言，语言有它自己的范围规则。一旦函数返回，函数中的局部变量会直接直接释放。你的编程语言就是依据这个工作的。

在堆中，也很难去定义。作用范围是由操作系统限定的，但是你的编程语言可能增加它自己的一些规则，去限定堆在应用程序中的范围。体系架构和操作系统是使用虚拟地址的，然后由处理器翻译到实际的物理地址中，还有页面错误等等。它们记录那个页面属于那个应用程序。不过你不用关心这些，因为你仅仅在你的编程语言中分配和释放内存，和一些错误检查（出现分配失败和释放失败的原因）。

**它们的大小由什么决定？**

答：依旧，依赖于语言，编译器，操作系统和架构。栈通常提前分配好了，因为栈必须是连续的内存块。语言的编译器或者操作系统决定它的大小。不要在栈上存储大块数据，这样可以保证有足够的空间不会溢出，除非出现了无限递归的情况（额，栈溢出了）或者其它不常见了编程决议。

堆是任何可以动态分配的内存的统称。这要看你怎么看待它了，它的大小是变动的。在现代处理器中和操作系统的工作方式是高度抽象的，因此你在正常情况下不需要担心它实际的大小，除非你必须要使用你还没有分配的内存或者已经释放了的内存。

**哪个更快一些？**

答：栈更快因为所有的空闲内存都是连续的，因此不需要对空闲内存块通过列表来维护。只是一个简单的指向当前栈顶的指针。编译器通常用一个专门的、快速的寄存器来实现。更重要的一点事是，随后的栈上操作通常集中在一个内存块的附近，这样的话有利于处理器的高速访问（译者注：局部性原理）。

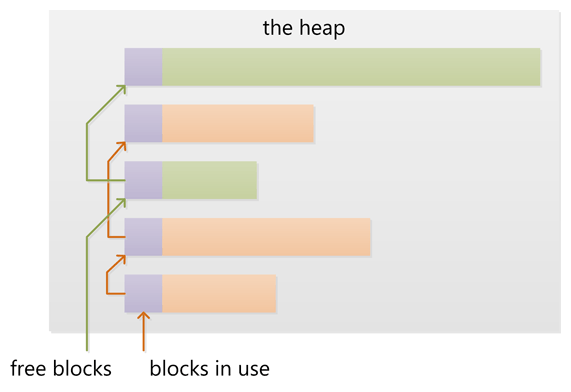
**答案四**

你问题的答案是依赖于实现的，根据不同的编译器和处理器架构而不同。下面简单的解释一下：

1. 栈和堆都是用来从底层操作系统中获取内存的。
2. 在多线程环境下每一个线程都可以有他自己完全的独立的栈，但是他们共享堆。并行存取被堆控制而不是栈。

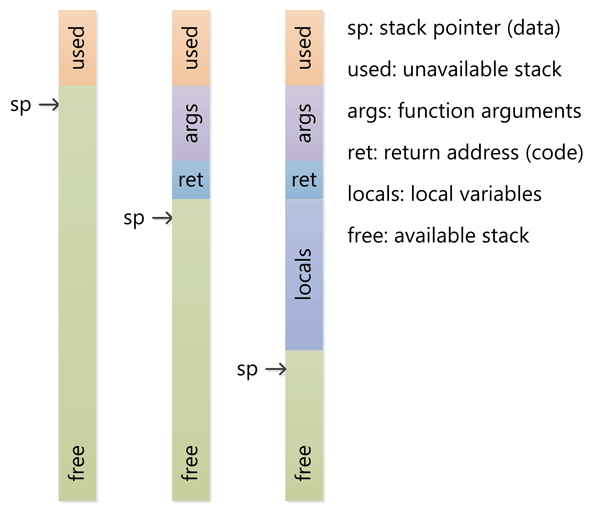
**堆:**

1. 堆包含一个链表来维护已用和空闲的内存块。在堆上新分配（用 new 或者 malloc）内存是从空闲的内存块中找到一些满足要求的合适块。这个操作会更新堆中的块链表。这些元信息也存储在堆上，经常在每个块的头部一个很小区域。
2. 堆的增加新快通常从地地址向高地址扩展。因此你可以认为堆随着内存分配而不断的增加大小。如果申请的内存大小很小的话，通常从底层操作系统中得到比申请大小要多的内存。
3. 申请和释放许多小的块可能会产生如下状态：在已用块之间存在很多小的空闲块。进而申请大块内存失败，虽然空闲块的总和足够，但是空闲的小块是零散的，不能满足申请的大小，。这叫做“堆碎片”。
4. 当旁边有空闲块的已用块被释放时，新的空闲块可能会与相邻的空闲块合并为一个大的空闲块，这样可以有效的减少“堆碎片”的产生。



