[第一章Javalang包 3](#_Toc452822558)

[本章关键词 3](#_Toc452822559)

[Character类 4](#_Toc452822560)

[String 类 6](#_Toc452822561)

[== 和equls的比较 7](#_Toc452822562)

[数据格式转换 11](#_Toc452822563)

[字符串的不变性 12](#_Toc452822564)

[总结 16](#_Toc452822565)

[第二章 Java.util 包 16](#_Toc452822566)

[本章关键词 16](#_Toc452822567)

[Date类 17](#_Toc452822568)

[Data类的构造方法 17](#_Toc452822569)

[Calendar类 18](#_Toc452822570)

[Calend类的常用方法 18](#_Toc452822571)

[Radom类 19](#_Toc452822572)

[Vector类 20](#_Toc452822573)

[Vector类的常用方法 20](#_Toc452822574)

[Hashset 21](#_Toc452822575)

[Treeset 22](#_Toc452822576)

[Hashmap类 23](#_Toc452822577)

[Collection补充 24](#_Toc452822578)

[总结 24](#_Toc452822579)

[第三章Java.IO包 25](#_Toc452822580)

[本章关键词 25](#_Toc452822581)

[Java.IO包简介 25](#_Toc452822582)

[File类 26](#_Toc452822583)

[Flie类的构造方法 26](#_Toc452822584)

[File类中的常用方法 26](#_Toc452822585)

[Stream流 27](#_Toc452822586)

[流的类型 28](#_Toc452822587)

[FileInputStream 28](#_Toc452822588)

[FileOutputStream 30](#_Toc452822589)

[小结 32](#_Toc452822590)

[总结 33](#_Toc452822591)

## 第一章Javalang包

### 本章关键词

|  |  |
| --- | --- |
| **单 词** | **说 明** |
| **language** | **语言** |
| **Integer** | **整数** |
| **character** | **字符** |
| **buffer** | **缓冲器** |
| **math** | **数学** |
| **parse** | **转换** |
| **equals** | **相等** |
| **compare** | **比较，相比** |
| **replace** | **替换，取代** |

java.lang包是java内置的一个基础包，其中包含了一系列程序中经常要用到的类；在默认情况下，每个java程序都会自动导入该包，因此无需在程序中显式地声明。

包装类：

原始数据类型和包装类对照

|  |  |
| --- | --- |
| **原始数据类型** | **包 装 类** |
| **boolean（布尔型）** | **Boolean** |
| **byte（字节型）** | **Byte** |
| **char（字符型）** | **Character** |
| **short（短整型）** | **Short** |
| **int（整型）** | **Integer** |
| **long（长整型）** | **Long** |
| **float（浮点型）** | **Float** |
| **double（双精度浮点型）** | **Double** |

### Character类

包装类的构造方法

public class LangDemo

{

public static void main(String[] args)

{

Boolean objBool = new Boolean(true);

Character objChar = new Character('X');

Integer objInt = new Integer(100);

Long objLong = new Long(2568);

Double objDou = new Double(3.1415);

System.out.println(objBool);

System.out.println(objChar);

System.out.println(objInt);

System.out.println(objLong);

System.out.println(objDou);

}

}

每个包装类都有一个静态的valueOf方法，用于将字符串转换成相应包装类的对象。

Eg

public class LangDemo {

public static void main(String[] args) {

String str = "120";

//如果转换失败，将会引发NumberFormatException异常

Byte objByte = Byte.valueOf(str);

Short objShort = Short.valueOf(str);

Integer objInt = Integer.valueOf(str);

Long objLong = Long.valueOf(str);

System.out.println(objByte);

System.out.println(objShort);

System.out.println(objInt);

System.out.println(objLong);

}

}

除了Character类以外，其它的包装类都有静态的parseXxx方法（Xxx指代具体的数据类型），用于将字符串转换成相对应的原始数据类型值。

public class ParseTest {

public static void main(String[] args) {

String str = "116";

//分别调用各个包装类的paseXxx方法对字符串进行转换，如果转换失败，将报异常

int i = Integer.parseInt(str);

short s = Short.parseShort(str);

byte b = Byte.parseByte(str);

long l = Long.parseLong(str);

float f = Float.parseFloat(str);

double d = Double.parseDouble(str);

boolean b = Boolean.parssBoolean(“true”);

System.out.println(i);

System.out.println(s);

System.out.println(b);

System.out.println(l);

System.out.println(f);

System.out.println(d);

System.out.println(b);

}

}

Character类中常用的方法

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **boolean isLetter(char ch)** | **判断字符ch是否为英文字母** |
| **boolean isDigit(char ch)** | **判断字符ch是否为0~9之间的数字** |
| **boolean isUpperCase(char ch)** | **判断字符ch是否为大写形式** |
| **boolean isLowerCase(char ch)** | **判断字符ch是否为小写形式** |
| **boolean isWhitespace(char ch)** | **判断字符ch是否为空格或换行符或回车符或制表符** |

以上方法都是静态方法，可以直接通过类名调用，返回值均为boolean类型，如果是返回true，否则返回false。

实例：

public class CharacterDemo {

public static void main(String[] args) {

char[] charArray = {'\*', '7', 'b', ' ', 'A'};

for (int i = 0; i < charArray.length; i++) {

if (Character.isDigit(charArray[i])) {

System.out.println(charArray[i] + "是一个数字。");

