**达内郑州嵩山路中心java方向各阶段重点知识点** 2017.7.10

**该文档由达内科技郑州嵩山路中心项目部原创,如需使用请保留此出处**

**若你身边有想学习JAVA、UI设计、Python、人工智能、大数据、Linux云计算等IT技能的,请联系达内郑州嵩山路中心各项目部当班JAVA项目经理老师,并且有现金推荐奖励:**

**​何新峰 18638581252**

**魏 凯 18503852425**

**刘旭凯 15669955261**

**赵小云 15617065713**

**注:文档中带有 "面" 标识的为重点面试题,未标注的为各阶段重点知识点**

1. **java编程语言的特点、优势？(面)**
   1. 简单，稳定。面向对象。跨平台性。
   2. 解释型语言(非纯解释性语言,先编译后运行)。
   3. 多线程。多态性，可靠，安全。
2. **谈谈你对Linux操作系统的认识?和windows有什么区别?常用命令有哪些? (面)**

Linux操作系统开源的操作系统、免费，主要用于服务器端操作系统,Java主要是服务器端开发，所以最终部署环境一般都是Linux.

Linux与Windows目录结构的区别:

1)文件系统不同: Linux:目录 Windows:盘符

2)外部设备映射不同: Linux:挂载点 Windows:盘符

3)安全级别不同: Linux:高 Windows:低

Linux常用命令:

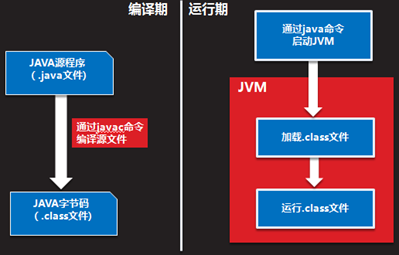
1)pwd:显示当前工作目录

2)ls:查看当前工作目录的内容

3)cd:改变当前工作目录 . :当前目录 .. :上一级目录

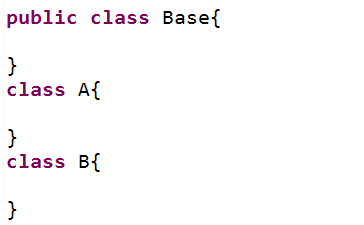
1. **Java的运行过程?(面)**

java遵循先编译后运行的过程,首先通过javac命令将JAVA源程序.java文件编译为JAVA字节码.class文件，而后通过java命令启动JVM，由JVM来加载.class文件并运行.class文件。



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

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

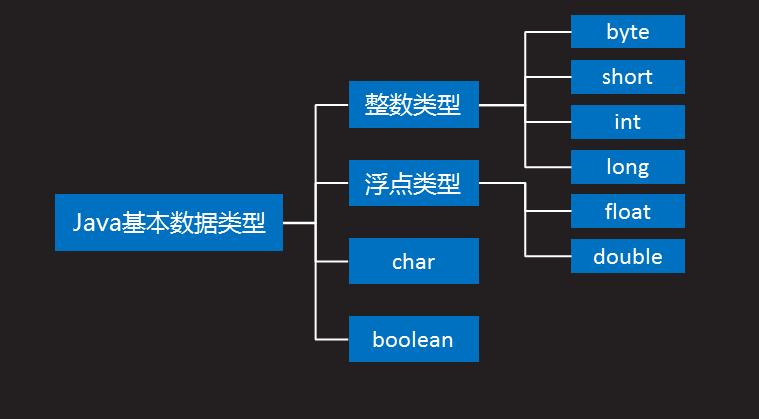


1. **简述：JDK，JRE,JVM的关系？(面)**

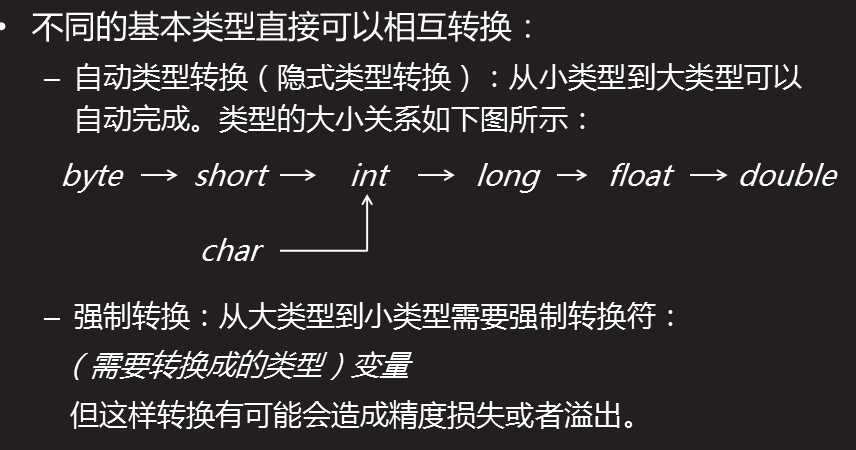
JDK：Java开发工具包，包含编写Java程序所必须的编译、运行等开发工具以及JRE。JRE：Java运行环境，提供了运行Java应用程序所必须的软件环境，包含有Java虚拟机（JVM）和丰富的系统类库。

JVM：Java虚拟机，提供了字节码文件（.class）的运行环境支持.

1. **java的八大基本类型以及各自的字节数和位数?(面)**



1. **基本类型之间的转换规则?(面)**



整数类型等号右边默认是int类型,如果想要表示long类型,需要在值后面加L

浮点型等号右边默认是double类型,如果想要表示float类型,需要在值后面加F

1. **short s1 = 1; s1 = s1 + 1;有什么错? short s1 = 1; s1 += 1;有什么错?(面)**

对于short s1 = 1; s1 = s1 + 1;由于s1+1运算时会自动提升表达式的类型，所以结果是int型，再赋值给short类型s1时，编译器将报告需要强制转换类型的错误。

对于short s1 = 1; s1 += 1;由于 +=是java语言规定的运算符，java编译器会对它进行特殊处理，因此可以正确编译。

1. **Java有没有goto关键字? (面)**

有,goto是java中的保留字，现在没有在java中使用。

1. **&和&&的区别? (面)**

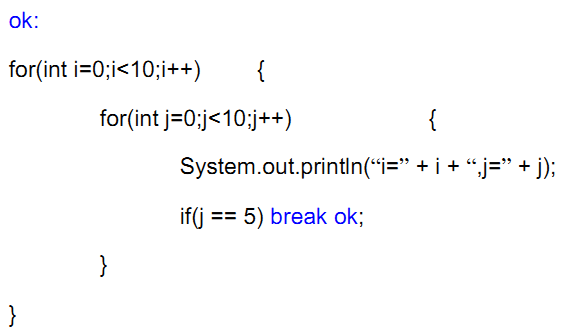
&和&&都可以用作逻辑与的运算符

**&:非短路与**,&左右两侧表达式如果为boolean类型,那么表示的是非短路与,左侧表达式为false时还会计算右侧表达式,若&两侧非boolean类型,那么是位与运算,获取来获取该整数的最低4个bit位，例如，0x31 & 0x0f的结果为0x01。

**&&:短路与**,表达式左侧如果为false,表达式右侧就不再进行运算.

1. **在JAVA中如何跳出当前的多重嵌套循环？(面)**

在Java中，要想跳出多重循环，可以在外面的循环语句前定义一个标号，然后在里层循环体的代码中使用带有标号的break语句，即可跳出外层循环。



1. **Math.round(11.5)等于多少? Math.round(-11.5)等于多少? (面)**

Math类中提供了三个与取整有关的方法：ceil、floor、round

ceil的英文意义是天花板，该方法就表示向上取整:

Math.ceil(11.3)的结果为12,Math.ceil(-11.3)的结果是-11

floor的英文意义是地板，该方法就表示向下取整:

Math.floor(11.6)的结果为11,Math.floor(-11.6)的结果是-12；

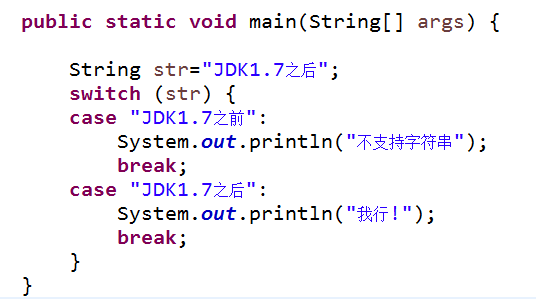
round方法:它表示“四舍五入”，Math.round(11.5)的结果为12，Math.round(-11.5)的结果为-11。

1. **switch 语句能否作用在byte上,能否作用在long上,能否作用在String上 ?(面)**

在switch(expr1)中,expr1只能是一个整数表达式,整数表达式可以是int基本类型或Integer包装类型,由于byte,short,char都可以自动转换成int型,所以可以使用.

long类型由于不能自动转换成int类型,所以不能使用.

关于字符串类型,在JDK是1.7版本之前，swicth case中不可以使用字符串,但在JDK1.7之后是可以用字符串的，这是最新版本新加入的特性.



1. **break,continue,return 的区别? (面)**

break:跳出循环(只能跳出一层循环)

continue:跳过本次循环体重剩余的语句进入下一次循环

return:结束当前方法

1. **使用final关键字修饰一个变量时，是引用不能变，还是引用的对象不能变？(面)**

使用final关键字修饰一个变量时，是指引用变量不能变，引用变量所指向的对象中的内容还是可以改变的,另外final修饰的方法不能被重写，修饰的类不可被继承，修饰的成员变量不可改变.

比如:final StringBuffer a=new StringBuffer("abc");

执行如下语句将报编译错误:

a=new StringBuffer("123");

但是,执行如下语句则可以通过编译:

a.append("def");

1. **内存管理？(面)**

JVM会将申请到的内存从逻辑上划分为三个区域：堆、栈、方法区。这三个区域分别用于存储不同的数据。

堆：用于存储使用new关键字所创建的对象以及对象的属性成员变量。

栈：用于存储程序运行时在方法中声明的所有的局部变量。

方法区：用于存放类的各种信息（包括方法）都在方法区存储。

1. **成员变量和局部变量的定义以及生命周期?(面)**

局部变量：

1) 定义在方法中；

2) 没有默认值，必须自行设定初始值；

3) 方法被调用时，存在栈中，方法调用结束时局部变量从栈中清除；

成员变量：

1) 定义在类中，方法外；

2) 由系统设定默认初始值，可以不显式初始化；

3) 所在类被实例化后，存在堆中，对象被回收时，成员变量失效；

1. **static关键字可以修饰什么?(面)**

static修饰的变量是静态变量,修饰的方法是静态方法,可以和final共同修饰常量,也还可以单独作为静态块存在.

static修饰的属性,方法都是独一份,可以被多个对象共用,因为与对象无关,属于类,所以直接用类名调用.

1. **静态变量和实例变量的区别？(面)**

在语法定义上的区别：静态变量前要加static关键字，而实例变量前则不加。

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

1. **冒泡排序的原理及代码实现?(面,代码为笔试题)**

冒泡的原理:

1)假设有4个数,4个数冒三轮

2)每一轮都是从第1个元素开始比，

每一次都是和它的下一个元素比

3)冒出来的就不带它玩了

代码实现:

需要用到双层for循环:

int arr[]={10,9,1,20,19,30,5};

public static void main(String[] args) {

for(int i=0;i<arr.length-1;i++){//轮:7个数比6轮

for(int j=0;j<arr.length-1-i;j++){//每轮次数:依次比6,5,4....

if(arr[j]>arr[j+1]){

int t=arr[j];

arr[j]=arr[j+1];

arr[j+1]=t;

}}}

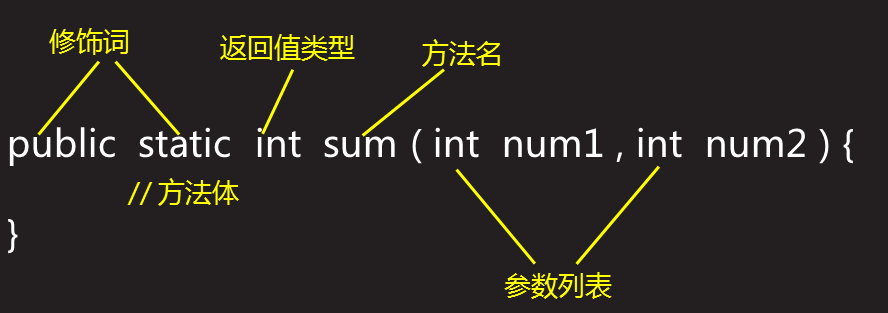
for(int i=0;i<arr.length;i++){

System.out.print(arr[i]+" ");

} }

1. **方法的定义,返回值?**

方法是封装特定的逻辑功能,一个方法只干一件事,可以被反复调用多次,避免代码重复，有利于代码维护、有利于团队协作开发.

