目录

[一.变量与运算符 2](#_Toc512364466)

[1.数据类型表 2](#_Toc512364467)

[2.转义字符 3](#_Toc512364468)

[3.数学运算符 3](#_Toc512364469)

[4.比较运算符或关系运算符、逻辑运算符 3](#_Toc512364470)

[5.位操作运算符 3](#_Toc512364471)

[6.运算符优先级 3](#_Toc512364472)

[7.类型转换 4](#_Toc512364473)

[8.代码注释与编程规范 4](#_Toc512364474)

[二. 流程控制 4](#_Toc512364475)

[1.if条件语句 4](#_Toc512364476)

[2. switch多分支语句 4](#_Toc512364477)

[3.循环语句 5](#_Toc512364478)

[三、内置类型与对象（库） 5](#_Toc512364479)

[1.String类 5](#_Toc512364480)

[四、数组 6](#_Toc512364481)

[1.一维数组 6](#_Toc512364482)

[2.二维数组 6](#_Toc512364483)

[3.Arrays类 6](#_Toc512364484)

[五、类和对象，继承与多态性 6](#_Toc512364485)

[1. 类的定义与实例化 6](#_Toc512364486)

[2. 类的继承 7](#_Toc512364487)

[3. Object类 8](#_Toc512364488)

[4.多态性 8](#_Toc512364489)

[5.抽象类与接口 8](#_Toc512364490)

[6.Java类包 9](#_Toc512364491)

[7.final与类和对象 9](#_Toc512364492)

[8.内部类 9](#_Toc512364493)

[六、泛型（模板）与集合（容器） 10](#_Toc512364494)

[1.定义泛型（Generics）类与使用 10](#_Toc512364495)

[2.泛型集合（collection）类 11](#_Toc512364496)

[3.泛型高级用法 12](#_Toc512364497)

[七、反射与注释Annotation 13](#_Toc512364498)

[1.反射概述 13](#_Toc512364499)

[2.访问构造方法 14](#_Toc512364500)

[3.访问成员变量 14](#_Toc512364501)

[4.访问方法 14](#_Toc512364502)

[5.Annotation 15](#_Toc512364503)

[6.说明注释与javadoc 16](#_Toc512364504)

[八、异常处理 17](#_Toc512364505)

[1.捕捉异常 17](#_Toc512364506)

[2.Java常见异常 17](#_Toc512364507)

[3.自定义异常和抛出异常 18](#_Toc512364508)

[4.异常的其他使用原则 18](#_Toc512364509)

[九、网络通信 18](#_Toc512364510)

[1.基本概念 18](#_Toc512364511)

[十、数据库操作 18](#_Toc512364512)

[1. JDBC常用类与接口 18](#_Toc512364513)

[十一、多线程 20](#_Toc512364514)

[1.继承Thread类 20](#_Toc512364515)

[2.实现Runnable接口 20](#_Toc512364516)

[3.线程的生命周期 20](#_Toc512364517)

[4.线程的优先级 21](#_Toc512364518)

[5.线程同步 21](#_Toc512364519)

[十二、I/O输入输出 21](#_Toc512364520)

[1.输入流 21](#_Toc512364521)

[2.输出流 22](#_Toc512364522)

[3. File类 22](#_Toc512364523)

[4.文件输入/输出流 22](#_Toc512364524)

[5.带缓存的输入/输出流 23](#_Toc512364525)

[6.数据输入/输出流 23](#_Toc512364526)

[7.ZIP压缩输入/输出流 23](#_Toc512364527)

[附录1. Java DOS指令 24](#_Toc512364528)

# 一.变量与运算符

声明

类型 变量名1 = 初值1, 变量名2 = 初值2;

final 类型 常量名=值; //定义常量关键字不同，必须定义时赋值

注意：Java中含有两种数据类型，一种是基本数据类型primitives，如int，float等；另一种是包装类类型，如Integer，Float等。前者是非对象型变量，只存储一个数值；后者是对象，不仅存储数据，还具有方法。除了简单算术计算可以使用基本数据类型，其他诸如泛型等功能都必须使用包装类。因此应该常用包装类，除非在特殊的需要考虑运行效率时。

## 1.数据类型表

|  |  |  |
| --- | --- | --- |
| 数据类型 | 内存空间（单位：位） | 取值范围 |
| byte | 8 | -128~127 |
| short | 16 | -32768~32767 |
| int | 32 | -2147483648~2147483647 |
| long | 64 | -9223372036854775~9223372036854775807 |
| float | 32 | 1.4E-45~3.4028235E38 |
| double | 64 | 4.9E-324~1.7976931348623157E308 |
| char | 16 | 0~65536 |
| boolean |  |  |

## 2.转义字符

|  |  |
| --- | --- |
| \ddd | 1~3位八进制数据所表示的字符 |
| \dxxxx | 4位16进制所表示的字符 |
| \’ | 单引号字符 |
| \\ | 反斜杠字符 |
| \t | 垂直制表符 |
| \r | 回车 |
| \n | 换行 |
| \b | 退格 |
| \f | 换页 |

## 3.数学运算符

！Java中没有和其他运算符结合的赋值运算符，如+=，-=等

|  |  |  |
| --- | --- | --- |
| 运算符 | 类别 | 作用 |
| + | 二元 | 求和，对于字符串起到连接作用 |
| - | 二元 | 其他运算符对于字符串无效 |
| \* | 二元 |  |
| / | 二元 |  |
| % | 二元 | 求余数 |
| + | 一元 | 正号 |
| - | 一元 | 取负 |
| ++, -- | 一元 | 自增自减，同C++ |
| = | 二元 | 同C++ |

## 4.比较运算符或关系运算符、逻辑运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 类别 | 作用 |
| ==, != | 二元 |  |
| <, <= | 二元 |  |
| >, >= | 二元 |  |
| ! | 一元 | 逻辑非 |
| &&, || | 二元 | 逻辑与，或 |

## 5.位操作运算符

|  |  |  |
| --- | --- | --- |
| &, |, ^,~ | 二元 | 按位与，或，异或，非 |
| >>, << | 二元 | var1=var2<<var3, 将var2的二进制值，左移var3位，赋给var1 |
| >>> | 二元 | 和右移类似，普通右移若最高位为1进位为1，最高位为0进位为0；无符号右移无论最高位取值，都进位为0 |

## 6.运算符优先级

|  |
| --- |
| () |
| +, - (一元) |
| ++, --(前缀), !, ~ |
| \*, /, % |
| +, -(二元) |
| <<, >>,>>> |
| <, >, <=, >= |
| ==, != |
| & |
| ^ |
| | |
| && |
| || |
| ? X:Y |
| = |
| ++, --（后缀） |

