目录

[1、运算符顺序 2](#_Toc490150539)

[2、HashMap、HashTable和concurrentHashMap的区别 2](#_Toc490150540)

[3、String和char[] 4](#_Toc490150541)

[4、request 4](#_Toc490150542)

[5、继承关系中的初始化顺序 4](#_Toc490150543)

[6、数字与字符串的连接 5](#_Toc490150544)

[7、接口的实现 5](#_Toc490150545)

[8、System.out.println() 5](#_Toc490150546)

[9、Java集合 5](#_Toc490150547)

[10、Object类的方法 7](#_Toc490150548)

[11、二维数组的声明 8](#_Toc490150549)

[12、垃圾收集器 8](#_Toc490150550)

[13、值传递和引用传递 8](#_Toc490150551)

[14、接口的修饰词 11](#_Toc490150552)

[15、ASCII 11](#_Toc490150553)

[16、继承中的private 11](#_Toc490150554)

[17、ORM 12](#_Toc490150555)

[18、static 12](#_Toc490150556)

[19、Collection和Collections 12](#_Toc490150557)

[20、bean 13](#_Toc490150558)

[21、String的==和equals 13](#_Toc490150559)

[22、数组复制效率 14](#_Toc490150560)

[23、JSP表达式 14](#_Toc490150561)

[24、逻辑与、逻辑或 15](#_Toc490150562)

[25、重写 15](#_Toc490150563)

[26、Servlet 16](#_Toc490150564)

[27、线程 20](#_Toc490150565)

[28、异常 24](#_Toc490150566)

[29、同步 27](#_Toc490150567)

[30、request的forward和redirect 27](#_Toc490150568)

[31、数据类型的自动转换和强制转换 28](#_Toc490150569)

[32、Applet 30](#_Toc490150570)

[33、JVM 30](#_Toc490150571)

[34、向上转型和向下转型 30](#_Toc490150572)

[35、内存区域 31](#_Toc490150573)

[36、int、Integer 32](#_Toc490150574)

[37、JSP的error 33](#_Toc490150575)

[38、继承 33](#_Toc490150576)

[39、Java Web 34](#_Toc490150577)

[40、ArrayList、Vector、LinkedList 34](#_Toc490150578)

[41、Iterator 35](#_Toc490150579)

[42、volatile 36](#_Toc490150580)

[43、ResultSet 36](#_Toc490150581)

[44、interface的修饰词 36](#_Toc490150582)

[45、TLS 37](#_Toc490150583)

[46、抽象类和接口的区别 37](#_Toc490150584)

[47、垃圾回收 38](#_Toc490150585)

[48、依赖注入 40](#_Toc490150586)

[49、Struts 40](#_Toc490150587)

[50、构造方法 40](#_Toc490150588)

[51、自动拆箱与自动装箱 41](#_Toc490150589)

[52、静态变量 42](#_Toc490150590)

[53、合法的标识符 42](#_Toc490150591)

[54、Java基础 42](#_Toc490150592)

[55、重载 44](#_Toc490150593)

[56、匿名内部类 44](#_Toc490150594)

[57、构造器 44](#_Toc490150595)

[58、JSP的四大作用域 45](#_Toc490150596)

[59、JDBC的Statement类 45](#_Toc490150597)

[60、JSP分页代码 45](#_Toc490150598)

[61、类变量的初始化 45](#_Toc490150599)

[62、静态内部类 46](#_Toc490150600)

[63、包装类 46](#_Toc490150601)

[64、Socket 46](#_Toc490150602)

[65、for循环执行顺序 47](#_Toc490150603)

[66、正则表达式 48](#_Toc490150604)

[67、并发 49](#_Toc490150605)

[68、基本类型与引用类型 49](#_Toc490150606)

[69、内联函数 50](#_Toc490150607)

[70、Hibernate 50](#_Toc490150608)

[71、Java的关键字 51](#_Toc490150609)

1、运算符顺序

先乘除再加减，括号优先，%同乘除

单目>运算>移位>比较>按位>逻辑>三目>赋值

2、HashMap、HashTable和concurrentHashMap的区别

HashMap和Hashtable两个类都实现了Map接口，二者保存K-V对（key-value对）。

HashTable不允许null值(key和value都不可以),HashMap允许null值(key和value都可以)。

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。

由所有HashMap类的“collection 视图方法”所返回的迭代器都是快速失败的：在迭代器创建之后，如果从结构上对映射进行修改，除非通过迭代器本身的 remove 方法，其他任何时间任何方式的修改，迭代器都将抛出ConcurrentModificationException。Hashtable和HashMap的区别主要是前者是同步的，后者是快速失败机制保证。

hashMap在单线程中使用大大提高效率，在多线程的情况下使用hashTable来确保安全。hashTable中使用synchronized关键字来实现安全机制，但是synchronized是对整张hash表进行锁定即让线程独享整张hash表，在安全同时造成了浪费。concurrentHashMap采用分段加锁的机制来确保安全。

在这里帮大家总结一下hashMap和hashtable方面的知识点吧： 1. 关于HashMap的一些说法： a) HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。HashMap的底层结构是一个数组，数组中的每一项是一条链表。 b) HashMap的实例有俩个参数影响其性能： “初始容量” 和 装填因子。 c) HashMap实现不同步，线程不安全。 HashTable线程安全。 d) HashMap中的key-value都是存储在Entry中的。 e) HashMap可以存null键和null值，不保证元素的顺序恒久不变，它的底层使用的是数组和链表，通过hashCode()方法和equals方法保证键的唯一性 f) 解决冲突主要有三种方法：定址法，拉链法，再散列法。HashMap是采用拉链法解决哈希冲突的。 注： 链表法是将相同hash值的对象组成一个链表放在hash值对应的槽位； 用开放定址法解决冲突的做法是：当冲突发生时，使用某种探查(亦称探测)技术在散列表中形成一个探查(测)序列。 沿此序列逐个单元地查找，直到找到给定 的关键字，或者碰到一个开放的地址(即该地址单元为空)为止（若要插入，在探查到开放的地址，则可将待插入的新结点存人该地址单元）。 拉链法解决冲突的做法是： 将所有关键字为同义词的结点链接在同一个单链表中 。若选定的散列表长度为m，则可将散列表定义为一个由m个头指针组成的指针数 组T[0..m-1]。凡是散列地址为i的结点，均插入到以T[i]为头指针的单链表中。T中各分量的初值均应为空指针。在拉链法中，装填因子α可以大于1，但一般均取α≤1。拉链法适合未规定元素的大小。 2. Hashtable和HashMap的区别： a) 继承不同。 public class Hashtable extends Dictionary implements Map public class HashMap extends AbstractMap implements Map b) Hashtable中的方法是同步的，而HashMap中的方法在缺省情况下是非同步的。在多线程并发的环境下，可以直接使用Hashtable，但是要使用HashMap的话就要自己增加同步处理了。 c) Hashtable 中， key 和 value 都不允许出现 null 值。 在 HashMap 中， null 可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为 null 。当 get() 方法返回 null 值时，即可以表示 HashMap 中没有该键，也可以表示该键所对应的值为 null 。因此，在 HashMap 中不能由 get() 方法来判断 HashMap 中是否存在某个键， 而应该用 containsKey() 方法来判断。 d) 两个遍历方式的内部实现上不同。Hashtable、HashMap都使用了Iterator。而由于历史原因，Hashtable还使用了Enumeration的方式 。 e) 哈希值的使用不同，HashTable直接使用对象的hashCode。而HashMap重新计算hash值。 f) Hashtable和HashMap它们两个内部实现方式的数组的初始大小和扩容的方式。HashTable中hash数组默认大小是11，增加的方式是old\*2+1。HashMap中hash数组的默认大小是16，而且一定是2的指数。 注： HashSet子类依靠hashCode()和equal()方法来区分重复元素。 HashSet内部使用Map保存数据，即将HashSet的数据作为Map的key值保存，这也是HashSet中元素不能重复的原因。而Map中保存key值的,会去判断当前Map中是否含有该Key对象，内部是先通过key的hashCode,确定有相同的hashCode之后，再通过equals方法判断是否相同。

【例题】关于HashMap与HashTable，以下说法错误的是（）

两者都是用key-value方式获取数据

Hashtable允许null值作为key和value，而HashMap不可以

HashMap不是同步的，而Hashtable是同步的

迭代HashMap采用快速失败机制，而Hashtable不是

【例题】在java7中,下列哪个说法是正确的

ConcurrentHashMap使用synchronized关键字保证线程安全（java7错，java8对）

HashMap实现了Collection接口

Arrays.asList方法返回java.util.ArrayList对象

SimpleDateFormat对象是线程不安全的

【例题】有关hashMap跟hashTable的区别，说法正确的是？

HashMap和Hashtable都实现了Map接口

HashMap是非synchronized，而Hashtable是synchronized

HashTable使用Enumeration，HashMap使用Iterator

Hashtable直接使用对象的hashCode，HashMap重新计算hash值，而且用与代替求模

3、String和char[]

其实都是引用传递，只是因为String是个特殊的final类，所以每次对String的更改都会重新创建内存地址并存储（也可能是在字符串常量池中创建内存地址并存入对应的字符串内容），但是因为这里String是作为参数传递的，在方法体内会产生新的字符串而不会对方法体外的字符串产生影响。（所以如果是return的话，就可以得到新建的String吧）

【例题】以下程序的输出结果是？

public class Example {

String str = new String("good");

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

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 static void change(String str, char ch[])

{

str = "test ok";//先开辟新内存，存储，之后将新地址给str，变成了类似值传递。

ch[0] = 'g';

}

}

good and abc

good and gbc

test ok and abc

test ok and gbc

4、request

request.getAttribute()方法返回request范围内存在的对象，而request.getParameter()方法是获取http提交过来的数据。getAttribute是返回对象，getParameter返回字符串。

5、继承关系中的初始化顺序

①初始化父类中的静态成员变量和静态代码块，按声明顺序，JVM类加载只执行一次；

②初始化子类中的静态成员变量和静态代码块，按声明顺序，JVM类加载只执行一次；

③初始化父类的普通成员变量和代码块，按声明顺序，再执行父类的构造方法；

④初始化子类的普通成员变量和代码块，按声明顺序，再执行子类的构造方法；

【例题】关于以下application,说法正确是什么？

public class Test {

static int x=10;

static {x+=5;}

public static void main(String[] args) //4

{

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

}

static{x/=3;};

}//9

4行与9行不能通过编译，因为缺少方法名和返回类型

编译通过，执行结果是：x=5

编译通过，执行结果是：x=3

9行不能通过编译，因为只能有一个静态初始化器

6、数字与字符串的连接

按从左往右的顺序看加号，逐个合并

加号左右有一方是字符串时表示“连接”

加号左右都是数字时表示“加法”

System.out.println(1+"10"+3+"2");//11032

System.out.println(1+2+"10"+3+"2");//31032

System.out.println(1+"10"+3+1+"2");//110312

7、接口的实现

既然是实现接口，就要实现接口的所有方法，相当于重写方法，方法的重写需要满足：三同一大一小（方法名、返回值类型（java1.5之前要求相同）、形参相同；访问权限>=重写前；抛出异常<=重写前）接口的方法都是public

