### 异常的分类

异常主要分为：Error、一般性异常、RuntimeException

Error：如果程序出现了Error，那么将无法恢复，只能重新启动程序，最典型的Error的异常是：OutOfMemoryError

一般性异常：出现了这种异常必须在程序里面显示的处理，否则程序无法编译通过

RuntimeException：此种异常可以不用显示的处理，例如被0除异常，java没有要求我们一定要处理。

**为何要用异常？**

JAVA号称安全性，异常考虑到可能出现的错误，

使用异常处理后：

1、把错误处理和真正的工作分开来；

2、代码更易组织，更清晰，复杂的工作任务更容易实现；

3、更安全，不至于由于一些小的疏忽而使程序意外崩溃了；

### throws

throws的作用是声明抛出异常，在方法声明的位置上使用throws关键字向上抛出异常。

**被finally控制的语句体一定会执行**，除非在执行finally语句体之前JVM退出(比如System.exit(0))，一般用于关闭资源

**final,finally和finalize的区别？**

**final**可以修饰类,不能被继承，修饰方法,不能被重写，修饰变量,只能赋值一次

**finally**是try语句中的一个语句体,不能单独使用,语句体中的语句一定会执行

**finalize**是Object中的一个方法,当没有引用指向这个对象时，由对象的垃圾回收器在回收之前调用此方法。

### throw和throws的区别

**throws**

用在方法声明后面，跟的是异常类名

可以跟多个异常类名，用逗号隔开

表示抛出异常，由该方法的调用者来处理

**throw**

用在方法体内，跟的是异常对象名

只能抛出一个异常对象名

表示抛出异常，由方法体内的语句处理

**Throwable类常用方法**

getMessage()返回异常发生时的详细信息

string toString()返回异常发生时的简要描述

string getLocalizedMessage()返回异常对象的本地化信息。使用Throwable的子类覆盖这个方法，可以声称本地化信息。如果子类没有覆盖该方法，则该方法返回的信息与getMessage（）返回的结果相同

void printStackTrace(:在控制台上打印Throwable对象封装的异常信息

**String**类在java.lang包下面，是Object类的直接子类，通过API或者源码可以看到，String类是final修饰的，这说明**String类不能被继承**。

字符串一旦创建好之后，里面的内容是不能被修改的，jvm会将双引号””中的内容存放在字符串常量池里面，常量池中的对象内容是不可修改的。创建字符串时，jvm会在常量池中创建一个字符串对象，创建另一个字符串时，jvm会去常量池中搜索，如果此时常量池中有相同的字符串对象，就不创建了，直接指向之前创建的字符串对象。

**StringBuffer**是一个字符串缓冲区，如果需要频繁的对字符串进行拼接时，建议使用StringBuffer。

**StringBuffer工作原理**  
**StringBuffer的底层是char数组，如果没有明确设定，则系统会默认创建一个长度为16的char类型数组，在使用时如果数组容量不够了，则会通过数组的拷贝对数组进行扩容，所以在使用StringBuffer时最好预测并手动初始化长度，这样能够减少数组的拷贝，从而提高效率。**

**String与StringBuffer的区别？**  
String是不可变字符序列，存储在字符串常量池中  
StringBuffer的底层是char数组，系统会对该数组进行扩容

### StringBuilder和StringBuffer的区别

通过API可以看到StringBuilder和StringBuffer里面的方法是一样的，那他们有什么区别呢？  
StringBuffer是jdk1.0版本中加入的，是线程安全的，效率低  
StringBuilder是jdk5版本加入的，是线程不安全的，效率高

**包装类**

Java里面8个基本数据类型都有相应的类，这些类叫做包装类，包装类都在java.lang包里面。

**包装类的优点**

可以在对象中定义更多的功能方法操作该数据，方便开发者操作数据，例如基本数据类型和字符串之间的转换。

**基本类型与包装类型的异同：**

在Java中，一切皆对象，但八大基本类型却不是对象。

声明方式的不同，基本类型无需通过new关键字来创建，而封装类型需new关键字。

存储方式及位置的不同，基本类型是直接存储变量的值保存在堆栈中能高效的存取，封装类型需要通过引用指向实例，具体的实例保存在堆中。

初始值的不同，封装类型的初始值为null，基本类型的的初始值视具体的类型而定，比如int类型的初始值为0，boolean类型为false；

使用方式的不同，比如与集合类合作使用时只能使用包装类型。

**int,Integer,String三种类型之间的转换**

**int–>Integer**

Integer i1 = Integer.valueOf(10);

**Integer–>int**

int i2 = i1.intValue();

**String–>Integer**

Integer i3 = Integer.valueOf("10");

**Integer–>String**

String s1 = i3.toString();

**String–>int**

int i4 = Integer.parseInt("123");

**int–>String**

String s2= 10 + "";

**自动装箱与拆箱**

**装箱**：将基本类型用它们对应的引用类型包装起来；

**拆箱**：将包装类型转换为基本数据类型；

**数组有没有length()这个方法? String有没有length()这个方法？**

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

**获取系统当前时间**

Date date = new Date();

Date d1 = new Date(0);//如果构造方法中参数传为0代表的是1970年1月1日

**创建日期格式化对象**

SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss SSS");

String str = sdf.format(date)