## 7.类型转换

①隐式类型转换规则

|  |  |  |
| --- | --- | --- |
| 操作数1的数据类型 | 操作数2的数据类型 | 转换后的数据类型 |
| 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 |

②显式类型转换

声明

(类型名)被转换的值

## 8.代码注释与编程规范

①代码注释

//单行注释

/\* \*/多行注释

/\*\* \*/文档注释

②编程规范

•被定义为final的常量使用全大写字母命名，并且中间使用下划线进行连接

•Java包的命名使用全小写

•类名的每个单词首字母大写

•函数或方法名首字母小写，其他的单词首字母大写

•普通变量，数据成员，对象的名称首字母小写

# 二. 流程控制

## 1.if条件语句

|  |  |  |
| --- | --- | --- |
| if(布尔表达式){  语句序列  } | if(表达式){  语句  }  else{  语句  } | if(表达式){  语句  }  else if(表达式){  语句  }... |

## 2. switch多分支语句

switch(表达式)

{

case 常量值1:

语句块

break;

case 常量值2:

语句块

break;

default:

语句块

break;

}

## 3.循环语句

|  |  |  |  |
| --- | --- | --- | --- |
| while(表达式)  {  语句  } | do  {  语句  }  while(表达式) | for(初始化;条件;增量)  {  语句  } | for(类型 变量:遍历对象)  {  语句  }  //称为foreach语句 |

# 三、内置类型与对象（库）

## 1.String类

1.1 声明和创建字符串

String str = new String(a)

String str = “…”

•String类的构造函数种类

String(char a[])

String(char a[], int offset, int length)

•Java中String类的连接可以使用“+”和普通字符串任意运算

1.2 String类的成员

公共成员：

length() //返回字符串长度

indexOf(substr) //查找第一次出现substr的索引

lastIndexOf(substr) //查找最后一次出现substr的索引

charAt(int index) //获取指定位置字符

substring(int beginIndex)

substring(int beginIndex, int endIndex) //截取子字符串

trim() //去除前导和尾部空格

replace(char oldChar, char newChar) //替换字符或者字符串

startsWith(String prefix)

endsWith(String suffix) //判断字符串是否以给定内容开始或结束

equals(String otherstr) //判断字符串是否和otherstr内容相同，用等号判定的是字符串地址

equalsIgnoreCase(String otherstr)

compareTo(String otherstr)

//按字典顺序比较字符串，若对象串在参数串之前返回负整数，反之返回正整数

toLowerCase() //大小写转换

toUpperCase()

split(String sign) //sign为分割字符串的分隔符，分割后会被删去，函数返回字符串数组

split(String sign, int limit) //limit限定拆分次数

1.3 格式化字符串和正则表达式

# 四、数组

## 1.一维数组

•两种声明方式

数组元素类型 数组名字[];

数组元素类型[] 数组名字;

•分配内存

数组名字 = new 数组元素类型[数组元素个数];

•两种初始化方式

例：

int arr[] = new int[]{1,2,3,4,};

int arr[] = {34,33,12};

## 2.二维数组

•两种声明方式

数组元素类型 数组名字[][];

数组元素类型[][] 数组名字;

•两种分配内存方式（功能不同）

例：

a = new int[2][4]

a = new int[2][];

a[0] = new int[2];

a[1] = new int[3];

•初始化

例：

int arr[][] = {{1,2,3},{4,5,6}};

## 3.Arrays类

**import** java.util.Arrays

•静态方法：

fill(int[] a, int value) //使用value填充所有a的元素

fill(int[] a, int fromIndex, int toIndex, int value) //在[fromIndex, toIndex)范围内填充

sort(arr) //对任意类型数组进行升序排序

copyOf(arr, int newlength) //复制（或截取数组）arr，返回新构建的数组

copyOfRange(arr, int fromIndex, int toIndex) //功能类似上例

binarySearch(arr, key)

binarySearch(arr, int fromIndex, int toIndex, key) //在范围内搜索

# 五、类和对象，继承与多态性

## 1. 类的定义与实例化

1.1 类的定义与声明

•例：

//默认的权限是private。这是该类在上一级定义域中的权限，如果上一级是package，则采用默认权限时不写出private

public class Book{

private String name;

public void setName(String name){

this.name = name; //this代表本类对象

}

}

public class StaticTest{

//定义静态成员或方法，关于static块或静态块的内容见基础知识额外积累

static double PI = 3.1415;

public static StaticTest method(){…}

public static void main(String[] args){…} //定义主方法（包含主方法的类叫主类）

}

•权限修饰符：（类对于包，和成员对于类的关系是类似的）

1.私有权限（private）

private可以修饰数据成员，构造方法，方法成员，不能修饰类（此处指外部类，即上一级为包，不包括内部类，即上一级为类）。被private修饰的成员，只能在定义它们的类中使用，在其他类中不能调用。

注意：private方法无法被子类直接访问，自然也无法重写，但是子类中可以重新定义一个与父类private方法同名的新方法。因为不是覆盖或重写，所以无法使用多态性。

2.默认权限（default）

类，数据成员，构造方法，方法成员，都能够使用默认权限，即不写任何关键字。默认权限即同包权限，同包权限的元素只能在定义它们的类中，以及同包的类中被调用。

3.受保护权限（protected）

protected可以修饰数据成员，构造方法，方法成员，不能修饰类（此处指外部类，不考虑内部类）。被protected修饰的成员，能在定义它们的类中，同包的类中被调用。如果有不同包的类想调用它们，那么这个类必须是定义它们的类的子类。

4.公共权限（public）

public可以修饰类，数据成员，构造方法，方法成员。被public修饰的成员，可以在任何一个类中被调用，不管同包或不同包，是权限最大的一个修饰符。

有关Java语言的修饰符，需要注意的问题有如下几个：

1.并不是每个修饰符都可以修饰类（指外部类），只有public和default可以。

2.所有修饰符都可以修饰数据成员，方法成员，构造方法。

3.为了代码安全起见，修饰符不要尽量使用权限大的，而是适用即可。比如，数据成员，如果没有特殊需要，尽可能用private。

4.修饰符修饰的是“被访问”的权限。

•对于Java中的函数（方法），Java不支持带默认参数的函数，也不支持运算符函数重载。

1.2 类的实例化与对象销毁

•例：

Book book = new Book();

//实例化类，使用构造函数构造新对象必须使用new。但仍然可以使用其他同类或同族对象来为被定义对象赋值，实际上对象的名称就是其指针。

book = null;

//将对象设为null，即对其进行销毁，必须是由new创建的对象

•对象的销毁一般由垃圾回收器自动执行，对于new创建的对象，可以手动赋值null以便交由gc自动销毁，对于非new创建的对象，可以重载其Object类继承而来的finalize()方法手动销毁。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的（该方法为protected）。另外可以用System.gc()方法强制启动垃圾回收器。

