**错题总结（基础知识）**

## Java部分

Day1:2018/5/18

### **1、HashMap**

底层由数组加链表实现，对于每一个key值都需要计算hash值，然后通过hash值来确定顺序，不是按照加入的顺序存放的，因此可以认为是无序的，最开始有Hashtable，Hashtable是不允许key和value的值为空的，但后来开发者认为有时候也会有key值为空的情况，因为可以允许null为空，通过查看HashMap的源代码就知道：

if(key = null) {putForNullKey(value);};

 HashMap如何实现?

 HashMap是不是线程安全?为什么非线程安全或者说哪里体现了非线程安全?HashMap的读写线程安全吗?

### 2、以.java为后缀的源文件

只能有一个与文件名相同的类，可以包含其他类，就是只能有一个public的类

### %和\*是同样的优先级

从左到右运算,10%3\*2 = 2

### 4、synchronized

Jvm是使用监视器锁来实现不同进程的异步执行，语法的关键字就是使用synchronized

### 5、++i / i++

Int i=5;int s = (i++)+(++i)+(i--)+(--i) s = 24

5+7+7+5=24

public class Test {  
 public static void main(String[] args) throws Exception {  
 Thread thread = new Thread((new Runnable() {  
 @Override  
 public void run() {  
 try {  
 Thread.*sleep*(2000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 System.*out*.print("2");  
 }  
 }));  
 thread.start();  
 thread.join();  
 System.*out*.print("1");  
 }  
}

public class Test {  
 public static void main(String[] args) throws Exception {  
 int i = 5;  
 int s = (i++);  
 s +=(++i);  
 s +=(i--);  
 s +=(--i);  
 System.*out*.println(s);  
 }  
}

### 6、Java数据区

运行时数据区包括：虚拟机栈区、堆区、方法区、本地方法栈、程序计数器

虚拟机栈区：线程私有、存放基本类型、对象的引用和returnAddress，在编译期间完成分配

堆区：Java堆，也称GC堆，所有线程共享，存放对象的实例和数组，Java堆是垃圾收集器管理的主要区域

方法区：所有线程共享，存储已被虚拟器编译后的代码等数据，这个区域的内存回收目标主要是针对常量池的对象的回收和对类型的卸载

程序计数器：线程私有，每个线程都有自己独立的程序计数器，用来指示下一条指令的地址

### 7、面向对象五大原则:

单一职责原则（SRP）、开放封闭原则（OCP）、里氏替换原则（LSP）、依赖倒置原则（DIP）、接口隔离原则（ISP）

****s(**** Single-Resposibility Principle ****):**** 单一职责原则

一个类，最好只做一件事，只有一个引起它的变化。单一职责原则可以看做是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因。

****o(**** Open-Closed principle ****):**** 开放封闭原则

软件实体应该是可扩展的，而不可修改的。也就是，对扩展开放，对修改封闭的

****l(**** Liskov-Substituion Principle ****):**** 里氏原则

子类必须能够替换其基类。这一思想体现为对继承机制的约束规范，只有子类能够替换基类时，才能保证系统在运行期内识别子类，这是保证继承复用的基础

****i(**** Interface-Segregation Principle ****):**** 接口隔离原则

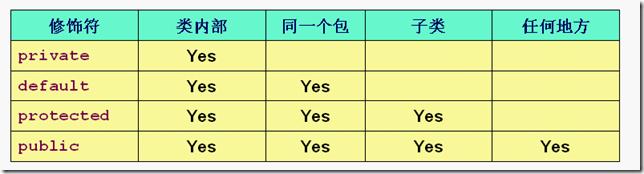
使用多个小的专门的接口，而不要使用一个大的总接口

****d(**** Dependecy-Inversion Principle ****):**** 依赖倒置原则

依赖于抽象。具体而言就是高层模块不依赖于底层模块，二者都同依赖于抽象；抽象不依赖于具体，具体依赖于抽象

### 8、访问权限

类定义前面只能是public或者default，而内部类是放在外部类中，和成员变量地位一致，所以有四种访问权限



### 9、main函数执行

是一个前台进程，前台进程是程序必须执行完成的，而后台进程是所有前台进程结束后结束，不管有没有完成，后台进程主要用于内存分配方面。

### 静态初始化代码块、构造代码块、构造方法执行顺序

在调用子类构造器之前，会先调用父类构造器，当子类构造器中没有使用"super(参数或无参数)"指定调用父类构造器时，是默认调用父类的无参构造器，如果父类中包含有参构造器，却没有无参构造器，则在子类构造器中一定要使用“super(参数)”指定调用父类的有参构造器，不然就会报错