**栈:**

1. 栈经常与 sp 寄存器（译者注：”stack pointer”，了解汇编的朋友应该都知道）一起工作，最初 sp 指向栈顶（栈的高地址）。
2. CPU 用 push 指令来将数据压栈，用 pop 指令来弹栈。当用 push 压栈时，sp 值减少（向低地址扩展）。当用 pop 弹栈时，sp 值增大。存储和获取数据都是 CPU 寄存器的值。
3. 当函数被调用时，CPU使用特定的指令把当前的 IP （译者注：“instruction pointer”，是一个寄存器，用来记录 CPU 指令的位置）压栈。即执行代码的地址。CPU 接下来将调用函数地址赋给 IP ，进行调用。当函数返回时，旧的 IP 被弹栈，CPU 继续去函数调用之前的代码。
4. 当进入函数时，sp 向下扩展，扩展到确保为函数的局部变量留足够大小的空间。如果函数中有一个 32-bit 的局部变量会在栈中留够四字节的空间。当函数返回时，sp 通过返回原来的位置来释放空间。
5. 如果函数有参数的话，在函数调用之前，会将参数压栈。函数中的代码通过 sp 的当前位置来定位参数并访问它们。
6. 函数嵌套调用和使用魔法一样，每一次新调用的函数都会分配函数参数，返回值地址、局部变量空间、嵌套调用的活动记录都要被压入栈中。函数返回时，按照正确方式的撤销。
7. 栈要受到内存块的限制，不断的函数嵌套/为局部变量分配太多的空间，可能会导致栈溢出。当栈中的内存区域都已经被使用完之后继续向下写（低地址），会触发一个 CPU 异常。这个异常接下会通过语言的运行时转成各种类型的栈溢出异常。（译者注：“不同语言的异常提示不同，因此通过语言运行时来转换”我想他表达的是这个含义）



\*函数的分配可以用堆来代替栈吗？

答：不可以的，函数的活动记录（即局部或者自动变量）被分配在栈上， 这样做不但存储了这些变量，而且可以用来嵌套函数的追踪。

堆的管理依赖于运行时环境，C 使用 malloc ，C++ 使用 new ，但是很多语言有垃圾回收机制。

栈是更低层次的特性与处理器架构紧密的结合到一起。当堆不够时可以扩展空间，这不难做到，因为可以有库函数可以调用。但是，扩展栈通常来说是不可能的，因为在栈溢出的时候，执行线程就被操作系统关闭了，这已经太晚了。

下面是一些不确定点:

1. 我没有听过 bookkeeping data 这种说法，故没有翻译。从上下文理解来看，可以想成是用来寄存器值？函数参数？返回地址？如果有了解具体含义的朋友，烦请告知。
2. 栈和堆栈是一回事，英文表达是 stack，堆是 heap。

### （★★★） JAVA优化

最近在想自己编程时是否注意过代码的效率问题，得出的答案是：没有。代码只是实现了功能，至于效率高不高没怎么关注，这应该是JAVA程序员进阶的时候需要考虑的问题，不再是单纯的实现功能，也不是完全依赖GC而不关注内存中发生了什么，而要考虑到代码的性能。下面是网上找的一篇关于JAVA代码优化的文章，觉得不错，就转载了。这里面设计到了JAVA基础和J2EE方面的优化建议，有时间会整理一下，现在先转载。

原文地址是：http://blog.163.com/user\_zhaopeng/blog/static/16602270820122105731329/

1、 尽量指定类的final修饰符 带有final修饰符的类是不可派生的。

   如果指定一个类为final，则该类所有的方法都是final。Java编译器会寻找机会内联（inline）所有的 final方法（这和具体的编译器实现有关）。此举能够使性能平均提高50% 。

2、 尽量重用对象。   
  特别是String 对象的使用中，出现字符串连接情况时应用StringBuffer 代替。由于系统不仅要花时间生成对象，以后可能还需花时间对这些对象进行垃圾回收和处理。因此，生成过多的对象将会给程序的性能带来很大的影响。

3、尽量使用局部变量

调用方法时传递的参数以及在调用中创建的临时变量都保存在栈（Stack）中，速度较快。其他变量，如静态变量、实例变量等，都在堆（Heap）中创建，速度较慢。另外，依赖于具体的编译器/JVM，局部变量还可能得到进一步优化。

4、不要重复初始化变量   
  默认情况下，调用类的构造函数时， Java会把变量初始化成确定的值：所有的对象被设置成null，整数变量（byte、short、int、long）设置成0，float和 double变量设置成0.0，逻辑值设置成false。当一个类从另一个类派生时，这一点尤其应该注意，因为用new关键词创建一个对象时，构造函数链 中的所有构造函数都会被自动调用。