## 2. 类的继承

例：

class Test{

public Test(){…} //构造函数不声明返回类型（也没有返回类型）

protected void doSomething(){…}

protected Test doIt(){…}

}

class Test2 extends Test{ //继承的格式

public Test2(){

super(); //使用super关键字调用父类构造函数与方法

super.doSomething();

//Java中子类的构造函数必须像这样在函数体中调用父类构造函数，不同于C++，因此

使用父类继承的构造函数的Java子类，其构造函数也无法为空

}

public void doSomething(){…} //重写父类方法，可以更改返回类型和权限

protected Test2 doIt(){…} //更改的返回类型必须是原来的类型的子类

}

## 3. Object类

•Java中的所有类都直接或间接继承自java.lang.Object类，包括用户自定义的类；Object类的方法包括clone(), finalize(), equals(), toString()等可重写的方法，和getClass(), notify(), notifyAll(), wait()等不可重写的final方法

①getClass()

用来获取对象的Class实例

例：getClass().getName() 获得类名称

②toString()

用来将对象转化为字符串，对于自定义类一般重写，对于基本类型即可直接使用

③equals()

用来判断两个对象是否内容相同，返回boolean类型；当然这种功能在自定义类中需重载更改实现。

## 4.多态性

4.1 instanceof

语法：Object instanceof Class

•这是一个条件表达式，返回boolean值，检验对象是否为类的实例，如果是即可使用强制类型转换

## 5.抽象类与接口

5.1 抽象类

•Java中，类不能继承多个基类，但是接口可以实现多个。

•只要有一个抽象方法，该类必须定义为抽象类，但是抽象类里可以有非抽象方法

•抽象方法没有函数体，具体相关知识和C++/C#中的知识类似

例：

public abstract class Test{

abstract void testAbstract();}

5.2 接口

例：接口定义

public interface DrawTest{

void draw(); //接口中所有方法均为公用抽象方法，但是省略abstract和public关键字，

没有函数体且省略{}

数据成员 //接口中定义的所有字段均隐式声明为public, static和final的，可省

} //接口中的方法不允许使用其他的权限修饰

例：接口和抽象类继承

public class Parallelogram extends Quadrangle implements DrawTest, DrawTest2

{}

•抽象类与接口都无法实例化，但是可以用于多态性，即接收派生类对象地址

例：接口继承接口

interface intf1{}

interface intf2 extends intf1{}

例：抽象类继承具体类并“实现”接口

interface intf1{}

class class1{}

abstract class abclass1 extends class1 implements intf1{}

//显然根据定义abclass1无需真正实现intf1接口

## 6.Java类包

•Java中每定义好一个类，编译之后，便成为一个.class文件

•习惯上将包与类和文件夹与文件位置一一对应起来，即某包中的类其.class文件位于以该包命名的文件夹中。但是这种对应关系不是必须的，如果同一个包中的两个类分属不同位置，只需将CLASSPATH环境变量分别指向这两个位置即可

•Java包的命名规则为全小写

例：创建包

package 包名 //必须是文件中第一个非注释行，表示文件中的类属于该包

例：导入包

import 包名 //可以用“\*”代表该包下的所有类，但是不包括子包中的类

import static java.lang.Math.max; //直接导入静态方法，即可直接使用，更加针对性的导入

## 7.final与类和对象

•final修饰的数据成员，字段都是不可更改的，这些数据如果不在声明时赋值，可以在构造函数中赋值

•final修饰的对象，其成员可以更改，但是final修饰的对象（实际上是一个引用）不能指向其他对象。若想限制不可更改成员的值，可以用static final加以限定

•final修饰的方法参数在方法中不可更改，因为一般对象采用引用传递参数，即不可以更改该对象的成员

•final修饰的方法，不能被子类重写，同时final方法比非final方法执行效率更高

•final修饰的类不能被继承，也不许进行任何改动；final类中的方法均隐式设置为final，数据可以被定义为final或非final形式

## 8.内部类

8.1成员内部类

•成员内部类可以随意访问其外部类成员，但外部类无法直接访问非静态内部类成员，需先实例化后通过内部类对象访问其成员。需要注意的是外部类可以访问内部类的私有成员。

•内部类可以单独实例化为对象（如下例），但是实例化过程必须考虑到其有外部类的特点，即内部类并不独立，只能在外部类或外部类的非静态成员方法中实现；而实例化的内部类对象显然也依赖于其外部类对象（如下例）

例：主方法中

OuterClass out = new OuterClass();

OuterClass.innerClass in = out.new innerClass(); //new运算符前加外部类对象（不是类名）

•成员内部类最基本的用途是：在一个外部类中定义多个private内部类，分别以不同的方式实现同一个接口。根据多态性，可以用这么一个接口关联任何一种实现方式，而不必创建多个外部类或者暴露实现的具体过程。

例：this关键字和内部类

public class TheSameName{

private int x;

private class Inner{

private int x = 9;

public void doit(int x){

x++; //形参x

this.x++; //调用内部类的变量x

TheSameName.this.x++;} //调用外部类的变量x

}

•成员内部类的另一个用途是定义为静态内部类，具体用法参考《Java基础知识额外积累》

8.2 局部内部类

•局部内部类即在类的成员方法中定义的类，这些局部内部类只能使用final类型的函数参数和外部类的所有成员。

8.3 匿名内部类

例：

interface OutInterface{}

class OuterClass{

public OutInterface doit(){

return new OutInterface(){

private int i = 0;

public int getValue(){

return i;}};}

•方法体中就地创建了实现OutInterface接口的匿名类

•匿名内部类编译后也会产生.class文件，名称为“外部类名$序号”，多个匿名类以序号区分

8.4 静态内部类

•少见

8.5 内部类的继承

例：

public class OutputInnerClass extends ClassA.ClassB{

public OutputInnerClass(ClassA a){

a.super(); //单独继承内部类需要用到其外部类的对象

…

}

# 六、泛型（模板）与集合（容器）

## 1.定义泛型（Generics）与使用

泛型由符号<T>标识，T可以用任意字母替代，但有一些约定如下，多个泛型符号由逗号隔开，如<T,U>：  
E — Element，常用在java Collection里，如：List<E>,Iterator<E>,Set<E>  
K,V — Key，Value，代表Map的键值对  
N — Number，数字  
T — Type，类型，如String，Integer等等

泛型可以以泛型类或泛型接口存在，也可以以泛型方法单独存在。

例：泛型类

public class ArrayClass<T>{ //无需像C++一样特别声明泛型类，直接声明在类的定义行

private T[] array;

public T[] get(){…}

}

例：泛型方法，每个<T>只限定于该方法的域，而非全类

public class StaticFans {

//静态函数

public static  <T> void StaticMethod(T a){}

//普通函数

public  <T> void OtherMethod(T a){}

}

例：

List<List<Integer>> list = new ArrayList<List<Integer>>();