【例题】如果一个接口Glass有个方法setColor()，有个类BlueGlass实现接口Glass，则在类BlueGlass中正确的是？ ( )

protected void setColor() { …}

void setColor() { …}

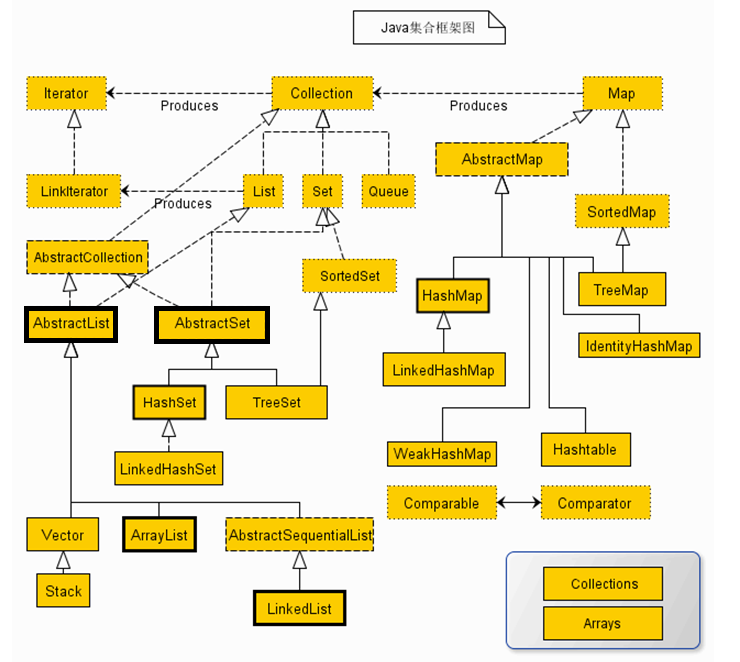
public void setColor() { …}

以上语句都可以用在类BlueGlass中

8、System.out.println()

System是java.lang包下的一个类，out为System的final静态成员（PrintStream类型），println()是PrintStream类的实例方法。

9、Java集合



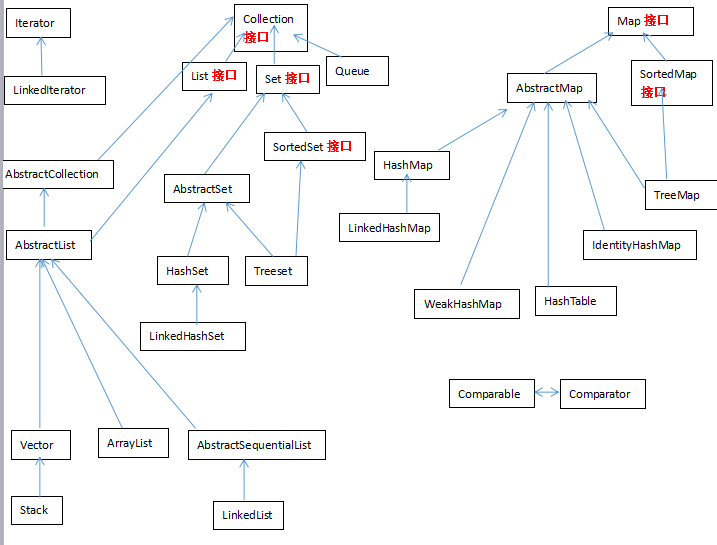
【例题】下面哪些类实现或继承了 Collection 接口？

HashMap

ArrayList

Vector

Iterator



10、Object类的方法

（1）clone方法

保护方法，实现对象的浅复制，只有实现了Cloneable接口才可以调用该方法，否则抛出CloneNotSupportedException异常。

（2）getClass方法

final方法，获得运行时类型。

（3）toString方法

该方法用得比较多，一般子类都有覆盖。

（4）finalize方法

该方法用于释放资源。因为无法确定该方法什么时候被调用，很少使用。

（5）equals方法

该方法是非常重要的一个方法。一般equals和==是不一样的，但是在Object中两者是一样的。子类一般都要重写这个方法。

（6）hashCode方法

该方法用于哈希查找，重写了equals方法一般都要重写hashCode方法。这个方法在一些具有哈希功能的Collection中用到。

一般必须满足obj1.equals(obj2)==true。可以推出obj1.hashCode()==obj2.hashCode()，但是hashCode相等不一定就满足equals。不过为了提高效率，应该尽量使上面两个条件接近等价。

（7）wait方法

wait方法就是使当前线程等待该对象的锁，当前线程必须是该对象的拥有者，也就是具有该对象的锁。wait()方法一直等待，直到获得锁或者被中断。wait(long timeout)设定一个超时间隔，如果在规定时间内没有获得锁就返回。

调用该方法后当前线程进入睡眠状态，直到以下事件发生。

①其他线程调用了该对象的notify方法。

②其他线程调用了该对象的notifyAll方法。

③其他线程调用了interrupt中断该线程。

④时间间隔到了。

此时该线程就可以被调度了，如果是被中断的话就抛出一个InterruptedException异常。

（8）notify方法

该方法唤醒在该对象上等待的某个线程。

（9）notifyAll方法

该方法唤醒在该对象上等待的所有线程。

11、二维数组的声明

int [][] table = new int[2][2];

int [][] table = new int[2][];

int [] table [] = new int[2][2];

int [] table [] = new int[2][];

【例题】已知 声明并初始化二维数组 int a[ ][ ]={{1,2},{3,4},{5,6}} ，则 a[1][1] 的值为（）

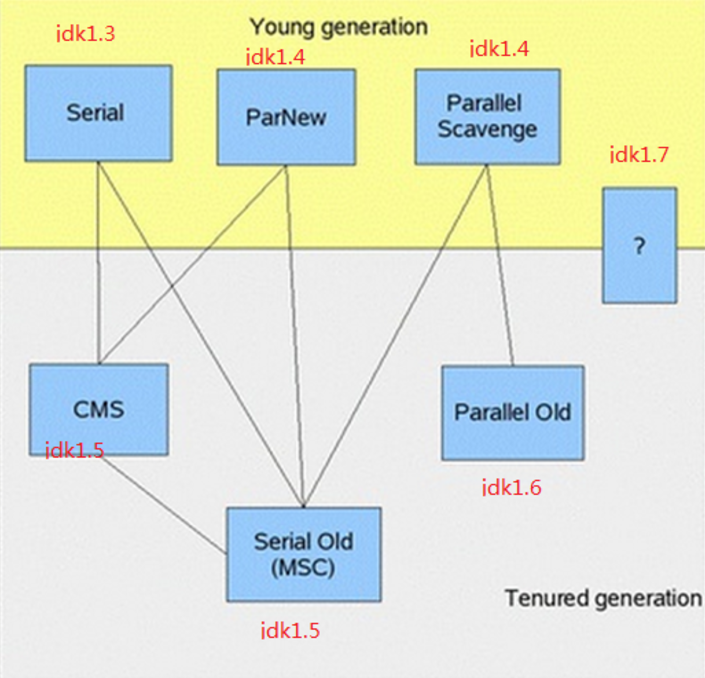
1

2

4

5

12、垃圾收集器



13、值传递和引用传递

【例题】给出以下代码,请给出结果

class Two{

Byte x;

}

class PassO{

public static void main(String[] args){

PassO p=new PassO();

p.start();

}

void start(){

Two t=new Two();

System.out.print(t.x+””);

Two t2=fix(t);

System.out.print(t.x+” ” +t2.x);

}

Two fix(Two tt){

tt.x=42;

return tt;

}

}

null null 42

null 42 42

0 0 42

0 42 42

An exception is thrown at runtime

Compilation

注意x是Byte类型，也就是byte的包装类型，属于引用类型。实例该类对象时，如果成员变量没有显示初始化那么Java默认初始化为null。该题中引用类型t作为形参进行传递，形参的改变会改变实参的值，所以再次打印t.x时已经变为42了。

【例题】given the following code,what will be the output?

class Value{

public int i=15;

}

public class Test{

public static void main(String argv[]){

Test t=new Test( );

t.first( );

}

public void first( ){

int i=5;

Value v=new Value( );// Value abc=new Value( );

v.i=25; // abc.i=25;

second(v,i); // second(abc,i)=25;

System.out.println(v.i);

}

public void second(Value v,int i){//这里新建了一个新的引用v，目前这个形参是指向abc

i = 0;

v.i = 20;//引用v指向abc，所以这里修改了abc的值

Value val = new Value( );

v = val; //引用v现在指向了val，之后就不是修改abc的值

System.out.println(v.i+" "+i);

}

}

15 0 20

15 0 15

20 0 20

0 15 20

Java中原始数据类型都是值传递，传递的是值的副本，形参的改变不会影响实际参数的值，引用传递的是引用类型数据，包括String、数组、列表、map、类对象等类型，形参与实参指向的是同一内存地址，因此形参改变会影响实参的值。（但String是特殊的final类，不改变）

【例题】以下代码结果是什么？

public class foo {

public static void main(String sgf[]) {

StringBuffer a=new StringBuffer(“A”);

StringBuffer b=new StringBuffer(“B”);

operate(a,b);

System.out.println(a+”.”+b);

}

static void operate(StringBuffer x,StringBuffer y) {

x.append(y);

y=x;

}

}

代码可以编译运行，输出“AB.AB”。

代码可以编译运行，输出“A.A”。

代码可以编译运行，输出“AB.B”。

代码可以编译运行，输出“A.B”。

引用a指向对象A

引用b指向对象B

引用x指向对象A

引用y指向对象B

在operate方法中，引用x指向的对象A被连接了B，对象A也就被改变为AB

然后又把引用y指向了x所指向的对象地址，也就是此时引用a,x,y指向同一个对象AB

而引用b没有发生任何变化，依旧指向对象B。

【例题】下列正确的有（ ）

call by value不会改变实际参数的数值

call by reference能改变实际参数的参考地址

call by reference不能改变实际参数的参考地址

call by reference能改变实际参数的内容

引用数据类型是引用传递（call by reference），基本数据类型是值传递（call by value）

值传递不可以改变原变量的内容和地址---》原因是java方法的形参传递都是传递原变量的副本，在方法中改变的是副本的值，而不适合原变量的

引用传递不可以改变原变量的地址，但可以改变原变量的内容---》原因是当副本的引用改变时，原变量的引用并没有发生变化，当副本改变内容时，由于副本引用指向的是原变量的地址空间，所以，原变量的内容发生变化。

结论： 1.值传递不可以改变原变量的内容和地址；

2.引用传递不可以改变原变量的地址，但可以改变原变量的内容

14、接口的修饰词

jdk1.7之前，只支持public和package；

jdk1.8支持接口内部实现静态方法，而且可以有（必须有）方法体具体实现。

【例题】java 接口的修饰符可以为（）

private

protected

final

abstract

接口只能用public和abstract修饰。only public & abstract are permitted。

内部接口only public, protected, private, abstract & static are permitted。

15、ASCII

【例题】执行语句“int a= '2'”后，a的值是（ ）

2

50

49

0

单引号是字符型 记住0是48， A是65。

16、继承中的private

public class Person{

private String name = "Person";

int age=0;

}

public class Child extends Person{

public String grade;

public static void main(String[] args){

Person p = new Child();

System.out.println(p.name);

}

}

输出：Person

没有输出

编译出错

运行出错

Java中对字段属性是静态绑定，方法成员是动态绑定，这里错在：在子类中试图访问父类的private字段，所以编译不通过，将private去掉就可访问，不是动态绑定的问题，它本来就属于静态绑定。