}

if (Character.isLetter(charArray[i])) {

System.out.println(charArray[i] + "是一个字母。");

}

if (Character.isWhitespace(charArray[i])) {

System.out.println(charArray[i] + "是一个空格。");

}

if (Character.isLowerCase(charArray[i])) {

System.out.println(charArray[i] + "是小写形式。");

}

if (Character.isUpperCase(charArray[i])) {

System.out.println(charArray[i] + "是大写形式。");

}

}

}

### String 类

Java中，字符串是String类的对象；可以通过使用String类提供的方法来完成对字符串的操作；创建一个字符串对象之后，将不能更改构成字符串的字符；每当更改了字符串版本时，就创建了一个新的字符串对象，并在其内包含所做的修改，原始字符串保持不变。

String类的构造方法共有13种重载方式，以下是常用的几个：

|  |  |
| --- | --- |
| **构造方法** | **说 明** |
| **String()** | **将创建一个空字符串** |
| **String(String original)** | **将新建一个字符串作为指定字符串的副本** |
| **String(char[] value)** | **将根据字符数组构造一个新字符串** |
| **String(byte[] tytes)** | **将通过转换指定的字节数组新建一个字符串** |

示例

public class StringDemo

{

public static void main(String[] args)

{

char[] aryChar = {‘I', 'C', ‘S', ‘S'};

String str1 = “ETC";

//利用一个字符串常量值创建新的字符串

String str2 = new String(“ICSSETC");

//利用一个字符型数组创建新的字符串

String str3 = new String(aryChar);

System.out.println(str1);

System.out.println(str2);

System.out.println(str3);

}

}

字符串长度

String类中提供length成员方法 ，用来获得字符串的长度，方法原型：int length()

该方法返回字符串中有效字符的个数。

public class StringDemo

{

public static void main(String[] args)

{

String str1 = "John Smith";

String str2 = new String("I Love Java");

System.out.println(str1.length());

System.out.println(str2.length());

}

}

### == 和equls的比较

1.基础类型直接比较用==比较的是值，一个基础一个引用==比较的也是值

2.引用类型不同的不能用==比较，编译报错,可以用equals进行比较(先比较数据类型，然后比较值)

3.两个Byte/Short/Integer/Long类型的比较注意数据缓存。-128-127之间的比较相等

public class StringDemo {

public static void main(String[] args) {

String str1 = “ICSS", str2 = “ICSS";

String str3 = new String(“ETC"), str4 = new String(“ETC");

if (str1 == str2) {

System.out.println("str1和str2指向同一字符串"); }

else {

System.out.println("str1和str2分别指向不同字符串"); }

if (str1.equals(str2)) {

System.out.println("str1和str2的内容完全相同"); }

else {

System.out.println("str1和str2的内容不相同"); }

if (str3 == str4) {

System.out.println("str3和str4指向同一字符串"); }

else {

System.out.println("str3和str4分别指向不同字符串"); }

if (str3.equals(str4)) {

System.out.println("str3和str4的内容完全相同");}

else {

System.out.println("str3和str4的内容不相同"); }

}

}

**equals(Object anObject))方法则是比较两个字符串中的内容是否相同，如果相等返回true，否则返回false。**

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **boolean**  **equalsIgnoreCase(String anotherString)** | **判断字符串anotherString是否与当前字符串相等，忽略大小写形式** |
| **int compareTo(String anotherString)** | **根据ASCII码比较字符串anoterString和当前字符串的大小，比较方式类似于C语言中的strcmp函数** |
| **boolean startsWith(String prefix)** | **判断当前字符串是否以字符串prefix为开头** |
| **boolean endsWith(String suffix)** | **判断当前字符串是否以字符串suffix为后缀** |

如果需要搜索某个字符（或某个子串）在字符串中是否出现过，这就要使用到indexOf方法和lastIndexOf方法

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **int indexOf(int ch)** | **搜索字符ch在当前字符串中第一次出现的索引，没有出现则返回-1** |
| **int indexOf(String str)** | **搜索字符串str在当前字符串中第一次出现的索引，没有出现则返回-1** |
| **int lastIndexOf(int ch)** | **搜索字符ch在当前字符串中最后一次出现的索引，没有出现则返回-1** |
| **int lastIndexOf(String str)** | **搜索字符串str在当前字符串中最后一次出现的索引，没有出现则返回-1** |

字符串搜索实例

public class StringDemo

{

public static void main(String[] args)

{

String strEmail = "java@sun.com";

int index;

System.out.println("E-mail地址：" + strEmail);

index = strEmail.indexOf('@');

System.out.println("@字符出现的索引：" + index); //4

index = strEmail.indexOf("sun");//5

System.out.println("字符串\"sun\"出现的索引：" + index);

index = strEmail.lastIndexOf('a');//3

System.out.println("a字符最后一次出现的索引：" + index);

}

}

提取字符串

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **char charAt(int index)** | **用于从指定位置提取单个字符，该位置由index指定，索引值必须为非负** |
| **String substring(int index)** | **用于提取从index指定的位置开始的字符串部分** |
| **String substring(int begin, int end)** | **用于提取 begin 和 end 位置之间的字符串部分,即*begin---->end-1*** |
| **String concat(String str)** | **用于连接两个字符串，并新建一个包含调用字符串的字符串对象** |
| **String replace(char oldChar, char newChar)** | **用于将调用字符串中出现oldChar指定的字符全部都替换为newChar指定的字符** |
| **replaceAll(String regex, String replacement)** | **用于将调用字符串中出现或者匹配regex的字符串全部都替换为replacement指定的字符** |
| **String trim()** | **用于返回一个前后不含任何空格的调用字符串的副本** |