//声明使用泛型类（或容器）时，经常用基类作为对象类型。如List即为基类，ArrayList是其实现类之一。如此可以利用多态性。

//另外声明容器的容器（如二维向量等）时，ArrayList实现类的内部维度必须仍然使用基类List声明。实质上，这种声明只是对外部维度的实现方式进行声明，内部维度的实现方式显然和内部元素（是List<Integer>元素而非底层Integer元素）的具体实现类型有关。

## 2.泛型集合（collection）类

**2.1 Collection接口**

|  |  |
| --- | --- |
| 方法 | 功能描述 |
| add(E e) | 添加对象e到该集合中 |
| remove(Object o) | 移除对象o |
| isEmpty() | 返回boolean值，判断当前集合是否为空 |
| iterator() | 返回在此集合的元素上使用的迭代器对象，用于遍历 |
| size() | 返回int型值，获取集合中元素个数 |

**2.2 List集合**

除了继承Collection的，还有两个方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| get(int index) | 获取指定索引位置的元素 |
| set(int index, Object obj) | 将指定索引位置的元素更改为obj |

例：

List<String> list = new ArrayList<>(); //利用接口多态性实例化，可省略后者的泛型实例化

•List用于存放多个元素，能够维护元素的次序，并且允许元素的重复。3个具体实现类的相关区别如下：  
①ArrayList是最常用的List实现类，内部是通过数组实现的，它允许对元素进行快速随机访问。数组的缺点是每个元素之间不能有间隔，当数组大小不满足时需要增加存储能力，就要讲已经有数组的数据复制到新的存储空间中。当从ArrayList的中间位置插入或者删除元素时，需要对数组进行复制、移动、代价比较高。因此，它适合随机查找和遍历，不适合插入和删除。  
②Vector与ArrayList一样，也是通过数组实现的，不同的是它支持线程的同步，即某一时刻只有一个线程能够写Vector，避免多线程同时写而引起的不一致性，但实现同步需要很高的花费，因此，访问它比访问ArrayList慢。  
③LinkedList是用链表结构存储数据的，很适合数据的动态插入和删除，随机访问和遍历速度比较慢。另外，他还提供了List接口中没有定义的方法，专门用于操作表头和表尾元素，可以当作堆栈、队列和双向队列使用。

•查看Java源代码，发现当数组的大小不够的时候，需要重新建立数组，然后将元素拷贝到新的数组内，ArrayList和Vector的扩展数组的大小不同。

•关于ArrayList和Vector区别如下：  
①ArrayList在内存不够时默认是扩展50% + 1个，Vector是默认扩展1倍。  
②Vector提供indexOf(obj, start)接口，ArrayList没有。  
③Vector属于线程安全级别的，但是大多数情况下不使用Vector，因为线程安全需要更大的系统开销。

2.2.1 Stack类

Stack除了Vector定义的所有方法，自己也定义了一些方法

|  |  |
| --- | --- |
| 方法 | 功能描述 |
| boolean empty() | 栈是否为空 |
| Object peek() | 查看栈顶部的对象 |
| Object pop() | 弹出栈顶 |
| Object push(Object element) | 压入栈顶 |
| int search(Object element) | 搜索对象，返回位置 |

**2.3 Set集合**

•Set集合中对象不能有重复值

|  |  |
| --- | --- |
| 方法 | 功能描述 |
| first() | 返回此Set中当前第一个（最低）元素 |
| last() | 返回此Set中当前最后一个（最高）元素 |
| comparator() | 返回对元素排序的比较器，如果使用自然顺序，返回null |
| headSet(E toElement) | 返回一个新的Set集合，新集合包含toElement（不包含）之前的所有对象 |
| subSet(E fromElement, E fromElement) | 返回一个新的Set集合，包含fromElement（包含）对象与fromElement（不包含）对象之间的所有对象 |
| tailSet(E fromElement) | 返回一个新的Set集合，新集合包含fromElement之后的所有对象 |

**2.4 Map集合**

•不继承Collection，不能包含重复的key，每个key只能映射一个值

|  |  |
| --- | --- |
| 方法 | 功能描述 |
| put(K key, V value) | 向集合中添加指定的key与value的映射关系 |
| containsKey(Object key) | 如果此映射包含指定key的映射关系，则返回true |
| containsValue(Object value) | 如果此映射将一个或多个key映射到指定值，则返回true |
| get(Object key) | 如果存在指定的key对象，则返回该对象对应的值，否则返回null |
| keySet() | 返回该集合中的所有key形成的Set集合 |
| values() | 返回该集合中所有值对象形成的Collection集合 |

## 3.泛型高级用法

# 七、反射与注释Annotation

## 1.反射概述

•反射功能用来查看对象和类的信息，支持该功能的包围java.lang.reflect

•使用反射功能需要获得类型为Class的对象，而从一个对象获得其类型的Class对象，可用Object类中的方法：

Class textFieldC = textField.getClass();

•通过反射可访问的主要描述信息，属于Class类的方法

|  |  |  |  |
| --- | --- | --- | --- |
| 组成部分 | 访问方法 | 返回值类型 | 说明 |
| 包路径 | getPackage() | Package | 获得该类的存放路径 |
| 类名称 | getName() | String | 获得该类名称 |
| 继承类 | getSuperclass() | Class | 获得该类的基类 |
| 实现接口 | getInterface() | Class [] | 获得该类实现的接口 |
| 构造方法 | getConstructors() | Constructor [] | 获得所有权限为public的构造方法 |
| getConstructor  (Class<?> … parameterTypes) | Constructor | 获得权限为public的由参数类型指定的构造方法 |
| getDeclaredConstructors() | Constructor [] | 获得所有构造方法，按声明次序返回 |
| getDeclaredConstructor  (Class<?> … parameterTypes) | Constructor | 获得由参数类型指定的构造方法 |
| 方法 | getMethods() | Method [] | 获得所有权限为public的方法 |
| getMethods(String name,  Class<?> … parameterTypes) | Method | 获得权限为public由名称和参数类型指定的方法 |
| getDeclaredMethods() | Method [] | 获得所有方法，按声明次序返回 |
| getDeclaredMethod(String name,  Class<?> … parameterTypes) | Method | 获得由名称和参数类型指定的方法 |
| 成员变量 | getFields() | Field [] | 获得所有权限为public的成员变量 |
| getField(String name) | Field | 获得权限为public的由名称指定的成员变量 |
| getDeclaredFields() | Field [] | 获得所有成员变量，按声明次序返回 |
| getDeclaredField(String name) | Field | 获得由名称指定的成员变量 |
| 内部类 | getClasses() | Class [] | 获得所有权限为public的内部类 |
| getDeclaredClasses() | Class [] | 获得所有内部类 |
| 内部类的  声明类 | getDeclaringClass() | Class | 若该类为内部类，返回其成员类，否则返回null |