**y表示年**

**M表示月**

**d表示日**

**H表示小时**

**m表示分钟**

**s表示秒**

**S表示毫秒**

**Math常用方法**

Math.abs()绝对值

Math.pow(a,b)计算a的b次方

Math.random()随机数

Math.floor()向下取整

Math.ceil()向上取整

Math.round()四舍五入

parseInt()字符串转换数字

**BigInteger类可以让超过Integer范围的数据进行运算，通常在对数字计算比较大的行业中应用的多一些**

由于在运算的时候，float类型和double很容易丢失精度，在金融、银行等对数值精度要求非常高的领域里面，就不能使用float或double了，为了能精确的表示、计算浮点数，Java提供了BigDecimal。

注意：**如果对计算的数据要求高精度时，必须使用BigDecimal类**

**DecimalFormat类**

DecimalFormat(",###").format(211321123)->211,321,123

DecimalFormat("00.000").format(3.1415926)->03.142

**泛型的优点**

可以统一集合中的数据类型，提高安全性

可以减少强制类型转换

**什么是IO流？**

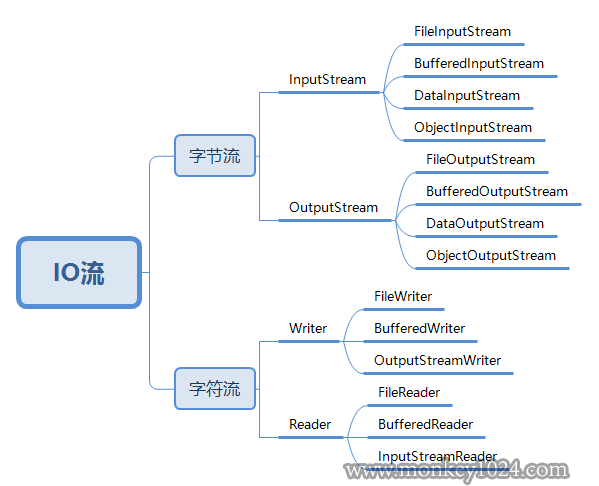
它是一种数据的流从源头流到目的地。比如文件拷贝，输入流和输出流都包括了。输入流从文件中读取数据存储到进程(process)中，输出流从进程中读取数据然后写入到目标文件。

**字节流和字符流的区别。**

字节流在JDK1.0中就被引进了，用于操作包含ASCII字符的文件。JAVA也支持其他的字符如Unicode，为了读取包含Unicode字符的文件，JAVA语言设计者在JDK1.1中引入了字符流。ASCII作为Unicode的子集，对于英语字符的文件，可以使用字节流也可以使用字符流。

**FileInputStream和FileOutputStream是什么？**

这是在拷贝文件操作的时候，经常用到的两个类。在处理小文件的时候，它们性能表现还不错，在大文件的时候，最好使用BufferedInputStream (或 BufferedReader) 和 BufferedOutputStream (或 BufferedWriter)



### IO流的分类

**按照流向分类：**

输入流（InputStream和Reader）：从硬盘或者别的地方读到内存里面

输出流（OutputStream和Writer）：从内存里面向硬盘或者别的地方输出

**按照操作类型分类：**

字节流（InputStream和OutputStream） : 字节流可以操作任何数据,因为在计算机中的数据都是以字节的形式存储的。

字符流（Reader和Writer） : 字符流只能操作纯字符数据，防止乱码。

**InputStream,OutputStream,Reader和Writer都是抽象类**

**使用File.separator解决不同系统的路径问题**

**缓冲区**

缓冲区可以简单地理解为一段内存区域。可以简单地把缓冲区理解为一段特殊的内存。某些情况下，如果一个程序频繁地操作一个资源（如文件或数据库），则性能会很低，此时为了提升性能，就可以将一部分数据暂时读入到内存的一块区域之中，以后直接从此区域中读取数据即可，因为读取内存速度会比较快，这样可以提升程序的性能。在字符流的操作中，所有的字符都是在内存中形成的，在输出前会将所有的内容暂时保存在内存之中，所以使用了缓冲区暂存数据。如果想在不关闭时也可以将字符流的内容全部输出，则可以使用Writer类中的flush()方法完成。

**乱码**

计算机中存储的数据都是二进制的数字，我们看到的文字信息是将二进制转换之后显示的，两者之间存在编码与解码的过程，其互相转换必须遵循某种规则，即编码和解码都遵循同一种规则才能将文字信息正常显示，如果编码跟解码使用了不同的规则，就会出现乱码的情况

**File类常用方法**

boolean isDirectory()判断是否是目录

boolean isFile()判断是否是文件

boolean exists()判断是否存在

boolean isHidden()判断是否隐藏

String getAbsolutePath()获取绝对路径

long length()获取文件大小。字节数

long lastModified()获取最后一次的修改时间，毫秒值

String[] list()获取文件路径下所有文件名（同一层级目录）

File[] listFiles()获取文件路径下所有File对象数组

### 序列化和反序列化的作用

在工作中有可能遇到多台机器远程通信的情况，如果要将机器A中的某个java对象传输到机器B上面，需要将这个java对象转换为字节序列然后进行传输。将对象转换为字节序列的过程叫做序列化，反之叫做反序列化。

如果不希望将某个属性序列化，可以使用transient声明该属性，在序列化时将忽略这个属性。

一个对象如果支持序列化，需要实现Serializable的接口，这个接口中没有任何方法，实现该接口后，JVM会给这个对象做特殊待遇

**什么是设计模式？**  
设计模式是可以重复利用的解决方案。软件开发的先驱或者前辈们将之前在开发中遇到的问题进行总结并给出了解决方案，后辈在遇到这些问题之后直接使用这些方案即可解决问题