提取字符串实例

public class StringDemo

{

public static void main(String[] args)

{

String str1 = " Java is OOP";

String str2 = new String(" ACCP");

System.out.println(str1.charAt(2));

System.out.println(str1.substring(5));

System.out.println(str1.substring(2, 9));

System.out.println(str1.concat(str2));

System.out.println(str1 + str2);

System.out.println(str1.replace('a', 'e'));

System.out.println(str1.trim());

}

}

对字符串中字符的大小写形式进行转换。

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **String toUpperCase()** | **返回当前字符串的全大写形式** |
| **String toLowerCase()** | **返回当前字符串的全小写形式** |

Eg

public class StringDemo

{

public static void main(String[] args)

{

String str1 = "Java is OOP";

String str2;

str2 = str1.toLowerCase();

System.out.println(str2);

str2 = str1.toUpperCase();

System.out.println(str2);

}

}

### 数据格式转换

在某些特定的场合，我们可能需要将字符串转化成其它格式的数据进行操作；

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **byte[] getBytes()** | **返回当前字符串转化成byte型数组的形式** |
| **char[] toCharArray()** | **返回当前字符串的字符数组形式，类似于C语言中字符串的保存形式** |

Stringbuffer类

String在操作的时候会产生新的字符串对象，导致大量的内存耗用构造一个其中不带字符的字符串缓冲区，初始容量为 16 个字符。

特点：

1：可以对字符串内容进行修改。

2：是一个容器。

3：是可变长度的。

4：缓冲区中可以存储任意类型的数据。

5：最终会通过toString方法变成字符串。

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **StringBuffer insert(int index, x x)** | **可以将数据插入到指定index位置，x可以为任何类型的数据** |
| **int length()** | **获得当前StringBuffer对象的长度** |
| **void setCharAt(int index, char ch)** | **使用 ch 指定的新值替换 index指定的位置上的字符** |
| **String toString()** | **转换为字符串形式** |
| **StringBuffer reverse()** | **将当前StringBuffer对象中的字符序列倒置** |
| **StringBuffer delete(int start, int end)** | **删除缓冲区中的数据，包含start，不包含end。** |
| **StringBuffer deleteCharAt(int index)** | **将删除 index 指定的索引处的字符** |
| **StringBuffer replace(int start, int end, String str)** | **此方法使用一组字符替换另一组字符。将用替换字符串从 start指定的位置开始替换，直到 end 指定的位置结束** |

### 字符串的不变性

在Java中一旦创建了字符串就不能直接更改，这就是字符串的不变性；而StringBuffer类正是针对此问题而提供的字符可变序列；StringBuffer与String是同等的类，唯一区别是可以进行更改。

Math类

* Math类中提供了一系列基本数学运算和几何运算的方法；
* 该类的**构造方法被修饰为private**，因此不能实例化；
* 该类中的**所有方法都是静态的**，可以通过类名直接调用；

该类被修饰为**final**，因此没有子类

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **static int abs(int a)** | **求a的绝对值，有4种重载，还有float，double和long** |
| **static double pow(double a, double b)** | **求a的b次方幂** |
| **static double sqrt(double a)** | **求a的平方根** |
| **static int round(float a)** | **求a的四舍五入取整结果** |
| **static double ceil(double a)** | **返回不小于a的最小整数值** |
| **static double floor(double a)** | **返回不大于a的最大整数值** |
| **static double sin(double a)** | **返回a的正弦值** |
| **static double cos(double a)** | **返回a的余弦值** |

* Math类中还包括两个常用的常量：
  + PI：圆周率π
  + E：自然常量
* 以上常量在Math类中都被声明成静态，可以直接通过类名进行访问。

Objet 类

ava中的类体系遵循单根结构，即任何一个类往上追溯都到达同一个父类；

Object类就是这个单根体系的根，也就是说它是其它所有类的共同父类；

如果用户定义的类没有扩展任何其它类，则默认扩展自Object类；

Object类中定义的一些方法，会被继承到所有类中 hashCode toString

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **boolean equals(Object obj)** | **判断当前对象是否与参数obj（内容）相等，如果有必要，应该在自定义的类中覆盖该方法** |
| **String toString()** | **返回当前对象的字符串表示，如果有必要，应该在自定义的类中覆盖该方法** |
| **Class getClass()** | **返回当前对象的类描述对象，此方法被继承到所有类中** |
| **protected void finalize()**  **throws Throwable** | **当前对象被垃圾回收时调用此方法（类似于C++的析构函数），但无法确定具体何时调用** |
| **public final void wait()**  **throws InterruptedException** | **使当前线程进入等待状态** |

class Student { //定义Student类，缺省继承于Object类

private String mName;

private int mAge;

public Student(String name, int age) { //构造方法

mName = name;

mAge = age;

}

public String toString() { //覆盖Object类中的toString方法

String str = "姓名：" + mName + ", 年龄：" + mAge + "岁";

return str;

