章节41-章节50

章节41 课时 159 04015_StringBuffer类

StringBuffer类的主要特点：

StringBuffer、StringBuilder、String的区别。

具体内容

回顾String类的特点：

String类对象有两种实例化方式：

直接赋值：只开辟一块堆内存空间，可以自动入池；

构造方法：开辟两块堆内存空间，不会自动入池，使用intern()手工入池；

任何一个字符串都是String类的匿名对象；

字符串一旦声明则不可改变，可以改变的只是String类对象的引用。

虽然在所有的项目里面，String类都一定要使用，可是String类有一个问题不得不进行重复，那么就是String类的内容不可改变。

为此在Java里面提供有另外一个类——StringBuffer类（里面的内容可以修改）。

String类的对象可以使用“+”进行字符串的连接操作，但是在StringBuffer里面必须使用append()方法进行追加：

方法：public StringBuffer append(数据类型 变量)

范例：观察StringBuffer基本使用

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        //String类可以直接赋值实例化，但是StringBuffer类不行。
        StringBuffer buf=new StringBuffer();
        buf.append("Hello").append(" World").append("!!!");
        change(buf);
        System.out.println(buf);
    }
    public static void change(StringBuffer temp){
        temp.append("\n").append("Hello MLDN.");
    }
}
```

String类	StringBuffer类
public final class String extends Object implements Serializable, Comparable<String>, CharSequence	public final class StringBuffer extends Object implements Serializable, CharSequence

发现String与StringBuffer都是CharSequence接口的子类。而在以后的开发之中，如果你看见某些方法的操作上出现的是CharSequence接口，那么应该立刻清楚，只需要传递字符串即可。

虽然String与StringBuffer类有着共同的接口，但是这两个类对象之间如果要转换不能够直接转换：

1、将String变为StringBuffer类对象有几种做法：

方式一：利用StringBuffer类的构造方法（public StringBuffer(String str)）

方式二：利用append()方法接收字符串public StringBuffer append(String str)

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        StringBuffer buf=new StringBuffer();
        buf.append("Hello");//将String变为StringBuffer
        System.out.println(buf);//String 类覆写的toString类
    }
}
```

2、将StringBuffer类变为String

利用toString()方法可以将StringBuffer转换为String

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        StringBuffer buf=new StringBuffer("Hello");
        String str=buf.toString();//将String变为StringBuffer
        System.out.println(str);//String 类覆写的toString类
    }
}
```

也可以利用String类的构造方法public String(StringBuffer buffer)实现StringBuffer与String的转换，在String类里面也提供有一个和StringBuffer比较的方法：

public boolean contentEquals(StringBuffer sb)

```
package cn.mldn.demo;
```

```
public class TestDemo{
   public static void main(String[] args) throws Exception {
      StringBuffer buf=new StringBuffer("Hello");
      System.out.println("hello".contentEquals(buf));
  }
}
```
String类中定义了许多的方法便于用户的开发，而在StringBuffer类里面也定义了许多的操作方法，而且部分方法与String类是正好互补的。
1、字符串反转public StringBuffer reverse()
```
package cn.mldn.demo;
public class TestDemo{
   public static void main(String[] args) throws Exception {
      StringBuffer buf=new StringBuffer("Hello");
      System.out.println(buf.reverse());
  }
}
```
2、在指定的索引位置增加数据：public StringBuffer insert(int offset,数据类型 变量)
```
package cn.mldn.demo;
public class TestDemo{
   public static void main(String[] args) throws Exception {
      StringBuffer buf=new StringBuffer("Hello");
      buf.insert(0, "MlDN ").insert(0, "你好");
      System.out.println(buf);
  }
}
```
3、删除部分数据：public StringBuffer delete(int start,int end)
```
package cn.mldn.demo;
public class TestDemo{
   public static void main(String[] args) throws Exception {
      StringBuffer buf=new StringBuffer("Hello World MLDN");
      System.out.println(buf.delete(5, 11));
  }
}
```
从JDK1.5之后增加了一个新的字符串操作类：StringBuilder类，其定义结构如下：public final class StringBuilder
extends Object
implements Serializable, CharSequence
发现StringBuffer类与StringBuilder类在定义上非常的相似；几乎连方法也一样。
面试题：请解释String、StringBuffer、StringBuilder的区别？
String的内容一旦声明则不可改变，而StringBuffer和StringBuilder声明的内容可以改变。
StringBuffer提供的方法都是同步方法，属于安全的线程操作，而StringBuilder类中的方法都属于异步方法，属于非线程安全的操作。
日后再开发之中，如果见到了字符串的应用，不需要思考95%使用的都是String类，只有字符串反复改变的时候会考虑使用StringBuffer或者StringBuilder类操作。
总结：
String类依然是最常用的字符串描述类，而StringBuffer（同步更安全）类由于出现时间较长，所以使用要比StringBuilder（异步更快）多。

章节41 课时 160 04016_Runtime类
Runtime类的主要作用
Runtime类的定义形式
具体内容：
在每一个JVM进程里面都会存在有一个Runtime类的对象，这个类的主要功能是取得一些与运行时有关的环境属性，或者创建新的进程等操作。
在Runtime类定义的时候它的构造方法已经被私有化了，这就属于**单例设计模式**的应用，因为要保证在整个进程里面只有唯一的一个Runtime类的对象。私有化必须定义一个static型的方法用于取得本类对象，所以在Runtime类里面提供有一个static型的方法，这个方法可以取得Runtime类的实例化对象：
public static Runtime getRuntime()
Runtime类是直接与本地运行有关的所有相关属性的集合，所以在Runtime类中定义了如下的方法：
public long totalMemory()//long类型一般用于日期、时间、内存大小。
返回所有可用内存空间：public long totalMemory()
返回最大可用内存空间：public long maxMemory()
返回空余内存空间：public long freeMermory()
范例：观察内存大小
```
package cn.mldn.demo;
public class TestDemo{
   public static void main(String[] args) throws Exception {
      Runtime run=Runtime.getRuntime();//取得Runtime类的实例化对象
      System.out.println("1、MAX="+run.maxMemory());
      System.out.println("1、TOTLE="+run.totalMemory());
```