方法包含:修饰词,返回值类型,方法名,参数列表,方法体,如下:

有返回值的方法可以使用return返回:

return 值; 结束方法的执行 返回结果给调用方

return; 结束方法的执行

1. **Overload和Override的区别？(面)**

Overload是重载的意思，Override是覆盖的意思，也就是重写。

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

重写Override表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。

1. **重写的两同两小一大规则?(面)**

重写要遵循"两同两小一大"原则:

两同:1)方法名相同 2)参数列表相同

两小: 1)子类方法的返回值类型小于或等于父类的

1.1)void时必须相等

1.2)基本类型时必须相等

1.3)引用类型时小于或相等:父类大,子类小

2)子类抛出的异常小于或等于父类的

一大:子类方法的访问权限大于或等于父类的

1. **什么是类?什么是对象?**

现实世界是由很多很多对象组成的,基于对象抽出了类

对象:真实存在的单个个体

类:类型、类别，代表一类个体, 一个类可以创建多个对象:同一类型的对象，结构相同，数据不同.

类是对象的模板，对象是类的具体的实例

类中可以包含:

1)所有对象所共有的属性/特征(静的)---------成员变量

2)所有对象所共有的行为(动的)--------------方法

比如:有一个人这个类,类中包含有人的一些特征(成员变量):年龄,性别,名字等,还有一些行为(方法):吃饭,睡觉等,通过这个类可以创建出张三,李四等很多个实例对象.

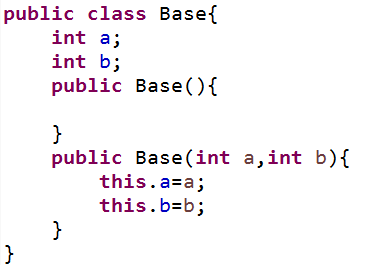
1. **构造方法? (面)**

构造方法也叫构造函数、构造器、构建器

1)常常用于给成员变量赋初值,与类同名，没有返回值类型,在创建对象时被自动调用

2)若自己不写构造，则编译器默认提供一个无参构造,若写了构造，则不再默认提供

3)构造方法可以重载,但不可被继承,不可以被重写.



1. **构造器Constructor是否可被重写override? (面)**

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

1. **this关键字的用法?(面)**

this:指代当前对象，哪个对象调指的就是哪个对象,只能用在方法中，方法中访问成员变量之前默认有个this.

this的用法:

1)this.成员变量名------------访问成员变量

2)this.方法名()--------------调用方法

3)this()---------------------调用构造方法

1. **super关键字的用法?(面)**

super:指代当前对象的父类对象

super的用法:

1)super.成员变量名----------访问父类的成员变量

2)super.方法名()------------调用父类的方法

3)super()-------------------调用父类的构造方法

注意: 构造子类之前必须先构造父类,子类构造方法中若没有调用父类的构造方法，则默认super()调父类的无参构造，若子类构造方法中调用了父类的构造方法，则不再默认提供super()调父类构造必须位于子类构造方法的第一句,默认存在.

且不能与this()同时使用,因为this()和super()都要求存在构造方法的第一行,会产生竞争,编译错误.

1. **向上造型?(面)**

父类型的引用指向子类的对象,多态的一种体现,能点出来什么，看引用的类型,调用方法看引用指向的对象.

1. **面向对象三大特征?(面)**

封装,继承,多态

封装:把描述一个对象的属性和行为封装成一个类,把具体的业务逻辑功能实现封装成一个方法,其次封装的意义还有效的保护属性通过访问修饰符私有化属性(成员变量),公有化方法.

继承:实现代码的复用,所有的子类所共有的行为和属性抽取为一个父类,所有的子类继承该类可具备父类的属性和行为,继承具有单一性和传递性.

多态:程序中定义的引用类型变量所指向的具体类型和调用具体的方法在编译期无法确定,而是在运行期才能确定该引用类型变量指向具体哪个对象而调用在哪个类中声明的方法.

多态的表现形式有强制类型转换,向上造型等,多态可分为行为多态和对象多态:

行为多态:同一个run(){}方法,不同的对象调用时会有不同的实现,猫调用时是跑,鱼调用时是游,鸟调用时是飞.

对象多态:同一个对象,可以被造型为不同的类型,比如同一个人对象,可以被造型为儿子,父亲,员工等.

1. **java中实现多态的机制是什么？(面)**

靠的是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象，而程序调用的方法在运行期才动态绑定，就是引用变量所指向的具体实例对象的方法，也就是内存里正在运行的那个对象的方法，而不是引用变量的类型中定义的方法。

1. **访问修饰符？**

1)public:公共的，任何类

2)protected:受保护的，本类、子类、同包类  
3)默认的:什么也不写，本类、同包类

4)private:私有的，本类



1. **abstract class抽象类和interface接口有什么区别?(面)**

含有abstract修饰符的class即为抽象类，抽象类不能创建的实例对象。含有抽象方法的类必须定义为abstract class.

接口（interface）可以说成是一种特殊的抽象类，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。

下面比较一下两者的语法区别：

* 1. 抽象类可以有构造方法，接口中不能有构造方法。
  2. 抽象类中可以有普通成员变量，接口中没有普通成员变量
  3. 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。
  4. 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不报错，但也不行,默认类型子类不能继承），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.
  5. 抽象类中可以包含静态方法，接口中JDK1.8之前不可以有不能包含静态方法和成员方法,JDK1.8之后可以包含.但成员方法必须使用default修饰
  6. 抽象类和接口中都可以包含静态成员变量，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。
  7. 一个类可以实现多个接口,用逗号隔开，但只能继承一个抽象类,接口不可以实现接口,但可以继承接口并且可以继承多个接口,用逗号隔开.

1. **内部类可以引用它的包含类的成员吗？有没有什么限制？(面)**

完全可以。如果不是静态内部类，那没有什么限制！

如果是静态内部类,那在这种情况下不可以访问外部类的普通成员变量，而只能访问外部类中的静态成员。

1. **什么是匿名内部类?**

1)若想创建一个类(子类)的对象，并且对象只被创建一个，此时该类不必命名，称

之为匿名内部类

2)匿名内部类中访问外部的变量，该变量必须是final的.

1. **super.getClass()方法调用结果? (面)**

下面程序的输出结果是多少？

public class Test extends Date{

public static void main(String[] args) {

new Test().test();

}

public void test(){

System.out.println(super.getClass().getName());

}

}

很奇怪，结果是Test

在test方法中，直接调用getClass().getName()方法，返回的是Test类名

原因在于:由于getClass()在Object类中定义成了final，子类不能覆盖该方法，所以，在test方法中调用getClass().getName()方法，其实就是在调用从父类继承的getClass()方法，等效于调用super.getClass().getName()方法，所以，super.getClass().getName()方法返回的也应该是Test。

如果想得到父类的名称，应该用如下代码：

getClass().getSuperClass().getName();

1. **一个对象在什么情况下会被GC回收？(面)**

当这个对象没有被任何引用指向或者引用为NULL时会被GC回收。

1. **JAVA中调用方法传递方式是值传递还是引用传递? (面)**

在java中调用方法传递的不管基本类型的值还是引用类型的引用,都是值传递,有

切只要一种传递方式:值传递

调用方法传入基本类型的值时传递的是值的一份拷贝,传入引用类型的引用时传递

的也是引用中存储的地址值,都是值传递,都是在栈中操作的.

1. **String是最基本的数据类型吗?(面)**

不属于，基本数据类型包括byte、int、char、long、float、double、boolean和short。

java.lang.String类是final类型的，因此不可以继承这个类、不能修改这个类。为了提高效率节省空间，我们应该用StringBuffer类。

1. **String这个类能否被继承? (面)**

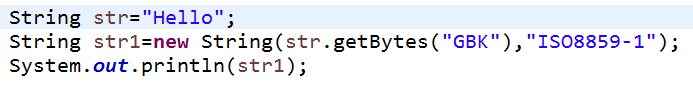
不可以,String类是final类型的，因此不可以继承这个类、不能修改这个类.

1. **如何将GBK编码的字符串转换成ISO8859-1编码的字符串?**

String类有一个重载的构造方法:

String(byte bytes[], String charsetName)

第一个参数为一个byte数组,第二个参数为指定的字符集类型



1. **String,StringBuffer,StringBuilder的区别?(面)**

String类是final修饰的,该类所表示的字符串对象不可变,一旦发生改变即创建新对象, 当我们通过字面量，常量来初始化一个字符串时，JVM首先会从字符串的常量池(一个JVM内部维护的内存区域，用来保存已经创建过的字符串对象)中查询用来保存该字符串的对象是否存在，若存在则直接引用，若不存在则创建该字符串对象并存入常量池，然后引用它.

StringBuffer和StringBuilder是可变字符串对象,对字符串的修改操作不会创建新对象,都是在原有对象基础上进行修改,内部有很多操作字符串的方法,比如append()等.另外StringBuffer是线程安全的，同步处理的，性能稍慢；StringBuilder是非线程安全的，并发处理的，性能稍快。

1. **String s="a"+"b"+"c"+"d"创建了几个对象?(面)**

创建了1个对象,编译器在编译时会先将4个单独的字符串编辑成一个"abcd"字符串.

1. **String str=new String("abcdefg");创建了几个对象?(面)**

创建了两个对象,一个str引用指向堆内存中的String对象,另外一个则是String类有参构造方法中的直接量字符串对象,该字符串对象在字符串常量池中.

把创建的对象过程拆分成两部分就比较直观了:

String s="abcdefg";

String str=new String(s);

延伸问题:

如果在执行一次String str1=new String("abcdefg");此时共创建了几个对象?

上一次操作创建了2个对象,这一次操作创建了1个对象,一共3个.

因为在第一次常量池中已经有一个"abcdefg"字符串对象,第二个创建时并没有创建新的,而是拿过来直接用,只是创建了1个str1指向堆内存中的String对象,共3个.

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引用(equals比较为true的字符串对象),如果常量池中没有该对应的字符串,则会把该字符串存入常量池,返回该字符串对象的引用.

它遵循对于任何两个字符串 s 和 t，当且仅当 s.equals(t) 为 true 时，s.intern() == t.intern() 才为 true.  
例如:

String str1 = "a";

String str2 = "b";

String str3 = "ab";

String str4 = str1 + str2;

String str5 = new String("ab");

System.out.println(str5.equals(str3));//true

System.out.println(str5 == str3);//false

System.out.println(str5.intern() == str3);//true

System.out.println(str5.intern() == str4);//false

第一、str5.equals(str3)这个结果为true，因为字符串的值的内容相同。

第二、str5 == str3对比的是引用的地址是否相同，由于str5采用new String引用第地址不相同,所以结果为false。

第三、当str5调用intern的时候，会检查字符串池中是否含有该字符串。由于之前定义的str3已经进入字符串池中，所以会得到相同的引用,所以比较为true。

第四，当str4 = str1 + str2后，str4的值也为”ab”,但是在字符串拼接的规则里, 如果+号两边有一个是变量,就会去创建新对象,str4并没有引用常量池中的地址.而是自己创建了一个新"ab"对象放在堆内存.所以比较为flase

1. **如何把一段逗号分割的字符串转换成一个数组? (面)**

String[] split(String regex):将字符串按照传入参数的规则进行拆分成字符串数组.