}

}

public class ToStringDemo { //容纳main方法

public static void main(String[] args) {

Student std = new Student("张三", 18);

System.out.println(std); //默认调用toString方法

}

}

Class类

* Java应用程序实际上都是由一个个对象组成，这些对象分别属于什么类，是从哪个类继承而来，这一系列的信息都是由Class类的实例来记录的；
* Class类的实例用于记录对象的类描述信息；
* 如果在程序运行时，需要**检索某个对象的相关类信息**，可以**调用该对象的getClass方法**来获得记录其描述信息的Class类实例；

Class类没有公共的构造方法，无法通过new运算符实例化，只能通过对象的getClass方法，或是通过Class的静态方法forName来获得实例

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **static Class**  **forName(String className)**  **throws ClassNotFoundException** | **使用参数className来指定具体的类，来获得相关的类描述对象，该方法有可能抛出类加载异常（ClassNotFoundException），必须捕捉** |
| **Class getSuperclass()** | **获得当前类描述对象的父类的描述对象** |
| **String getName()** | **返回当前类描述对象的类名称** |

类描述对象实例

public class ClassDemo {

public static void main(String[] args) {

try {

/\*使用forName方法获得任意一个类的类描述对象

forName方法有可能抛异常，必须捕捉\*/

Class cls = Class.forName("java.lang.StringBuffer");

//循环打印父类信息，直到没有父类

while (cls != null) {

System.out.println(cls);

cls = cls.getSuperclass();

}

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

}

### 总结

java.lang包是Java中最基础的一个包，也是唯一一个无需显示声明就默认导入包；

包装类共有8个，它们可以以对象的形式封装原始类型的数据；

Java中的字符串以String类对象的形式存放，该类中提供一系列对字符串进行操作的方法；

StringBuffer类是另一种形式的字符串，与String类的区别是允许对自身的内容进行修改；

Math类提供一系列进行算术计算的方法，这些方法都是静态的；

Object类是其它所有类的父类；

Class类的实例记录了对象的类信息。

# 第二章 Java.util 包

## 本章关键词

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **int nextInt()** | **从随机数生成器返回下一个整型值** |
| **long nextLong()** | **从随机数生成器返回下一个长整型值** |
| **float nextFloat()** | **从随机数生成器返回0.0到1.0之间的下一个浮点值** |
| **double nextDouble()** | **从随机数生成器返回0.0到1.0之间的下一个双精度值** |
| **double nextGaussian()** | **从随机数生成器返回下一个高斯分布的双精度值。中间值为0.0，而标准差为1.0** |

## Date类

Date类对象用来表示时间和日期；该类提供一系列操纵日期和时间各组成部分的方法；Date类最多的用途是获取系统当前的日期和时间。

### Data类的构造方法

|  |  |
| --- | --- |
| **构 造 方 法** | **说 明** |
| **Date()** | **使用系统当前时间创建日期对象** |
| **Date(long date)** | **使用自1970年1月1日以后的指定毫秒数创建日期对象** |
| **Date(int year, int month, int date)** | **创建指定年、月、日的日期对象** |
| **Date(int year, int month, int date,int hrs, int min, int sec)** | **创建指定年、月、日、时、分、秒的日期对象** |

Data类的常用方法

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **boolean after(Date when)** | **如果当前日期对象在when指定的日期对象之后，返回true，否则返回false** |
| **boolean before(Date when)** | **如果当前日期对象在when指定的日期对象之前，返回true，否则返回false** |
| **void setTime(long time)** | **设置日期对象，以表示自1970年1月1日起的指定毫秒数** |
| **boolean equals(Object obj)** | **如果两个日期对象完全相同，返回true，否则返回false** |
| **String toString()** | **返回日期的格式化字符串，包括星期几** |
| **String toLocalString()** | **返回本地时间** |

Data实例

public class DateDemo

{

public static void main(String[] args)

{

Date date = new Date(); //获得当前的系统日期和时间

System.out.println("今天的日期是：" + date);

long time = date.getTime(); //获得毫秒数

System.out.println("自1970年1月1日起以毫秒为单位的时间(GMT):" + time);

//截取字符串中表示时间的部分

String strDate = date.toString();

String strTime = strDate.substring(11, (strDate.length() - 4));

System.out.println(strTime);

strTime = strTime.substring(0, 8);

System.out.println(strTime);

}

}

## Calendar类

Calendar类也是用来操作日期和时间的类，但它可以以整数形式检索类似于年、月、日之类的信息；

**Calendar类是抽象类，无法实例化，**要得到该类对象只能通过调用getInstance方法来获得；

Calendar对象提供为特定语言或日历样式实现日期格式化所需的所有时间字段

### Calend类的常用方法

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **Calendar getInstance()** | **返回默认地区和时区的Calendar对象** |
| **int get(int fields)** | **返回调用对象中fields指定部分的值** |
| **void set(int fields, int value)** | **将value中指定的值设置到fields指定的部分** |
| **void add(int fields, int amount)** | **将amount值添加到fields指定的时间或日期部分** |
| **Date getTime()** | **返回与调用对象具有相同时间的Date对象** |
| **Object clone()** | **返回调用对象的副本** |
| **void clear()** | **清除当前对象中所有的时间组成部分，从1970年初始值开始** |
| **boolean after(Object obj)** | **如果调用对象时间在obj之后，返回true** |
| **boolean before(Object obj)** | **如果调用对象时间在obj之前，返回true** |
| **boolean equals(Object obj)** | **判断调用对象与obj是否相等** |