17、ORM

【例题】一般情况下，以下哪个选项不是关系数据模型与对象模型之间匹配关系？

表对应类

记录对应对象

表的字段对应类的属性

表之间的参考关系对应类之间的依赖关系

一般关系数据模型和对象数据模型之间有以下对应关系：表对应类，记录对应对象，表的字段对应类的属性

18、static

【例题】有关下述Java代码描述正确的选项是\_\_\_\_。

public class TestClass {

private static void testMethod(){

System.out.println("testMethod");

}

public static void main(String[] args) {

((TestClass)null).testMethod();

}

}

编译不通过

编译通过，运行异常，报NullPointerException

编译通过，运行异常，报IllegalArgumentException

编译通过，运行异常，报NoSuchMethodException

编译通过，运行异常，报Exception

运行正常，输出testMethod

（1）此处是类对方法的调用，不是对象对方法的调用。

（2）方法是static静态方法，直接使用"类.方法"即可，因为静态方法使用不依赖对象是否被创建。null可以被强制类型转换成任意类型（不是任意类型对象），于是可以通过它来执行静态方法。

（3）非静态的方法用"对象.方法"的方式，必须依赖对象被创建后才能使用，若将testMethod()方法前的static去掉，则会报空指针异常 。此处也验证了（2）的观点

当然，不管是否静态方法，都是已经存在的，只是访问方式不同。

注：private变量在同个类中是可见的，包括main方法。

19、Collection和Collections

【例题】对Collection和Collections描述正确的是

Collection是java.util下的类，它包含有各种有关集合操作的静态方法

Collection是java.util下的接口，它是各种集合结构的父接口

Collections是java.util下的接口，它是各种集合结构的父接口

Collections是java.util下的类，它包含有各种有关集合操作的静态方法

java.util.Collection 是一个集合接口。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。

java.util.Collections 是一个包装类。它包含有各种有关集合操作的静态多态方法。此类不能实例化，就像一个工具类，服务于Java的Collection框架。

20、bean

以下代码定义了一个变量，如何输出这个变量的值？

<bean:define id="stringBean" value="helloWorld"/>

<% String myBean = (String)pageContext.getAttribute(“stringBean”,PageContext.PAGE\_SCOPE);%>

<%=myBean%>

<bean:write name=“helloworld”/>

<bean:write name=“stringBean”/>

<%=stringBean%>

21、String的==和equals

【例题】下面的输出结果是什么？

public class Demo {

public static void main(String args[])

{

String str=new String("hello");

if(str=="hello")

{

System.out.println("true");

}

else {

System.out.println("false");

}

}

}

true

false

String重写了equals，比较字符串是否相等，但==仍判断的是对象引用地址是否相同。

【例题】Java中equal和==的区别是什么？

public class Test1{

public static void main(String args[]){

String a="1234";

String b="1234";

String c = new String("1234");

System.out.println(a==b);

System.out.println(a==c);

System.out.println(a.equals(c));

}

}

true false true

true true false

true false false

true true true

equals：是用来比较两个String是否相等的。

==：是用来判断两个String的地址是否相同。

第一次String a="1234"时，会在常量池中创建一个常量1234,String b=1234时，常量池中已经有了该常量，所以直接取，a和b的地址一样，所以地址值相等；String c = newString("1234")重新new了对象，在堆内存中开辟了新的空间，所以地址值不相等，而equals方法比较的是值是否相等。

【例题】假设有以下代码String s = "hello"；String t = “hello”；char c [ ] = {'h','e','l','l','o'}；下列选项中返回false的语句是？

s.equals (t)；

t.equals (c)；

s==t；

t.equals (new String ("hello"))；

22、数组复制效率

java语言的下面几种数组复制方法中，哪个效率最高？

for 循环逐一复制

System.arraycopy

Array.copyOf

使用clone方法

System.arraycopy>clone>Arrays.copyOf>for循环

23、JSP表达式

【例题】JSP 表达式的写法：

<% expression %>

<=% expression %>

<%= expression %>

<expression/>

专业术语叫 EL表达式

EL 全名为Expression Language，就是为了替代<%= %>脚本表达式。

在j2ee1.4以前默认是不支持el，如果需要需要指定page指令[isELIgnored="true | false" ]为false，j2ee4.0后默认支持el

其主要作用为:

EL主要作用：

1、获取数据：

EL表达式主要用于替换JSP页面中的脚本表达式，以从各种类型的web域中检索java对象、获取数据。(某个web域 中的对象，访问javabean的属性、访问list集合、访问map集合、访问数组)

2、执行运算：

利用EL表达式可以在JSP页面中执行一些基本的关系运算、逻辑运算和算术运算，以在JSP页面中完成一些简单的逻辑运算。${user==null}

3、获取web开发常用对象

EL 表达式定义了一些隐式对象，利用这些隐式对象，web开发人员可以很轻松获得对web常用对象的引用，从而获得这些对象中的数据。

4、调用Java方法

EL表达式允许用户开发自定义EL函数，以在JSP页面中通过EL表达式调用Java类的方法。

很明显 C 选项是对的。

如果本题中的表达式为以下内容:

<%

List list = new ArrayList();

list.add("1");

list.add("2");

list.add("3");

list.add("4");

pageContext.setAttribute("list",list);

%>

那么A 选项是对的。

24、逻辑与、逻辑或

【例题】根据下面的代码，

String s = null;

会抛出NullPointerException异常的有（）。

if( (s!=null) & (s.length()>0) )

if( (s!=null) && (s.length()>0) )

if( (s==null) | (s.length()==0) )

if( (s==null) || (s.length()==0) )

出现异常主要是因为执行了s.length()这句话，所以A、B、C、D选项中只要没有执行到这句话的都是不会报空指针异常的。然后考查的第二点是短路逻辑运算符和非短路逻辑运算符。&和|为非短路逻辑运算符，无论两个语句为什么，一定会执行完毕，所以A、C两个是会报出异常的。

25、重写

方法的重写（override）两同两小一大原则：

方法名相同，参数类型相同

子类返回类型小于等于父类方法返回类型，

子类抛出异常小于等于父类方法抛出异常，

子类访问权限大于等于父类方法访问权限。

【例题】在java中重写方法应遵循规则的包括（）

访问修饰符的限制一定要大于被重写方法的访问修饰符

可以有不同的访问修饰符

参数列表必须完全与被重写的方法相同

必须具有不同的参数列表

【例题】观察以下代码：

class Car extends Vehicle

{

public static void main (String[] args)

{

new Car(). run();

}

private final void run()

{

System. out. println ("Car");

}

}

class Vehicle

{

private final void run()

{

System. out. println("Vehicle");

}

}

Car

Vehicle

Compiler error at line 3

Compiler error at line 5

Exception thrown at runtime

首先final声明的方法是不能被覆盖的，但是这里并不错误，因为方法是private的，也就是子类没有继承父类的run方法，因此子类的run方法跟父类的run方法无关，并不是覆盖。new Car().run()也是调用子类的run方法。

26、Servlet

【例题】Servlet的生命周期可以分为初始化阶段，运行阶段和销毁阶段三个阶段，以下过程属于初始化阶段是（）。

加载Servlet类及.class对应的数据

创建servletRequest和servletResponse对象

创建ServletConfig对象

创建Servlet对象

Servlet的生命周期一般可以用三个方法来表示：

（1）init()：仅执行一次，负责在装载Servlet时初始化Servlet对象

（2）service() ：核心方法，一般HttpServlet中会有get,post两种处理方式。在调用doGet和doPost方法时会构造servletRequest和servletResponse请求和响应对象作为参数。

（3）destory()：在停止并且卸载Servlet时执行，负责释放资源

初始化阶段：Servlet启动，会读取配置文件中的信息，构造指定的Servlet对象，创建ServletConfig对象，将ServletConfig作为参数来调用init()方法。所以选ACD。B是在调用service方法时才构造的

【例题】常用的servlet包的名称是？

java.servlet

javax.servlet

servlet.http

javax.servlet.http

使用 Java 技术开发 WEB 应用程序 , 深入了解 Servlet 的机制对应用的开发将有重要的推动作用 . 而想深入了解 Servlet 的机制就不得不了解 javax.servlet 包 .

javax.servlet 包中包含了 7 个接口 ,3 个类和 2 个异常类 , 它们分别是 :

接口 :RequestDispatcher,Servlet,ServletConfig,ServletContext,ServletRequest,ServletResponse 和 SingleThreadModel

类 :GenericServlet,ServletInputStream 和 ServletOutputStream

异常类 :ServletException 和 UnavailableException

Servlet 的生命周期

在 Servlet 的接口中定义了一个 Servlet 的生命周期方法 , 分别是 Init,Service 和 Destroy

演示了 Servlet 生命周期方法的简单 Servlet:

import javax.servlet.\*;

import java.io.IOException;

public class PrimitiveServlet implements Servlet {

public void init(ServletConfig config) throws ServletException {

System.out.println("init");

}

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

System.out.println("service");

}

public void destroy() {

System.out.println("destroy");

}

public String getServletInfo() {

return null;

}

public ServletConfig getServletConfig() {

return null;

}

}

在 Servlet 中如何获取 ServletConfig 对象 ?

在 Servlet 的 Init 方法中 ,Servlet Container 将会传入一个 ServletConfig 对象 , 开发人员可以通过这个对象获取在 web.xml 文件中定义的 Servlet 初始化参数 .

下面是一个获取 Servlet 初始参数的示例 :

import javax.servlet.\*;

import java.util.Enumeration;

import java.io.IOException;

public class ConfigDemoServlet implements Servlet {

public void init(ServletConfig config) throws ServletException {

Enumeration parameters = config.getInitParameterNames();

while (parameters.hasMoreElements()) {

String parameter = (String) parameters.nextElement();

System.out.println("Parameter name : " + parameter);

System.out.println("Parameter value : " +

config.getInitParameter(parameter));

}

}

public void destroy() {

}

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

}

public String getServletInfo() {

return null;

}

public ServletConfig getServletConfig() {

return null;

}

}

如何获取 ServletContext 对象 ?

可以通过 ServletConfig 对象的 getServletContext 方法获取 ServletContext 对象

import javax.servlet.\*;

import java.util.Enumeration;

import java.io.IOException;

public class ContextDemoServlet implements Servlet {

ServletConfig servletConfig;

public void init(ServletConfig config) throws ServletException {

servletConfig = config;

}

public void destroy() {

}

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

ServletContext servletContext = servletConfig.getServletContext();

Enumeration attributes = servletContext.getAttributeNames();

while (attributes.hasMoreElements()) {

String attribute = (String) attributes.nextElement();

System.out.println("Attribute name : " + attribute);

System.out.println("Attribute value : " +

servletContext.getAttribute(attribute));

}

System.out.println("Major version : " +

servletContext.getMajorVersion());

System.out.println("Minor version : " +

servletContext.getMinorVersion());

System.out.println("Server info : " + servletContext.getServerInfo());

}

public String getServletInfo() {

return null;

}

public ServletConfig getServletConfig() {

return null;

}

}

如何在 Servlet 之间共享信息 ?

我们可以通过 ServletContext 来维护在不同 Servlet 之间共享的信息 .

如何解决 Servlet 的多 Thread 问题 ?