**类的初始化顺序是**： 1、初始化父类中的静态成员变量和静态代码块。 2、初始化子类中的静态成员变量和静态代码块。 3、初始化父类中的普通成员变量和代码块，在执行父类中的构造方法。 4、初始化子类中的普通成员变量和代码块，在执行子类中的构造方法。

1.静态代码块 2.构造代码块3.构造方法的执行顺序是1>2>3;明白他们是干嘛的就理解了。  
1.静态代码块：是在类的加载过程的第三步初始化的时候进行的，主要目的是给类变量赋予初始值。  
2.构造代码块：是独立的，必须依附载体才能运行，Java会把构造代码块放到每种构造方法的前面，用于实例化一些共有的实例变量，减少代码量。  
3.构造方法：用于实例化变量。  
1是类级别的，2、3是实例级别的，自然1要优先23.  
在就明白一点：对子类得主动使用会导致对其父类得主动使用，所以尽管实例化的是子类，但也会导致父类的初始化和实例化，且优于子类执行。

### 11、字符读写过程

一个文件中的字符要写到另外一个文件中，首先要进行的是建立文件字符输入流，首先需要读取这个文件，所以要先建立输入流，然后再写到另外一个文件中，这时候建立输出流

### 12、try...catch...finally块

如果try中有return，是先将return的值保存在一个临时栈中，等到finally块执行完以后，程序就可以返回了，但是，如果finally块中**有return语句（不是赋值语句）**，那么会覆盖之前的值，所以返回的是finally块中的值

### 13、Suspend()和resume()

俩个方法配套使用，suspend使线程进入阻塞状态，并且不会自动恢复，必须其对应的resume()方法被调用才能使线程重新进入可执行状态

### 14、自动数据类型转换

从低到高，顺序：

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

### 15、重写方法（override）原则：

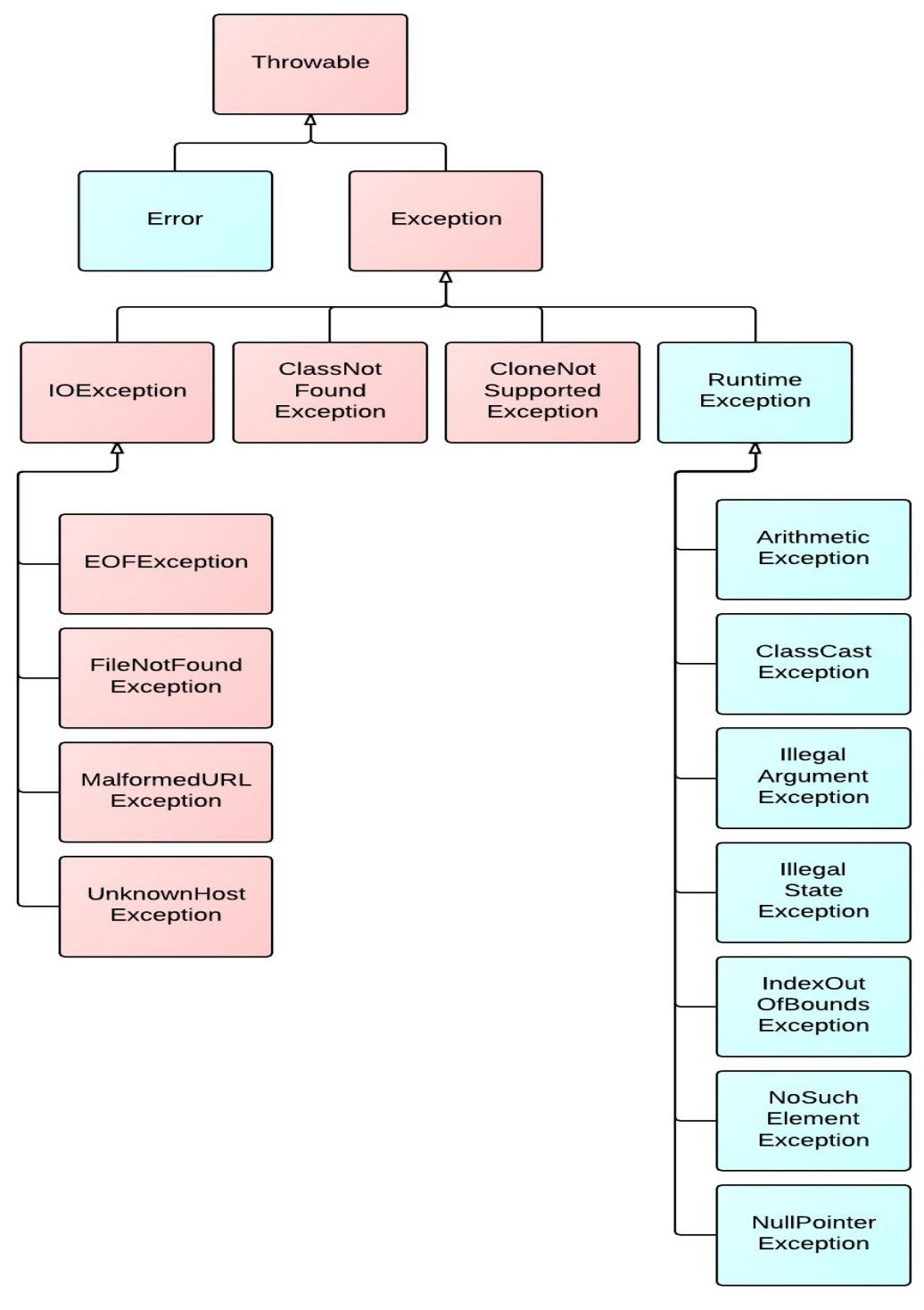
两同两小一大原则：函数名称，参数列表相同；子类返回类型、子类抛出异常小于等于父类；子类访问权限大于等于父类