比如:

String str="java,php,c++,c#,web";

String [] array=str.split(",");

拆分后的结果为字符串数组.

1. **final,finalize,finally的区别?(面)**

final:修饰符:

final修饰的类不能被继承,

final修饰的变量不可改变,在声明的同时初始化.

final修饰的方法不可被重写.

finalize:方法名

在垃圾回收器将内存中的没有引用指向的对象清空之前,调用finalize() 进行清理工作.

finally:异常处理中的程序块

在异常处理时,使用finally块来进行必要的清理工作,不论是否有异常发生,finally语句块中的语句都会被执行,如果发生了异常,则执行catch语句,再执行finally块.

1. **String这个类常用的处理字符串的方法? (面)**

indexOf:检索指定字符在字符串中第一次出现的位置.

substring:用来截取当前字符串,含头不含尾

trim:去除字符串两边的空白

charAt:给定一个下标,获取该下标对应的字符

startsWith:是否以给定字符串开始.

endsWith:是否以给定字符串结尾

toUpperCase:将当前字符串引文部分全部转换为大写

toLowerCase:将当前字符串英文部分全部转换为小写

valueOf:将基本类型的值转换为字符串

1. **StringBuilder这个类常用的处理字符串的方法?**

append(String str)：追加字符串；

insert (int dstOffset，String s)：插入字符串；

delete(int start，int end)：删除字符串；

replace(int start，int end，String str): 替换字符串；

reverse()：字符串反转.

1. **==和equals方法究竟有什么区别？(面)**

==比较基本数据类型时比较的时值是否相等，比较引用类型时比较的是对象的地址值是

否相同，也就是否是同一对象。

未重写的equals方法调用的是Object的方法,用法==一样,重写后的equals方法是用

于比较两个独立对象的内容是否相同，就好比去比较两个人的长相是否相同，它比较的

两个对象是独立的。例如对于下面的代码：

String a=new String("foo");

String b=new String("foo");

两条new语句创建了两个对象，然后用a和b这两个变量分别指向了其中一个对象，

这是两个不同的对象，它们的地址是不同的，所以表达式a==b将返回false，而这两

个对象中的内容是相同的，所以，表达式a.equals(b)将返回true.

String这个类重写过equals方法,StringBuffer,StringBuilder未重写过equals方法.

1. **包装类的自动拆装箱？(面)**

自动拆装箱是JDK1.5以后的一个新特性：

基本类型转换为包装类叫自动装箱，比如int转换成Integer

包装类转换成基本类型叫自动拆箱，比如Integer转换成int。

将基本类型转换为包装类时，我们可以调用包装类的一个静态方法valueOf():

Integer i = Integer.valueOf(1);

Double d = Double.valueOf(1.1):

将包装类转换为基本类型时，我们可以使用包装类的方法xxxValue()方法

Integer i = new Integer(1);

int n = i.intValue();

Double d = new Double(1.1);

double dn = d.doubleValue();

1. **int i=128;如下情况的比较结果是?(面)**

Integer i1=new Integer(i);

Integer i2=new Integer(i);

System.out.println(i1==i2);

System.out.println(i1.equals(i2));

输出的结果是false true

因为i1和i2使用new关键字new了两次,在堆内存中创建了2个对象,所以==比较为false,内容相同equals比较为true

Integer i3=Integer.valueOf(i);

Integer i4=Integer.valueOf(i);

System.out.println(i3==i4);

System.out.println(i3.equals(i4));

输出结果为false true

在包装类的内存中有一块区域,缓存着Integer的byte范围内的值(-128~127),如果未超出此范围,则直接在该缓存区取值,并不会在堆中创建新对象,如果超出此范围则会在堆内存中创建新对象.

如上代码i=128超出了取值范围,则在i3和i4创建了两个对象,所以双等号比较为false,内容相同equals比较为true

注意:如果i=127,通过new关键字创建的对象也是两个对象,通过valueOf()创建的对象为同一个对象.

1. **怎么把一个基本类型的值转换成字符串?(面)**

int a=100;

两种方式:

1)String str=a+"";

2)String str=String.valueOf(a);

反过来把一个字符串数值转成一个基本类型:

String str="100";

int a=Integer.parseInt(str);

1. **数组有没有length()这个方法? String有没有length()这个方法？集合有没有length()这个方法? (面)**

数组没有length()这个方法，有length的属性.

String有length()这个方法.

集合没有length()方法,有size()方法.

1. **Collection和 Collections的区别?(面)**

Collection是集合类的上级接口，继承与他的接口主要有Set和List.

List:称为可重复集, 重复元素指的是equals方法比较为true的元素

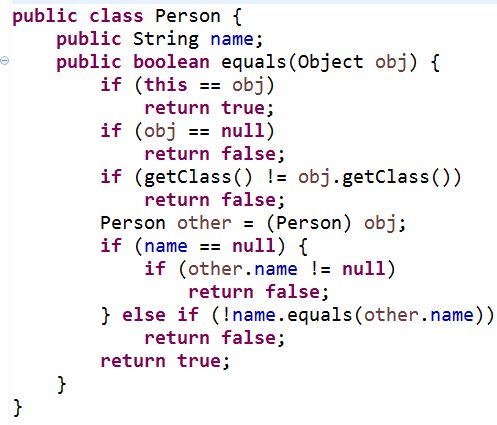
Set:称为不可重复集, 该集合中是不能将相同的元素存入集合两次

Collections是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索

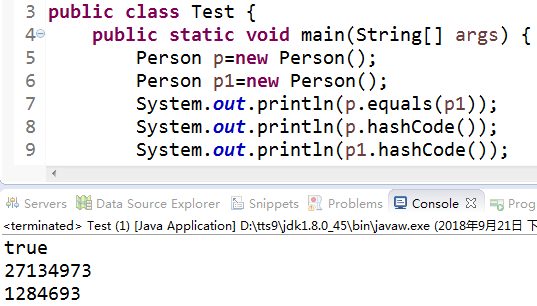
排序、线程安全化等操作。

1. **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? (面)**

这句话是对的。首先看以下Person,只重写了equals方法时:



运行结果如下:equals比较为true,但却有不同的hashCode值,此时调用的是Object的hashCode方法.

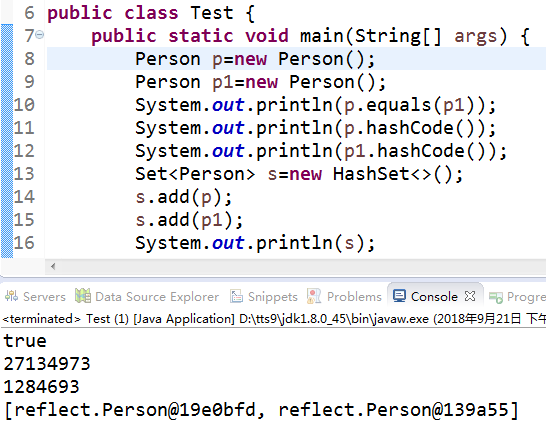


因为equest比较相等,hashcode不相等无法保证是同一个对象,因为equest比较的属性好比身份证上的名字,hashcode值好比身份证号,两个都叫张三的人他们的身份证号不同,就不是同一个人,为了避免这种情况会同时重写两个方法保证他们是同一个对象.

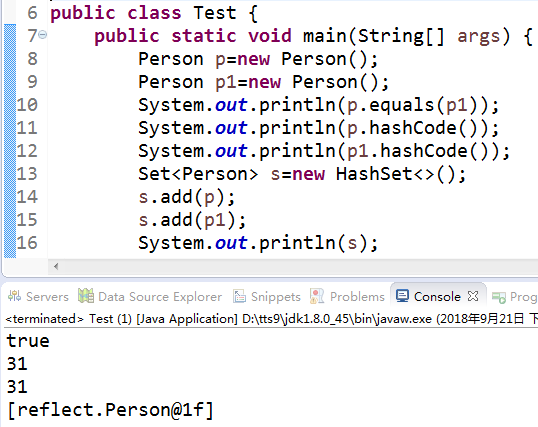
如果对象要保存在HashSet或HashMap中，它们的equals相等，那么，它们的hashcode值就必须相等.

如果不是要保存在HashSet或HashMap，则与hashcode没有什么关系了，这时候hashcode不等是可以的，例如ArrayList存储的对象就不用实现hashcode，但通常都会去实现的,因为我们不确定这个对象在某个时刻会不会放在HashSet或HashMap中.

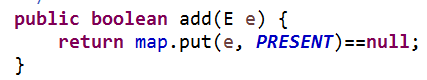
未重写hashcode()方法在存入HashSet时为两个对象.



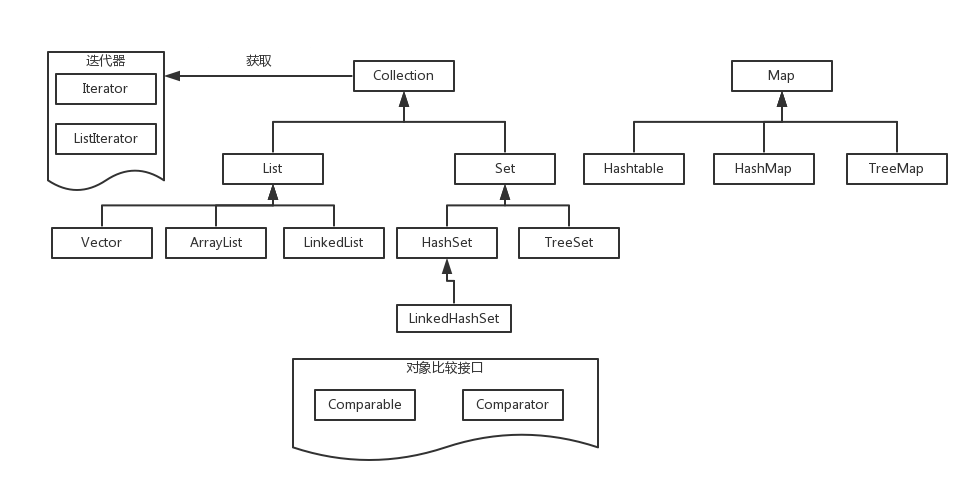
重写hashCode()方法后再存入HashSet时为重复对象.



原因在于HashSet的add方法内部是map,map是根据key的hashcode值来决定键值对的存储位置.



1. **List和set集合,Map集合的区别以及他们的实现类有哪些?有什么区别?(面)**

****

List是可重复集合,Set是不可重复集合,这两个接口都实现了Collection父接口.

Map未继承Collection,而是独立的接口, Map是一种把键对象和值对象进行映射的集

合，它的每一个元素都包含了一对键对象和值对象, Map中存储的数据是没有顺序的，

其key是不能重复的,它的值是可以有重复的。

List的实现类有ArrayList, Vector和LinkedList.

ArrayList和Vector内部是线性动态数组结构,在查询效率上会高很多,Vector是线程安全的,相比ArrayList线程不安全的,性能会稍慢一些.

LinkedList:是双向链表的数据结构存储数据,在做查询时会按照序号索引数据进行前向或后向遍历,查询效率偏低,但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的实现类有HashSet和TreeSet;

HashSet: 内部是由哈希表（实际上是一个 HashMap 实例）支持的。它不保证 set元素的迭代顺序.

TreeSet: TreeSet使用元素的自然顺序对元素进行排序，或者根据创建 set 时提供的 Comparator 进行排序.