**5、 SQL语句尽量使用大写**

在JAVA + 数据库的应用系统开发中，java中内嵌的SQL语句尽量使用大写的形式，以减轻数据库解析器的解析负担。

1. **I/O操作中需要及时释放资源**

  Java 编程过程中，进行数据库连接、I/O流操作时务必小心，在使用完毕后，即使关闭以释放资源。因为对这些大对象的操作会造成系统大的开销，稍有不慎，会导致严重的后果。

1. **保证过期对象的及时回收**

由于JVM的有其自身的GC机制，不需要程序开发者的过多考虑，从一定程度上减轻了开发者负担，但同时也遗漏了隐患，过分的创建对象会消耗系统的大量内 存，严重时会导致内存泄露，因此，保证过期对象的及时回收具有重要意义。

JVM回收垃圾的条件是：对象不在被引用；然而，JVM的GC并非十分的机智，即使对象满足了垃圾回收的条件也不一定会被立即回收。所以，建议我们在对象使用完毕，应手动置成null。

**8、 在使用同步机制时，应尽量使用方法同步代替代码块同步。**

**9、 尽量减少对变量的重复计算**

**10、尽量采用lazy loading 的策略，即在需要的时候才开始创建。**

**11、慎用异常**

  异常对性能不利。抛出异常首先要创建一个新的对象。Throwable接口的构造函数调用名为fillInStackTrace（）的本地 （Native）方法，fillInStackTrace（）方法检查堆栈，收集调用跟踪信息。只要有异常被抛出，VM就必须调整调用堆栈，因为在处理过 程中创建了一个新的对象。**异常只能用于错误处理，不应该用来控制程序流程。**

1. **不要在循环中使用：Try {} catch（） {}应把其放置在最外层.**

**13、StringBuffer 的使用**

StringBuffer表示了可变的、可写的字符串。   
有三个构造方法 :   
StringBuffer （）; //默认分配16个字符的空间   
StringBuffer （int size）; //分配size个字符的空间   
StringBuffer （String str）; //分配16个字符+str.length（）个字符空间   
你可以通过StringBuffer的构造函数来设定它的初始化容量，这样可以明显地提升性能。  
 这里提到的构造函数是StringBuffer（int length），length参数表示当前的StringBuffer能保持的字符数量。你也可以使用ensureCapacity（int minimumcapacity）方法在StringBuffer对象创建之后设置它的容量。首先我们看看StringBuffer的缺省行为，然后再找 出一条更好的提升性能的途径。   
 StringBuffer在内部维护一个字符数组，当你使用缺省的构造函数来创建StringBuffer对象的时候，因为没有设置初始化字符长 度，StringBuffer的容量被初始化为16个字符，也就是说缺省容量就是16个字符。当StringBuffer达到最大容量的时候，它会将自身 容量增加到当前的2倍再加2，也就是（2\*旧值+2）。如果你使用缺省值，初始化之后接着往里面追加字符，在你追加到第16个字符的时候它会将容量增加到 34（2\*16+2），当追加到34个字符的时候就会将容量增加到70（2\*34+2）。无论何事只要StringBuffer到达它的最大容量它就不得 不创建一个新的字符数组然后重新将旧字符和新字符都拷贝一遍――这也太昂贵了点。所以总是给StringBuffer设置一个合理的初始化容量值是错不了 的，这样会带来立竿见影的性能增益。StringBuffer初始化过程的调整的作用由此可见一斑。所以，使用一个合适的容量值来初始化 StringBuffer永远都是一个最佳的建议。

**14、合理的使用Java类 java.util.Vector。**

简单地说，一个Vector就是一个java.lang.Object实例的数组。Vector与数组相似，它的元素可以通过整数形式的索引访问。但是，Vector类型的对象在创建之后，对象的大小能够根据元素的增加或者删除而扩展、缩小。请考虑下面这个向Vector加入元素的例子：   
Object bj = new Object（）;   
Vector v = new Vector（100000）;   
for（int i=0; i<100000; i++） {

v.add（0,obj）;

}  
  
  除非有绝对充足的理由要求每次都把新元素插入到Vector的前面，否则上面的代码对性能不利。在默认构造函数中，Vector的初始存储能力 是10个元素，如果新元素加入时存储能力不足，则以后存储能力每次加倍。Vector类就对象StringBuffer类一样，每次扩展存储能力时，所有 现有的元素都要复制到新的存储空间之中。下面的代码片段要比前面的例子快几个数量级：   
 Object bj = new Object（）;   
 Vector v = new Vector（100000）;   
 for（int i=0; i<100000; i++） {

v.add（obj）;