•注1：获得非declared内容的方法将包含从基类继承过来的方法，而declared方法将只返回本类中定义的变量与方法

•注2：获得指定的方法的具体操作如下，如获取参数类型依次为String和int的构造方法

objectClass.getDeclaredConstructor(String.class, int.class);

objectClass.getDeclaredConstructor(new Class[]{String.class, int.class});

## 2.访问构造方法

•Constructor类中提供的常用方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| isVarArgs() | 查看构造方法是否允许带有可变数量的参数，返回boolean |
| getParameterTypes() | 按照声明顺序以Class[]形式获得构造方法的各参数类型 |
| getExceptionTypes() | 以Class[]形式获得构造方法可能抛出的异常类型 |
| newInstance(Object … initargs) | 通过该构造方法利用指定参数创建一个对象，可默认 |
| setAccessible(boolean flag) | 如果构造方法权限为私有，默认是不许用newInstance方法创建对象。如果先执行此方法，将入口参数设为true，则可以创建 |
| getModifiers() | 获得可解析出该构造方法所采用修饰符的整数 |

•针对getModifiers()方法返回的整数，可以用java.lang.reflect.Modifier类中的静态方法解析

|  |  |
| --- | --- |
| 静态方法 | 说明 |
| isPublic(int mod) | 查看是否被public修饰符修饰，返回boolean |
| isProtected(int mod) | 类似上一个 |
| isPrivate(int mod) | 类似上一个 |
| isStatic(int mod) | 类似上一个 |
| isFinal(int mod) | 类似上一个 |
| toString(int mod) | 以字符串形式返回mod对应的修饰符 |

## 3.访问成员变量

•Field类中提供的常用方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| getName() | 获得该成员变量的名称 |
| getType() | 获得表示该成员变量类型的Class对象 |
| get(Object obj) | 获得指定对象obj中成员变量的值，返回Object类型 |
| set(Object obj, Object value) | 将指定对象obj中成员变量的值设置为value |
| getInt(Object obj) | 获得指定对象obj中类型为int的成员变量的值 |
| setInt(Object obj, int i) | 将指定对象obj中类型为int的成员变量的值设置为i |
| getFloat(Object obj) | 获得指定对象obj中类型为float的成员变量的值 |
| setFloat(Object obj, float f) | 将指定对象obj中类型为float的成员变量的值设置为f |
| getBoolean(Object obj) | 获得指定对象obj中类型为boolean的成员变量的值 |
| setBoolean(Object obj, boolean z) | 将指定对象obj中类型为boolean的成员变量值设置为z |
| setAccessible(boolean flag) | 此方法可以设置是否忽略权限限制，直接访问private等私有成员变量 |
| getModifiers() | 获得可以解析出该成员变量所采用修饰符的整数 |

## 4.访问方法

•Method类的常用方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| getName() | 获得方法名称 |
| getParameterTypes() | 按照声明顺序以Class[]形式获得方法的各参数类型 |
| getReturnType() | 以Class对象形式获得方法的返回值类型 |
| getExceptionTypes() | 以Class[]形式获得方法可能抛出的异常类型 |
| invoke(Object obj, Object … args) | 利用指定参数args执行指定对象obj中的该方法，返回Object型 |
| isVarArgs() | 查看该方法是否允许带有不定长参数 |
| getModifiers() | 获得可以解析出修饰符的整数 |

## 5.Annotation

•定义Annotation的方法：

例1：

public @interface 注释类名称{ //继承java.lang.annotation.Annotation接口

String value();}

//不包含任何成员的注释类叫做marker annotation，如果只含一个成员，通常其名称为value

//可用的成员类型有String, Class, primitive, enumerated, annotation

例2：

public @interface 名称{

String describe() default “<…>”;

Class type() default void.class;} //可以为成员设置默认值

•定义注释类型之前，可以使用@Target和@Retention分别设置该注释类适用的程序元素种类和其有效范围。

对于@Target()使用枚举类ElementType中的值设置：

|  |  |
| --- | --- |
| ANNOTATION\_TYPE | 用于注释类型 |
| TYPE | 用于类，接口和枚举以及注释类型 |
| CONSTRUCTOR | 用于构造方法 |
| FIELD | 用于成员变量和枚举常量 |
| METHOD | 用于方法 |
| PARAMETER | 用于参数 |
| LOCAL\_VARIABLE | 用于局部变量 |
| PACKAGE | 用于包 |

例：使用多个值时

@Target({ElementType.FIELD, ElementType.METHOD})

对于@Retention()使用枚举类RetentionPolicy中的值设置：

|  |  |
| --- | --- |
| SOURCE | 不编译注释到类文件中 |
| CLASS | 编译注释到类文件中，但是在运行时不加载到JVM中，不设置@Retention时默认采用 |
| RUNTIME | 运行时加载到JVM中 |

•使用注释的方法很简单，想要注释什么程序元素（@Target）就将注释对象添加在该元素前即可。然后为注释提供构造参数即可。不加构造参数的注释对象可以省略后面的()，而相对的定义注释时，其数据成员加()，原因待发掘。

例：注释函数的参数，也可以省略注释的成员变量名

public void setId(@Parameter\_Annotation(describe = “编号”, type = int.class) int id){…}

•添加过注释的Java类可以通过反射在其他地方访问，有以下几个常用方法：

Constructor，Field和Method类中的函数

①isAnnotationPresent(Parameter\_Annotation.class)

//查看是否含有某种注释类型

②getAnnotation(Parameter\_Annoation.class)

//获得对应类型的注释对象

③getAnnotations() //获得所有注释对象的数组

Constructor和Method类中的函数

①getParameterAnnotations() //返回方法所有参数的注释对象的二维数组（每个参数的注释使用一个数组），按照声明顺序。

•通过以上方法取得注释对象后，可以调用其成员获得取值，和一般的对象一样。但是注意注释对象的成员名称后带括号。

## 6.说明/文档注释与javadoc

Java 支持三种注释方式。前两种分别是 // 和 /\* \*/，第三种被称作说明注释，它以 /\*\* 开始，以 \*/结束。说明注释允许你在程序中嵌入关于程序的信息。你可以使用 javadoc 工具软件来生成信息，并输出到同名的html文件中。生成的html文件可按照不同class进行分割，用以便捷产生程序document（具体如何操作依据javadoc版本和Java版本）。说明注释，使你更加方便的记录你的程序信息。同时一些高级的IDE可以通过说明注释来帮助检查代码内容。

在开始的 /\*\* 之后，第一行或几行是关于类、变量和方法的主要描述。之后，你可以包含一个或多个各种各样的 @ 标签。每一个 @ 标签必须在一个新行的开始或者在一行的开始紧跟星号(\*)。多个相同类型的标签应该放成一组。例如，如果你有三个 @see 标签，可以将它们一个接一个的放在一起。所有的说明注释需要刚好放在被注释内容的前面。

下面是一个类的说明注释的实例：

/\*\*\* 这个类绘制一个条形图

\* @author runoob

\* @version 1.2

\*/

public class …..