Map接口有三个实现类:Hashtable,HashMap,TreeMap,LinkedHashMap

Hashtable: 内部存储的键值对是无序的是按照哈希算法进行排序,与HashMap最大的区别就是线程安全.键或者值不能为null,为null就会抛出空指针异常

HashMap: 内部存储的键值对是无序的是按照哈希算法进行排序,与Hashtable最大的区别就是非线程安全的,键或值可以为null

TreeMap: 基于红黑树(red-black tree)数据结构实现, 按 key 排序,默认的排序方式是升序.

LinkedHashMap:有序的Map集合实现类,相当于一个栈,先put进去的最后出来,先进后出.

1. **List和 Map区别?(面)**

一个是存储单列数据的集合，另一个是存储键和值这样的双列数据的集合，List中存储

的数据是有顺序，并且允许重复；Map中存储的数据是没有顺序的，其key是不能重

复的,它的值是可以有重复的。

1. **Java Bean规范？Bean属性是哪一个? (面)**

创建一个类时要遵循java Bean规范：

1）实现可序列化接口。

2）私有属性（成员变量）和公有方法。

3）无参构造器。

4）GET和SET方法。

其中：get和set方法后面跟着的属性首字母小写后是Bean属性，比如：getName/setName中的Name将首字母小写后name为bean属性,Bean属性和成员变量是可以不一样的.

1. **怎么实现集合中引用类型元素的自定义排序?**

当一个类实了Comparable接口后,接口要求必须重写方法comparaTo,该方法的作用

是定义当前对应于参数给定对象比较大小规则

方法要求返回一个整数,该整数不关注具体值,只关注取值范围,即:

当返回值>0当前对象大于参数对象

当返回值<0当前对象小于参数对象

当返回值=0两个对象相等

然后在使用时直接调用Collections.sort(list);传入要比较的集合就会按照自己定义的比

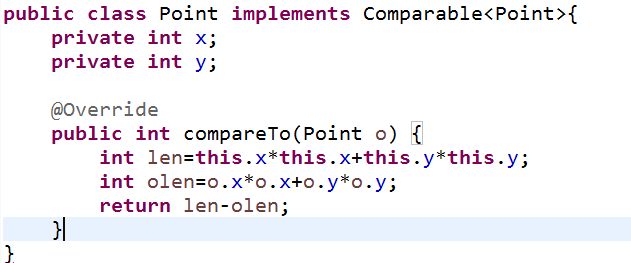
较规则进行比较.

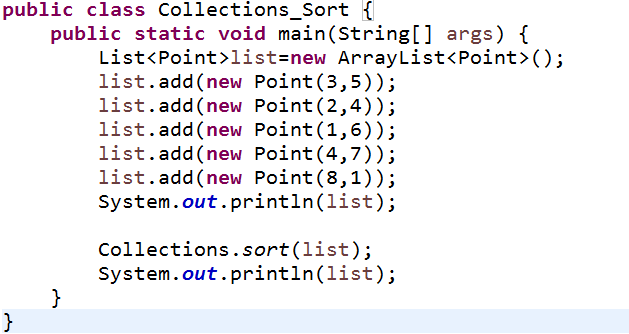
注: 该方法在排序自定义元素时,对我们的程序有侵入性

侵入性:当我们使用一个功能时,该功能要求我们为其修改的代码越多,侵入性越强高侵入

性不利于程序扩展,应尽量避免

侵入性的弊端是一旦该功能不再需要时,之前修改的代码都不具意义,增加后期维护成本





1. **Collections重载的sort方法(两个参数)?**

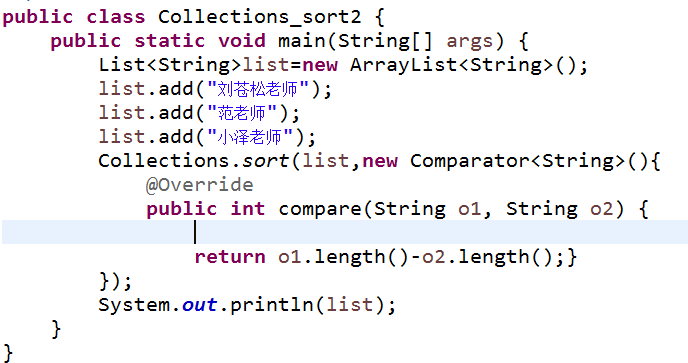
static void sort(List list,Comparator c):第一个参数是集合,第二个参数是比较器

该方法会根据给定的比较器中定义的比较规则对集合元素比较大小后排序

该方法解决两个问题:

1:由于提供了额外比较器,不再要求元素必须实现Comparable接口,较少侵入性

2:不使用元素自身比较规则,可以避免由于元素比较规则不满足实际排序需求带来的问题



1. **集合遍历之迭代器?(面)**

遍历集合使用统一的方式:迭代器:

Iterator iterator():集合提供的该方法可以获取一个可遍历当前集合的迭代器实现类

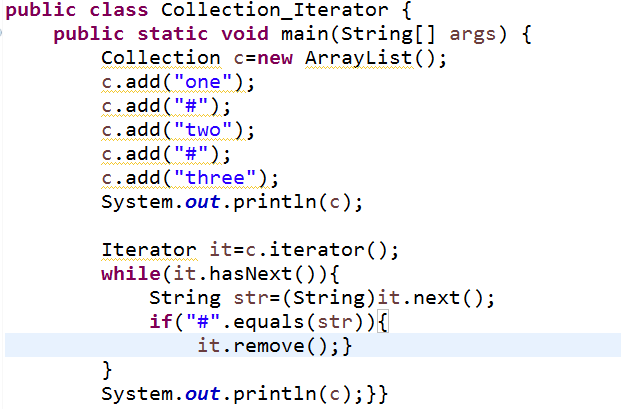
迭代器实现类中有几个常用方法:

boolean hasNext():判断集合中是否还有元素可以遍历

E next():获取下一个元素, 返回的是Object类型,需要强制转型

remove():迭代期间删除集合元素的方法

注: 迭代期间不能使用集合的remove,可以使用迭代器的remove

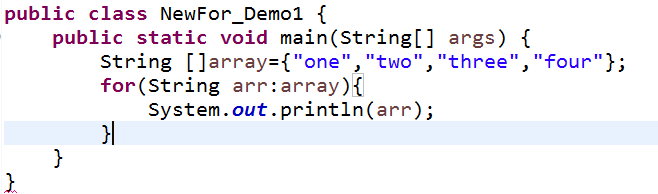


1. **增强for循环?(面)**

新循环,又称为:增强for循环,for each, 是Java1.5之后推出的一个新特性.

只能用来遍历数组和集合, 是编译器认可,而非虚拟机认可,编译器会将代码中的新循环遍历数组改变为传统for循环遍历,将新循环遍历集合改为迭代器遍历集合.

以数组为例:



1. **List集合有哪些常用方法?**

List接口继承自Collection接口, 是可重复集,并且有序. 提供了一系列支持使用下标操作元素的方法:

E get(index): 获取指定下标处的元素

E set(int index,E e):将给定的元素设置到指定位置处,返回值原位置元素

E remove(int index):删除并返回指定下标处的元素,返回为原位置的元素

boolean remove(Object o): 删除集合中与给定元素equals比较为true的元素

boolean add(E e):向集合内添加一个元素

void add(int index,E e): 在指定位置插入规定元素

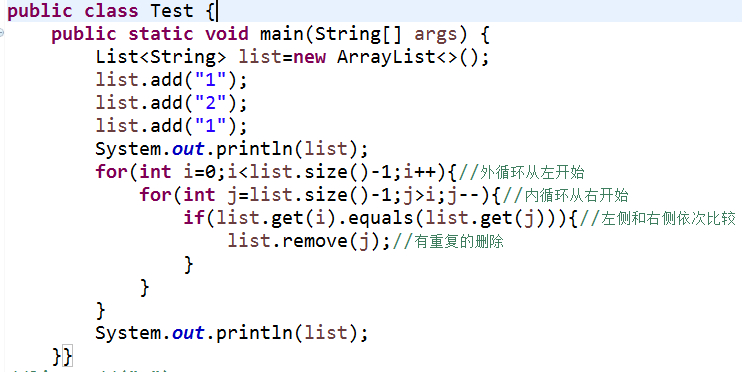
void clear():清空集合

List subList(int start,int end): 截取当前集合中指定范围的子集.

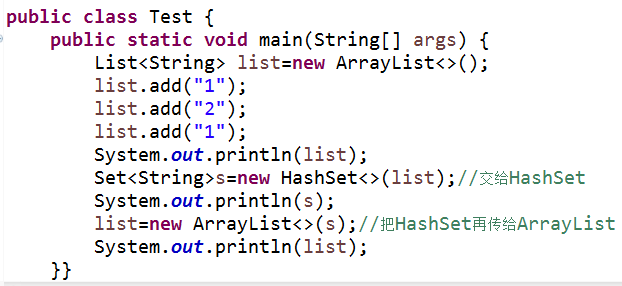
注: 对子集的操作就是对原集合中对应元素的操作

1. **怎么对ArrayList集合中的元素去重?(面)**
2. 使用遍历比较方式:两层循环,外层从左侧,内层从右侧,左侧的每一个元素依次和右

侧开始的元素进行比较,有重复则删除



1. 把ArrayList传给HashSet,然后把去重后的HashSet再赋值给ArrayList



1. **Map集合有哪些常用方法?**

Value put(Key k,Value v): 将给定的Key - Value 对存入到Map中返回值为被替换的value.注:Map不允许有重复的Key,若存入的key已经存在与Map中,则是替换value操作,返回值就是被替换的value,若存入的key在Map中不存在,则返回值为Null

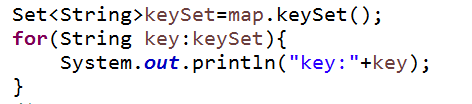
Value get(Key k): 根据给定的key获取对应的value.

Value remove(Key k): 根据给定的key删除这组键值对,返回值是该键值对的value若没有,则返回null.

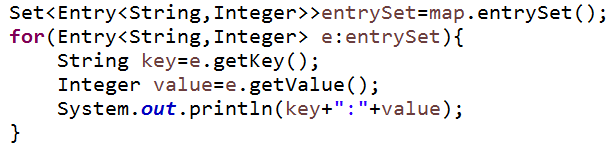
1. **遍历Map有几种方式?(面)**

一共三种方式:

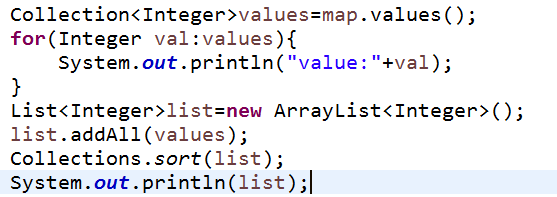
1) Set<k> keySet():遍历所有的key, 将当前Map中所有的key存入到一个Set集合后返回



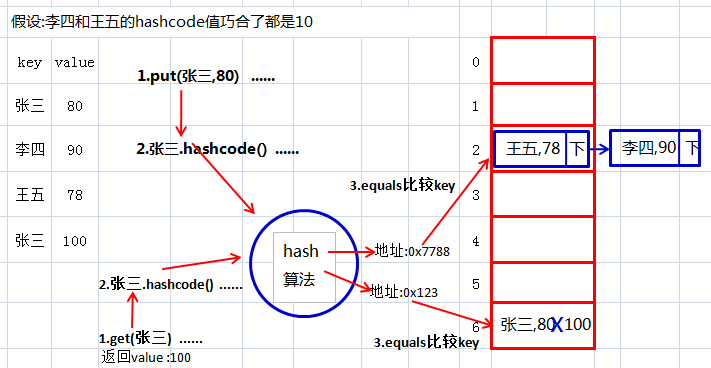
2) Set<Entry>entrySet():遍历每一组键值对, Entry是Map的内部类,一个Entry就表示当前Map对象的实例,其每一个实例用于表示Map中的一组键值对



3) Collection values():遍历所有的value, 将当前Map中所有value存入一个集合中并返回



1. **HashMap内部实现原理?(面)**



1)当调用put方法往HashMap中存键值对时,会把key的hashCode值交给Hash

算法进行计算,算出来一个地址值,该地址值决定着该键值对存在数组中的那个位置.