如果 Servlet 需要读写外部资源 , 我们需要考虑 Thread 的问题 , 我们可以使用声明性接口 SingleThreadModel 来避免多 Thread 之间的资源冲突问题 . 但是需要注意的是 , 如果 Servlet 仅仅只是读外部资源的话 , 我们通常不应该实现这个接口 . 如果实现这个接口 ,Servlet 在同一时刻只能服务一个用户请求 , 后至的用户请求必须在队列中等待。

【例题】在J2EE中，使用Servlet过滤器，需要在web.xml中配置（）元素

<filter>

<filter-mapping>

<servlet-filter>

<filter-config>

Servlet过滤器的配置包括两部分：

第一部分是过滤器在Web应用中的定义，由<filter>元素表示，包括<filter-name>和<filter-class>两个必需的子元素

第二部分是过滤器映射的定义，由<filter-mapping>元素表示,可以将一个过滤器映射到一个或者多个Servlet或JSP文件，也可以采用url-pattern将过滤器映射到任意特征的URL。

27、线程

【例题】以下程序运行的结果为（）

public class Example extends Thread{

@Override

public void run(){

try{

Thread.sleep(1000);

}catch (InterruptedException e){

e.printStackTrace();

}

System.out.print("run");

}

public static void main(String[] args){

Example example=new Example();

example.run();

System.out.print("main");

}

}

run main

main run

main

run

不能确定

这个类虽然继承了Thread类，但是并没有真正创建一个线程。创建一个线程需要覆盖Thread类的run方法，然后调用Thread类的start()方法启动。这里直接调用run()方法并没有创建线程，跟普通方法调用一样，是顺序执行的。

【例题】假设 a 是一个由线程 1 和线程 2 共享的初始值为 0 的全局变量，则线程 1 和线程 2 同时执行下面的代码，最终 a 的结果不可能是（）

boolean isOdd = false;

for(int i=1;i<=2;++i)

{

if（i%2==1）isOdd = true；

else isOdd = false；

a+=i\*(isOdd?1:-1)；

}

-1

-2

0

1

易知：每个线程对a 均做了两次读写操作，分别是 “ +1 ” 和 “ -2 ”

而题目问了是最终a 的结果，所以 a 的结果取决于各自线程对 a 的先后读写的顺序

结论：a的可能取值为-1、0、-2

【例题】下列哪些情况可以终止当前线程的运行？

当一个优先级高的线程进入就绪状态时

当该线程调用sleep()方法时

当创建一个新线程时

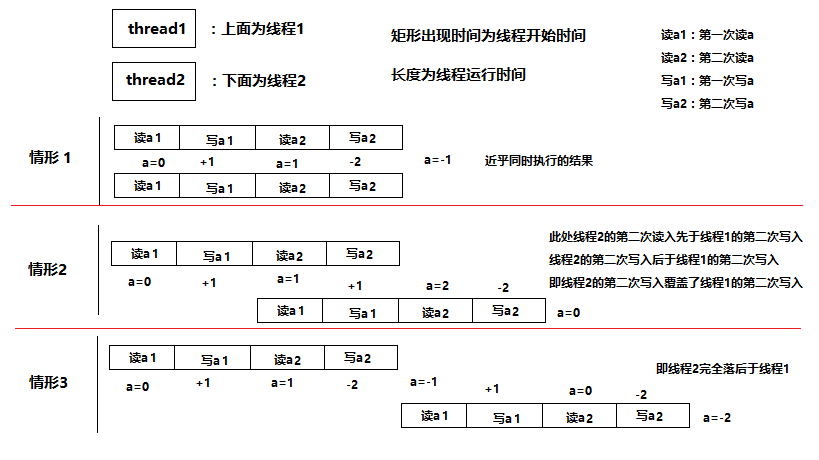
抛出一个异常时

优先级高的并不一定会马上执行。

sleep方法会阻塞一个线程并不会终止

创建一个新的线程时也不会终止另一个线程

当抛出一个异常后程序会结束，所以线程也会被终止



【例题】java中下面哪个能创建并启动线程（）

public class MyRunnable implements Runnable {

public void run() {

//some code here

}

}

new Runnable(MyRunnable).start()

new Thread(MyRunnable).run()

new Thread(new MyRunnable()).start()

new MyRunnable().start()

C正确

首先：创建并启动线程的过程为：定义线程—》实例化线程—》启动线程。

一 、定义线程： 1、扩展java.lang.Thread类。 2、实现java.lang.Runnable接口。

二、实例化线程：

1、如果是扩展java.lang.Thread类的线程，则直接new即可。

2、如果是实现了java.lang.Runnable接口的类，则用Thread的构造方法：

Thread(Runnable target)

Thread(Runnable target, String name)

Thread(ThreadGroup group, Runnable target)

Thread(ThreadGroup group, Runnable target, String name)

Thread(ThreadGroup group, Runnable target, String name, long stackSize)

所以A、D的实例化线程错误。

三、启动线程：在线程的Thread对象上调用start()方法，而不是run()或者别的方法。

所以B的启动线程方法错误。

【例题】以下JAVA程序的输出是什么（）

public class HelloSogou{

public static synchronized void main(String[] a){

Thread t=new Thread(){

public void run(){Sogou();}

};

t.run();

System.out.print("Hello");

}

static synchronized void Sogou(){

System.out.print("Sogou");

}

}

HelloSogou

SogouHello

Hello

结果不确定

我觉得误区有两个：一个是run和start区别，Thread.run()是调用方法，Thread. start()是启动线程；另一个是锁持有问题。这个题是调用方法，和多线程就无关。本题只有一个线程，持有HelloSogou.class锁。那么，就是另一个问题：同步方法调用另一个同步方法的锁问题？

public synchronized void methodA(int a, int b){}

public synchronized void methodB(int a）{

methodA(a, 0);

}

首先要明白两个问题，1.锁的对象是谁？2.谁持有了锁？

假设方法A和B是在同一个类Test中的两个方法。

Test t=new Test();

t.methodB();

调用methodB()方法，获得锁，锁是对象t；锁谁持有？当前线程（不可以说是methodB持有该锁），methodB又调用methodA，也需要锁t，该线程已持有t，当然可以直接调用methodA。

类比到此题，只有一个主线程，调用main，持有HelloSogou.class锁，那当然可以直接调用Sogou方法。

第二，如果是t.statrt()，那么这个题，静态同步函数的锁是该类的字节码文件.class。此题中，main函数和Sogou方法都是static的，所以持有相同锁 HelloSogou.class ，那么，在main线程（main 是一个线程也是一个进程 ）中又开了一个线程，调用Sogou方法，锁会冲突。

我的分析是：调用main函数（一个线程），main函数开启另一个线程，并启动，但是main函数和Sogou方法是同一个锁，所以main函数执行完毕后才会释放锁，Sogou方法才会执行，这就是为什么，换成start，是HelloSogou。

第三，将Sogou方法的锁改为其他.class锁，那么，HelloSogou和SogouHello都可能出现。因为没有互斥现象了，变为抢占式的了。

【例题】下列程序的运行结果

public static void main(String args[]) {

Thread t = new Thread() {

public void run() {

pong();

}

};

t.run();

System.out.print("ping");

}

static void pong() {

System.out.print("pong");

}

}

pingpong

pongping

pingpong和pongping都有可能

都不输出

这里需要注意Thread的start和run方法

用start方法才能真正启动线程，此时线程会处于就绪状态，一旦得到时间片，则会调用线程的run方法进入运行状态。

而run方法只是普通方法，如果直接调用run方法，程序只会按照顺序执行主线程这一个线程。

28、异常



【例题】下面有关JAVA异常类的描述，说法错误的是？

异常的继承结构：基类为Throwable，Error和Exception继承Throwable，RuntimeException和IOException等继承Exception。

非RuntimeException一般是外部错误(非Error)，其必须被 try{}catch语句块所捕获。

Error类体系描述了Java运行系统中的内部错误以及资源耗尽的情形，Error不需要捕捉。

RuntimeException体系包括错误的类型转换、数组越界访问和试图访问空指针等等，必须被 try{}catch语句块所捕获。

运行时异常故名思议就是程序在运行的时候出现异常，隐含的一个前提就是程序在编译时是检测不到异常的存在，作者本人也并不知道是否自己的代码中含有运行时异常，所以根本也不可能提前使用try{}catch{}捕获。

异常是指程序运行时（非编译）所发生的非正常情况或错误，当程序违反了语义规则，jvm就会将出现的错误表示一个异常抛出。

异常也是java的对象，定义了基类 java.lang.throwable作为异常父类。这些异常类又包括error和exception两大类

error类异常主要是运行时逻辑错误导致，一个正确程序中是不应该出现error的。当出现error一般jvm会终止。

exception表示可恢复异常，包括检查异常和运行时异常。检查异常是最常见异常比如 io异常、sql异常，都发生在编译阶段。这类通过try、catch捕捉

而运行时异常，编译器没有强制对其进行捕捉和处理。一般都会把异常向上抛出，直到遇到处理代码位置，若没有处理块就会抛到最上层，多线程用thread.run()抛出，单线程用main()抛出。常见的运行异常包括空指针异常、类型转换异常、数组越界异常、数组存储异常、缓冲区溢出异常、算术异常等。

【例题】在java的一个异常处理中，可以包含多个的语句块是 （ ）。

try

finally

throws

catch

异常处理一般格式：

捕获异常：

try{

//代码块

}catch(异常类型，例如：Exception e){

//需要抛出的异常，例如：e.printStackTrace();

}catch(异常类型){

//需要抛出的异常

}finally{

//必定执行的代码块

}

所以说在一个异常处理中catch语句块是可以多个的，也就是可以抛出多个异常！

【例题】请问所有的异常类皆直接继承于哪一个类？（）

java.applet.Applet

java.lang.Throwable

java.lang.Exception

java.lang.Error

Throwable有两个子类exception(异常)和error(错误)。

【例题】关于异常的编程，以下描述错误的是：（ ）

在有除法存在的代码处，抛不抛出异常均可

int i=Integer.parseInt(”123a”);将产生NumberFarmatException

int a[]=null; a[0]=1; 将产生ArraylndexOutOfBoundsException

输入输出流编程中，读和写时都必须抛出IOException

A.除数为0 等ArithmeticException，是RuntimException的子类。而运行时异常将由运行时系统自动抛出，不需要使用throw语句。Java编译器 允许忽略运行时异常，一个方法可以既不捕捉，也不声明抛出运行时异常。C.产生NullPointerException。

【例题】在异常处理中，如释放资源，关闭数据库、关闭文件应由（ ）语句来完成。

try子句

catch子句

finally子句

throw子句

finally子句一般是用来清理这些资源的。

【例题】以下关于JAVA语言异常处理描述正确的有？

throw关键字可以在方法上声明该方法要抛出的异常。

throws用于抛出异常对象。

try是用于检测被包住的语句块是否出现异常，如果有异常，则抛出异常，并执行catch语句。

finally语句块是不管有没有出现异常都要执行的内容。

Java语言中的异常处理包括声明异常、抛出异常、捕获异常和处理异常四个环节。

throw用于抛出异常。

throws关键字可以在方法上声明该方法要抛出的异常，然后在方法内部通过throw抛出异常对象。

try是用于检测被包住的语句块是否出现异常，如果有异常，则抛出异常，并执行catch语句。

cacth用于捕获从try中抛出的异常并作出处理。

finally语句块是不管有没有出现异常都要执行的内容。

【例题】下面程序的输出结果是？

package algorithms.com.guan.javajicu;

public class TestDemo {

public static String output ="";

public static void foo(int i){

try{

if(i == 1){

throw new Exception();

}

}catch(Exception e){

output += "2";

return ;