常用的@标签如下：

|  |  |  |
| --- | --- | --- |
| **标签** | **描述** | **示例** |
| @author | 标识一个类的作者 | @author description |
| @deprecated | 指名一个过期的类或成员 | @deprecated description |
| {@docRoot} | 指明当前文档根目录的路径 | Directory Path |
| @exception | 标志一个类抛出的异常 | @exception exception-name explanation |
| {@inheritDoc} | 从直接父类继承的注释 | Inherits a comment from the immediate surperclass. |
| {@link} | 插入一个到另一个主题的链接 | {@link name text} |
| {@linkplain} | 插入一个到另一个主题的链接，但是该链接显示纯文本字体 | Inserts an in-line link to another topic. |
| @param | 说明一个方法的参数 | @param parameter-name explanation |
| @return | 说明返回值类型 | @return explanation |
| @see | 指定一个到另一个主题的链接 | @see anchor |
| @serial | 说明一个序列化属性 | @serial description |
| @serialData | 说明通过writeObject( ) 和 writeExternal( )方法写的数据 | @serialData description |
| @serialField | 说明一个ObjectStreamField组件 | @serialField name type description |
| @since | 标记当引入一个特定的变化时 | @since release |
| @throws | 和 @exception标签一样. | The @throws tag has the same meaning as the @exception tag. |
| {@value} | 显示常量的值，该常量必须是static属性。 | Displays the value of a constant, which must be a static field. |
| @version | 指定类的版本 | @version info |

# 八、异常处理

## 1.捕捉异常

•异常状况也是一种对象，出问题时，由所在方法创建然后抛出

•所有异常类是Throwable类的子孙类，其派生了Exception类与Error类。后者错误比较严重，程序一般无法运行，前者通过适当异常处理可以保持程序运行。RuntimeException是Exception的一种

例：

try{代码}

catch(Exceptiontype1 e){代码}

catch(Exceptiontype2 e){代码}

catch(Exception e){代码} //可选择添加，但必须用最后一个catch块处理其他一般异常类型

finally{代码}

//无论try-catch结构运行结果为何，finally块一定会执行。前提是必须正常或异常执行try块（和catch块）之后才会执行finally块，即便try或catch块中存在控制转移语句（return、throw、break、continue等），在离开该块前也须执行finally块。只有在try块之前控制转移（即没有执行到try块）或通过指令或极端条件终止程序，才不会执行finally块。

•抛出异常后，根据类型选择对应的catch块执行，之后便继续后面的代码，try块中异常之后的代码不被执行

•异常处理，即catch块中常用函数如下：（均为Exception类非静态成员函数）

①getMessage()：输出错误性质

②toString()：给出异常的类型和性质

③printStackTrace()：指出异常的类型，性质，栈层及在程序中的位置

## 2.Java常见异常

常见错误有Java自带的一些异常类型，通过查看这些异常的信息和类型即可得知出错原因：

|  |  |
| --- | --- |
| 异常 | 说明 |
| ClassCastException | 类型转换异常 |
| ClassNotFoundException | 未找到相应类异常 |
| ArithmeticException | 算数异常（RE） |
| ArrayIndexOutOfBoundsException | 数组下标越界（RE） |
| ArrayStoreException | 数组中包含不兼容的值（RE） |
| SQLException | 操作数据库异常 |
| SecurityException | 安全性异常（RuntimeException） |
| NullPointerException | 空指针异常（RE） |
| NoSuchFieldException | 字段未找到异常 |
| NoSuchMethodException | 方法未找到 |
| NumberFormatException | 字符串转换为数字出异常 |
| NegativeArraySizeException | 数组元素个数为负（RE） |
| StringIndexOutOfBoundsException | 字符串索引超出范围 |
| IOException | 输入输出异常 |
| IllegalAccessException | 不允许访问某类异常 |
| IllegalArgumentException | 非法参数（RE） |
| InstantiationException | 使用newInstance()方法创建无法实例化的类时 |
| EOFEException | 文件已结束 |
| FileNotFoundException | 文件未找到 |

## 3.自定义异常和抛出异常

•throws是不在本方法中处理异常时向上抛出异常所使用的，可以多级向上抛出，只需在各个函数定义时声明throws即可，最后处理异常的方法要用到try-catch代码块

例：定义抛出异常的方法，throws指定可能抛出的异常类型，可以是自定义或常用异常类

int method1(int number1) throws Exceptiontype1, Exceptiontype2 {}

•throw则是用来抛出用户自定义异常类型对象的，常用异常不需显式抛出

例：

int method1(int number1) throws Exceptiontype1{

if(…){

throw new Exceptiontype1(...);}…}

//throw后面接创建异常对象的语句即可

•定义自定义异常类

public class Exceptiontype1 extends Exception{}

//所有的异常类型都继承Exception类

## 4.异常的其他使用原则

①子类覆盖父类的方法时，若父类方法有抛出异常，则子类必须抛出相同的异常或该异常的子类；如果父类抛出多个异常，那么子类必须抛出这些异常的一个子集，即子类不能抛出比父类多的异常类型

# 九、网络通信

## 1.基本概念

•IP协议，TCP和UDP协议

•局域网（Local Area Network, LAN）和广域网（Wide Area Network, WAN）：无数的LAN和WAN组成Internet。

•IP地址：IPv4，用4个字节共32位代表一个IP地址，常用每个字节的十进制数以点号分隔的方式表达。IPv6使用16个字节表示IP地址

•端口（port）：真实的端口即计算机连接到网络的物理连接口，而网络程序设计中的端口则是假想的连接装置，并被规定为一个在0~65535之间的整数。不同的服务使用不同的端口。0~1023之间的端口数一般用于知名的网络服务和应用，普通的网络服务使用端口数更大的端口。

•套接字（socket）：用于将应用程序和端口连接起来，也是一个假想的连接装置，Java中体现为一个类。

# 十、数据库操作

本节只是作为一个大致了解JDBC内容的概览

## 1. JDBC常用类与接口

•数据库存在于网络中，而且有多种厂商和类型，使用Java通用的JDBC API显然无法直接调用所有的数据库。实际上是用JDBC和数据库驱动程序取得连接，后者将JDBC指令转化为与自己兼容的形式。