2)找到该存储地址位置时会在调用equals方法来比较是否存在相同的key,如果不存在则把该键值对存在该位置,如果存在则替换其value值.

3)如果在调用put方法时传入key不相同,但是key的hashCode值交给Hash算法进行计算时得到了相同的地址值,那么就会产生链表.会在同一个地址上存入两个key不相同的键值对,先存入的键值对被放到链表的尾部,后存入放到链表首部.

4)产生链表会影响查询效率,所以被存入的键值对对象都会重写equals和hashcode方法,但链表不能完全避免,原因是可能存在巧合key不相同但HashCode值相同.

5)当调用get方法时传入key,把key的hashCode值交给Hash算法进行计算,算出来一个地址值,通过该地址找到元素后在调用equals方法来比较key是否存在,若存在则返回其value值.

1. **重写equals方法的规则?(面)**

1).如果传过来的obj是null那么就返回false

2).如果obj==this,传过来的obj和当前对象时同一个对象,返回true

3).如果obj 和当前对象是相同类型,则返回当前对象和obj的属性的比较,否则返回false

public boolean equals(Object obj){

if(obj==null){

return false;}

if(obj==this){

return true;}

if(obj instanceof Point){

Point p=(Point)obj;

return this.x==p.x&&this.y==p.y;}

return false;}

1. **为什么要重写hashcode()和equals()?(面)**

打个比方,一个名叫张三的人去住酒店,在前台登记完名字就去了99层100号房间,此时警察来前台找叫张三的这个人住在哪间房,经过查询,该酒店住宿的有50个叫张三的,需要遍历查询,查询起来很不方便.

那么就换另外一种登记方式,前台登记时登记身份证号,警察来前台找身份证号为发现身份证号也存在重复,经过哈希算法进行计算后相同的hashcode值被分到了一个房间然后产生链表,链表查询效率非常慢,然后警察找的时候也会遇到问题

那么只能换第三种登记方式了,前台登记时同时登记身份证号和名字,这样警察来找的时候同时按照两个条件去查,这样就能直接锁定要找的人在哪个房间.

在程序中:登记的前台好比哈希算法,名字是对比好比equals方法,身份证号的对比好比hashcode方法只有equals和hashcode都满足的时候才能确保是同一个对象.

当我们重写一个类的equals方法时就应当连同重写

hashcode方法.并且两个方法应满足:

1:一致性,即:当两个对象equals比较为true,那么hashcode值应当相等.反之亦然.

因为当两个对象hashcode值相等,但是equals比较为false,那么在HashMap中

会产生链表,影响查询性能.

2:成对重写,即重写equals就应当重写hashcode.

1. **File是对文件做哪些操作?有哪些常用方法?**

File只是访问文件属性,比如:名字,可读,可写,修改时间,字节大小,是否可运行权等,但是不能访问文件内部的内容.另外还可以通过File创建文件和目录.

File file=new File();可以传入文件路径及文件名来访问该文件,默认是当前项目根目录.

访问文件属性常用方法:

file.getName():获取文件名

file.length():获取文件字节长度

file.canRead():是否可读

file.canWrite():是否可写

file.canExecute():是否可运行权

file.lastModified():最后修改时间

file.isHidden():是否被隐藏

创建文件常用方法:

file.exists():判断文件是否存在:true表示存在

file.createNewFile():创建新文件,依据file中的路径和文件名进行创建

file.delete():删除文件

创建目录常用方法:

File dir=new File();可以传入一个文件路径及目录名,不加文件类型后缀就是目录

dir.exists():判断目录是否存在:true表示存在

dir.mkdir():创建目录,注:mkdir也是Linucs的创建目录命令

dir.getName():获取目录名

dir.delete():删除目录

dir.mkdirs();创建嵌套多级目录:目录名为最内层的目录名

注意:删除多级目录时:

由于delete()删除目录时需要保证目录内部使空的才可以,如果要删除多级目录时

需要先将内部的文件或目录删除

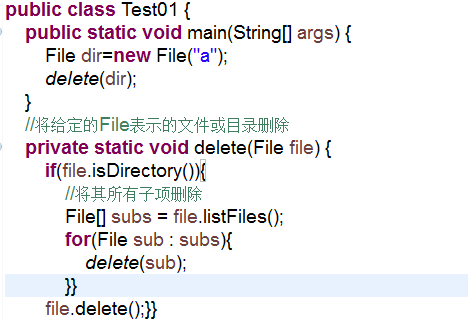
dir.listFiles():获取一个目录下的所有子项

dir.isDirectory():判断当前File表示的是否为目录

dir.isFile():判断当前File表示的是否为文件

例如:删除多级目录先判断目录内部是文件还是目录,如果是目录则可以获取其内部

所以子项,是文件就直接删除



文件过滤器:

FileFilter filter=new FileFilter();//需重写accept(File file)方法:

public boolean accept(File file) {

String name=file.getName();

System.out.println("正在过滤:"+name);

return name.startsWith(".");}

dir.listFiles(filter):方法允许传入一个文件过滤器FileFilter,然后将当前目录中满足

过滤器要求的所有子项返回

1. **RandomAccessFile是对文件做哪些操作?常用方法有哪些?**

RandomAccessFile用来读写文件数据内容的类, 是基于指针进行读写,即:总是在执行当前位置读写字节,有两种常用模式:

1):只读模式 r: RandomAccessFile raf=new RandomAccessFile("raf.dat","r");

2).读写模式rw : RandomAccessFile raf=new RandomAccessFile("raf.dat","rw");

第一个参数:文件路径及文件名,第二个参数:模式r rw

注意:

读写模式下,如果在读取文件的时目录下没有该文件,他会自动创建一个该名字的文件.

只读模式下不会创建,并会抛出异常

常用方法:

void write(int d):向文件中写入1个字节,写的内容是指定int值对应2进制的低八位int read():读取一个字节,并以10进制的int值返回,返回的数字应在0-255之间,若返回-1,则表示读到文件末尾了,每次读取后自动移动文件指针, 准备下次读取

int read(byte[]data):一次性尝试读取给定的字节数组总长度的字节量,并将读取到的

字节存入到该数组中,返回值为实际读到的字节量,若返回值为-1,表示本次没有读到任何数据(文件末尾)

void write(byte[]data):将给定的字节数组中所有字节一次性写出

void write(byte[]data,int s,int len):将给定的字节数组从下标s处的连续len个字节一次性写出

byte[] getBytes():把当前字符串按照系统默认字符集转换成一组字节.

byte[] getBytes(String csn): 按照指定的字符集将当前字符串转换为一组字节常用字符集名称：GBK,UTF-8

long getFilePointer():获取当前指针的位置

void seek(long pos):将指针移动到指定位置

void writeInt(int d):将int值4个字节一次性写出，底层还是位移运算操作了4次.

另外还有重载的写出double,byte等基本类型的方法,此处省略

**怎么将读到的字节转成字符串:**

String构造方法支持4个参数: 每次读取的字节量,从0开始读取，读取len个字节，使用UTF-8编码,如下:

String str=new String(data,0,len,"UTF-8");把读到字节按照指定格式转换为字符串

1. **IO流是做什么的?流的分类?常用的流有哪些?(面)**

IO流指的是输入输出流，用来处理设备上的数据。这里的设备指硬盘，内存，键盘录入，网络传输等。

1)根据数据的流向来分：

输出流：是用来写数据的，是由程序（内存）--->外界设备

输入流：是用来读数据的，是由外界设备--->程序（内存）

2)根据流数据的格式来分：

字节流：处理声音或者图片等二进制的数据的流，比如InputStream

字符流：处理文本数据（如txt文件）的流，比如InputStreamReader

3)根据流数据的包装过程来分：

节点流:又称为低级流,特点是:数据源明确,真实负责读写数据的流

处理流:又称为高级流,特点是:不能单独存在(没意义),用来处理其他流,所有高级流都封装了某些特定功能的读写操作,目的是简化我们的读写操作具体的流:

1)字节流：

InputStream与OutputStream

InputStream是所有字节输入流的父类，其定义了基础的读取方法

OutputStream是所有字节输出流的父类，其定义了基础的写出方法

包含：

文件流：FileOutputStream是文件的字节输出流，我们使用该流可以以字节为单位将

数据写入文件(默认覆盖模式,第二个参数设施为true可改变为追加模式)。

FileInputStream是文件的字节输入流，我们使用该流可以以字节为单位读取

文件内容。

缓冲流：BufferedOutputStream缓冲字节输出流,需使用flush方法将缓存的字节写

出或者在关闭流时一次性写出.

BufferedInputStream是缓冲字节输入流.

对象流：ObjectOutputStream是用来对对象进行序列化的输出流。

ObjectInputStream是用来对对象进行反序列化的输入流。

2)字符流：Reader和Writer

Reader是所有字符输入流的父类。

Writer是所有字符输出流的父类。

字符流是以字符(char)为单位读写数据的。一次处理一个unicode。字符流都是高级

流，其底层都是依靠字节流进行读写数据的。

包含：

转换流：InputStreamReader：字符输入流，使用该流可以设置字符集

OutputStreamWriter:字符输出流，使用该流可以设置字符集

字符缓冲流:

PrintWriter具有自动行刷新的缓冲该字符输出流,需第二个参数设置为true

BufferedReader是缓冲字符输入流

注: PrintWriter写出字符串时我们通常不使用Writer提供的write()相关方法，而是使用重载的print和println方法.

BufferedReader读取字符串时可以使用readLine()连续读取一行字符串，直到读取到换行符为止，返回的字符串中不包含该换行符,未读到返回null



1. **什么是序列化和反序列化?在IO流中哪种流是处理序列化和反序列化的?(面)**

在java中,可以将任何的对象都转换成一组字节存储到硬盘等载体上,这个过程称为序列化,反序列化就是把一组字节读取出来转换成原对象的过程.

在序列化和反序列化的过程中需要让对象的实体类实现序列化接口Serializable,生成序列化ID,通过该ID进行序列化和反序列化,并且在转换过程中需要保证ID的一致性.

在IO流中,负责处理序列化和反序列化的两个流分别是:字节对象流中的:

对象输出流:ObjectOutputStream:序列化

对象输入流:ObjectInputStream:反序列化

1. **java中的异常处理机制?(面)**

当程序中抛出一个异常后，程序从出现异常的代码处跳出，JVM检测寻找和try关键字匹配的处理该异常的catch块，如果找到，将控制权交到catch块中的代码，从catch中继续往下执行，try块中发生异常的代码之后不再执行。如果没有找到处理该异常的catch块，在所有的finally块代码被执行和当前线程的所属的ThreadGroup(线程组)的uncaughtException(未捕获的异常)方法被调用后，出现异常的当前线程被中止。

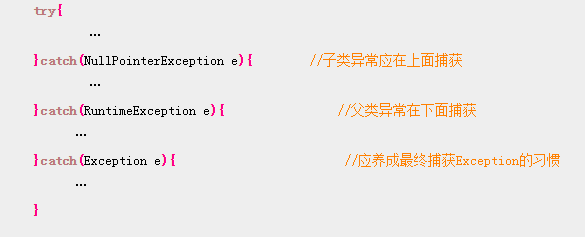
1. **java异常处理时try...catch...finally各表示什么?(面)**

try:将一块可能发生异常的代码使用大括号包起来，称为监控区域.

catch: try中发生的异常处理该异常的代码块

finally: 为异常处理提供一个统一的出口,不管是否发生异常一定会执行的代码块,可以用来关闭文件等操作.

try后面可以跟catch和finally,但至少跟其中之一,不能单独存在,也可以跟多个catch,多个catch时异常类型应从小到大排序(子类型在上,父类型在下顺序).