### 16、ArrayList 、 LinkedList 、 HashMap

HashMap实现Map接口，它允许任何类型的键和值对象，并允许将null用作键或值 ,ArrayList和LinkedList均**实现**了List接口,ArrayList的访问速度比LinkedList快

ArrayList list = new ArrayList(20);中的list扩充**0**次！默认ArrayList的长度是10个，所以如果你要往list里添加20个元素肯定要扩充一次（扩充为原来的1.5倍），但是这里显式指明了需要多少空间，所以就一次性为你分配这么多空间，也就是不需要扩充了。

### 17、Java中的异常API



(1). 粉红色的是受检查的异常(checked exceptions),其必须被 try{}catch语句块所捕获,或者在方法签名里通过throws子句声明.受检查的异常必须在编译时被捕捉处理,命名为 Checked Exception 是因为Java编译器要进行检查,Java虚拟机也要进行检查,以确保这个规则得到遵守.

(2). 绿色的异常是运行时异常(runtime exceptions),需要程序员自己分析代码决定是否捕获和处理,比如 空指针,被0除...

(3). 而声明为Error的，则属于严重错误，如系统崩溃、虚拟机错误、动态链接失败等，这些错误无法恢复或者不可能捕捉，将导致应用程序中断，Error不需要捕捉。

**运行异常和error一样，不用我们捕获或者抛出**

### 18、JAVA反射机制

主要提供了以下功能：

在**运行时**判断一个对象所属的类、在**运行时**构造一个类的对象、在**运行时**判断一个类所具有的成员变量和方法、在**运行时**调用一个对象的方法

### 19、捕获到的异常

不仅可以在当前方法中处理，还可以将异常抛给调用它的上一级方法来处理。

### 20、JSP生命周期

JSP会先解释成Servlet源文件，然后编译成Servlet类文件，每当用户端运行JSP时，jsp service()方法都会运行一次。

6.8

### 21、抽象类和接口总结：

抽象类：

1. 抽象类中可以构造方法
2. 抽象类中可以存在普通属性，方法，静态属性和方法。
3. 抽象类中**可以**存在抽象方法。
4. 如果一个类中有一个抽象方法，那么当前类一定是抽象类；抽象类中不一定有抽象方法。
5. 抽象类中的抽象方法，需要有子类实现，如果子类不实现，则子类也需要定义为抽象的。
6. 抽象方法必须为public或者protected（因为如果为private，则不能被子类继承，子类便无法实现该方法），缺省情况下默认为public。
7. 抽象类不能用来创建对象；

