# NIO

JDK1.4中才引入

1. NIO和IO最大的区别是数据打包和传输方式。IO是以流的方式处理数据，而NIO是以块的方式处理数据。
2. Buffer是一个对象，它包含一些要写入或读出的数据。在NIO中，数据是放入buffer对象的，而在IO中，数据是直接写入或者读到Stream对象的。应用程序不能直接对 Channel 进行读写操作，而必须通过 Buffer 来进行，即 Channel 是通过 Buffer 来读写数据的。

Java IO是面向流的，这意味着我们需要每次从流中读取一个或多个字节，直到读取完所有字节；NIO是面向缓冲的，也就是说会把数据读取到一个缓冲区中，然后对缓冲区中的数据进行相应处理。

Java IO是阻塞IO，而NIO是非阻塞IO。

Java NIO中存在一个称为选择器（selector）的东西，它允许你把多个通道（channel）注册到一个选择器上，然后使用一个线程来监视这些通道：若这些通道里有某个准备好可以开始进行读或写操作了，则开始对相应的通道进行读写。而在等待某通道变为可读/写期间，请求对通道进行读写操作的线程可以去干别的事情。

# Java1.6,1.7,1.8

# Java异常

## 常见异常

1. [Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/liu_jian140126/article/details/_blank).lang.NullPointerException(空指针异常) 调用了未经初始化的对象或者是不存在的对象
2. java.lang.ClassNotFoundException

指定的类不存在

1. java.lang.NumberFormatException字符串转换为数字异常
2. java.lang.IndexOutOfBoundsException

数组下标越界异常

1. java.lang.IllegalArgumentException

方法的参数错误

6.java.lang.ArithmeticException

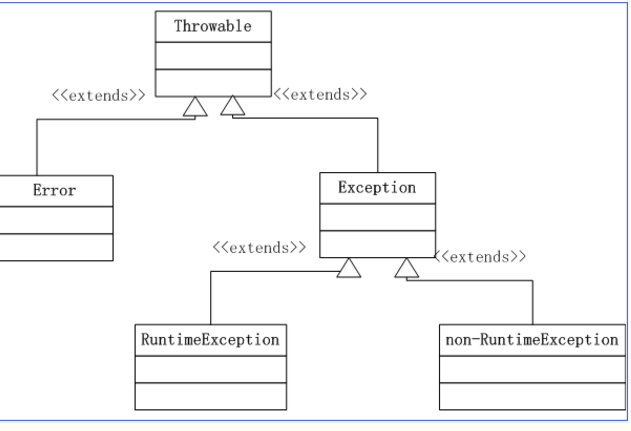
数学运算异常

7.java.lang.ClassCastException

数据类型转换异常

当试图将对某个对象强制执行向下转型，但该对象又不可转换又不可转换为其子类的实例时将引发该异常

## 结构解说



java中error和exception的区别

Error类和Exception类的父类都是throwable类，他们的区别是：

Error类一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢等。对于这类错误的导致的应用程序中断，仅靠程序本身无法恢复和和预防，遇到这样的错误，建议让程序终止。

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

Exception类又分为运行时异常（Runtime Exception）和受检查的异常(Checked Exception )，运行时异常;ArithmaticException,IllegalArgumentException，编译能通过，但是一运行就终止了，程序不会处理运行时异常，出现这类异常，程序会终止。而受检查的异常，要么用try。。。catch捕获，要么用throws字句声明抛出，交给它的父类处理，否则编译不会通过。

典型的RuntimeException包括NullPointerException, IndexOutOfBoundsException, IllegalArgumentException等；非RuntimeException包括IOException, ClassNotFoundException等。

而按照编译器检查方式划分，异常又可以分为检查型异常（CheckedException）和非检查型异常（UncheckedException）。Error和RuntimeException合起来称为UncheckedException，之所以这么称呼，是因为编译器不检查方法是否处理或者抛出这两种类型的异常，因此编译期间出现这种类型的异常也不会报错，默认由虚拟机提供处理方式。除了Error和RuntimeException这两种类型的异常外，其它的异常都称为Checked异常。

1.关于catch需要注意的几点：

1)、参数的异常类型必须是Throwable类或者其子类。

2)、从上往下的catch语句，其参数类型必须按照从子类到父类顺序，因为一旦匹配到一个类型，就会忽略往后的catch。比如IOException必须放到Exception前面，否则编译器会报错。