1. **java中异常的分类?(面)**

Java异常结构中定义有Throwable(可抛出)类，Exceotion(异常)和Error(错误)是其派生的两个子类。

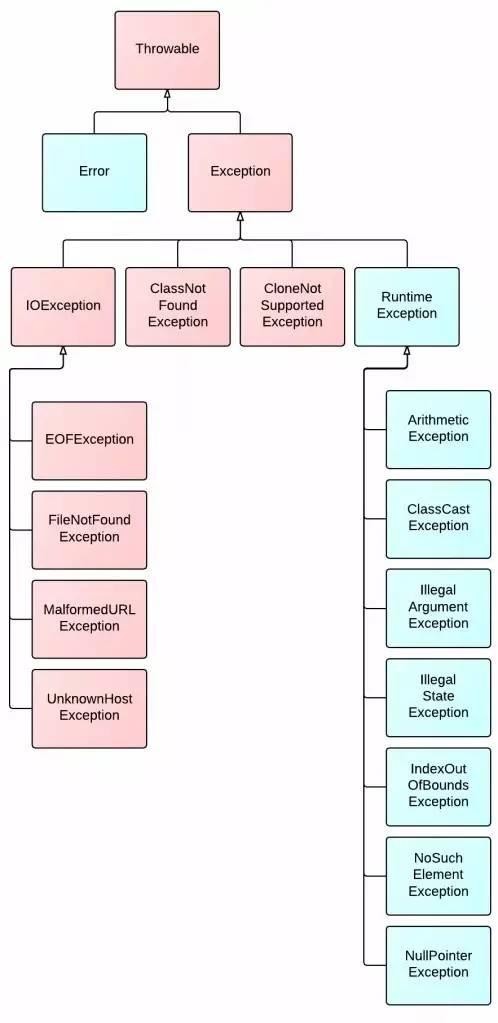
**Exception（异常）:**表示由于网络故障、文件损坏、设备错误、用户输入非法等情况导致的异常，这类异常是可以通过Java异常捕获机制处理的。

**Error（错误）:** 表示程序无法处理的错误，表示运行应用程序中较严重问题，例如:JVM内存溢出等。

**Exception（异常）:又分为检查型异常和运行时异常**

**检查型异常:需要经过编辑器检查处理的异常,对于声明抛出异常的方法,必须经过强制处理或者自己捕获处理,否则编译无法通过,此类异常继承Exception,典型的检查性异常有IOException,ClassNotFoundException等.**

**运行时异常:编辑期间编译器不做检查,在运行时才会抛出的异常,此类异常继承RuntimeException.运行时异常有NullPointerException,** **NumberFormatException,** **ArrayIndexOutOfBoundsException等.**



1. **try、catch、finally语句块的执行顺序(无return情况下)?(面)**

1)当try没有捕获到异常时：

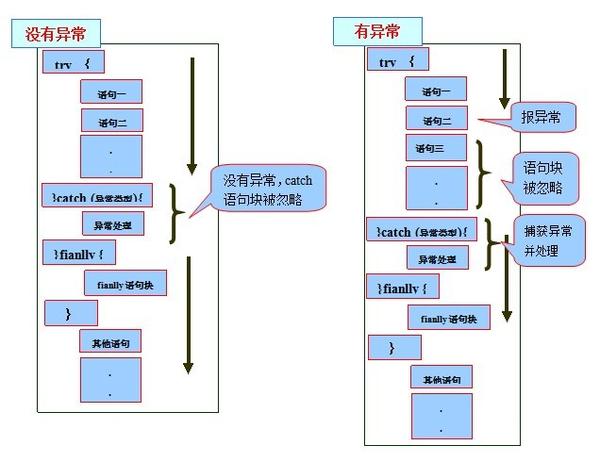
try语句块中的语句逐一被执行，程序将跳过catch语句块执行finally语句块和其后的语句；

2)当try捕获到异常，catch语句块里没有处理此异常的情况：

当try语句块里的某条语句出现异常时，而没有处理此异常的catch语句块时，此异常将会抛给JVM处理，finally语句块里的语句还是会被执行，但finally语句块后的语句不会被执行；

3)当try捕获到异常，catch语句块里有处理此异常的情况：

在try语句块中是按照顺序来执行的，当执行到某一条语句出现异常时，程序将跳到catch语句块，并与catch语句块逐一匹配，找到与之对应的处理程序，其他的catch语句块将不会被执行，而try语句块中，出现异常之后的语句也不会被执行，catch语句块执行完后，执行finally语句块里的语句，最后执行finally语句块后的语句；



1. **try、catch、finally语句块的执行顺序(有return返回值的情况)?(面)**

1)try里有return ,catch和finally中无return,未发生异常:

try中return前的语句执行完毕后跳过return语句和catch语句块,先执行finally块中的代码,然后在回到try中执行return语句结束当前方法.

2)try里有return, catch中有return,finally中无return, 发生异常:

try中发生异常的代码之后不再执行,进入catch块中处理异常,但暂时先不执行catch中的return,先进入finally块执行finally块中代码然后再回到catch块中执行return语句结束方法.

3)try里有return,catch中有return,finally中有return,发生异常:

try中发生异常的代码之后不在执行,进入catch块中处理异常,然后执行catch中的return取出返回值将值临时保存在栈中但不是执行return结束方法,然后进入finally块中执行代码最后执行finally中取出返回值覆盖try中的return取出的返回值结束方法.

4)try里有return,cacth中有return,finally中有return,没有发生异常

try中代码包括return都会执行,但是并不会从try中return结束该方法,而是执行try中return取出返回值将值临时保存在栈中,然后会向后继续执行finally块中代码并执行finally中return取出返回值覆盖try中的return取出的返回值结束方法.

可参考如下代码并使用DEBUG方式进行各种情况测试观察结果:

public static void main(String[] args) {

System.out.println(test()); }

public static int test(){

int num = 10;

try{

System.out.println("try");

return num += 80;

}catch(Exception e){

System.out.println("error");

}finally{

if (num > 20){

System.out.println("num>20 : " + num); }

System.out.println("finally");

num = 100;

return num; } }

注意:虽然3和4实现在finally中加入return并未报错,但是不建议在finally中加return语句.,会造成两个如下后果:

1) 如果catch块中捕获了异常, 并且在catch块中将该异常throw给上级调用者进行处理, 但finally中return了, 那么catch块中的throw就失效了, 上级方法调用者是捕获不到异常的.

2) finally块中的return语句会覆盖前面的return语句（try块、catch块中的return语句）

1. **try{}里有一个return语句,那么紧跟在整个try后的finally{}里的代码会不会执行,什么时候执行,在return前还是return后?(面)**

会执行,在return前执行.请参考上一题的分析过程.

1. **throw和throws的区别?(面)**

throws:是用来声明一个方法可能抛出的所有异常信息,多个异常之间逗号隔开,写在方 法上, 通常不用显示的捕获异常，可由系统自动将所有捕获的异常信息抛给上级调用者.

throw:是指抛出的一个具体的异常类型,写在方法的内部, 需要用户自己捕获相关的异

常，而后在对其进行相关包装，最后在将包装后的异常信息抛出.

假设自定义了一个IllegalAgeException异常,此处省略:

public class Exception\_throw {

public static void main(String[] args){

Person p=new Person();

try {

p.setAge(10);

} catch (IllegalAgeException e) {

e.printStackTrace();

System.out.println("出错了!");

}

System.out.println("年龄是:"+p.getAge()); }}

class Person{

private int age;

public int getAge(){

return age;}

public void setAge(int age) throws IllegalAgeException{

if(age<0||age>100){

throw new IllegalAgeException("年龄不合法");}

this.age=age;

}}

1. **子类方法重写父类含有throws异常抛出声明的方法时throws重写规则?(面)**

假设有一个父类:

public class Aoo {

public void dosome() throws IOException,AWTException{

}}

那么子类继承父类后重写父类方法时:

1) 允许不再抛出任何异常

public void dosome(){}

}

2) 允许抛出部分异常

public void dosome() throws IOException{}

}

3) 允许抛出子类型异常

public void dosome() throws FileNotFoundException{

}

4) 不允许抛出额外异常

public void dosome() throws SQLException{

}

5) 不允许抛出父类型异常

public void dosome() throws Exception{

}

1. **Java常见异常有哪些,请至少说出5个异常及造成出现该异常的原因?(面)**

### ****RuntimeException子类:****

### ArrayIndexOutOfBoundsException:  数组下标越界异常。当对数组的下标为负数或大于等于数组大小时抛出。

### ArithmeticException: 算术条件异常。如：整数除零等。

### NullPointerException:空指针异常, 调用null对象的实例方法、访问null对象的属性、计算null对象的长度、使用throw语句抛出null等等.

ClassNotFoundException:  找不到类异常。根据字符串形式的类名在遍CLASSPAH之后找不到对应名称的class文件时，抛出该异常。

NegativeArraySizeException  数组长度为负异常

ArrayStoreException 数组中包含不兼容的值抛出的异常

SecurityException 安全性异常

IllegalArgumentException 非法参数异常

ClassCastException 类型转换异常类

ArrayStoreException 数组中包含不兼容的值抛出的异常

SQLException 操作数据库异常类

NoSuchFieldException 字段未找到异常

NoSuchMethodException 方法未找到抛出的异常

NumberFormatException 字符串转换为数字抛出的异常

StringIndexOutOfBoundsException 字符串索引超出范围抛出的异常

IllegalAccessException 不允许访问某类异常

* **IOException的子类:**

IOException：操作输入流和输出流时可能出现的异常。

EOFException 文件已结束异常

FileNotFoundException 文件未找到异常

1. **怎么写一个自定义异常?(面)**

创建一个类继承Exception或者RuntimeException

继承Exception时自定义异常属于检查型异常,编译器会做检查,会主动提示抛出.

继承RuntimeException时自定义异常属于运行时异常,编译期不会检查,运行时会抛出相关异常.

1. **Throwable中常用API?**

getCause()：返回抛出异常的原因。如果 cause 不存在或未知，则返回 null。

getMessage()：返回异常的消息信息。

printStackTrace()：输出错误信息，用来跟踪异常事件发生时执行堆栈的内容.

1. **进程和线程的区别?(面)**

进程（process）就是一块包含了某些资源的内存区域,操作系统又利用进程将进程的内存区域分为多个不同的功能模块,而这些功能模块被称为线程,一个程序就是一个进行,而程序的不同功能就是不同的线程,比如:QQ是一个进行,QQ的添加好友功能,聊天功能,群功能等都是不同的线程.进程中的多个线程是"同时"(切换)并发执行.

进程中包含多个线程,至少包含一个线程,线程跟随进行,是进程中的一个个工作单元.

在一个程序中如果想让"同时"执行很多个操作,那么就需要用到多线程,比如玩游戏,背景音乐,人物移动,战斗都是"同时"执行的,可以采用多线程.

1. **多线程的运行状态有哪些?(面)**

New(新建)：当我们创建一个线程时该线程并没有纳入线程调度处于一个new状态。

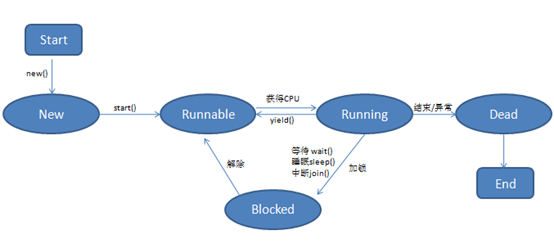
Runnable(就绪):当调用线程的start方法后，该线程纳入线程调度的控制，其处于一个可运行状态，等待分配时间片段以并发运行。

Running(运行):当该线程被分配到时间片段后被CPU运行，该线程处于running状态。

Blocked(阻塞):当线程在运行过程中可能会出现阻塞现象，比如等待用户输入信息等。但阻塞状态不是百分百出现的，具体要看代码中是否有相关需求。

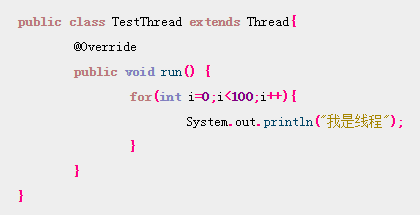
Dead(结束):当线程的任务全部运行完毕，或在运行过程中抛出了一个未捕获的异常，那么线程结束，等待GC回收.