}finally{

output += "3";

}

output += "4";

}

public static void main(String[] args) {

foo(0);

foo(1);

System.out.println(output);

}

}

出错

342

34234

3423

解题要点：

1、try中没有抛出异常，则catch语句不执行，如果有finally语句，则接着执行finally语句，继而接着执行finally之后的语句；

2、try中抛出异常，有匹配的catch语句，则catch语句捕获，如果catch中有return语句，则要在finally执行后再执行。

29、同步

【例题】下面有关java的一些细节问题，描述错误的是？

构造方法不需要同步化

一个子类不可以覆盖掉父类的同步方法

定义在接口中的方法默认是public的

容器保存的是对象的引用

构造方法每次都是构造出新的对象，不存在多个线程同时读写同一对象中的属性的问题，所以不需要同步。

如果父类中的某个方法使用了 synchronized关键字，而子类中也覆盖了这个方法，默认情况下子类中的这个方法并不是同步的，必须显示的在子类的这个方法中加上 synchronized关键字才可。当然，也可以在子类中调用父类中相应的方法，这样虽然子类中的方法并不是同步的，但子类调用了父类中的同步方法，也就相当子类方法也同步了。详见：http://blog.csdn.net/welcome000yy/article/details/8941644

接口里面的变量为常量，其实际是 public static final ；接口里面的方法为抽象方法，其实际是public abstract。

30、request的forward和redirect

【例题】下面有关forward和redirect的描述，正确的是() ？

forward是服务器将控制权转交给另外一个内部服务器对象，由新的对象来全权负责响应用户的请求

执行forward时，浏览器不知道服务器发送的内容是从何处来，浏览器地址栏中还是原来的地址

执行redirect时，服务器端告诉浏览器重新去请求地址

forward是内部重定向，redirect是外部重定向

redirect默认将产生301 Permanently moved的HTTP响应

1.从地址栏显示来说

forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址栏还是原来的地址.

redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL.

2.从数据共享来说

forward:转发页面和转发到的页面可以共享request里面的数据.

redirect:不能共享数据.

3.从运用地方来说

forward:一般用于用户登陆的时候,根据角色转发到相应的模块.

redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等.

4.从效率来说

forward:高.

redirect:低.

31、数据类型的自动转换和强制转换

【例题】Java类Demo中存在方法func0、func1、func2、func3和func4，请问该方法中，哪些是不合法的定义？( )

public class Demo{

　　float func0()

　　{

　　　　byte i=1;

　　　　return i;

　　}

　　float func1()

　　{

　　　　int i=1;

　　　　return;

　　}

　　float func2()

　　{

　　　　short i=2;

　　　　return i;

　　}

　　float func3()

　　{

　　　　long i=3;

　　　　return i;

　　}

　　float func4()

　　{

　　　　double i=4;

　　　　return i;

　　}

}

func1？？？？

func2

func3

func4

数据类型的转换，分为自动转换和强制转换。自动转换是程序在执行过程中 “ 悄然 ” 进行的转换，不需要用户提前声明，一般是从位数低的类型向位数高的类型转换；强制类型转换则必须在代码中声明，**转换顺序不受限制**。

**自动数据类型转换**

自动转换按从低到高的顺序转换。不同类型数据间的优先关系如下：

低 ---------------------------------------------> 高

byte,short,char-> int -> long -> float -> double

运算中，不同类型的数据先转化为同一类型，然后进行运算，转换规则如下：

|  |  |  |
| --- | --- | --- |
| 操作数1的类型 | 操作数2的类型 | 转换后的类型 |
| byte 、 short 、 char | int | int |
| byte 、 short 、 char 、 int | long | long |
| byte 、 short 、 char 、 int 、 long | float | float |
| byte 、 short 、 char 、 int 、 long 、 float | double | double |

**强制数据类型转换**

强制转换的格式是在需要转型的数据前加上 “( )” ，然后在括号内加入需要转化的数据类型。有的数据经过转型运算后，精度会丢失，而有的会更加精确

【例题】设int x=1,float y=2,则表达式x/y的值是：（）

0

1

2

以上都不是

【例题】下列方法定义中， 正确的是（）

int x( ){ char ch=’a’; return (int)ch; }

void x( ){ ...return true; }

int x( ){ ...return true; }

int x( int a, b){ return a+b; }

A.Java中涉及到byte、short和char类型都可以强制转化为int，符合返回类型 A正确

B.方法中定义为void 应该没有返回值，但返回值为boolean类型 B错

C. 方法中类型为int，应该返回int类型值，但是 返回值为boolean类型 C错

D.方法应该定义为int(int a,int b)，所以D错

32、Applet

【例题】在 Applet 中，方法执行的顺序是 ?（ ）

init(), start(), paint(),stop(),destroy().

init(),paint(),start(),stop(),destroy().

paint(),start(),stop(),destroy().

init(), start(),stop(),destroy(),paint().

init 初始化，start启动运行，paint 窗口最小化等时候需要暂停，stop 停止，destory 销毁。

【例题】当使包含 applet 程序的页面从最小化恢复时，以下选项中的哪个方法将被执行？（ ）

paint()

start()

destroy()

stop()

applet页面刚打开时，程序调用init（）然后调用start（）再然后paint（）

用户离开applet页面，程序自动调用stop（），用户关闭浏览器，程序触发destroy（）

并且paint（）在每一次浏览器显示页面时被调用。

最小化点开-->页面重新显示，之前进程没丢，只有paint（）。

【例题】在Java Applet程序用户自定义的Applet子类中，一般需要重载父类的( )方法来完成一些画图操作。

start( )

stop( )

init( )

paint( )

33、JVM

【例题】下面对JVM叙述不正确的是：

JVM的全称是Java Virtual Machine

JVM是一种计算机硬件技术，它是Java程序的运行平台

JVM是在计算机硬件系统上用软件实现的一台假想机

Java程序在执行时．JVM把Java字节码解释成机器码

JVM也就是Java Virtual Machine，即java虚拟机，是在现有的平台上，如Windows，Linux等，运行一个软件，虚拟出一台接口统一的计算机，依次实现java语言的跨平台特性。

34、向上转型和向下转型

【例题】下列程序执行后结果为( )

class A {

public int func1(int a, int b) {

return a - b;

}

}

class B extends A {

public int func1(int a, int b) {

return a + b;

}

}

public class ChildClass {

public static void main(String[] args) {

A a = new B();

B b = new B();

System.out.println("Result=" + a.func1(100, 50));

System.out.println("Result=" + b.func1(100, 50));

}

}

Result=150Result=150

Result=100Result=100

Result=100Result=150

Result=150Result=100

其实很简单，涉及转型的题目，分为向上或者向下转型。

关键的来了，不论向上或者向下转型，都是一句话，**“编译看左边，运行看右边”。也就是编译时候，会看左边引用类型是否能正确编译通过，运行的时候是调用右边的对象的方法。**

就本题来说，编译时候会发现左边满足条件所以编译通过，运行时候又会调用右边也就是 class B 的方法，所以答案都是150。

35、内存区域

【例题】String str1 = “abc”，“abc”分配在内存哪个区域？

堆

栈

字符串常量区

寄存器

用new创建的对象在堆区

函数中的临时变量在栈区

java中的字符串在字符串常量区

【例题】String str = new String(“abc”)，“abc”在内存中是怎么分配的？

堆

栈

字符串常量区

寄存器

“abc”保存在常量池，str引用的对象保存在堆里，而java7中又把常量池移到了堆中，所以这题题目就不够严谨。

【例题】下列Java代码中的变量a、b、c分别在内存的\_\_\_\_存储区存放。

class A {

private String a = “aa”;

public boolean methodB() {

String b = “bb”;

final String c = “cc”;

}

}

堆区、堆区、堆区

堆区、栈区、堆区

堆区、栈区、栈区

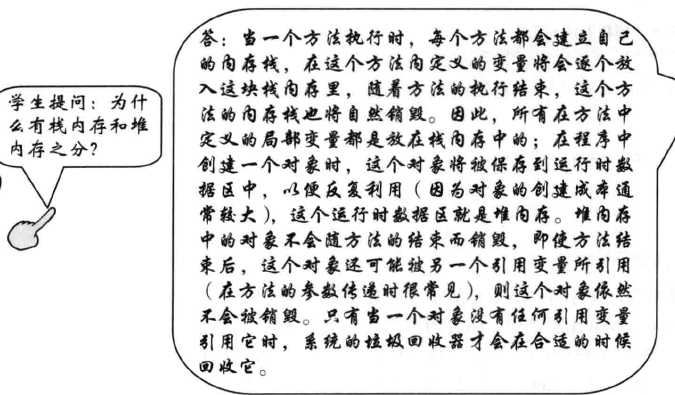
堆区、堆区、栈区

静态区、栈区、堆区

静态区、栈区、栈区

a是类中的成员变量，存放在堆区

b、c都是方法中的局部变量，存放在栈区



36、int、Integer

【例题】在jdk1.5的环境下，有如下4条语句：

Integer i01 = 59;

int i02 = 59;

Integer i03 =Integer.valueOf(59);

Integer i04 = new Integer(59)。

以下输出结果为false的是：

System.out.println(i01== i02);

System.out.println(i01== i03);

System.out.println(i03== i04);

System.out.println(i02== i04);

JVM中一个字节以下的整型数据会在JVM启动的时候加载进内存，除非用new Integer()显式的创建对象，否则都是同一个对象。也就是i01、i02是同一个对象，i04是新对象，i03是基本数据类型。

所有只有i04是一个新对象，其他都是同一个对象。所以A，B选项为true

C选项i03和i04是两个不同的对象，返回false

D选项i02是基本数据类型，比较的时候比较的是数值，返回true（自动拆箱？）

【例题】设有下面两个赋值语句：

a = Integer.parseInt("1024");

b = Integer.valueOf("1024").intValue();

下述说法正确的是（）

a是整数类型变量，b是整数类对象。

a是整数类对象，b是整数类型变量。

a和b都是整数类对象并且它们的值相等。

a和b都是整数类型变量并且它们的值相等。

intValue()是把Integer对象类型变成int的基础数据类型；

parseInt()是把String 变成int的基础数据类型；

Valueof()是把String 转化成Integer对象类型；（现在JDK版本支持自动装箱拆箱了。）

本题：parseInt得到的是基础数据类型int，valueof得到的是装箱数据类型Integer，然后再通过valueInt转换成int，所以选择D

37、JSP的error

在 myjsp.jsp 中，关于下面的代码说法错误的是： ( )

<%@ page language="java" import="java.util.\*" errorPage="error.jsp" isErrorPage="false" %>

该页面可以使用 exception 对象

该页面发生异常会转向 error.jsp

存在 errorPage 属性时，isErrorPage 是默认为 false

error.jsp 页面一定要有isErrorPage 属性且值为 true

exception是JSP九大内置对象之一，其实例代表其他页面的异常和错误。只有当页面是错误处理页面时，即isErroePage为 true时，该对象才可以使用。对于C项，errorPage的实质就是JSP的异常处理机制,发生异常时才会跳转到 errorPage指定的页面，没必要给errorPage再设置一个errorPage。所以当errorPage属性存在时， isErrorPage属性值为false。