Calendar实例

public class CalendarDemo

{

public static void main(String[] args)

{

//创建包含有当前系统时间的Calendar对象

Calendar cal = Calendar.getInstance();

//打印Calendar对象的各个组成部分的值

System.out.print("当前系统时间：");

System.out.print(cal.get(Calendar.YEAR) + "年");

System.out.print((cal.get(Calendar.MONTH) + 1) + "月");

System.out.print(cal.get(Calendar.DATE) + "日 ");

System.out.print(cal.get(Calendar.HOUR\_OF\_DAY) + ":");

System.out.print(cal.get(Calendar.MINUTE) + ":");

System.out.println(cal.get(Calendar.SECOND));

//将当前时间添加30分钟，然后显示日期和时间

cal.add(Calendar.MINUTE, 30);

Date date = cal.getTime();

System.out.println("将当前时间添加30分钟后的时间：" + date);

}

}

## Radom类

Random类专门用来生成随机数；

该类的构造方法有2种重载方式。

经常使用new Random创建一个随机数对象，然后调用nextInt(参数)，去规定随机数的范围。

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **int nextInt()** | **从随机数生成器返回下一个整型值** |
| **long nextLong()** | **从随机数生成器返回下一个长整型值** |
| **float nextFloat()** | **从随机数生成器返回0.0到1.0之间的下一个浮点值** |
| **double nextDouble()** | **从随机数生成器返回0.0到1.0之间的下一个双精度值** |
| **double nextGaussian()** | **从随机数生成器返回下一个高斯分布的双精度值。中间值为0.0，而标准差为1.0** |

## Vector类

* Vector类与ArrayList类和LinkedList类很相似，最大的区别在于**Vector是线程同步**的
* 如果**在多线程的程序中要使用到集合框架**，并且不希望线程与线程之间相互干扰，那么**Vector是不错的选择**；
* **Vector**类**继承**于***AbstractList类***，并**实现**了**List接口**。

### Vector类的常用方法

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **int size()** | **返回Vector对象的大小，即元素的数量** |
| **boolean isEmpty()** | **判断Vector对象是否为空，为空返回true，否则返回false** |
| **void clear()** | **清空Vector对象中的所有元素** |
| **boolean add(Object element)** | **向Vector对象中添加一个元素，该元素可以是任何类的对象** |
| **Object remove(int index)** | **从Vector对象中删除指定索引位置的元素** |
| **Object get(int index)** | **返回指定索引位置的元素** |
| **Object set(int index, Object elem)** | **将元素elem存放到由index指定的索引位置上** |
| **int indexOf(Object element)** | **判断element在Vector对象中是否存在，存在返回对应的索引，否则返回-1** |
| **方 法 原 型** | **说 明** |
| **int capacity()** | **返回Vector对象的容量，即可以存放元素的个数** |
| **void addElement(Object element)** | **将指定元素插入到Vector对象的末尾处** |
| **void insertElementAt(Object elem, int index)** | **将指定元素插入到指定索引位置** |
| **void setElementAt(Object elem, int index)** | **将指定对象替换位于指定索引处的对象** |
| **Object ElementAt(int index)** | **检索位于指定索引处的元素** |
| **boolean contains(Object elem)** | **如果Vector对象包含指定元素，返回true** |
| **Object firstElement()** | **返回Vector对象中的第一个元素** |
| **Object lastElement()** | **返回Vector对象中的最后一个元素** |
| **void removeAllElements()** | **删除Vector对象中的所有元素** |
| **void copyInto(Object[] anArray)** | **将Vector对象中的元素复制到指定数组中** |
| **void setSize(int newSize)** | **根据newSize的值设置Vector对象的容量** |

Vector类实例

public class VectorDemo

{

public static void main(String[] args)

{

Vector vec = new Vector(); //创建空的Vector

//往Vector中添加元素

vec.addElement("Java");

vec.addElement("C#");

vec.addElement("Oracle");

vec.addElement("C++");

vec.addElement("HTML");

System.out.println(vec.toString()); //打印Vector中的元素

vec.removeElement("C++"); //删除其中的元素

System.out.println(vec.toString());

}

}

## Hashset

不能保证元素的排列顺序，顺序有可能发生变化

不是同步的,即线程不安全

HashSet 是哈希表实现的,HashSet中的数据是无序的，可以放入null，但只能放入一个null(无重复)

当向HashSet集合中存入一个元素时，HashSet会调用该对象的hashCode()方法来得到该对象的hashCode值， 然后根据 hashCode值来决定该对象在HashSet中存储位置。如果这个位置上没有元素，它就可以直接存储在这个位置上，不用再进行任何比较了；如果这个位置上已经有元素了， 就调用它的equals方法与新元素进行比较，相同的话就不存了，不相同就散列其它的地址。

LinkedHashset

LinkedHashSet继承了HashSet集合，同样是根据元素的hashCode值来决定元素的存储位置，但是它同时使用链表维护元素的次序。

这样使得元素看起 来像是以插入顺序保存的，也就是说，当遍历该集合时候,LinkedHashSet将会以元素的添加顺序访问集合的元素。

LinkedHashSet在迭代访问Set中的全部元素时，性能比HashSet好，但是插入时性能稍微逊色于HashSet。

Ps：linkedhashset和hashset比较 前者对数值的添加速度大于hashset的速度，后者对数值的移除速度高于前者，因为后者是通过移动集合号中的元素获得，前者使用指针进行指向。