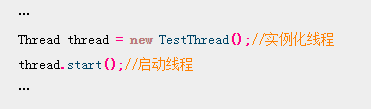
当使用new线程创建完毕后调用start方法后线程进入就绪状态,线程调度系统将就绪状态的线程转为运行状态,遇到synchronized语句时,由运行状态转为阻塞,当该线程获得synchronized对象锁后,由阻塞状态转为运行,在这种情况下可以调用wait 方法转为挂起状态,当线程关联的代码执行完毕后,线程变为结束状态.



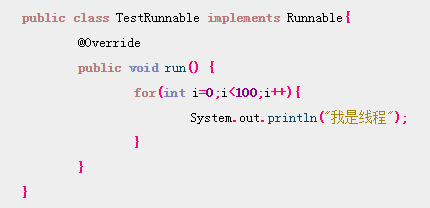
1. **创建线程的几种方式,有什么区别?(面)**

1)继承Thread类重写run方法





2)实现Runnable接口重写run方法





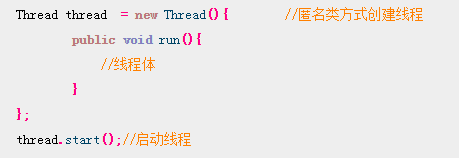
实现Runnable接口的好处:

1).java仅支持单继承,实现接口可以把继承的机会留给其他类,实现接口更灵活一些

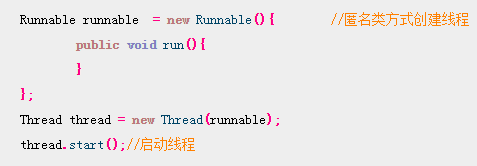
2). 可以将线程与线程要执行的任务分离开减少耦合度

另外:一个线程仅需要一个实例时可以使用匿名内部类创建,这种方式有助于简化代码

继承Thread类:



实现Runnable接口:



1. **线程并发(异步)和同步的区别?(面)**

线程的并发是指线程在执行时是交替执行的,一会执行a线程,一会执行b线程,一会执行c线程,而这些操作是线程调度分配给线程的时间片段决定的,每个线程被分配的时间都不是固定的, 但多个线程并发读写同一个临界资源时候会发生"线程并发安全问题“

注: 常见的临界资源: 多线程共享实例变量 多线程共享静态公共变量

比如:一家四口去饭店吃饭,菜以上来,但是只有一双筷子,他们想了个办法,每人夹一口菜然后把筷子传递给下一个人,轮流使用筷子去夹菜.

线程的同步指的是线程的执行不是交替的,而是有顺序的执行,等a线程执行完毕后,b线程才能执行,需要使用synchronized同步锁给对象上锁.

比如: 一家四口去饭店吃饭,菜以上来,但是只有一双筷子,他们想了个办法,一个人先吃饱,然后筷子传递给下一个人,依次类推.当一个人使用筷子的时候其他人不能使用.

1. **多线程有几种实现方法?同步有几种实现方式?(面)**

多线程有两种实现方法:分别是继承Thread类和实现Runnable接口.

同步实现由两种方式: 1)synchronized 2)wait与notify配合使用

synchronized是同步锁,可以实现同步效果.

线程调用wait方法在挂起时会释放掉锁,而被notify唤醒时会去对象锁定池重新获得对象锁,所以也能达到同步效果.

1. **启动一个线程是用run()还是start()?(面)**

启动线程调用start()方法是线程进入就绪状态,之后可以被线程调度调度为运行状态.

而run()是线程要具体执行的关联代码,要线程具体要执行的工作.

1. **线程中常用API?**

static void yield():使当前线程主动让出当次CPU时间片回到Runnable状态，等待分配时间片。

void join():当前线程会开始阻塞，直到调用join方法的线程中run方法执行完毕.

1. **线程的优先级?**

线程的切换是由线程调度控制的,我们无权干涉,但是提高线程的优先级来最大程度的改善线程获取时间片的几率。

线程的优先级被划分为10级，值分别为1-10，其中1最低，10最高.

可以三个常量表示最大,最小,其中默认优先级为5.

Thread.MIN\_PRIORITY,

Thread.MAX\_PRIORITY,

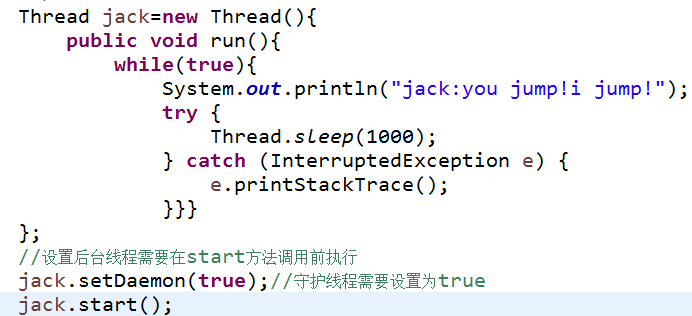
Thread.NORM\_PRIORITY

设置优先级的方法为:

void setPriority(int priority)

1. **守护线程? (面)**

守护线程又称为后台线程, 与普通线程的一个最主要区别在于结束时机. 当一个进程结束时,所有运行的守护线程都会被强制结束. 当一个进程中的所有前台(普通)线程结束时,进程就会结束. GC就是运行在一个守护线程上的。



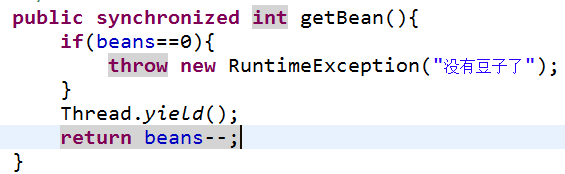
注意: 设置后台线程需要在start方法调用前执行.

1. **synchronized同步锁的不同使用场景?(面)**

java中有一个关键字名为:synchronized,该关键字是同步锁，用于将某段代码变为同步操作，从而解决线程并发安全问题.

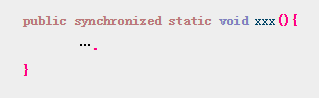
**synchronized加在普通方法上**:当一个普通方法是用synchronized修饰后,该方法称为:"同步方法" 即:多个线程不能"同时"执行该方法,只能有先后顺序同步执行.锁定的是当前对象,也就是this.

注:只针对同一个对象访问同一个方法会产生同步,两个不同的对象访问各自的同步方法不会产生同步.



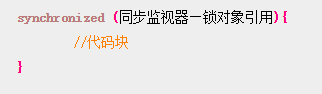
**synchronized加在静态方法上:**当一个静态方法是用synchronized修饰后, 该方法一定具有同步效果,因为只有1份,跟对象无关, 实际上上锁的对象为当前方法所属类的类对象,即:Class的一个实例.

注:与普通方法加锁有区别,在静态方法上加锁由于锁的是当前类,所以通过该类创建出来的多个对象访问该静态方法都会出现同步.



**synchronized代码块:** 为了有效缩小同步范围,来提高多线程并发执行效率,可以使用同步块,同步块写在方法内部,同步方法内的某一段代码.

注:同步块需要自行指定"同步监视器",即:上锁的对象,只要保证需要排队执行的多个线程看到是同一个对象即可!,一般写的是this

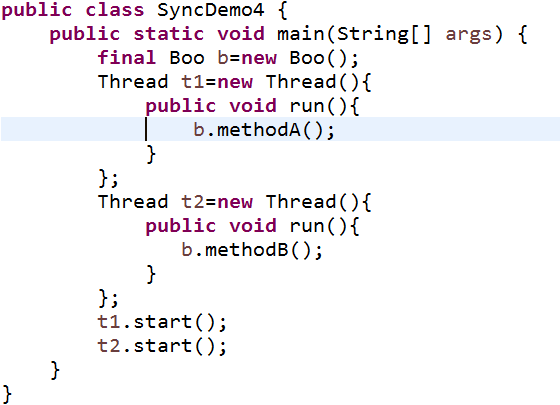


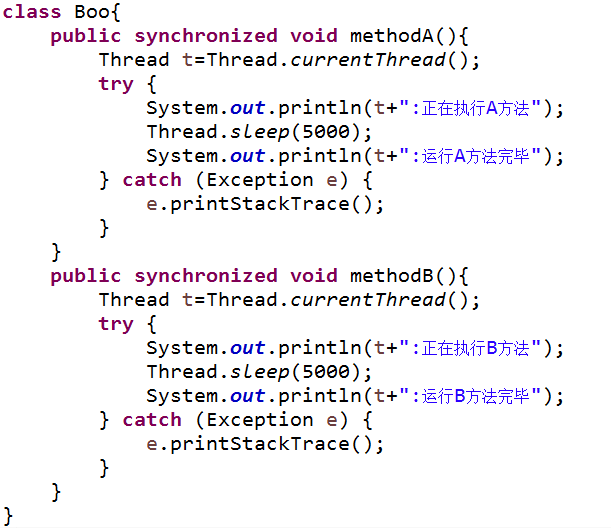
1. **线程中的互斥锁?(面)**

Synchronized也是互斥锁: 当修饰的是两段不同的代码,但是同步监视器对象是同一个的时候,那么就具有了互斥性.

如下案例:

Boo类中有两个Synchronized同时修饰2个不同的方法,锁定的都是当前对象this,在访问该对象的两个不同同步方法时都会将当前对象锁定,互相牵制,谁也不让谁,必须等一个执行完另外一个才能执行.

****

****

注意:上述案例如果Boo里有一个synchronized修饰的是静态方法,那么将不会产生互斥,因为静态方法用synchronized修饰锁定的是当前.class,而普通方法使用synchronized修饰锁定的是当前对象,也就是this, 同步监视器监视的对象并非同一个,所以不会产生互斥.

1. **多线程死锁?(面)**

理解多线程死锁前先理解这个小故事:

假设有A、B、C、D四个人在一起吃饭，每个人左右各有一只筷子。所以，这其中要是有一个人想吃饭，他必须首先拿起左边的筷子，再拿起右边的筷子。现在，我们让所有的人同时开始吃饭。那么就很有可能出现这种情况。每个人都拿起了左边的筷子，或者每个人都拿起了右边的筷子，为了吃饭，他们现在都在等另外一只筷子。此时每个人都想吃饭，同时每个人都不想放弃自己已经得到的一那只筷子。所以，事实上大家都吃不了饭。

**死锁:** 就是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。

线程死锁可能发生在以下的情况：

1)当两个线程相互调用Thread.join();

2)当两个线程使用嵌套的同步块时，一个线程占用了另一个线程的必需的锁，互相等待时被阻塞，就有可能出现死锁。

死锁一般都是由于对共享资源的竞争所引起的。但对共享资源的竞争又不一定就会发生死锁,死锁的发生必需满足4个必要条件：

* 互斥: 所谓互斥就是进程在某一时间内独占资源。
* 等待/持有: 一个进程因请求资源而阻塞时，对已获得的资源保持不放。
* 非抢占: 进程已获得资源，在末使用完之前，不能强行剥夺。
* 形成等待环: 若干进程之间形成一种头尾相接的循环等待资源关系。

**如何解决死锁:**可以从死锁的四个条件去解决，只要破坏了一个必要条件，那么死锁问题就解决了,在java中使用多线程的时候一定要考虑是否有死锁的问题.

1. **sleep和wait这两个方法有什么区别?(面)**

sleep方法是Thread类的方法,可以在传入指定毫秒数让线程阻塞给定的毫秒数,阻塞期间不会释放掉对象锁,只是让出cup时间片给其他线程执行,一旦给定的毫秒数时间到了会再次去争抢时间片,运行线程.

wait方法是Object类的方法,调用该方法可以让线程处于挂起状态, 让出cpu给其他线程执行,挂起状态下会释放掉对象锁,进入对象等待锁定池,当调用notifiy或notiflyAll方法唤醒该线程,让线程进入对象锁定池重新获得对象锁,然后进入runnable状态重新等待cup分配时间片.