}  
  
  同样的规则也适用于Vector类的remove（）方法。由于Vector中各个元素之间不能含有“空隙”，删除除最后一个元素之外的任意其 他元素都导致被删除元素之后的元素向前移动。也就是说，从Vector删除最后一个元素要比删除第一个元素“开销”低好几倍。  
 假设要从前面的Vector删除所有元素，我们可以使用这种代码：   
 for（int I=0; I<100000; I++） {   
 v.remove（0）;   
 }  
 但是，与下面的代码相比，前面的代码要慢几个数量级：   
 for（int I=0; I<100000; I++） {   
 v.remove（v.size（）-1）;   
 }  
 从Vector类型的对象v删除所有元素的最好方法是：   
 v.removeAllElements（）;  
 假设Vector类型的对象v包含字符串“Hello”。考虑下面的代码，它要从这个Vector中删除“Hello”字符串：   
 String s = “Hello”;   
 int i = v.indexOf（s）;   
 if（I != -1） v.remove（s）;  
 这些代码看起来没什么错误，但它同样对性能不利。在这段代码中，indexOf（）方法对v进行顺序搜索寻找字符串 “Hello”，remove（s）方法也要进行同样的顺序搜索。改进之后的版本是：   
 String s = “Hello”;   
 int i = v.indexOf（s）;   
 if（I != -1） v.remove（i）;  
  这个版本中我们直接在remove（）方法中给出待删除元素的精确索引位置，从而避免了第二次搜索。一个更好的版本是：   
String s = “Hello”; v.remove（s）;  
 最后，我们再来看一个有关Vector类的代码片段：

for（int I=0; I++;I < v.length）  
  如果v包含100,000个元素，这个代码片段将调用v.size（）方法100,000次。虽然size方法是一个简单的方法，但它仍旧需要 一次方法调用的开销，至少JVM需要为它配置以及清除堆栈环境。在这里，for循环内部的代码不会以任何方式修改Vector类型对象v的大小，因此上面 的代码最好改写成下面这种形式：   
 int size = v.size（）; for（int I=0; I++;I<size）  
 虽然这是一个简单的改动，但它仍旧赢得了性能。毕竟，每一个CPU周期都是宝贵的。

**15、当复制大量数据时，使用System.arraycopy（）命令。**

1. **代码重构：增强代码的可读性。**

**17、不用new关键词创建类的实例**

  用new关键词创建类的实例时，构造函数链中的所有构造函数都会被自动调用。但如果一个对象实现了Cloneable接口，我们可以调用它的 clone（）方法。clone（）方法不会调用任何类构造函数。   
 在使用设计模式（Design Pattern）的场合，如果用Factory模式创建对象，则改用clone（）方法创建新的对象实例非常简单。例如，下面是Factory模式的一个 典型实现：   
public static Credit getNewCredit（） {   
 return new Credit（）;   
}   
 改进后的代码使用clone（）方法，如下所示：   
private static Credit BaseCredit = new Credit（）;   
public static Credit getNewCredit（） {   
 return （Credit） BaseCredit.clone（）;   
}   
上面的思路对于数组处理同样很有用。

1. **乘法和除法,用移位操作替代乘法操作可以极大地提高性能。**
2. **在JSP页面中关闭无用的会话。**   
    一个常见的误解是以为session在有客户端访问时就被创建，然而事实是直到某server端程序调用 HttpServletRequest.getSession（true）这样的语句时才被创建，注意如果JSP没有显示的使用 <%@pagesession=”false”%> 关闭session，则JSP文件在编译成Servlet时将会自动加上这样一条语句HttpSession session = HttpServletRequest.getSession（true）;这也是JSP中隐含的session对象的来历。由于session会消耗内 存资源，因此，如果不打算使用session，应该在所有的JSP中关闭它。   
   对于那些无需跟踪会话状态的页面，关闭自动创建的会话可以节省一些资源。使用如下page指令：<%@ page session=”false”%>

**20、JDBC与I/O**

如果应用程序需要访问一个规模很大的数据集，则应当考虑使用块提取方式。默认情况下，JDBC每次提取32行数据。举例来说，假设我们要遍历一个5000 行的记录集，JDBC必须调用数据库157次才能提取到全部数据。如果把块大小改成512，则调用数据库的次数将减少到10次。

**21、Servlet与内存使用**

许多开发者随意地把大量信息保存到用户会话之中。一些时候，保存在会话中的对象没有及时地被垃圾回收机制回收。从性能上看，典型的症状是用户感到系统周期 性地变慢，却又不能把原因归于任何一个具体的组件。如果监视JVM的堆空间，它的表现是内存占用不正常地大起大落。   
 解决这类内存问题主要有二种办法。第一种办法是，在所有作用范围为会话的Bean中实现HttpSessionBindingListener接口。这 样，只要实现valueUnbound（）方法，就可以显式地释放Bean使用的资源。  
 另外一种办法就是尽快地把会话作废。大多数应用服务器都有设置会话作废间隔时间的选项。另外，也可以用编程的方式调用会话的 setMaxInactiveInterval（）方法，该方法用来设定在作废会话之前，Servlet容器允许的客户请求的最大间隔时间，以秒计。