3)、可以有一个或者多个catch语句，甚至如果有finally语句的情况下，可以没有catch语句，如try-finally。

想要捕获多个异常，可以使用多个catch语句，JDK7以后提供了另外一种方式：多重捕获（multi-catch)。

2.4)、不要忽略异常。空的catch块会使异常达不到应有的目的，除非诸如关闭FileInputStream的时候，因为你还没有改变文件的状态，因此不必执行任何恢复动作，并且已经从文件中读取到所需要的信息，因此不用终止正在进行的操作。

关于finally需要注意的几点：

1)、finally中的代码总是会被执行，除非在执行try或者catch语句时虚拟机退出（System.exit(1))。

2)、finally块可以做一些资源清理工作，如关闭文件、关闭游标等操作。

3)、finally块不是必须的。

另外，如果在try和finally块中都执行了return语句，最终返回的将是finally中的return值。

## 自定义异常

1)、将检查型异常转换为非检查型异常。

2)、在产生异常时封装上下文信息、定义异常码、收集环境对象，有利于信息的传递。

# 快速失败安全失败

当错误发生时，如果系统立即关闭，即是快速失败，系统不会继续运行。运行中发生错误，它会立即停止操作，错误也会立即暴露。而安全失败系统在错误发生时不会停止运行。它们隐蔽错误，继续运行，而不会暴露错误。这两种模式，孰优孰优，是系统设计中常讨论的话题，在此，我们只讨论java中的快速失败和安全失败迭代器。

Java快速失败与安全失败迭代器 :

java迭代器提供了遍历集合对象的功能，集合返回的迭代器有快速失败型的也有安全失败型的，快速失败迭代器在迭代时如果集合类被修改，立即抛出ConcurrentModificationException异常，而安全失败迭代器不会抛出异常，因为它是在集合类的克隆对象上操作的。我们来看看快速失败和 安全失败迭代器的具体细节。

java快速失败迭代器 :

大多数集合类返回的快速失败迭代器在遍历时不允许结构性修改（结构性修改指添加，删除和更新一个元素） 当遍历的同时被结构性修改，就会抛出ConcurrentModificationException异常，而当集合是被迭代器自带的方法（如remove()）修改时，不会抛出异常。

**快速失败迭代器运行原理**：

1. 所有的集合类都维护着一个对象数组（Object[]）,用来存储元素, 快速失败迭代器直接从数组中获取元素，在迭代过程中，总是假定该内部数组不会被修改。为了判断这个集合是否被修改，它们使用名为modCount的内部标识，当集合被修改，该标识也会更新。迭代器每次调用next()方法，都会检查modCount，如果发现modCount被更新，就会抛出ConcurrentModificationException异常。
2. 原理：迭代器在遍历时直接访问集合中的内容，并且在遍历过程中使用一个 modCount 变量。集合在被遍历期间如果内容发生变化，就会改变modCount的值。每当迭代器使用hashNext()/next()遍历下一个元素之前，都会检测modCount变量是否为expectedmodCount值，是的话就返回遍历；否则抛出异常，终止遍历。

    注意：这里异常的抛出条件是检测到 modCount！=expectedmodCount 这个条件。如果集合发生变化时修改modCount值刚好又设置为了expectedmodCount值，则异常不会抛出。因此，不能依赖于这个异常是否抛出而进行并发操作的编程，这个异常只建议用于检测并发修改的bug。

ArrayList，Vector，HashMap等集合返回的迭代器都是快速失败类型的

Java安全失败迭代器 :

安全失败迭代器在迭代中被修改，不会抛出任何异常，因为它是在集合的克隆对象迭代的，所以任何对原集合对象的结构性修改都会被迭代器忽略，但是这类迭代器有一些缺点，其一是它不能保证你迭代时获取的是最新数据，因为迭代器创建之后对集合的任何修改都不会在该迭代器中更新，还有一个缺点就是创建克隆对象在时间和内存上都会增加一些负担。