38、继承

【例题】关于继承的说法正确的是（）。

子类将继承父类所有的数据域和方法

子类将继承父类的其可见的数据域和方法

子类只继承父类public方法和数据域

子类只继承父类的方法，而不继承数据域

在一个子类被创建的时候，首先会在内存中创建一个父类对象，然后在父类对象外部放上子类独有的属性，两者合起来形成一个子类的对象。所以所谓的继承使子类拥有父类所有的属性和方法其实可以这样理解，子类对象确实拥有父类对象中所有的属性和方法，但是父类对象中的私有属性和方法，子类是无法访问到的，只是拥有，但不能使用。就像有些东西你可能拥有，但是你并不能使用。所以子类对象是绝对大于父类对象的，所谓的子类对象只能继承父类非私有的属性及方法的说法是错误的。可以继承，只是无法访问到而已。

【例题】对于子类的构造函数说明，下列叙述中错误的是（ ）。

子类不能继承父类的无参构造函数。

子类可以在自己的构造函数中使用super关键字来调用父类的含参数构造函数，但这个调用语句必须是子类构造函数的第一个可执行语句。

在创建子类的对象时，若不含带参构造函数，将先执行父类的无参构造函数，然后再执行自己的无参构造函数。

子类不但可以继承父类的无参构造函数，也可以继承父类的有参构造函数。

构造函数不能被继承，构造方法只能被显式或隐式的调用。

39、Java Web

【例题】在Web应用程序的文件与目录结构中，web.xml是放置在( )中。

WEB-INF目录

conf目录

lib目录

classes目录

（1）/WEB-INF/web.xml 是部署描述文件

（2）/WEB-INF/classes 用来放置应用程序用到的自定义类(.class)，必须包括包(package)结构。

（3）/WEB-INF/lib 用来放置应用程序用到的JAR文件。

40、ArrayList、Vector、LinkedList

【例题】ArrayList和Vector主要区别是什么？

Vector与ArrayList一样，也是通过数组实现的，不同的是Vector支持线程的同步

Vector与ArrayList一样，也是通过数组实现的，不同的是ArrayList支持线程的同步

Vector是通过链表结构存储数据，ArrayList是通过数组存储数据

上述说法都不正确

Vector & ArrayList 的主要区别

1）同步性:Vector是线程安全的，也就是说是同步的 ，而ArrayList 是线程不安全的，不是同步的。

2）数据增长:当需要增长时，Vector默认增长为原来一倍，而ArrayList却是原来的50%，这样，ArrayList就有利于节约内存空间。

如果涉及到堆栈，队列等操作，应该考虑用Vector，如果需要快速随机访问元素，应该使用ArrayList 。

扩展知识：

1. Hashtable & HashMap

Hashtable和HashMap它们的性能方面的比较类似 Vector和ArrayList，比如Hashtable的方法是同步的,而HashMap的不是。

2. ArrayList & LinkedList

ArrayList的内部实现是基于内部数组Object[],所以从概念上讲,它更象数组，但LinkedList的内部实现是基于一组连接的记录，所以，它更象一个链表结构，所以，它们在性能上有很大的差别：

从上面的分析可知，在ArrayList的前面或中间插入数据时,你必须将其后的所有数据相应的后移,这样必然要花费较多时间，所以,当你的操作是在一列数据的后面添加数据而不是在前面或中间,并且需要随机地访问其中的元素时,使用ArrayList会提供比较好的性能； 而访问链表中的某个元素时,就必须从链表的一端开始沿着连接方向一个一个元素地去查找,直到找到所需的元素为止，所以,当你的操作是在一列数据的前面或中间添加或删除数据，并且按照顺序访问其中的元素时，就应该使用LinkedList了。

【例题】ArrayLists和LinkedList的区别，下述说法正确的有？

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。

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

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

ArrayList的空间浪费主要体现在在list列表的结尾预留一定的容量空间，而LinkedList的空间花费则体现在它的每一个元素都需要消耗相当的空间。

A. 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。 //正确，这里的所谓动态数组并不是那个“ 有多少元素就申请多少空间 ”的意思，通过查看源码，可以发现，这个动态数组是这样实现的，如果没指定数组大小，则申请默认大小为10的数组，当元素个数增加，数组无法存储时，系统会另个申请一个长度为当前长度1.5倍的数组，然后，把之前的数据拷贝到新建的数组。

- ---------------------------------------------------------------------

B. 对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。//正确，ArrayList是数组，所以，直接定位到相应位置取元素，LinkedLIst是链表，所以需要从前往后遍历。

-----------------------------------------------------------------------

C. 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。//正确，ArrayList的新增和删除就是数组的新增和删除，LinkedList与链表一致。

-------------------------------------------------------------------------

D. ArrayList的空间浪费主要体现在在list列表的结尾预留一定的容量空间，而LinkedList的空间花费则体现在它的每一个元素都需要消耗相当的空间。//正确，因为ArrayList空间的增长率为1.5倍，所以，最后很可能留下一部分空间是没有用到的，因此，会造成浪费的情况。对于LInkedList的话，由于每个节点都需要额外的指针，所以，你懂的。

41、Iterator

【例题】list是一个ArrayList的对象，哪个选项的代码填到//todo delete处，可以在Iterator遍历的过程中正确并安全的删除一个list中保存的对象？（）

Iterator it = list.iterator();

int index = 0;

while (it.hasNext())

{

Object obj = it.next();

if (needDelete(obj)) //needDelete返回boolean，决定是否要删除

{

//todo delete

}

index ++;

}

it.remove();

list.remove(obj);

list.remove(index);

list.remove(obj,index);

Iterator支持从源集合中安全地删除对象，只需在Iterator上调用remove()即可。这样做的好处是可以避免ConcurrentModifiedException，当打开 Iterator 迭代集合时，同时又在对集合进行修改。有些集合不允许在迭代时删除或添加元素，但是调用Iterator的remove()方法是个安全的做法。

如果在循环的过程中调用集合的remove()方法，就会导致循环出错，例如：

for(int i=0;i<list.size();i++){

list.remove(...);

}

循环过程中list.size()的大小变化了，就导致了错误。

所以，如果你想在循环语句中删除集合中的某个元素，就要用迭代器iterator的remove()方法，因为它的remove()方法不仅会删除元素，还会维护一个标志，用来记录目前是不是可删除状态，例如，你不能连续两次调用它的remove()方法，调用之前至少有一次next()方法的调用。

42、volatile

【例题】volatile关键字的说法错误的是

能保证线程安全

volatile关键字用在多线程同步中，可保证读取的可见性

JVM保证从主内存加载到线程工作内存的值是最新的

volatile能禁止进行指令重排序

出于运行速率的考虑，java编译器会把经常经常访问的变量放到缓存（严格讲应该是工作内存）中，读取变量则从缓存中读。但是在多线程编程中,内存中的值和缓存中的值可能会出现不一致。volatile用于限定变量只能从内存中读取，保证对所有线程而言，值都是一致的。但是volatile不能保证原子性，也就不能保证线程安全。

43、ResultSet

【例题】ResultSet中记录行的第一列索引为？

-1

0

1

以上都不是

ResultSet跟普通的数组不同，索引从1开始而不是从0开始

44、interface的修饰词

【例题】不能用来修饰interface的有（ ）

private

public

protected

static

就访问来说，可以用public,默认修饰符，而不能用private，因为接口本身就是为了让别的类或接口使用的，用private就没有了意义。

protected也是不可以的，ptotected的访问权限是包内对象以及包外继承该类的包外对象，其实和public意义也就一样了，也就没必要选择protected。

static是用来修饰变量或者方法的，而不修饰类。

（final同样不能修饰接口）

在jdk1.8时，支持static修饰接口

45、TLS

【例题】对于线程局部存储TLS(thread local storage)，以下表述正确的是

解决多线程中的对同一变量的访问冲突的一种技术

TLS会为每一个线程维护一个和该线程绑定的变量的副本

每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了

Java平台的java.lang.ThreadLocal是TLS技术的一种实现

同一全局变量或者静态变量每个线程访问的是同一变量，多个线程同时访存同一全局变量或者静态变量时会导致冲突，尤其是多个线程同时需要修改这一变量时，通过TLS机制，为每一个使用该全局变量的线程都提供一个变量值的副本，每一个线程均可以独立地改变自己的副本，而不会和其它线程的副本冲突。

46、抽象类和接口的区别

【例题】在Jdk1.7中，下述说法中抽象类与接口的区别正确的有哪些？

抽象类中可以有普通成员变量，接口中没有普通成员变量。

抽象类和接口中都可以包含静态成员常量。

一个类可以实现多个接口，但只能继承一个抽象类

抽象类中可以包含非抽象的普通方法，接口中的方法必须是抽象的，不能有非抽象的普通方法。

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。另外，接口和抽象类在方法上有区别：

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。

3. 抽象类中可以有普通成员变量，接口中没有普通成员变量

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

5. 抽象类中可以包含静态方法，接口中不能包含静态方法

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，但只能继承一个抽象类。二者在应用方面也有一定的区别：接口更多的是在系统架构设计方法发挥作用，主要用于定义模块之间的通信契约。而抽象类在代码实现方面发挥作用，可以实现代码的重用，例如，模板方法设计模式是抽象类的一个典型应用，假设某个项目的所有Servlet类都要用相同的方式进行权限判断、记录访问日志和处理异常，那么就可以定义一个抽象的基类，让所有的Servlet都继承这个抽象基类，在抽象基类的service方法中完成权限判断、记录访问日志和处理异常的代码，在各个子类中只是完成各自的业务逻辑代码。

【例题】下列描述错误的是?

类只能继承一个父类，但是可以实现多个接口

抽象类自身可以定义成员而接口不可以

抽象类和接口都不能被实例化(忽略匿名内部类)

一个类可以有多个基类和多个基接口

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。

47、垃圾回收

【例题】对于一个已经不被任何变量引用的对象，当垃圾回收器准备回收该对象所占用的内存时，将自动调用该对象的哪个方法（）

finalize

notify

notifyAll

hashCode

垃圾回收过程中的对象销毁–Finalization

就在移除一个对象并回收它的内存空间之前，Java垃圾回收器将会调用各个实例的finalize()方法，这样实例对象就有机会可以释放掉它占用的资源。尽管finalize()方法是保证在回收内存空间之前执行的，但是对具体的执行时间和执行顺序是没有任何保证的。多个实例之间的finalize()执行顺序是不能提前预知的，甚至有可能它们是并行执行的。程序不应该预先假设实例执行finalize()的方法，也不应该使用finalize()方法来回收资源。

在finalize过程中抛出的任何异常都默认被忽略掉了，同时对象的销毁过程被取消。

JVM规范并没有讨论关于弱引用的垃圾回收，这是明确声明的。具体的细节留给实现者决定。

垃圾回收是由守护进程执行的。

【例题】如何放掉一个指定占据的内存空间？（）

调用free()方法

代用system.gc()方法

赋值给该项对象的引用为null

程序员无法明确强制垃圾回收器运行

在《java虚拟机》一书中明确讲了，释放掉占据的内存空间是由gc完成，但是程序员无法明确强制其运行，该空间在不被引用的时候不一定会立即被释放，这取决于GC本身，无法由程序员通过代码控制。