**22、使用缓冲标记**

一些应用服务器加入了面向JSP的缓冲标记功能。例如，BEA的WebLogic Server从6.0版本开始支持这个功能，Open Symphony工程也同样支持这个功能。JSP缓冲标记既能够缓冲页面片断，也能够缓冲整个页面。当JSP页面执行时，如果目标片断已经在缓冲之中，则 生成该片断的代码就不用再执行。页面级缓冲捕获对指定URL的请求，并缓冲整个结果页面。对于购物篮、目录以及门户网站的主页来说，这个功能极其有用。对 于这类应用，页面级缓冲能够保存页面执行的结果，供后继请求使用。

**23、选择合适的引用机制**

在典型的JSP应用系统中，页头、页脚部分往往被抽取出来，然后根据需要引入页头、页脚。当前，在JSP页面中引入外部资源的方法主要有两 种：include指令，以及include动作。   
 include指令：例如<%@ include file=”copyright.html” %>。该指令在编译时引入指定的资源。在编译之前，带有include指令的页面和指定的资源被合并成一个文件。被引用的外部资源在编译时就确定， 比运行时才确定资源更高效。   
include动作：例如<jsp:include page=”copyright.jsp” />。该动作引入指定页面执行后生成的结果。由于它在运行时完成，因此对输出结果的控制更加灵活。但时，只有当被引用的内容频繁地改变时，或者在对 主页面的请求没有出现之前，被引用的页面无法确定时，使用include动作才合算。

**24、及时清除不再需要的会话**

为了清除不再活动的会话，许多应用服务器都有默认的会话超时时间，一般为30分钟。当应用服务器需要保存更多会话时，如果内存容量不足，操作系统会把部分 内存数据转移到磁盘，应用服务器也可能根据“最近最频繁使用”（Most Recently Used）算法把部分不活跃的会话转储到磁盘，甚至可能抛出“内存不足”异常。在大规模系统中，串行化会话的代价是很昂贵的。当会话不再需要时，应当及时 调用HttpSession.invalidate（）方法清除会话。HttpSession.invalidate（）方法通常可以在应用的退出页面调 用

1. **不要将数组声明为：public static final 。**

**26、HashMap的遍历效率讨论**

经常遇到对HashMap中的key和value值对的遍历操作，有如下两种方法：Map<String,String[]> paraMap = new HashMap<String, String[]>（）;   
…………….//第一个循环

Set<String> appFieldDefIds = paraMap.keySet（）;   
 for （String appFieldDefId : appFieldDefIds） {   
 String[] values = paraMap.get（appFieldDefId）;   
 ……   
 }  
//第二个循环   
for（Entry<String, String[]> entry : paraMap.entrySet（））{   
 String appFieldDefId = entry.getKey（）;   
 String[] values = entry.getValue（）;   
 …….   
}  
  
第一种实现明显的效率不如第二种实现。   
 分析如下 Set<String> appFieldDefIds = paraMap.keySet（）; 是先从HashMap中取得keySet  
代码如下：   
public Set<K> keySet（） {   
 Set<K> ks = keySet;   
 return （ks != null ? ks : （keySet = new KeySet（）））;   
}  
private class KeySet extends AbstractSet<K> {   
 public Iterator<K> iterator（） {   
 return newKeyIterator（）;   
}   
public int size（） {   
 return size;   
}   
public boolean contains（Object o） {   
 return containsKey（o）;   
}   
public boolean remove（Object o） {   
 return HashMap.this.removeEntryForKey（o） != null;   
}   
public void clear（） {   
 HashMap.this.clear（）;   
}   
其实就是返回一个私有类KeySet, 它是从AbstractSet继承而来，实现了Set接口。  
再来看看for/in循环的语法   
 for（declaration : expression）   
 statement  
在执行阶段被翻译成如下各式   
for（Iterator<E> #i = （expression）.iterator（）; #i.hashNext（）;）{   
 declaration = #i.next（）;   
 statement   
}  
因此在第一个for语句for （String appFieldDefId : appFieldDefIds） 中调用了HashMap.keySet（）.iterator（）  
而这个方法调用了newKeyIterator（）  
Iterator<K> newKeyIterator（） {   
 return new KeyIterator（）;   
}   
private class KeyIterator extends HashIterator<K> {   
 public K next（） {   
 return nextEntry（）.getKey（）;   
}   
所以在for中还是调用了