ConcurrentHashMap返回的迭代器是安全失败迭代器：

# 谈谈面向对象

环境所谓的面向对象就是将我们的程序模块化，对象化，把具体事物的特性属性和通过这些属性来实现一些动作的具体方法放到一个类里面，这就是封装。封装是我们所说的面相对象编程的特征之一。除此之外还有继承和多态。继承有点类似与我们生物学上的遗传，就是子类的一些特征是来源于父类的，儿子遗传了父亲或母亲的一些性格，或者相貌，又或者是运动天赋。有点种瓜得瓜种豆得豆的意思。面向对象里的继承也就是父类的相关的属性，可以被子类重复使用，子类不必再在自己的类里面重新定义一回，父类里有点我们只要拿过来用就好了。而对于自己类里面需要用到的新的属性和方法，子类就可以自己来扩展了。当然，会出现一些特殊情况，就是我们在有一些方法在父类已经定义好了，但是子类我们自己再用的时候，发现，其实，我们的虽然都是计算工资的，但是普通员工的工资计算方法跟经理的计算方法是不一样的，所以这个时候，我们就不能直接调用父类的这个计算工资的方法了。这个时候我们就需要用到面向对象的另一个特性，多态。对，就是多态，我们要在子类里面把父类里面定义计算工资的方法在子类里面重新实现一遍。多态包含了重载和重写。重写很简单就是把子类从父亲类里继承下来的方法重新写一遍，这样，父类里相同的方法就被覆盖了，当然啦，你还是可以通过super.CaculSalary方法来调用父类的工资计算方法。而重载就是类里面相同方法名，不同形参的情况，可以是形参类型不同或者形参个数不同，或者形参顺序不同，但是不能使返回值类型不同。

1. 封装就是将对象的属性和行为特征包装到一个程序单元（即类）中，把实现细节隐藏起来，通过公用的方法来展现类对外提供的功能，提高了类的内聚性，降低了对象之间的耦合性。
2. 继承是对原有类的拓展，父类和子类之间是一般和特殊的关系，子类是一种特殊的父类。此外，子类还可以通过重写来改变父类中的方法，重写可以改变方法的返回类型和访问权限，不能改变方法名称。
3. 多态是建立在继承的基础上的，是指子类类型的对象可以赋值给父类类型的引用变量，但运行时仍表现子类的行为特征。也就是说，同一种类型的对象执行同一个方法时可以表现出不同的行为特征。

# 接口与抽象类

1.语法层面上的区别

　　1）抽象类可以提供成员方法的实现细节，而接口中只能存在public abstract 方法；

　　2）抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的；

　　3）接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；

1. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。

2.设计层面上的区别

　　1）抽象类是对一种事物的抽象，即对类抽象，而接口是对行为的抽象。抽象类是对整个类整体进行抽象，包括属性、行为，但是接口却是对类局部（行为）进行抽象。

1. 设计层面不同，抽象类作为很多子类的父类，它是一种模板式设计。而接口是一种行为规范，它是一种辐射式设计。

1、 抽象层次不同

1. 抽象类是对类抽象，而接口是对行为的抽象。抽象类是对整个类整体进行抽象，包括属性、行为，但是接口却是对类局部（行为）进行抽象。

2、 跨域不同

1. 抽象类所跨域的是具有相似特我们知道抽象类是从子类中发现公共部分，然后泛化成抽象类点的类，而接口却可以跨域不同的类。，子类继承该父类即可，但是接口不同。实现它的子类可以不存在任何关系，共同之处。例如猫、狗可以抽象成一个动物类抽象类，具备叫的方法。鸟、飞机可以实现飞Fly接口，具备飞的行为，这里我们总不能将鸟、飞机共用一个父类吧！所以说抽象类所体现的是一种继承关系，要想使得继承关系合理，父类和派生类之间必须存在"is-a" 关系，即父类和派生类在概念本质上应该是相同的。对于接口则不然，并不要求接口的实现者和接口定义在概念本质上是一致的， 仅仅是实现了接口定义的契约而已。

3、 设计层次不同

1. 对于抽象类而言，它是自下而上来设计的，我们要先知道子类才能抽象出父类，而接口则不同，它根本就不需要知道子类的存在，只需要定义一个规则即可，至于什么子类、什么时候怎么实现它一概不知。比如我们只有一个猫类在这里，如果你这是就抽象成一个动物类，是不是设计有点儿过度？我们起码要有两个动物类，猫、狗在这里，我们在抽象他们的共同点形成动物抽象类吧！所以说抽象类往往都是通过重构而来的！但是接口就不同，比如说飞，我们根本就不知道会有什么东西来实现这个飞接口，怎么实现也不得而知，我们要做的就是事前定义好飞的行为接口。所以说抽象类是自底向上抽象而来的，接口是自顶向下设计出来的。