【例题】以下哪项陈述是正确的？

垃圾回收线程的优先级很高，以保证不再使用的内存将被及时回收

垃圾收集允许程序开发者明确指定释放哪一个对象

垃圾回收机制保证了JAVA程序不会出现内存溢出

进入”Dead”状态的线程将被垃圾回收器回收

以上都不对

A: 垃圾回收在jvm中优先级相当相当低。

B：垃圾收集器（GC）程序开发者只能推荐JVM进行回收，但何时回收，回收哪些，程序员不能控制。

C：垃圾回收机制只是回收不再使用的JVM内存，如果程序有严重BUG，照样内存溢出。

D：进入DEAD的线程，它还可以恢复，GC不会回收

【例题】在Java中，对于不再使用的内存资源，如调用完成的方法，“垃圾回收器”会自动将其释放。（ ）

正确

错误

方法调用时，会创建栈帧在栈中，调用完是程序自动出栈释放，而不是gc释放

【例题】以下哪些jvm的垃圾回收方式采用的是复制算法回收

新生代串行收集器

老年代串行收集器

并行收集器

新生代并行回收收集器

老年代并行回收收集器

cms收集器

两个最基本的java回收算法：复制算法和标记清理算法

复制算法：两个区域A和B，初始对象在A，继续存活的对象被转移到B。此为新生代最常用的算法

标记清理：一块区域，标记要回收的对象，然后回收，一定会出现碎片，那么引出

标记-整理算法：多了碎片整理，整理出更大的内存放更大的对象

两个概念：新生代和年老代

新生代：初始对象，生命周期短的

永久代：长时间存在的对象

整个java的垃圾回收是新生代和年老代的协作，这种叫做分代回收。

P.S：

Serial New收集器是针对新生代的收集器，采用的是复制算法

Parallel New（并行）收集器，新生代采用复制算法，老年代采用标记整理

Parallel Scavenge（并行）收集器，针对新生代，采用复制收集算法

Serial Old（串行）收集器，新生代采用复制，老年代采用标记整理

Parallel Old（并行）收集器，针对老年代，标记整理

CMS收集器，基于标记清理

G1收集器：整体上是基于标记整理 ，局部采用复制

综上：新生代基本采用复制算法，老年代采用标记整理算法。cms采用标记清理。

48、依赖注入

【例题】下面关于依赖注入(DI)的说法不正确的是()

只有通过Spring才能实现依赖注入(DI)

依赖注入的主要目的是解耦合

常见的依赖注入方式有Setter和构造方法

依赖注入是一种思想，或者说是一种设计模式，在java中是通过反射机制实现，与具体框架无关。

Spring依赖注入（DI）的三种方式，分别为：

1． 接口注入

2． Setter 方法注入

3． 构造方法注入

49、Struts

【例题】关于struts项目中的类与MVC模式的对应关系，说法错误的是

Jsp文件实现视图View的功能

ActionServlet这一个类是整个struts项目的控制器

ActionForm、Action都属于Model部分

一个struts项目只能有一个Servlet

Model： MVC系统中的Model部分从概念上可以分为两类――系统的内部状态，和改变系统状态的动作。Struts为Model部分提供了Action和ActionForm对象：所有的Action处理器对象都是开发者从Struts的Action类派生的子类。Action处理器对象封装了具体的处理逻辑，调用业务逻辑模块，并且把响应提交到合适的View组件以产生响应。Struts提供的ActionForm组件对象，它可以通过定义属性描述客户端表单数据。开发者可以从它派生子类对象，利用它和Struts提供的自定义标记库结合可以实现对客户端的表单数据的良好封装和支持，Action处理器对象可以直接对它进行读写，而不再需要和request、response对象进行数据交互。通过ActionForm组件对象实现了对View和Model之间交互的支持。Struts通常建议使用一组JavaBean表示系统的内部状态，根据系统的复杂度也可以使用像Entity EJB 和 Session EJB等组件来实现系统状态。Struts建议在实现时把"做什么"（Action）和"如何做"（业务逻辑）分离。这样可以实现业务逻辑的重用。

50、构造方法

【例题】Which of the following can be applied to constructors:

final

static

synchronized

native

None of these

识别合法的构造方法；

1 构造方法可以被重载，一个构造方法可以通过this关键字调用另一个构造方法，this语句必须位于构造方法的第一行；

2 当一个类中没有定义任何构造方法，Java将自动提供一个缺省构造方法；

3 子类通过super关键字调用父类的一个构造方法；

4 当子类的某个构造方法没有通过super关键字调用父类的构造方法，通过这个构造方法创建子类对象时，会自动先调用父类的缺省构造方法

5 构造方法不能被static、final、synchronized、abstract、native修饰，但可以被public、private、protected修饰；

6 构造方法不是类的成员方法；

7 构造方法不能被继承

51、自动拆箱与自动装箱

【例题】下列 java 程序输出结果为\_\_\_\_\_\_。

int i=0;

Integer j = new Integer(0);

System.out.println(i==j);

System.out.println(j.equals(i));

true,false

true,true

false,true

false,false

对于不同的环境结果不同

程序无法执行

本题是一个自动拆装箱的考题（自动拆装箱JDK需在1.5上）

1、基本型和基本型封装型进行“==”运算符的比较，基本型封装型将会自动拆箱变为基本型后再进行比较，因此Integer(0)会自动拆箱为int类型再进行比较，显然返回true；

2、两个Integer类型进行“==”比较，如果其值在-128至127，那么返回true，否则返回false, 这跟Integer.valueOf()的缓冲对象有关，这里不进行赘述。

3、两个基本型的封装型进行equals()比较，首先equals()会比较类型，如果类型相同，则继续比较值，如果值也相同，返回true

4、基本型封装类型调用equals(),但是参数是基本类型，这时候，先会进行自动装箱，基本型转换为其封装类型，再进行3中的比较。

int a=257;

Integer b=257;

Integer c=257;

Integer b2=57;

Integer c2=57;

System.out.println(a==b);

//System.out.println(a.equals(b)); 编译出错，基本型不能调用equals()

System.out.println(b.equals(257.0));

System.out.println(b==c);

System.out.println(b2==c2);

因此上面的代码的结果因此为 true, false, false, true

52、静态变量

【例题】如下代码的 输出结果是什么？

public class Test {

public int aMethod() {

static int i = 0;

i++;

return i;

}

public static void main (String args[]) {

Test test = new Test();

test.aMethod();

int j = test.aMethod();

System.out.println(j);

}

}

0

1

2

编译失败

静态变量只能在类主体中定义，不能在方法中定义

53、合法的标识符

【例题】下列哪个选项是合法的标识符？（）

123

\_name

class

first

Java标识符由 数字、字母、下划线(\_)、美元符号($) 组成， 首位不能是数字 。并且 Java关键字不能作为标识符 。

54、Java基础

【例题】下列说法错误的有（ ）

能被java.exe成功运行的java class文件必须有main()方法

J2SDK就是Java API

Appletviewer.exe可利用jar选项运行.jar文件

能被Appletviewer成功运行的java class文件必须有main()方法

B选项中J2SDK是编程工具，不是API。

C选项中 Appletviewer.exe 就是用来解释执行java applet应用程序的,简单理解就是没有main函数的继承applet类的java类。

D选项中 能被Appletviewer成功运行的java class文件没有main()方法。

【例题】下列说法正确的有（ ）

环境变量可在编译source code时指定

在编译程序时，所能指定的环境变量不包括class path

javac一次可同时编译数个Java源文件

javac.exe能指定编译结果要置于哪个目录（directory）

a选项-d即可设置系统属性

c选项一次编译多个java文件用javac \*.java. 即可编译当前目录下的所有java文件

d选项－s指定存放生成的源文件的位置

【例题】下列说法错误的有（ ）

Java面向对象语言容许单独的过程与函数存在

Java面向对象语言容许单独的方法存在

Java语言中的非静态方法属于类中的成员（member）

Java语言中的方法必定隶属于某一类（对象），调用方法与过程或函数相同

java不允许单独的方法，过程或函数存在,需要隶属于某一类中。——AB错

java语言中的方法属于对象的成员,而不是类的成员。不过，其中静态方法属于类的成员。——C错

【例题】下列哪个选项是Java调试器？如果编译器返回程序代码的错误，可以用它对程序进行调试。

java.exe

javadoc.exe

jdb.exe

javaprof.exe

java.exe是java虚拟机

javadoc.exe用来制作java文档

jdb.exe是java的调试器

javaprof.exe是剖析工具

【例题】在Java语言中，下列关于字符集编码（Character set encoding）和国际化（i18n）的问题，哪些是正确的？

每个中文字符占用2个字节，每个英文字符占用1个字节

假设数据库中的字符是以GBK编码的，那么显示数据库数据的网页也必须是GBK编码的。

Java的char类型，通常以UTF-16 Big Endian的方式保存一个字符。

实现国际化应用常用的手段是利用ResourceBundle类

A 显然是错误的，Java一律采用Unicode编码方式，每个字符无论中文还是英文字符都占用2个字节。

B 也是不正确的，不同的编码之间是可以转换的，通常流程如下：

将字符串S以其自身编码方式分解为字节数组，再将字节数组以你想要输出的编码方式重新编码为字符串。

例：String newUTF8Str = new String(oldGBKStr.getBytes("GBK"), "UTF8");

C 是正确的。Java虚拟机中通常使用UTF-16的方式保存一个字符

D 也是正确的。ResourceBundle能够依据Local的不同，选择性的读取与Local对应后缀的properties文件，以达到国际化的目的。

55、重载

【例题】在类Tester中定义方法如下，

public double max(int x, int y) { // 省略 }

则在该类中定义如下哪个方法头是对上述方法的重载(Overload)?

public int max(int a, int b) {}

public int max(double a, double b) {}

public double max(int x, int y) {}

private double max(int a, int b) {}

Java 重载的规则：

1、必须具有不同的参数列表；

2、可以有不同的返回类型，只要参数列表不同就可以；（即不能只是返回类型不同）

3、可以有不同的访问修饰符；

4、可以抛出不同的异常；

5、方法能够在一个类中或者在一个子类中被重载。

56、匿名内部类

【例题】关于匿名内部类叙述正确的是？ ( )

匿名内部类可以继承一个基类，不可以实现一个接口

匿名内部类不可以定义构造器

匿名内部类不能用于形参

以上说法都不正确

匿名内部类的创建格式为：

new 父类构造器（参数列表）|实现接口（）{

//匿名内部类的类体实现

}

使用匿名内部类时，必须继承一个类或实现一个接口

匿名内部类由于没有名字，因此不能定义构造函数

匿名内部类中不能含有静态成员变量和静态方法

57、构造器

【例题】下列说法正确的有（ ）

class中的constructor不可省略

constructor必须与class同名，但方法不能与class同名

constructor在一个对象被new时执行

一个class只能定义一个constructor

其实普通的类方法是可以和类名同名的，和构造方法唯一的区分就是，构造方法没有返回值。

58、JSP的四大作用域

【例题】Web程序中，当前用户上下文信息应该保存在下面哪个对象中（）

page

request

session

Application

JSP 四大作用域： page (作用范围最小)、request、session、application（作用范围最大）。

存储在application对象中的属性可以被同一个WEB应用程序中的所有Servlet和JSP页面访问。（属性作用范围最大）

存储在session对象中的属性可以被属于同一个会话（浏览器打开直到关闭称为一次会话，且在此期间会话不失效）的所有Servlet和JSP页面访问。

存储在request对象中的属性可以被属于同一个请求的所有Servlet和JSP页面访问（在有转发的情况下可以跨页面获取属性值），例如使用PageContext.forward和PageContext.include方法连接起来的多个Servlet和JSP页面。