接口：

1）在接口中只有方法的声明，没有方法体。

1. 在接口中只有常量，因为定义的变量，在编译的时候都会默认加上public static final
2. 在接口中的方法，永远都被public来修饰。
3. 接口中没有构造方法，也不能实例化接口的对象。
4. 接口可以实现多继承
5. 接口中定义的方法都需要有实现类来实现，如果实现类不能实现接口中的所有方法则实现类定义为抽象类。

|  |
| --- |
| **interface**中的方法默认为**public** **abstract** 的 ，变量默认为**public** **static** **final** |

**区别与联系**

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。

**哪些情况下需要使用抽象类？**

当一个类的一个或多个方法是抽象方法时；当类是一个抽象类的子类，并且不能为任何抽象方法提供任何实现细节或方法体时；当一个类实现一个接口，并且不能为任何抽象方法提供实现细节或方法体时

### 22、ant和maven总结：

Ant和Maven都是基于Java的构建(build)工具。理论上来说，有些类似于（Unix）C中的make ，但没有make的缺陷。Ant是软件构建工具，Maven的定位是软件项目管理和理解工具。

Ant：

没有一个约定的目录结构，必须明确让ant做什么，什么时候做，然后编译，打包，没有生命周期，必须定义目标及其实现的任务序列 没有集成依赖管理

Maven：

拥有约定，知道你的代码在哪里，放到哪里去拥有一个生命周期，例如执行 mvn install 就可以自动执行编译，测试，打包等构建过程只需要定义一个pom.xml,然后把源码放到默认的目录，Maven帮你处理其他事情，拥有依赖管理，仓库管理

6.10

### 23、重载和重写区别

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 类 | 参数 | 返回值 | 权限 | 异常 |
| 重载 | 同一类 | 参数类型，个数不同 | 无关 | 无关 | 无 |
| 重写 | 子类父类 | 必须相同 | 相同 | 大于等于原函数 | 不能新的或更宽泛的 |

### 24、类中各块执行顺序

|  |  |  |  |
| --- | --- | --- | --- |
| 块名称 | 含义 | 执行时间 | 执行次数 |
| 静态块 | 用static申明 | Jvm加载类时执行 | 只执行一次 |
| 构造块 | 类中直接用{}定义 | 每一次创建类对象时执行 | 创建次数 |

所以总的执行顺序是：**静态块>mian()>构造块>构造方法**

6.11

### 25、Java语言的数组复制方法

For循环逐一复制、System.arraycopy、Array.copyOf、使用clone方法。效率最高的是**System.arraycopy**

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

解析：System类源码中给出了arraycopy的方法，是native方法，也就是本地方法，肯定是最快的

Arrays.copyOf(注意是Arrays类，不是Array)的实现，在源码中是调用System.copyOf的

6.13

### 26、SPRING的事务传播特性

6.15

### 27、Java内联函数的使用

**函数调用过程：**

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。

**内联函数：**

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。

C语言，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）

C++，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。

### 28、run方法和start方法

通过调用Thread类的start()方法来启动一个线程，这时此线程是处于就绪状态，并没有运行

run方法是thread的一个普通方法调用

调用start方法后，一旦得到cpu时间片，就开始执行run()方法

### 29、自定义异常

if(i>10)throw new Exception("something’swrong!");

### 30、Volatile关键字作用

出于运行速率的考虑，java编译器会把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**

### 31、线程安全的map实现

在JDK 1.5及其更高版本环境实现：

Map map = new ConcurrentHashMap();

Map map = Collections.synchronizedMap(new HashMap())；

### JSP内置对象及方法

request、out、application、config

### ThreadLocal类

ThreadLocal是采用哈希表的方式来为每个线程都提供一个变量的副本

ThreadLocal保证各个线程间数据安全，每个线程的数据不会被另外线程访问和破坏

### forEach循环

通过**break**跳出Array的froEach循环

### &&和&区别

& is bitwise. && is logical.

& evaluates **both sides** of the operation.

&& evaluates the left side of the operation, if it's true, it continues and evaluates the right side.