## Treeset

TreeSet是SortedSet接口的唯一实现类，TreeSet可以确保集合元素处于排序状态。

TreeSet 是二差树实现的,Treeset中的数据是自动排好序的，不允许放入null值(默认是升序)

TreeSet支持两种排序方式，自然排序 和定制排序。其中自然排序为默认的排序方式。向TreeSet中加入的应该是同一个类的对象。

TreeSet判断两个对象不相等的方式是两个对象通过equals方法返回false，或者通过CompareTo方法比较返回0。

**自然排序**

自然排序使用要排序元素的**CompareTo（Object obj）**方法来比较元素之间大小关系，然后将元素按照升序排列。

Java提供了一个**Comparable接口**，该接口里定义了一个***compareTo(Object obj)***方法，该方法返回一个**整数**值，实现了该接口的对象就可以比较大小。obj1.compareTo(obj2)方法如果返回0，则说明被比较的两个对象相等，如果返回一个正数，则表明obj1大于obj2，如果是 负数，则表明obj1小于obj2。

如果我们将两个对象的equals方法总是返回true，则这两个对象的compareTo方法返回应该返回0

**定制排序**

自然排序是根据集合元素的大小，以升序排列，如果要定制排序，应该使用**Comparator接口**，实现 **int compare(T o1,T o2)**方法

Treeset提供的一些方法

first();//**返回排序好的第一个元素**

last();//**返回排序好的最后一个元素**

lower(T t);//**小于指定元素之前的一个元素**

higher(T t);//**大于指定元素之后的一个元素**

subSet(T a,T b);//返回SortedSet,**大于等于a小于b的元素**

headSet(T a);//**返回小于指定元素的所有元素**

tailSet(T a);//**返回大于等于指定元素的所有元素**

注意:

set集合、list集合都不是线程安全的，当多个线程访问集合并且有超过一个线程修改了该集合，就必须手动方式保证集合同步。

Java提供了一个工具类----->Collections

如：

Set<Integer> nums = Collections.synchronizedSet(new HashSet<Integer>());

## Hashmap类

HashMap以键值对的形式存储元素；

对于HashMap来说，不存在索引，也就是说不可以通过索引来访问元素，只能通过键去访问值

由于没有索引，所以HashMap中元素的存放是没有顺序的；

HashMap类继承了AbstractMap类，并实现了Map接口。

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **int size()** | **返回HashMap对象的大小，即元素的个数** |
| **boolean isEmpty()** | **判断HashMap对象是否为空，为空返回true，否则返回false** |
| **void clear()** | **清空HashMap对象中的所有元素** |
| **Object put(Object key, Object value)** | **向HashMap对象中添加一个元素，必须指定该元素的键和值** |
| **Object remove(Object key)** | **通过键对象删除相对应的值对象** |
| **Object get(Object key)** | **通过键对象查找相对应的值对象** |
| **boolean containsKey(Object key)** | **查找指定的键对象在HashMap对象中是否存在** |
| **boolean containsValue(Object value)** | **查找指定的值对象在HashMap对象中是否存在** |

Hashmap 示例

public class HashDemo {

public static void main(String[] args) {

HashMap hm = new HashMap(); //创建空的HashMap

//往HashMap中添加元素

hm.put("93-07", "张三");

hm.put("84-12", "李四");

hm.put("102-20", "王五");

hm.put("91-04", "郑六");

hm.put("111-17", "田七");

//打印出HashMap中的元素

System.out.println(hm.toString());

//通过键对象查找值对象

System.out.println("学号91-04的学生是" + hm.get("91-04"));

System.out.println("学号84-12的学生是" + hm.get("84-12"));

hm.remove("93-07"); //通过键对象删除元素

System.out.println(hm.toString());

}

}

## Collection补充

* **Collection**是集合类的上级接口，继承与他的接口**主要有Set 和List**.
* **Collections是针对集合类的一个帮助类**，他提供一系列**静态方法**实现对各种集合的搜索、排序、线程安全化等操作.

## 总结

java.util包中包含一系列常用的工具类和集合框架类；

Date类和Calendar类用来操作日期和时间；

可以使用Random类的对象来生成随机数；

集合框架提供了一系列复杂的数据结构和优越的算法；

ArrayList类可以创建动态数组；

LinkedList类可以创建链表结构；

Vector类可以创建线程同步的动态数组；

熟悉Set接口下的HashSet/TreeSet的区别联系

HashMap类可以创建键值对集合。

1）HashSet

元素无序的，并且不能重复

线程不安全

hashSet是哈希表实现的 ，可以放入null,但是只能放入1个null.

hashCode()方法来确定你要存放位置。如果这个位置上没有元素，他就可以直接存放在这个位置上。

如果有元素了，就调用它的equals()方法与新元素进行比较。相同的话就不存放了，不想通就散列到其他位置。

2）LinkedHashSet

同样根据元素的hashCode值来确定的元素位置。链表维护关系

3）TreeSet

sortedSet接口的唯一实现类。保证元素处于排序状态

TreeSet是二叉树实现的。不可以放入null值。

应该使用Comparator接口，实现 int compare(T o1,T o2)方法

Collections.synchronizedList(new HashSet(Integer));