在第二个循环for（Entry<String, String[]> entry : paraMap.entrySet（））中使用的Iterator是如下的一个内部类  
private class EntryIterator extends HashIterator<Map.Entry<K,V>> {   
 public Map.Entry<K,V> next（） {   
 return nextEntry（）;   
}   
 此时第一个循环得到key，第二个循环得到HashMap的Entry效率就是从循环里面体现出来的第二个循环此致可以直接取key和value值 而第一个循环还是得再利用HashMap的get（Object key）来取value值现在看看HashMap的get（Object key）方法   
public V get（Object key） {   
 Object k = maskNull（key）;   
 int hash = hash（k）;   
 int i = indexFor（hash, table.length）; //Entry[] table   
 Entry<K,V> e = table;   
 while （true） {   
 if （e == null）   
 return null;   
 if （e.hash == hash && eq（k, e.key））   
 return e.value;   
 e = e.next;   
}    
 其实就是再次利用Hash值取出相应的Entry做比较得到结果，所以使用第一中循环相当于两次进入HashMap的Entry中而第二个循环取得Entry的值之后直接取key和value，效率比第一个循环高。其实按照Map的概念来看也应该是用第二个循环好一点，它本来就是key和value的值对，将key和value分开操作在这里不是个好选择。

25、array（数组） 和 ArryList的使用

array（[]）：最高效；但是其容量固定且无法动态改变；   
ArrayList：容量可动态增长；但牺牲效率；   
基于效率和类型检验，应尽可能使用array，无法确定数组大小时才使用ArrayList！

ArrayList是Array的复杂版本   
 ArrayList内部封装了一个Object类型的数组，从一般的意义来说，它和数组没有本质的差别，甚至于ArrayList的许多方法，如 Index、IndexOf、Contains、Sort等都是在内部数组的基础上直接调用Array的对应方法。   
 ArrayList存入对象时，抛弃类型信息，所有对象屏蔽为Object，编译时不检查类型，但是运行时会报错。   
注：jdk5中加入了对泛型的支持，已经可以在使用ArrayList时进行类型检查。   
从这一点上看来，ArrayList与数组的区别主要就是由于动态增容的效率问题了

**27、尽量使用HashMap 和ArrayList ,除非必要，否则不推荐使用HashTable和Vector ，后者由于使用同步机制，而导致了性能的开销。**

### （★★）XML 部分

**102、XML 文档定义有几种形式？它们之间有何本质区别？解析XML 文档有哪几种方式？【基础】★**

答：1）两种形式：dtd 以及schema；

2）本质区别：schema 本身是xml 的，可以被XML 解析器解析（这也是从DTD上发展schema 的根本目的）；

3）解析方式：有DOM,SAX,STAX 等：

DOM:处理大型文件时其性能下降的非常厉害。这个问题是由DOM 的树结构所造成的，这种结构占用的内存较多，而且DOM 必须在解析文件之前把整个文档装入内存,适合对XML 的随机访问；SAX:不同于DOM,SAX 是事件驱动型的XML 解析方式。它顺序读取XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理XML 文件，适合对XML 的顺序访问；

STAX:Streaming API for XML （StAX）。

**103、你在项目中用到了xml 技术的哪些方面?如何实现的? 【中等难度】★★**

答:用到了数据存贮，信息配置两方面。在做数据交换平台时，将不能数据源的数据组装成XML 文件，然后将XML 文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同XML 文件中还原相关信息进行处理。在做软件配置时，利用XML 可以很方便的进行，软件的各种配置参数都存贮在XML 文件中。

**104、用jdom 解析xml 文件时如何解决中文问题?如何解析? 【较难】★★★**

答：看如下代码,用编码方式加以解决

package test;

import java.io.\*;

public class DOMTest{

private String inFile = "c:\people.xml";

private String outFile = "c:\people.xml";

public static void main（String args[]）{

new DOMTest（）;

}

public DOMTest（）{

try{

javax.xml.parsers.DocumentBuilder builder =

javax.xml.parsers.DocumentBuilderFactory.

newInstance（）.newDocumentBuilder（）;

org.w3c.dom.Document doc = builder.newDocument（）;

org.w3c.dom.Element root = doc.createElement（"老师"）;

org.w3c.dom.Element wang = doc.createElement（"王"）;

org.w3c.dom.Element liu = doc.createElement（"刘"）;

wang.appendChild（doc.createTextNode（"我是王老师"））;

root.appendChild（wang）;

doc.appendChild（root）;

javax.xml.transform.Transformer transformer =

javax.xml.transform.TransformerFactory.

newInstance（）.newTransformer（）;

transformer.setOutputProperty（

javax.xml.transform.OutputKeys.ENCODING,"gb2312"）;

transformer.setOutputProperty（

javax.xml.transform.OutputKeys.INDENT, "yes"）;

transformer.transform（new

javax.xml.transform.dom.DOMSource（doc）,

new javax.xml.transform.stream.StreamResult（outFile））;

}catch （Exception e）{

System.out.println （e.getMessage（））;