### JDBC

Statement继承自Wrapper、PreparedStatement继承自Statement、CallableStatement继承自PreparedStatement。

Statement接口提供了执行语句和获取结果的基本方法； PreparedStatement接口添加了处理 IN 参数的方法； CallableStatement接口添加了处理 OUT 参数的方法。

Statement、PreparedStatement 和 CallableStatement三种方式来执行查询语句，其中 Statement 用于**通用查询**，普通的不带参的查询SQL支持批量更新,批量删除**;**PreparedStatement 用于执行参数化查询，**可变参数的SQL,编译一次,执行多次,效率高;安全性好，有效防止Sql注入等问题;** 支持批量更新,批量删除;而 CallableStatement则是用于存储过程，继承自PreparedStatement,支持带参数的SQL操作；支持调用存储过程,提供了对输出和输入/输出参数(INOUT)的支持;

对于PreparedStatement来说，数据库可以使用已经编译过及定义好的执行计划，由于 PreparedStatement 对象已预编译过，所以其执行速度要快于 Statement 对象”

### sleep和wait区别

sleep和wait的区别有：

1，这两个方法来自不同的类分别是Thread和Object

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

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

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

### i++

package algorithms.com.guan.javajicu;

public class Inc {

public static void main(String[] args) {

Inc inc = new Inc();

int i = 0;

inc.fermin(i);

i= i ++;

System.out.println(i);

}

void fermin(int i){

i++;

}

}

**输出：0**

原因：jvm里面有两个存储区，一个是暂存区（是一个堆栈，以下称为堆栈），另一个是变量区。jvm会这样运行这条语句，　JVM把count值（其值是0）拷贝到临时变量区。 步骤2　count值加1，这时候count的值是1。 步骤3　返回临时变量区的值，注意这个值是0，没修改过。 步骤4　返回值赋值给count，此时count值被重置成0。 c/c++中没有另外设置一个临时变量或是临时空间来保存i，所有操作都是在一个内存空间中完成的，所以在c/c++中是1。

### 基本数据类型比较

两个数值进行二元操作时，会有如下的转换操作：

如果两个操作数其中有一个是double类型，另一个操作就会转换为double类型。

否则，如果其中一个操作数是float类型，另一个将会转换为float类型。

否则，如果其中一个操作数是long类型，另一个会转换为long类型。

否则，两个操作数都转换为int类型。

### Java语言-解释性语言

编译型语言的首先将源代码编译生成机器语言，再由机器运行机器码（二进制）。像C/C++等都是编译型语言。程序执行效率高，依赖编译器，跨平台性差些。如C、C++、Delphi等

解释性语言在运行程序的时候才翻译，比如解释性basic语言，专门有一个解释器能够直接执行basic程序，每个语句都是执行的时候才翻译。这样解释性语言每执行一次就要翻译一次，效率比较低。如JavaScript、VBScript、Perl、Python、Ruby、MATLAB 等等

Java不同于一般的编译语言和解释语言。它首先将源代码编译成字节码，然后依赖各种不同平台上的虚拟机来解释执行字节码，从而实现了“一次编写，到处运行”的跨平台特性， 所以说java是一种半编译半解释型的语言

### Java字节流 字符流

stream结尾都是字节流，reader和writer结尾都是字符流 两者的区别就是读写的时候一个是按字节读写，一个是按字符。 实际使用通常差不多。 在读写文件需要对内容按行处理，比如比较特定字符，处理某一行数据的时候一般会选择字符流。 只是读写文件，和文件内容无关的，一般选择字节流

字节流：

InputStream

|-- FileInputStream (基本文件流）

|-- BufferedInputStream

|-- DataInputStream

|-- ObjectInputStream

字符流

Reader

|-- InputStreamReader (byte->char 桥梁）

|-- BufferedReader (常用）

Writer

|-- OutputStreamWriter (char->byte 桥梁）

|-- BufferedWriter

|-- PrintWriter （常用）

### 引用传递 值传递

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；

### 实例变量,局部变量,类变量和final变量

定义在类中的变量是类的**成员变量**，可以不进行初始化，Java会**自动**进行初始化，如果是引用类型认默初始化为null,如果是基本类型例如int则会默认初始化为0