当我们多个线程访问集合并且超过一个线程修改了这个集合对象，就必须手动方式保证set集合同步

# 第三章Java.IO包

## 本章关键词

|  |  |
| --- | --- |
| **单 词** | **说 明** |
| **input** | **输入** |
| **output** | **输出** |
| **file** | **文件，档案** |
| **directory** | **目录** |
| **stream** | **流** |
| **write** | **写，书写** |
| **read** | **读，阅读** |
| **source** | **源头，来源** |
| **destination** | **目标，目的地** |

## Java.IO包简介

java.io包也是Java内置的包，其中包含一系列对文件和目录的属性进行操作，对文件进行读写操作的类；

程序中如果要使用到该包中的类，对文件或流进行操作，则必须显式地声明如下语句：

import java.io.\*;

## File类

File类的对象不但可以表示文件，还可以表示目录，在程序中一个File类对象可以代表一个文件或目录；

当创建一个文件对象后，就可以利用它来对文件或目录的属性进行操作，如：文件名、最后修改日期、文件大小等等；

需要注意的是，**File对象并不能直接对文件进行读/写操作，只能查看文件的属性；**

### Flie类的构造方法

|  |  |
| --- | --- |
| **构 造 方 法** | **说 明** |
| **File(String pathname)** | **指定文件（或目录）名和路径创建文件对象** |

Eg：

如：

//在当前目录下创建一个与aaa.txt文件名相关联的文件对象

File f1 = new File("aaa.txt");

//指明详细的路径以及文件名，请注意双斜线

File f2 = new File("D:\\Java\\Hello.java");

### File类中的常用方法

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **boolean exists()** | **判断文件是否存在，存在返回true，否则返回false** |
| **boolean isFile()** | **判断是否为文件，是文件返回true，否则返回false** |
| **boolean isDirectory()** | **判断是否为目录，是目录返回true，否则返回false** |
| **String getName()** | **获得文件的名称** |
| **String getAbsolutePath()** | **获得文件的绝对路径** |
| **long length()** | **获得文件的长度（字节数）** |
| **boolean createNewFile()**  **throws IOException** | **创建新文件，创建成功返回true，否则返回false，有可能抛出IOException异常，必须捕捉** |
| **boolean delete()** | **删除文件，删除成功返回true，否则返回false** |
| **File[] listFiles()** | **返回文件夹内的子文件与子文件夹的数组** |

Eg：

public class FileDemo

{

public static void main(String[] args)

{

//创建一个文件对象

File file = new File("test.txt");

//显示与文件有关的属性信息

System.out.println("文件或目录是否存在：" + file.exists());

System.out.println("是文件吗：" + file.isFile());

System.out.println("是目录吗：" + file.isDirectory());

System.out.println("名称：" + file.getName());

System.out.println("绝对路径：" + file.getAbsolutePath());

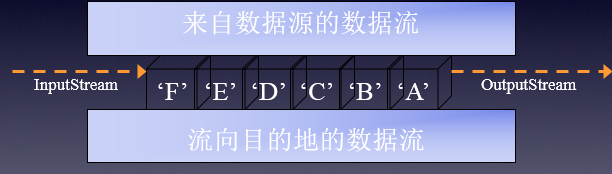
System.out.println("文件大小：" + file.length());

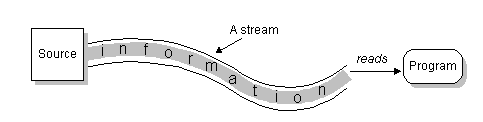
}

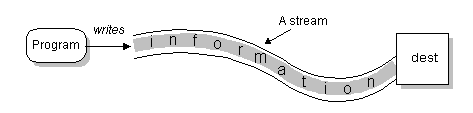
}

## Stream流

流是指一连串流动的数据信号，是以先进先出的方式发送和接收数据的通道。







### 流的类型

* 根据流动方向的不同，流分为输入流和输出流；
* 对于输入和输出流，由于**传输格式**的不同，又分为字节流和字符流：
  + 字节流是指8位的通用字节流，以**字节**为基本单位，在java.io包中，对于字节流进行操作的类大部分继承于**InputStream（输入字节流）**类和**OutputStream（输出字节流）**类；
  + 字符流是指16位的Unicode字符流，以**字符**（两个字节）为基本单位，非常适合处理字符串和文本，对于字符流进行操作的类大部分继承于**Reader（读取流）**类和**Writer（写入流）**类。

#### FileInputStream

FileInputStream类称为文件输入流，继承于InputStream类，是进行文件读操作的最基本类；

它的作用是将文件中的数据输入到内存中，我们可以利用它来读文件；由于它属于字节流，因此在读取Unicode字符（如中文）的文件时可能会出现问题。

##### FileInputStream类的构造方法

|  |  |
| --- | --- |
| **构 造 方 法** | **说 明** |
| **FileInputStream(File file)**  **throws FileNotFoundException** | **使用File对象创建文件输入流对象，如果文件打开失败，将抛出异常** |
| **FileInputStream(String name)**  **throws FileNotFoundException** | **使用文件名或路径创建文件输入流对象，如果文件打开失败，将抛出异常** |
| **方 法 原 型** | **说 明** |
| **int read()**  **throws IOException** | **读取文件中的数据，一次读取一个字节，读取的数据作为返回值返回，如果读到文件末尾则返回-1，有可能抛异常，必须捕捉** |
| **int read(byte[] b)**  **throws IOException** | **读取文件中的数据，将读到的数据存放到byte型数组中，并返回读取的字节的数量，未读到数据返回-1，有可能抛异常，必须捕捉** |
| **void close()**  **throws IOException** | **关闭流对象，有可能抛异常，必须捕捉** |