注意: 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。

1. **当一个线程进入一个对象的synchronized方法后,其他线程是否可进入此对象的其他方法?(面)**

该问题需要分不同的情况去解释:

1)这个对象的其他方法前是否加了synchronized关键字,如果加了,那么会产生互斥,如果没有加,则可以访问.

2)如果这个方法内部调用了wait方法,则可以进入其他synchronized修饰的方法,因为wait会释放掉对象锁.

3)如果其他方法都加了synchronized修饰,并且内部没有调用wait方法,则不能,因为多个方法有synchronized产生互斥.

4)如果其他方法是static类型方法,则可以访问,静态方法使用synchronized修饰锁定的是当前.class,而普通方法锁定的是当前对象,即this,两者锁定的对象不是同一个,所以不能同步.

1. **线程池有几种?**

当一个程序中大量创建线程，并在任务结束后销毁，会给系统带来过度消耗资源，以及过度切换线程的危险，可能导致系统崩溃,此情况可以使用线程池来解决。

线程池有两个主要作用: 1)控制线程数量 2)重用线程

线程池的执行过程: 首先创建一些线程，它们的集合称为线程池，当服务器接受到一个客户请求后，就从线程池中取出一个空闲的线程为之服务，服务完后不关闭该线程，而是将该线程还回到线程池中。

ExecutorService是java提供的用于管理线程池的类。

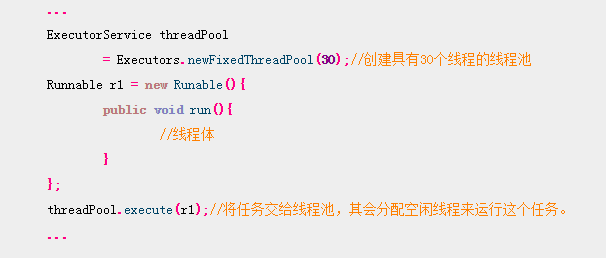
Executors.newCachedThreadPool():创建一个可根据需要创建新线程的线程池，但是在以前构造的线程可用时将重用它们。

Executors.newFixedThreadPool(int nThreads): 创建一个可重用固定线程集合的线程池，以共享的无界队列方式来运行这些线程。

Executors.newScheduledThreadPool(int corePoolSize): 创建一个线程池，它可安排在给定延迟后运行命令或者定期地执行。

Executors.newSingleThreadExecutor():创建一个使用单个 worker 线程的 Executor，以无界队列方式来运行该线程。

例如，创建一个有固定线程数量的线程池：



1. **什么是JAVA反射机制?(面)**

反射是Java 动态执行机制，可以实现动态执行功能：

1.反射提供了在运行时判断任意一个对象所属的类型并可以检查解析类型的内部结构。

2.反射可以动态加载类型，并能够动态创建对象

3.反射可以动态访问对象的属性。

4.反射可以动态执行对象的方法。

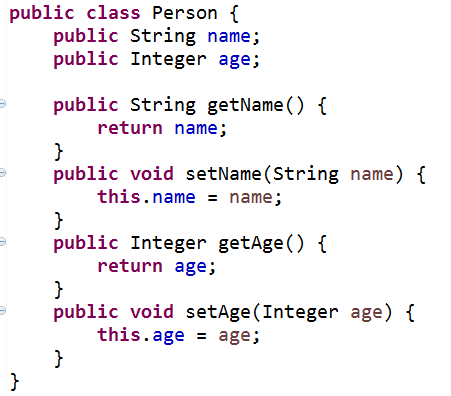
5.利用反射API还可以动态的访问，不可见的属性和方法,打破面向对象的封装性.

优点:可以动态执行！在运行期间根据业务功能动态执行方法、访问属性，最大限度发

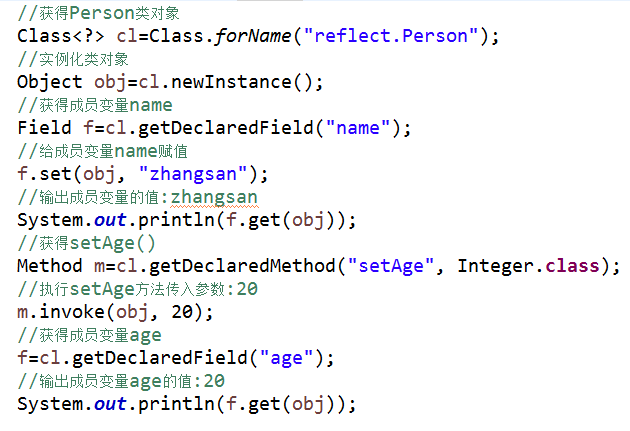
挥了java的灵活性。

缺点：对性能有影响，这类操作总是慢于直接执行java代码。

有一个实体类Person:



通过反射机制获取该类的类对象和内部的属性及方法,并给属性赋值和调用方法:

JDBC中，利用反射动态加载了数据库驱动程序。

Web服务器中利用反射调用了Sevlet的服务方法。

很多框架用到反射机制，注入属性，调用方法，如Hibernate、Struts2、Spring等

1. **用最有效率的方法算出2乘以8等于几?(面)**

可以使用二进制中的移位运算:<<

2 << 3

计算机中移位运算性能远远超过乘法,如果一个数n乘以2的m次幂,则可以优化为n<<m

8是2^3,所以向左移动3位,相当于计算公式:n\*2\*2\*2,如果是9能优化吗?不能,必须是2的m次幂,可以是2,4,8,16,32,64,128等。

二进制中的移位则是:

0000 0010

将其向左移动3位,则是在最右侧补3个0

0001 0000

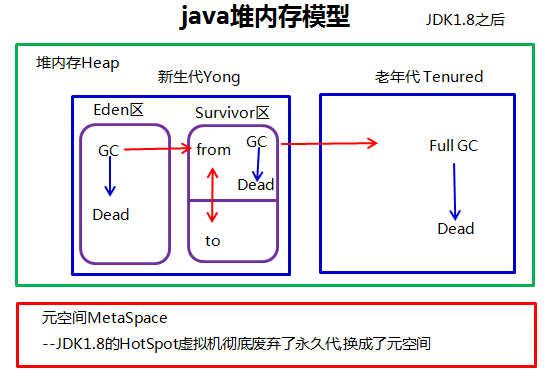
1. **JVM堆内存的分代管理和GC的垃圾回收机制(面)**

**堆内存:**

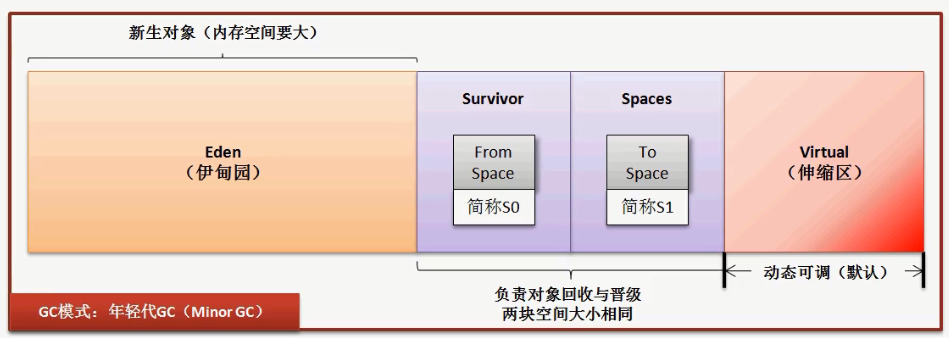
存放对象的实例,垃圾收集器管理的主要区域,分为新生代,老年代

新生代又分为Eden区域, Survivor区域

Survivor区域又分为form,to区域



新生代Eden:新对象和没有达到一定年龄的对象保存在该区域



老年代Tenured:新创建的超大对象,长时间被使用的对象,经过GC清理后还存活的对象

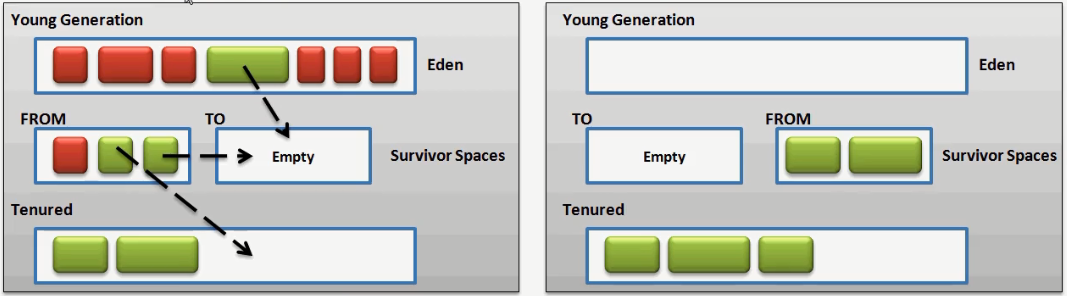
元空间:像一些方法中操作的参数,临时的对象都保存在元空间

1.创建一个新对象,会把这个新对象的实例放在新生代的Eden区域,当Eden区空间不足,无法保存创建新的对象时则会触发MinerGC进行清理

2.当经过一次MinorGC进行清理后Eden区还存活的一些对象会通过复制算法把它复制到Survivor区(存活区)的from 区(原to区).

3.Survivor区的两块区域是相同大小的两块区域,是可以互相交换的,交换以后form的叫to,to的叫from,交换的过程中会把其中一个的对象复制到另外一个,保证有一个是空的.

--经历一次MinorGC,如果Eden区有对象被保存到Survivor区的原to区时,然后对原from区域中进行一次清理,把清理后还依然存活的对象复制到原to区,然后把原to区更名为from区,把原form区更名为to区,这样就保证每次to区都是空的



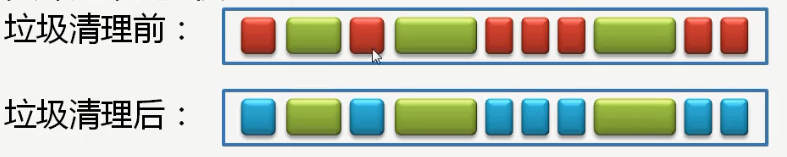
4.当原from区进行MinorGC进行清理后往原to区进行复制的时候,原to区复制一部分对象后满了的情况下,会将原form区的剩余对象复制到老年代区域.

--另外Survivor区中的对象每熬过一次MinorGC年龄就会增长一次,还可以通过设置年龄阈值参数:-XX:MaxTenuringThreshold,当年龄增到到我们设定的阈值时,将该对象直接复制到老年代中.

5.老年代区域没有可用空间时会触发Full GC,Full GC会扫描整个老年代区域将还存活的对象进行标记,然后把不存活的对象进行清除.

优点:空间内存活对象多的情况下,效率高.

缺点:直接回收不存活对象所占用的内存空间,回收后造成整个空间不连贯,产生内存碎片.



堆内存空间满的情况会抛出OutOfMemoryError异常.

**GC的垃圾回收策略:**

串行策略,并行策略,并发策略

**GC的垃圾回收算法:**

复制,标记-清除,标记-清除-整理