**局部变量**是定义在方法中的变量，必须要进行初始化，否则不同通过编译

被static关键字修饰的变量是静态的，静态变量随着类的加载而加载，所以也被称为**类变量**

被final修饰的变量是**常量**

### 装箱拆箱

在jdk1.5的环境下，有如下4条语句：

|  |  |
| --- | --- |
| 1  2  3  4 | Integer i01 = 59;  **int** i02 = 59;  Integer i03 =Integer.valueOf(59);  Integer i04 = **new** Integer(59); |

Integer i01=59 的时候，会调用 Integer 的 valueOf 方法，

|  |  |
| --- | --- |
| 1  2  3  4  5 | **public** **static** Integer valueOf(**int** i) {  **assert** IntegerCache.high>= 127;  **if** (i >= IntegerCache.low&& i <= IntegerCache.high)  **return** IntegerCache.cache[i+ (-IntegerCache.low)];  **return** **new** Integer(i); } |

这个方法就是返回一个 Integer 对象，只是在返回之前，看作了一个判断，判断当前 i 的值是否在 [-128,127] 区别，且 IntegerCache 中是否存在此对象，如果存在，则直接返回引用，否则，创建一个新的对象。

在这里的话，因为程序初次运行，没有 59 ，所以，直接创建了一个新的对象。

int i02=59 ，这是一个基本类型，存储在栈中。

Integer i03 =Integer.valueOf(59); 因为 IntegerCache 中已经存在此对象，所以，直接返回引用。

Integer i04 = new Integer(59) ；直接创建一个新的对象。

System. out .println(i01== i02); i01 是 Integer 对象， i02 是 int ，这里比较的不是地址，而是值。 Integer 会自动拆箱成 int ，然后进行值的比较。所以，为真。

System. out .println(i01== i03); 因为 i03 返回的是 i01 的引用，所以，为真。

System. out .println(i03==i04); 因为 i04 是重新创建的对象，所以 i03,i04 是指向不同的对象，因此比较结果为假。

System. out .println(i02== i04); 因为 i02 是基本类型，所以此时 i04 会自动拆箱，进行值比较，所以，结果为真。

### Static

静态块仅在类加载时执行一次、static方法中不能有用this调用的方法、可以用对象名来访问类中的静态方法(public权限)

### 46、InterruptedException

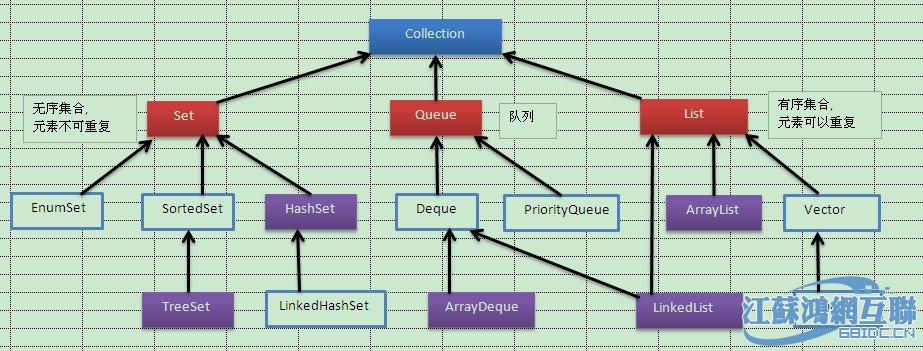
抛InterruptedException的代表方法有：

java.lang.Object 类的 wait 方法

java.lang.Thread 类的 sleep 方法

java.lang.Thread 类的 join 方法

### Collection接口



### Java注释

/\*..................\*/中可以嵌套//注释，不能嵌套/\*..........\*/注释。

### 48、

## 数据结构部分

### 冒泡排序，快排，和归并排序及优缺点和优化

### jvm的垃圾回收机制和垃圾收集器

### jdk1.8 concurrenthashmap 的新的特性，有没有看过源码

HashMap & ConcurrentHashMap介绍

### synchronized & Lock

### 设计模式

### JVM（GC、垃圾收集器）

### B、B+树、红黑树

数据库索引，对索引的理解，索引的优点，索引的分类。

hash索引和b树索引的区别，适用场景。

使用b树有什么优势，b树和b+树的区别，b树和红黑树有什么区别。

### top k问题

### 网络方面有osi七层，tcp/ip五层，分别有哪些协议及作用

## 在线编程