**1.1 Connection接口**

•是用来表示和数据库连接状态的对象的类

|  |  |
| --- | --- |
| createStatement() | 创建Statement对象 |
| createStatement(int resultSetType,  int resultSetConcurrency) | 类似上例，该Statement可生产具有给定类型，并发性和可保存性的ResultSet对象 |
| preparedStatement(String sql) | 创建预处理对象preparedStatement |
| isReadOnly() | 查看当前Connection的读取模式是否为只读 |
| setReadOnly() | 设置当前Connection为只读，默认是非只读 |
| commit() | 使所有上一次提交/回滚后进行的更改持久化，并释放Connection对象当前持有的所有数据库锁 |
| roolback() | 取消当前事务中进行的所有更改，并释放此Connection当前持有的所有数据库锁 |
| close() | 立即释放此Connection所有资源 |

•创建Connection不是用其构造函数，而是使用数据库驱动程序相关的方法

**1.2 Statement接口**

•Statement派生PreparedStatement，后者再派生CallableStatement。这些声明对象是由Connection对象创建的，用来执行SQL语句

•Statement用于执行不带参数的简单SQL语句，PreparedStatement用于执行动态SQL语句，后者因为预处理了SQL指令使得执行效率更高。

|  |  |
| --- | --- |
| execute(String sql) | 执行静态的SELECT语句，可能返回多个结果集 |
| executeQuery(String sql) | 执行给定SQL，返回单个ResultSet对象 |
| clearBatch() | 清空当前SQL命令列表 |
| executeBatch() | 将一批命令提交给数据库执行，若全部成功，返回更新计数组成的数组，顺序与SQL语句对应 |
| addBatch(String sql) | 将给定的SQL命令添加到当前对象的命令列表，前提是驱动程序支持批量处理 |
| close() | 释放此Statement占用的资源 |

**1.3 PreparedStatement接口**

•PreparedStatement和某一句或一种SQL语句相对应，但是不指定具体参数，如果需要多次执行某一种语句，可以使用这个对象，并通过成员方法修改参数即可。效率高过直接使用Statement。

|  |  |
| --- | --- |
| executeQuery() | 执行本对象中的SQL查询，返回ResultSet |
| executeUpdate() | 执行前面包含的参数的动态SQL增删改语句，又称为DML（Data Manipulation Language）语句，不是查询语句。也可以是DDL语句 |
| clearParameters() | 清楚当前所有参数的值 |
| setInt(int index, int k) | 将指定位置的参数设为int值，index表示第几个通配符?位置，而此函数还有支持其他类型的类似的形式：Float, Long, Double, Boolean, Date, String, Null |

**1.4 DriverManager类**

•用来管理和与数据库驱动取得连接的类

|  |  |
| --- | --- |
| getConnection(String url, String user, String  password) | 静态成员，获取与数据库连接对象 |
| setLoginTimeout(int second) | 设定登录到某一数据库时最大等待时间（秒） |
| printIn(String message) | 打印一条消息到当前JDBC日志流 |

# 十一、多线程

进程：一个包含有自身地址的程序，每个独立执行程序都称为进程。操作系统分配给每个进程一段有限的使用CPU的时间（CPU时间片），CPU在不同的进程中来回切换执行。进程也可以称作任务（操作系统范畴内），实际上就是一个程序。

线程：进程中的执行流程。一个进程中可以同时包括多个线程，每个线程也可以得到一小段程序执行时间。进程相对于CPU，就是线程相当于进程。

## 1.继承Thread类

Thread类构造函数：  
public Thread(String threadName)  
public Thread()

实现多线程步骤：  
继承Thread类之后，重写其void run()函数以包含所需功能，最后通过其成员start()方法在主函数中启动线程（启动线程不是用run()）。

## 2.实现Runnable接口

这个方式适合无法继承Thread类的情况（已经继承了其他类），Thread类本质上实现了Runnable接口。后一种方法不考虑性能因素可以代替前一种。

Thread类其他构造函数：  
public Thread(Runnable r)  
public Thread(Runnable r, String name)

实现步骤：  
为需要实现多线程的类implements Runnable接口（或使用匿名类实现Runnable接口，至少定义run()即可）。建立对象后即可通过调用上面两个构造函数创建Thread类实例（这个过程可以理解为将Runnable接口对象和新建立的Thread对象关联），然后调用start()启动线程。

## 3.线程的生命周期

线程生命周期有7种状态：  
①出生状态——线程被创建时处于的状态，在start()被调用之前  
②就绪状态/可执行状态——调用start()方法后，未得到CPU资源时  
③运行状态——获得CPU资源时，且没有其他原因阻碍其运行时  
④等待状态——调用wait()方法后，必须用Thread类notify()或notifyAll()方法唤醒当前或全部处于等待状态的线程  
⑤休眠状态——调用sleep(int sleeptime)成员方法后，等待一定时间后恢复就绪  
⑥阻塞状态——等待其他系统的输入/输出响应时的状态  
⑦死亡状态——run()方法执行完毕后

**3.1 线程的休眠**

sleeptime是以毫秒为单位的休眠时间，时间一到该线程会进入就绪状态。在此之前，不会做任何响应。sleep()是Thread类静态成员函数，由Thread.sleep(…)调用，通常使用在run()方法体中。该方法执行时可能抛出InterruptedException异常（具体原因见“线程的中断”），因此要置于try/catch中。

**3.2 线程的加入**

使用Thread类的成员方法join()来在某个线程中加入（调用）另一个线程，一般使用在被插入线程的run()方法体中，当加入线程执行完毕后，才继续执行原被插入线程。

**3.3 线程的中断**

合理的使用线程和终止线程的方式是在run()方法体中定义一个无限循环while(true){}，然后使用布尔型标记控制循环的停止if (isContinue) break，进而结束进程。将isContinue设为私有成员，通过公共方法来控制其值，进而实现可在外部中断线程的功能。

另一种终止线程的方法（适用于睡眠或等待状态下的线程）：仍然使用无限循环，但使用Thread类成员方法interrupt()来打断线程并抛出InterruptedException异常，在异常处理catch块中来break循环。同时catch块中也可以关闭数据库连接和关闭Socket连接等。很明显，interrupt()方法要在线程本体外调用，类似start()

**3.4 线程的礼让**

对于不支持多任务的操作系统来说，必须手动控制线程的CPU资源分配。可以使用Thread类中的yield()方法来实现。该方法只是“提醒”当前运行线程，并不保证其必然让出资源。