Eg:

public class FileInputStreamDemo1 {

public static void main(String[] args) {

try {

File file = new File("test.txt"); //创建文件对象

//使用文件对象创建文件输入流对象，相当于打开文件

FileInputStream fis = new FileInputStream(file);

for (int i = 0; i < file.length(); i++) {

char ch = (char)(fis.read()); //循环读取字符

System.out.print(ch);

}

System.out.println();

fis.close(); //关闭流

} catch (FileNotFoundException fnfe) {

System.out.println("文件打开失败。");

} catch (IOException ioe) {

ioe.printStackTrace();

}

}

}

#### FileOutputStream

FileOutputStream类称为文件输出流，继承于OutputStream类，是进行文件写操作的最基本类；

它的作用是将内存中的数据输出到文件中，我们可以利用它来写文件。

##### FileOutputStream类的构造方法

|  |  |
| --- | --- |
| **构 造 方 法** | **说 明** |
| **FileOutputStream(File file)**  **throws FileNotFoundException** | **使用File对象创建文件输出流对象，如果文件打开失败，将抛出异常** |
| **FileOutputStream(File file, boolean append)**  **throws FileNotFoundException** | **使用File对象创建文件输出流对象，并由参数append指定是否追加文件内容，true为追加，false为不追加，异常情况同上** |
| **FileOutputStream(String name)**  **throws FileNotFoundException** | **直接使用文件名或路径创建文件输出流对象，异常情况同上** |
| **FileOutputStream(String name, boolean append)**  **throws FileNotFoundException** | **直接使用文件名或路径创建文件输出流对象，并由参数append指定是否追加，异常情况同上** |

FileOutputStream常用方法

|  |  |
| --- | --- |
| **方 法 原 型** | **说 明** |
| **void write(int b)**  **throws IOException** | **往文件中写数据，一次写一个字节，有可能抛异常，必须捕捉** |
| **void write(byte[] b)**  **throws IOException** | **往文件中写数据，将byte数组中的数据全部写入到文件中，有可能抛异常，必须捕捉** |
| **void close()**  **throws IOException** | **关闭流对象，有可能抛异常，必须捕捉** |

EG；

public class FileOutputStreamDemo1

{

//在函数内部不进行异常处理，将异常抛出函数外部

public static void main(String[] args) throws IOException

{

String str = "Hello world!";

File file = new File("test.txt"); //创建文件对象

//通过文件对象创建文件输出流对象

//附加第二个参数true，指定进行文件追加，默认为不追加

FileOutputStream fos = new FileOutputStream(file, true);

//逐个将字符写入到文件中

for (int i = 0; i < str.length(); i++)

{

fos.write(str.charAt(i));

}

fos.close(); //关闭流

}

}

Eg2

public class FileOutputStreamDemo2

{

//在函数内部不进行异常处理，将异常抛出函数外部

public static void main(String[] args) throws Exception

{

String str = "I Love Java";

//通过文件名创建文件输出流对象

FileOutputStream fos = new FileOutputStream("test.txt");

//将字符串转化为字节数组

byte[] buffer = str.getBytes();

//将字节数组中包含的数据一次性写入到文件中

fos.write(buffer);

//关闭流

fos.close();

}

}

#### 小结

FileInputStream类和FileOutputStream类是成对出现的，一个进行输入（读文件）操作，一个进行输出（写文件）操作；

由于采用字节方式进行数据传输，不必考虑数据的格式问题，这两个类对文件操作的效率较高；

可以使用这两个类完成复制文件的操作

public class CopyFileDemo {

public static void main(String[] args) throws IOException {

File srcFile = new File("src.dat"); //源文件对象

File destFile = new File("dest.dat"); //目标文件对象

if (!(destFile.exists())) { //判断目标文件是否存在

destFile.createNewFile(); //如果不存在则创建新文件

}

//使用源文件对象创建文件输入流对象

FileInputStream fis = new FileInputStream(srcFile);

//使用目标文件对象创建文件输出流对象

FileOutputStream fos = new FileOutputStream(destFile);

byte[] buf = new byte[1024]; //创建字节数组，作为临时缓冲

System.out.println("开始复制文件...");

while (fis.read(buf) != -1) { //循环从文件输入流中读取数据

fos.write(buf); //写入到文件输出流中

}

System.out.println("文件复制成功！");

fis.close(); //关闭流

fos.close();

}

}

## 总结

java.io包是Java内置的包，其中包含一系列对输入/输出进行操作的类；

File类的对象可以访问文件（或目录）的属性，但不可以进行读/写操作；

从方向上讲，流分为输入流和输出流，但从格式上区分的话，流分为字节流和字符流；

使用FileInputStream类和FileOutputStream类以字节流方式对文件进行读/写操作，这是操作文件最基本的两个类；