|  |  |  |
| --- | --- | --- |
| ****参数**** | ****抽象类**** | ****接口**** |
| 默认的方法实现 | 它可以有默认的方法实现 | 接口完全是抽象的。它根本不存在方法的实现 |
| 实现 | 子类使用****extends****关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现。 | 子类使用关键字****implements****来实现接口。它需要提供接口中所有声明的方法的实现 |
| 构造器 | 抽象类可以有构造器 | 接口不能有构造器 |
| 与正常Java类的区别 | 除了你不能实例化抽象类之外，它和普通Java类没有任何区别 | 接口是完全不同的类型 |
| 访问修饰符 | 抽象方法可以有****public****、****protected****和****default****这些修饰符 | 接口方法默认修饰符是****public****。你不可以使用其它修饰符。 |
| main方法 | 抽象方法可以有main方法并且我们可以运行它 | 接口没有main方法，因此我们不能运行它。 |
| 多继承 | 抽象方法可以继承一个类和实现多个接口 | 接口只可以继承一个或多个其它接口 |
| 速度 | 它比接口速度要快 | 接口是稍微有点慢的，因为它需要时间去寻找在类中实现的方法。 |
| 添加新方法 | 如果你往抽象类中添加新的方法，你可以给它提供默认的实现。因此你不需要改变你现在的代码。 | 如果你往接口中添加方法，那么你必须改变实现该接口的类。 |

# Final关键字

## 1. final关键字的含义

　　http://www.cnblogs.com/liun1994/p/6691094.html

final表面意思就是不可更改的,恒量的意思；类似于C语言中的const关键字，指的是无法改变的量，这与静态标量static是有区别的，静态变量指的是只有一份存储空间，值是可以改变的。使用final一定原因是出于软件设计的角度，因为别人看到final这个关键字就知道是什么意思，达到心领神会的效果，但也正是由于这种"语义"的存在，在程序设计中要谨慎使用，以免误用。

　　在Java中final修饰的就是常量，而且变量名要大写；小写也可以

Math类：public static final double E = 2.7182818284590452354;public static final double PI = 3.14159265358979323846;

......java源码中好多变量都用final修饰 如string

## 2. final的作用怎么用

final根据修饰位置的不同作用也不相同，针对

### 三种情况：

**1）修饰变量**，被final修饰的变量必须要初始化，赋初值后不能再重新赋值。

　　　　　　注意：局部变量不在我们讨论的范畴，因为局部变量本身就有作用范围，不使用private、public等词修饰。

**2）修饰方法**，被final修饰的方法代表不能重写。

“使用final方法的原因有两个。第一个原因是把方法锁定，以防任何继承类修改它的含义；第二个原因是效率。在早期的Java实现版本中，会将final方法转为内嵌调用。但是如果方法过于庞大，可能看不到内嵌调用带来的任何性能提升。在最近的Java版本中，不需要使用final方法进行这些优化了。“

　　因此，如果只有在想明确禁止 该方法在子类中被覆盖的情况下才将方法设置为final的。

　　注：类的private方法会隐式地被指定为final方法。

**3）修饰类**，被final修饰的类，不能够被继承。

　　　　　　注意：final修饰的类，类中的所有成员方法都被隐式地指定为final方法。

## **final修饰变量**

　　　　被final修饰的变量必须显示的初始化，初始化可以以三种方式：1）定义时初始化，2）在构造器中设置值，3）在非静态块中为final实例变量设置值。

　　　　final修饰变量指的是：这个变量被初始化后便不可改变，这里不可改变的意思对基本类型来说是其值不可变，而对于对象变量来说其引用不可变，即不能再指向其他的对象。

[IMG_256](http://www.cnblogs.com/liun1994/p/javascript:void(0);)

public class Test01{

final int x1= 10000;

final int x2;

final int x3;

{

x2 = 20000;

}

Public exe3(){

this.x3 = 3000;

}

}

[IMG_257](http://www.cnblogs.com/liun1994/p/javascript:void(0);)

　　如果final修饰的变量是对象类型，那么不可更改指的是该变量不可以再指向别的对象，但是对象的值（只是对象的引用不能改变，对象的内容可变）时可以更改的，比如：

final Operate operate = new Operate() ;// operate有一个普通变量i初始化为10

operate.i = 11;

operate.i = 12;