}

}

}

**105、编程用JAVA 解析XML 的方式。【较难】★★★**

答：用SAX 方式解析XML，XML 文件如下：

<?xml version="1.0" encoding="gb2312"?>

<person>

<name>王小明</name>

<college>信息学院</college>

<telephone>6258113</telephone>

<notes>男,1955 年生,博士，95 年调入海南大学</notes>

</person>

事件回调类SAXHandler.java ：

import java.io.\*;

import java.util.Hashtable;

import org.xml.sax.\*;

public class SAXHandler extends HandlerBase{

private Hashtable table = new Hashtable（）;

private String currentElement = null;

private String currentValue = null;

public void setTable（Hashtable table）{

this.table = table;

}

public Hashtable getTable（）{

return table;

}

public void startElement（String tag, AttributeList attrs）

throws SAXException{

currentElement = tag;

}

public void characters（char[] ch, int start, int length）

throws SAXException{

currentValue = new String（ch, start, length）;

}

public void endElement（String name） throws SAXException{

if （currentElement.equals（name））

table.put（currentElement, currentValue）;

}

}

**JSP 内容显示源码,SaxXml.jsp:★★★**

<HTML>

<HEAD>

<TITLE>剖析XML 文件people.xml</TITLE>

</HEAD>

<BODY>

<%@ page errorPage="ErrPage.jsp"

contentType="text/html;charset=GB2312" %>

<%@ page import="java.io.\*" %>

<%@ page import="java.util.Hashtable" %>

<%@ page import="org.w3c.dom.\*" %>

<%@ page import="org.xml.sax.\*" %>

<%@ page import="javax.xml.parsers.SAXParserFactory" %>

<%@ page import="javax.xml.parsers.SAXParser" %>

<%@ page import="SAXHandler" %>

<%

File file = new File（"c:\people.xml"）;

FileReader reader = new FileReader（file）;

Parser parser;

SAXParserFactory spf = SAXParserFactory.newInstance（）;

SAXParser sp = spf.newSAXParser（）;

SAXHandler handler = new SAXHandler（）;

sp.parse（new InputSource（reader）, handler）;

Hashtable hashTable = handler.getTable（）;

out.println（"<TABLE BORDER=2><CAPTION>" +

"教师信息表</CAPTION>"）;

out.println（"<TR><TD>姓名</TD>" + "<TD>" +

（String）hashTable.get（new String（"name"）） +"</TD></TR>"）;

out.println（"<TR><TD>学院</TD>" + "<TD>" +（String）hashTable.get（new String（"college"））+"</TD></TR>"）;

out.println（"<TR><TD>电话</TD>" + "<TD>" +

（String）hashTable.get（new String（"telephone"））+ "</TD></TR>"）;

out.println（"<TR><TD>备注</TD>" + "<TD>" +（String）hashTable.get（new String（"notes"））+ "</TD></TR>"）;

out.println（"</TABLE>"）;

%>

</BODY>

</HTML>

### （★★）OOA/D 与UML 部分

### （★）UML 是什么？常用的几种图？【基础】

答：UML 是标准建模语言；常用图包括：用例图,静态图（包括类图、对象图和包图）,行为图,交互图（顺序图,合作图）,实现图。

### （★）编程题: 写一个Singleton 出来。【基础】

答：Singleton 模式主要作用是保证在Java 应用程序中，一个类Class 只有一个实例存在。举例：定义一个类，它的构造函数为private 的，它有一个static的private 的该类变量，在类初始化时实例话，通过一个public 的getInstance

方法获取对它的引用,继而调用其中的方法。

第一种形式:

public class Singleton {

private Singleton（）{}

private static Singleton instance = new Singleton（）;

public static Singleton getInstance（）{

return instance;

}

}

第二种形式:

public class Singleton {

private static Singleton instance = null;

public static synchronized Singleton getInstance（）{

if （instance==null）

instance＝new Singleton（）;

return instance;

}

}

其他形式: 定义一个类，它的构造函数为private 的，所有方法为static 的。一般认为第一种形式要更加安全些。

### （★★）你对软件开发中迭代的含义的理解

答：软件开发中，各个开发阶段不是顺序执行的，应该是并行执行,也就是迭代的意思。这样对于开发中的需求变化，及人员变动都能得到更好的适应。

### （★★）JDK是什么？JRE是什么？

JDK是Java Development Kit的缩写，是Java的开发工具包，主要包含了各种类库和工具，当然也包含了另外一个JRE

JRE是Java Runtime Environment的缩写，是Java程序的运行环境

### （★★）什么是java的平台无关性？

java跨平台是同一java源码被编译后得到的字节码文件可以在不同的操作系统上运行。

注意编译器和解释器的区别：

编译器是将java文件编译为class字节码文件，而解释器是解释执行字节码文件。

不同的系统用各系统下的解释器来运行程序，而编译文件不用用那个系统下的编译器都可以。

### （★）在一台电脑上配置java环境，path起什么作用？如何配置？

path的作用是在DOS环境下，能在任意位置使用JDK目录中bin文件夹中的可执行程序，来编译执行java程序

配置：在环境变量中找到path变量，把bin文件夹的绝对路径加上即可。

### （★★）Java 中会存在内存泄漏吗，请简单描述。

内存泄露就是指一个不再被程序使用的对象或变量一直被占据在内存中。Java 使用有向图的方式进行垃圾回收管理，可以消除引用循环的问题，例如有两个对象，相互引用，只要它们和根进程不可达的，那么GC也是可以回收它们的