存储在pageContext对象中的属性仅可以被当前JSP页面的当前响应过程中调用的各个组件访问，例如，正在响应当前请求的JSP页面和它调用的各个自定义标签类。

59、JDBC的Statement类

【例题】下面哪个不是标准Statement类？

Statement

PreparedStatement

CallableStatement

BatchedStatement

Statement在JDBC中相当于SQL语句的载体

A，Statement是最基本的用法，采用字符串拼接的方式，存在注入漏洞

B，PreparedStatement对Statement中的SQL语句进行预编译，同时检查合法性，效率高

C，CallableStatement接口扩展 PreparedStatement，用来调用存储过程,它提供了对输出和输入/输出参数的支持。CallableStatement 接口还具有对 PreparedStatement 接口提供的输入参数的支持。

D，不是标准的Statement类

60、JSP分页代码

【例题】JSP分页代码中，哪个步骤次序是正确的？

先取总记录数，得到总页数，最后显示本页的数据。

先取所有的记录，得到总页数，再取总记录数，最后显示本页的数据。

先取总页数，得到总记录数，再取所有的记录，最后显示本页的数据。

先取本页的数据，得到总页数，再取总记录数，最后显示所有的记录。

61、类变量的初始化

【例题】关于下列程序段的输出结果，说法正确的是：（ ）

public class MyClass{

static int i;

public static void main(String argv[]){

System.out.println(i);

}

}

有错误，变量i没有初始化。

null

1

0

类变量在不设置初始值时，会进行默认值赋值，而局部方法中声明的变量则必须进行初始化，他不会进行默认值赋值。

Java中基本数据类型默认值为： 　　short: 0 　　int: 0 　　long:0 　　float: 0.0 　　double: 0.0 　　char:0 　　String:null 　　Java中对象引用默认值为null

62、静态内部类

【例题】静态内部类不可以直接访问外围类的非静态数据，而非静态内部类可以直接访问外围类的数据，包括私有数据。（ ）

正确

错误

静态内部类可以直接访问外部类的静态成员。

实例内部类可以直接访问外部类的各个访问级别的成员。

63、包装类

【例题】关于Float，下列说法错误的是()

Float是一个类

Float在java.lang包中

Float a=1.0是正确的赋值方法

Float a= new Float(1.0)是正确的赋值方法

1. Float是类，float不是类.

2. 查看JDK源码就可以发现Byte，Character，Short，Integer，Long，Float，Double，Boolean都在java.lang包中.

3. Float正确复制方式是Float f=1.0f,若不加f会被识别成double型,double无法向float隐式转换.

4. Float a= new Float(1.0)是正确的赋值方法，但是在1.5及以上版本引入自动装箱拆箱后，会提示这是不必要的装箱的警告，通常直接使用Float f=1.0f.

64、Socket

【例题】关于 Socket 通信编程，以下描述错误的是：（ ）

服务器端通过new ServerSocket()创建TCP连接对象

服务器端通过TCP连接对象调用accept()方法创建通信的Socket对象

客户端通过new Socket()方法创建通信的Socket对象

客户端通过new ServerSocket()创建TCP连接对象

Client通过 new socket(ip,端口号)来建立与服务端连接

65、for循环执行顺序

【例题】以下程序的输出结果是

public class Print{

static boolean out(char c){

System.out.println(c);

return true;

}

public static void main(String[] argv){

int i = 0;

for(out('A');out('B') && (i<2);out('C')){

i++;

out('D');

}

}

}

ABDCBDCB

BCDABCD

编译错误

运行错误

for循环的执行顺序用如下表达式：

for(expression1;expression2;expression3){

expression4;

}

执行的顺序应该是：

1）第一次循环，即初始化循环。

首先执行表达式expression1（一般为初始化语句）；再执行expression2（一般为条件判断语句），判断expression1是否符合expression2的条件；如果符合，则执行expression4，否则，停止执行；最后执行expression3。

2）第N（N>=2）次循环

首先执行expression2，判断在expression3是否符合在expression2要求；如果符合，则继续执行在expression4，否则，停止执行。最后执行在expression3。如此往复，直至expression3不满足在expression2条件是为止。

总结：

总的来说，执行的顺序是一致的。先条件判断（expression2），再函数体执行（expression4），最后for执行（expression3）。往复......区别在于，条件判断的对象。第一次判断时，对象为初始化语句（expression1），后续的判断对象为执行后的结果（expression3）。

66、正则表达式

【例题】Java中用正则表达式截取字符串中第一个出现的英文左括号之前的字符串。比如：北京市（海淀区）（朝阳区）（西城区），截取结果为：北京市。正则表达式为（）

".\*?(?=\\()"

".\*?(?=\()"

".\*(?=\\()"

".\*(?=\()"

String str = "北京市(海淀区)(朝阳区)(西城区)";

String patStr = ".\*?(?=\\()";

Pattern pattern = Pattern.compile(patStr);

Matcher matcher = pattern.matcher(str);

if(matcher.find())

{

System.out.println(matcher.group(0));

}

else

{

System.out.println(matcher.group(0));

}

1.什么是正则表达式的贪婪与非贪婪匹配

如：String str="abcaxc";

Patter p="ab\*c";

贪婪匹配:正则表达式一般趋向于最大长度匹配，也就是所谓的贪婪匹配。如上面使用模式p匹配字符串str，结果就是匹配到：abcaxc(ab\*c)。

非贪婪匹配：就是匹配到结果就好，就少的匹配字符。如上面使用模式p匹配字符串str，结果就是匹配到：abc(ab\*c)。

2.编程中如何区分两种模式

默认是贪婪模式；在量词后面直接加上一个问号？就是非贪婪模式。

量词：{m,n}：m到n个

\*：任意多个

+：一个到多个

？：0或一个

以上来自博主的博客，然后这道题目

.表示除\n之外的任意字符

\*表示匹配0-无穷

+表示匹配1-无穷

(?=Expression) 顺序环视，(?=\\()就是匹配正括号

懒惰模式正则：

src=".\*? (?=\\()"

结果：北京市

因为匹配到第一个"就结束了一次匹配。不会继续向后匹配。因为他懒惰嘛。

67、并发

【例题】下列关于java并发的说法中正确的是：

copyonwritearraylist适用于写多读少的并发场景

readwritelock适用于读多写少的并发场景

concurrenthashmap的写操作不需要加锁，读操作需要加锁

只要在定义int类型的成员变量i的时候加上volatile关键字，那么多线程并发执行i++这样的操作的时候就是线程安全的了

CopyOnWriteArrayList适合使用在读操作远远大于写操作的场景里，比如缓存。

ReadWriteLock 当写操作时，其他线程无法读取或写入数据，而当读操作时，其它线程无法写入数据，但却可以读取数据 。适用于读取远远大于写入的操作。

ConcurrentHashMap是一个线程安全的HashTable，它的主要功能是提供了一组和HashTable功能相同但是线程安全的方法。ConcurrentHashMap可以做到读取数据不加锁，并且其内部的结构可以让其在进行写操作的时候能够将锁的粒度保持地尽量地小，不用对整个ConcurrentHashMap加锁。

volatile能保证能保证可见性，基本数据类型的读取和赋值具有原子性。因此volatile能保证基本数据类型的读取和赋值是线程安全的。对应i++操作，实际上包括了读取和赋值2个操作，volatile不能保证原子性，因此不是线程安全的。

68、基本类型与引用类型

【例题】以下哪个类型是基本数据类型（）

int

String

Byte

Float

基本类型四类八种，其余全是引用类型

四类八种：

整数（byte short int long)

浮点数(float double)

字符型(char)

逻辑性(boolean)

注意大小写，大写的是封装基本类型的引用类型。

byte - Byte

short - Short

int - Integer

long - Long

float - Float

double - Double

char - Charactor

boolean - Boolean

69、内联函数

【例题】在（ ）情况下适宜采用内联函数。

函数代码小，频繁调用

函数代码多，频繁调用

函数体含有递归语句

函数体含有循环语句

在说内联函数之前，先说说函数的调用过程。

调用某个函数实际上将程序执行顺序转移到该函数所存放在内存中某个地址，将函数的程序内容执行完后，再返回到转去执行该函数前的地方。这种转移操作要求在转去前要保护现场并记忆执行的地址，转回后先要恢复现场，并按原来保存地址继续执行。也就是通常说的压栈和出栈。因此，函数调用要有一定的时间和空间方面的开销。那么对于那些函数体

代码不是很大，又频繁调用的函数来说，这个时间和空间的消耗会很大。

那怎么解决这个性能消耗问题呢，这个时候需要引入内联函数了。内联函数就是在程序编译时，编译器将程序中出现的内联函数的调用表达式用内联函数的函数体来直接进行替换。显然，这样就不会产生转去转回的问题，但是由于在编译时将函数体中的代码被替代到程序中，因此会增加目标程序代码量，进而增加空间开销，而在时间代销上不象函数调用时那么大，可见它是以目标代码的增加为代价来换取时间的节省。

在大学里学习写C代码时，我们都学到将一些简短的逻辑定义在宏里。这样做的好处是，在编译器编译的时候会将用到该宏的地方直接用宏的代码替换。这样就不再需要象调用方法那样的压栈、出栈，传参了。性能上提升了。内联函数的处理方式与宏类似，但与宏又有所不同，内联函数拥有函数的本身特性（类型、作用域等等）

写过C++代码的应该都知道，在C++里有个内联函数，使用inline关键字修饰。另外，写在Class定义内的函数也会被编译器视为内联函数。

那么，在java中的内联函数长什么模样呢？在java中使用final关键字来指示一个函数为内联函数，例如：

public final void method1() {

//TODO something

}

这个指示并不是必需的。final关键字只是告诉编译器，在编译的时候考虑性能的提升，可以将final函数视为内联函数。但最后编译器会怎么处理，编译器会分析将final函数处理为内联和不处理为内联的性能比较了。

70、Hibernate

【例题】在 hibernate 开发中，关于 POJO 类对象的状态说法正确的是（）。

自由状态（Transient）：实体在内存中自由存在，与数据库中的记录无关

持久状态（Persistent）：实体处于由Hibernate框架所管理的状态，对应了数据库中的一条记录，同时与某个session实例发生了关联

游离状态（Detached）：在session 关闭之后,可以使对象从持久状态转换到游离状态。

不能将对象从游离状态转换成持久态

Session的update()，saveOrUpdate()和lock()方法使游离状态变为持久化状态。

持久化对象的特征：

位于一个session事例的缓存中,可以说持久化对象总是被一个session事例关联；

持久化对象和数据库中的相关记录对应；

Session在清理缓存时,会根据持久化对象的属性变化来同步更新数据库；

Session的save()方法把临时状态变为持久化状态；

Session的update()、saveOrUpdate()和lock()方法使游离状态变为持久化状态。

71、Java的关键字

【例题】true、false、null、sizeof、goto、synchronized 哪些是Java关键字？

true

false

null

sizeof

goto

synchronized

goto和const是保留字也是关键字。

1.Java 关键字列表 (依字母排序 共50组)：

abstract, assert, boolean, break, byte, case, catch, char, class, const（保留关键字）, continue, default, do, double, else, enum, extends, final, finally, float, for, goto（保留关键字）, if, implements, import, instanceof, int, interface, long, native, new, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, try, void, volatile, while

2.保留字列表 (依字母排序 共14组)，Java保留字是指现有Java版本尚未使用，但以后版本可能会作为关键字使用：

byValue, cast, false, future, generic, inner, operator, outer, rest, true, var, goto （保留关键字） , const （保留关键字）, null