## 4.线程的优先级

当多个线程同处于就绪状态时，系统会根据优先级来决定首先运行哪个线程。优先级较低的线程并非永远不会被运行，只是几率较小。

Thread类中的数据成员MIN\_PRIORITY(常数1)、MAX\_PRIORITY(常数10)、NORM\_PRIORITY(常数5)定义了优先级的范围和默认值。每个新产生的线程都继承父线程的优先级。

使用Thread类成员方法setPriority(int priority)来为线程设置优先级。一般在start()启动线程之前就设置好线程的优先级和名称（可以使用构造函数或成员方法setName(String threadName)来完成）

## 5.线程同步

线程同步、线程安全都是解决多线程之间对共享资源的并发处理问题。原理上和所有并发问题类似，使用同步锁机制。Java中有两种方式可以实现同步锁。

注：共享资源一般定义在实现Runnable接口的类中，由该类实例建立（关联）的Thread对象都使用（共享）其数据成员。

**1. 同步块**

将对共享资源的操作放在同步块中，同一时间只有一个线程能够运行同步块中的内容。将同步块置于run()方法中适当的区域即可。同步块的使用方法和catch(Exception e)块类似。

同步块的建立：（Object为任意对象，可以使用空字符串“”）  
synchronized(Object){块内容}

**2. 同步方法**

原理上类似，同步方法在同一时间只能被一个线程对象调用，因此只在同步方法中对共享资源操作即可实现线程安全

同步方法的定义：  
public synchronized void function(){}

# 十二、I/O输入输出

负责输入输出的类都在java.io包中

## 1.输入流

InputStream是字节输入流抽象类，诸多以字节为单位输入流的基类，这些派生输入流类名称都以\*InputStream结尾，其常用方法如下：（所有的输入流类都有的方法，个别除外）  
read()  
read(byte[] b)  
mark(int readlimit)  
reset()  
skip(long n)  
markSupported()  
close()

Reader是字符输入流抽象类，诸多以字符为单位输入流的基类，这些派生输入流类名称都以\*Reader结尾，常用方法和InputStream类中的方法类似

## 2.输出流

OutputStream是字节输出流基类，常用方法如下：  
write(int b) 将指定字节写入输出流  
write(byte[] b)  
write(byte[] b, int off, int len)  
flush() 彻底完成输出并清空缓存区  
close() 关闭输出流

Writer是字符输出流基类，可类推

## 3. File类

File类是代表磁盘文件的对象，但是它并不是具体的文件，比如某个File对象存在，但其代表的具体文件可能不存在（没有创建，或刚被删除等）。另外，File类对象可以是某种文件，也可以是文件夹。

**1.文件对象的创建与删除**

File类的构造函数：  
File(String pathname)——创建一个封装该路径或文件的File对象。路径可以是绝对路径（带有盘符的完整路径），也可以是相对的，如pathname只是“/”表示当前路径下  
File(String parent, String child)——创建一个封装前后两个路径组合在一起的路径的File  
File(File f, String child)——f即文件夹创建的文件对象，和上一种构造方式类似

**2. 对文件对象的操作**

|  |  |  |
| --- | --- | --- |
| 方法 | 返回值 | 说明 |
| canRead() | boolean | 判断文件是否是可读的 |
| canWrite() | boolean | 判断文件是否可被写入 |
| exists() | boolean | 判断文件是否存在 |
| getAbsolutePath() | String | 获取文件的绝对路径 |
| getName() | String | 获取文件的名称 |
| getParent() | String | 获取文件的父路径 |
| isDirectory() | boolean | 判断文件是否是一个目录 |
| isFile() | boolean | 判断文件是否存在 |
| isHidden() | boolean | 判断文件是否是隐藏文件 |
| lastModified() | long | 获取文件最后修改时间 |
| length() | long | 获取文件长度（字节单位） |
| listFiles() | File[] | 返回当前路径下的所有文件（夹）构建的File对象 |

## 4.文件输入/输出流

FileInputStream类构造方法：FileOutputStream类与此类似  
FileInputStream(String name)  
FileInputStream(File file)

FileReader和FileWriter类用于对文件进行字符输入输出

## 5.带缓存的输入/输出流

带缓存的IO流才支持skip()、mark()、reset()功能。Java中的缓存流不是独立创建的，而是“包装”在一个输入或输出流类对象上的，以优化其性能。

BufferedInputStream的构造函数：BufferedOutputStream与此类似  
BufferedInputStream(InputStream in)  
BufferedInputStream(InputStream in, int size)

BufferedReader的常用方法：  
read()  
readLine()

BufferedWriter的常用方法：  
write(String s, int off, int len)  
flush()  
newLine()

## 6.数据输入/输出流

允许程序以与机器无关的方式从底层输入输出，不必关心数值的字节类型

构造方法：  
DataInputStream(InputStream in)  
DataOutputStream(OutputStream out)

DataOutputStream的写入字符串  
writeBytes(String s) 将字符串s中所有字符的低字节写入  
writeChars(String s) 将s中的每一个字符的两个字节都写入  
writeUTF(String s) 先写入UTF字符串长度，再写入UTF编码

DataInputStream只有一个方法：  
readUTF()：只有writeUTF方法写入了字符串长度，才能以字符形式读取数据

## 7.ZIP压缩输入/输出流

类包Java.util.zip，主要类ZipOutputStream（用于压缩）、ZipInputStream（用于解压缩）、ZipEntry（目录进入点，代表文件在压缩包中的形态，亦即压缩包内的文件）。

ZipOutputStream构造方法：（必须使用待压缩文件名构造的文件输出流对象FileOutputStream作参数）  
ZipOutputStream(OutputStream out)

ZipOutputStream常用方法：  
putNextEntry(ZipEntry e)——压缩文件过程即将文件写入压缩包的过程，每写入一个文件，需要先使用此方法创建其在压缩包中的“代表”实体  
write(byte[] b, int off, int len)  
finish()  
setComment(String comment)

ZipInputStream构造方法：（道理类似，必须用待解压文件包创建的文件输入流对象来构造）  
ZipInputStream(InputStream in)

常用方法：  
read(byte[] b, int off, int len)  
available()  
closeEntry()  
skip(long n)  
getNextEntry()——道理和压缩相反，解压过程相当于依照压缩包内容条目在压缩包外创建同名同路径的文件  
createZipEntry(String name)

ZipEntry常用方法：  
ZipEntry(String entryName)——构造方法，条目名即文件名  
getName()

# 附录1. Java DOS指令

|  |  |
| --- | --- |
| javac -d ./bin/ ./com/lzw/\*.java | 编译com/lzw路径下的所有java源文件并将编译好的文件输出至bin文件夹下，bin文件夹中也沿用com/lzw的路径形式，bin和com文件夹都位于当前文件夹或路径下 |
|  |  |
|  |  |