```
        System.out.println("1、MAX="+run.freeMemory());
        String str="";
        for(int x=0;x<2000;x++){
            str +=x;//产生大量垃圾
        }
        System.out.println("2、MAX="+run.maxMemory());
        System.out.println("2、TOTLE="+run.totalMemory());
        System.out.println("2、MAX="+run.freeMemory());
        run.gc();//释放垃圾空间
        System.out.println("3、MAX="+run.maxMemory());
        System.out.println("3、TOTLE="+run.totalMemory());
        System.out.println("3、MAX="+run.freeMemory());
    }
}
```

如果一旦产生了过多的垃圾之后，那么就会改变可用的内存空间的大小。可是在Runtime里面有一个方法：public void gc()可以释放掉垃圾空间。

面试题：请解释什么叫GC？如何处理

GC（Garbage Collector）垃圾收集器，指的是释放无用的内存空间；

GC会由系统不定期进行自动的回收，或者调用Runtime类中的gc()方法手工回收。

实际上Runtime类还有一个更加有意思的功能，就是它可以调用本机的可执行程序，并且创建进程。

执行程序：

public Process exec(String command)throws IOException

范例：执行进程

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Runtime run=Runtime.getRuntime();//取得Runtime实例化对象
        Process pro=run.exec("mspaint.exe");//调用本机可执行程序
        Thread.sleep(2000);
        pro.destroy();//销毁进程
    }
}
```

这样的操作一般意义不大，作为娱乐就行。

总结：

1、Runtime类使用了单例设计模式，必须通过内部的getRuntime（）方法才可以取得Runtime类对象

2、Runtime类提供了gc()方法可以用于手工释放内存。

章节41 课时 161 04017_System类

知识点：

1、如何计算某个代码的执行时间

2、进行垃圾手机操作

具体内容了解：

之前一直使用的System.out.println()就属于System类的操作功能，只不过这个功能由于牵扯到IO部分，所以留到以后继续讲解。

在System类里面之前也使用过一个System.arraycopy()方法实现数组拷贝，而这个方法的真实定义如下：

数组拷贝：public static void arraycopy(Object src,int srcPos,Object dest,int destPos,int length)，在System类里面定义由一个重要的方法：

取得当前的系统时间：public static long currentTimeMillis()

范例：请统计出某项操作的执行时间

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        long start=System.currentTimeMillis();
        String str="";
        for(int x=0;x<3000;x++){
            str+=x;
        }
        long end=System.currentTimeMillis();
        System.out.println("本次执行所花费的时间："+(end-start));
    }
}
```

如果要想统计出所花费的毫秒时间，就用long型数据直接进行数学计算后得来。在System类里面定义了一个操作方法：public static void gc()，这个gc方法并不是一个新定义的方法，而是间接调用了Runtime类中的gc()方法。

对象产生一定会调用构造方法，可以进行一些处理操作，但是某一个对象如果要被回收了，那么连一个收尾的机会都没有。那么就可以考虑些Object类中的finalize()方法完成。

finalize()方法：protected void finalize()throws Throwable

在对象回收时，就算抛出了任何的异常，也不会影响到整个程序的正常执行。