System.out.println(operate.i); //输出12

上述是自定义类，即便是数组，List等集合类型，所保存的值也是可以更改的。

## 3. final和static的区别

　　static作用于成员变量用来表示只保存一份副本，而final的作用是用来保证变量不可变，看一下网上的一个例子：

[IMG_258](http://www.cnblogs.com/liun1994/p/javascript:void(0);)

public class Test {

public static void main(String[] args) {

MyClass myClass1 = new MyClass();

MyClass myClass2 = new MyClass();

System.out.println(myClass1.i);

System.out.println(myClass1.j);

System.out.println(myClass2.i);

System.out.println(myClass2.j);

}

}class MyClass {

public final double i = Math.random();

public static double j = Math.random();

}//运行结果，两次打印，i的值都是一样的，j的值时不同的。

[IMG_259](http://www.cnblogs.com/liun1994/p/javascript:void(0);)

## 4. 其他final相关的知识

　　1）使用final关键字，如果编译器能够在编译阶段确定某变量的值，那么编译器就会把该变量当做编译期常量来使用。如果需要在运行时确定，那么编译器就不会优化相关代码。

[IMG_260](http://www.cnblogs.com/liun1994/p/javascript:void(0);)

public class Test {

public static void main(String[] args) {

String a = "hello2";

final String b = "hello";

String d = "hello";

String c = b + 2;

String e = d + 2;

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

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

}

}//final类型，在编译阶段能够确定值。//非final类型在编译阶段确定不了输出：truefalsepublic class Test {

public static void main(String[] args) {

String a = "hello2";

final String b = getHello();

String c = b + 2;

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

}

public static String getHello() {

return "hello";

}

}//即便是final类型，编译阶段也确定不了值。输出false

[IMG_261](http://www.cnblogs.com/liun1994/p/javascript:void(0);)

　　2）注意不要将final与finally、finalize()等搞混。

　　3）将类、方法、变量声明为final能够提高性能，这样JVM就有机会进行估计，然后优化。

1. 接口中的变量都是public static final 的。

# Sleep和wait

Java中的多线程是一种抢占式的机制，而不是分时机制。抢占式的机制是有多个线程处于可运行状态，但是只有一个线程在运行。   
共同点 ：   
1. 他们都是在多线程的环境下，都可以在程序的调用处阻塞指定的毫秒数，并返回。   
2. wait()和sleep()都可以通过interrupt()方法 打断线程的暂停状态 ，从而使线程立刻抛出InterruptedException。   
如果线程A希望立即结束线程B，则可以对线程B对应的Thread实例调用interrupt方法。如果此刻线程B正在wait/sleep/join，则线程B会立刻抛出InterruptedException，在catch() {} 中直接return即可安全地结束线程。   
需要注意的是，InterruptedException是线程自己从内部抛出的，并不是interrupt()方法抛出的。对某一线程调用 interrupt()时，如果该线程正在执行普通的代码，那么该线程根本就不会抛出InterruptedException。但是，一旦该线程进入到 wait()/sleep()/join()后，就会立刻抛出InterruptedException 。   
不同点 ：    
1.每个对象都有一个锁来控制同步访问。Synchronized关键字可以和对象的锁交互，来实现线程的同步。   
sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法。   
2.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   
3.sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常

4.sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用sleep不会释放对象锁。

5.wait是Object类的方法，对此对象调用wait方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出notify方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。

# 集合中为什么只能存基本类型

1.这个你必须得知道集合中是怎么存放元素的，集合中存放的可都是对象的引用，实际内容都在堆上面或者方法区里面，但是基本数据类型是在栈上分配空间的。随时就被收回的。但是通过自动包装类就可以把基本类型转为对象类型，存放引用就解决了这个问题。

2.

首先，集合的存在就是为了方便对多个对象的操作才出现的，集合是存储对象最常用的一种方式，也就是说，集合的从有到无都是因为对象，人们发现要保存多个对象很麻烦，于是便发明了集合，集合是依赖对象而生的，所以就对基本数据类型"不感兴趣"，但是，现在基本数据类型都有了其对应的封装的对象，而且有了自动拆箱和装箱功能，基本数据类型和其对应对象之间的转换变得很方便，想把基本数据类型存入集合中，直接存就可以了，系统会自动将其装箱成封装类

然后加入集合当中