java中内存泄露的发生场景，通俗地说，就是程序员可能创建了一个对象，以后一直不再使用这个对象，这个对象却一直被引用，即这个对象无用但是却无法被垃圾回收器回收的，这就是java中的内存泄露，一定要让程序将各种分支情况都完整执行到程序结束，然后看某个对象是否被使用过，如果没有，则才能判定这个对象属于内存泄露。

2.如果一个外部类的实例对象的方法返回了一个内部类的实例对象，这个内部类对象被长期引用了，即使那个外部类实例对象不再被使用，但由于内部类持久外部类的实例对象，这个外部类对象将不会被垃圾回收，这也会造成内存泄露。

3.当一个对象被存储进HashSet集合中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进HashSet集合中时的哈希值就不同了，在这种情况下，即使在contains方法使用该对象的当前引用作为的参数去HashSet集合中检索对象，也将返回找不到对象的结果，这也会导致无法从HashSet集合中单独删除当前对象，造成内存泄露。

### （★★）系统之间有哪些数据交互方式?

我查找的资料题目是：应用系统之间数据传递的几种方式

答案

Socket方式

ftp/文件共享服务器方式

数据库共享数据方式

message方式

具体见博客：https://blog.csdn.net/erhei0317/article/details/52600012

### （★）如下程序，最终执行结果是什么?A

public class Demo {

public static void main（String[] args） {

int x = 10;

double y = 20.2;

long z = 10L;

String str = ""+ x + y \* z;

System.out.println（str）;

}

}

A、10202.0 B、0212.0 C、302.0 D、1020.210

### （★）如下程序，最终执行结果是什么?A

public class Demo {

public static void main（String[] args） {

int i =50;

i = i++ \* 2;

System.out.println（i）;

}

}

A、100 B、101 C、102 D、None of above

### （★）如下程序，最终执行结果是什么?B

public class Example {

String str = new String（"hello"）;

char[] ch = {'a','b'};

public static void main（String[] args） {

Example ex = new Example（）;

ex.change（ex.str,ex.ch）;

System.out.print（ex.str + " and "）;

System.out.print（ex.ch）;

}

public void change（String str, char[] ch） {

str = "test ok";

ch[0] = 'c';

}

}

A、hello and ab B、hello and cb C、hello and a D、None of above

### （★）如下程序，最终执行结果是什么?B

calss C {

C（） {System.out.print（"C"）;}

}

calss A {

C c = new C（）;

A（） {

this（"A"）;

System.out.print（"A"）;

A（String s） {System.out.print（"s"）;}

}

}

calss Test extends A {

Test（） {

super（"B"）;

System.out.print（"B"）;

}

public static void main（String[]args）{

new Test（）;

}

}

A、BB B、CBB C、BAB D、None of above

### （★★）下面代码输出内容是什么？

public static void changeStr（String str）{

str = “welcome”;

}

public static void main （String[] args）{

String str = “1234”;

changeStr（str）;

System.out.println（str）;

}

答案;1234

### （★★）JDK的Collections.sort（）是采用何种排序方式？请就一个简单的数组通过此方式来实现排序

归并排序，它是稳定排序，当数据接近有序的时候，效率更高

public static int[] sort（int[] a,int low,int high）{

int mid = （low+high）/2;

if（low<high）{

sort（a,low,mid）;

sort（a,mid+1,high）;

//左右归并

merge（a,low,mid,high）;

}

return a;

}

public static void merge（int[] a, int low, int mid, int high） {

int[] temp = new int[high-low+1];

int i= low;

int j = mid+1;

int k=0;

// 把较小的数先移到新数组中

while（i<=mid && j<=high）{

if（a[i]<a[j]）{

temp[k++] = a[i++];

}else{

temp[k++] = a[j++];

}

}

// 把左边剩余的数移入数组

while（i<=mid）{

temp[k++] = a[i++];

}

// 把右边边剩余的数移入数组

while（j<=high）{

temp[k++] = a[j++];

}

// 把新数组中的数覆盖nums数组

for（int x=0;x<temp.length;x++）{

a[x+low] = temp[x];

}

}

### （★★）1MB内存空间的存下100万个8位数吗?如果能，怎么存储?如果不能，那么最少需要多大的内存空间？

不能

Int 类型 -2147483648——2147483647 4 Byte

1kb = 1024 Byte 4000000

1mb = 1024 kb 3906.25

1GB = 1024 mb 3.8146972656