```
package cn.mldn.demo;
```

```
class Member{
    public Member(){
        System.out.println("噼里啪啦，祸害出生了。");
    }
    @Override
    protected void finalize() throws Throwable {//本包和不同包的子类访问
        System.out.println("祸害死了，全世界欢庆！");
        throw new Exception("老子十八年后继续祸害全世界，于是自己死了！");
    }
}
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Member mem=new Member();//会出现一些出生时的辅助操作
        mem=null;//会产生垃圾
        System.gc();//手工处理垃圾收集
    }
}
```

构造方法是留给对象初始化时使用的，而finalize（）方法是留给对象回收前使用的。

面试题：请解释final、finally、finalize的区别？

final：定义不能被继承的类、不能被覆写的方法、常量；

finally：关键字，异常的统一出口；

finalize：方法，Object类提供的方法(protected void finalize()throws Throwable），指的是对象回收前的收尾方法，即使出现了异常也不会导致程序中断执行。

总结：

1、System类可以使用currentTimeMillies()方法取得当前的系统时间；

2、System类中的gc()方法就直接调用了“Runtime.getRuntime().gc()”方法。

章节41 课时 162 04018_对象克隆（了解）

本次预计讲解的知识点：

1、清除对象克隆的操作结构；

2、巩固接口的作用

具体内容；

对象克隆指的就是对象的赋值操作，在Object类里面提供有一个专门的克隆方法。

对象克隆：protected Object clone()throws CloneNotSupportedException此方法上抛出一个“CloneNotSupportedException”异常。如果要实现对象克隆的类没有实现Cloneable接口（接口由抽象方法和全局常量来组成），那么就会抛出此异常。但是Cloneable接口看不见方法，此为标识接口，表示一种操作能力。

范例：实现克隆操作

```
package cn.mldn.demo;
class Book implements Cloneable{//此类的对象可以被克隆
    private String title;
    private double price;
    public Book(String title,double price){
        this.title=title;
        this.price=price;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String toString() {
        return "书名："+this.title+",价格："+this.price;
    }
    //由于此类需要对象克隆操作，所以才需要进行方法的覆写
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();//调用父类的克隆方法
    }

}
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Book bookA=new Book("Java开发",79.8);
        Book bookB=(Book)bookA.clone();
        bookB.setTitle("jsp开发");
        System.out.println(bookA);
        System.out.println(bookB);
    }
}
```

对象克隆的操作，理论价值大于实际价值，因为在实际的工作里面很少会用到对象克隆的操作。重点在于标识接口上，以后依然会见到没有方法的接口，这样的接口就好比通行证一样，表示的是能力。

总结
标识接口：没有任何方法定义，只是一个空接口的声明。

章节42 课时 163 04019_数字操作类（Math类）

本次预计讲解的知识点
本次主要讲解的是数学操作类的使用
1、Math类；
2、Random类
3、大数字操作类
具体类容：
1、Math(类)了解
Math就是一个专门进行数学计算的操作类，里面提供了一系列的数学计算方法。
Eclipse中写上类名称，按ctrl+shift+o是用来导入包的
在Math类里面提供的一切方法都是static型的方法，因为Math类里面没有普通属性。
在这个Math类里面实际上只有一个方法能够引起我们的注意：
四舍五入：public static long round(double a)
范例：观察四舍五入

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        System.out.println(Math.round(15.5));//16
        System.out.println(Math.round(-15.5));//-15
        System.out.println(Math.round(-15.51));//-16
    }
}
```

如果进行负数四舍五入的时候，操作的数据小数位大于0.5才仅为，小于等于0.5不进位。

章节42 课时 164 04020_数字操作类（Random类）
Random类：java.util包下的
这个类的主要功能是取得随机数的操作类。
范例：产生10个不大于100的正整数（0~99）

```
package cn.mldn.demo;

import java.util.Random;

public class TestDemo{
    public static void main(String[] args) throws Exception {
        Random rand=new Random();
        for(int x=0;x<10;x++){
            System.out.print(rand.nextInt(100)+"、");
        }
    }
}
```

既然Random可以产生随机数，下面就希望利用其来实现一个36选7的功能。
最大值到36，所以来讲设置边界的数值就是37了，并且里面不能够有0或者是重复的数据。

```
package cn.mldn.demo;

import java.util.Random;

public class TestDemo{
    public static void main(String[] args) throws Exception {
        Random rand=new Random();
        int data []=new int [7];//开辟一个7个元素的数组
        int foot=0;//数组操作的脚标
        while(foot<7){//不知道多少次循环可以保存完数据，所以使用while循环
            int t=rand.nextInt(37);//生成一个不大于37的随机数
            if(!isRepeat(data,t)){//重复
                data[foot++]=t;//j保存数据
            }
        }
        java.util.Arrays.sort(data);
        for(int x=0;x<data.length;x++){
            System.out.print(data[x]+"、");
        }
    }
        /**
```

```
        *此方法主要是判断是否存在有重复的内容,但是不允许保存0
        *@param temp指的是已经保存的数据
        *@param num是新生成的数据
        *@return 如果存在，那么返回true，否则返回false
        */
    public static boolean isRepeat(int temp[],int num){
        if(num==0){//没有必要判断了
            return true;//
        }
        for(int x=0;x<temp.length;x++){
            if(temp[x]==num){
                return true;
            }
        }
        return false;
    }
}
```

章节42 课时 165 04021_数字操作类（BigInteger）

大整数操作类：BigInteger

如果说现在要操作的数据值很大，那么首先想到的应该是double，那么如果说现在计算的结果超过了double

```
package cn.mldn.demo;

public class TestDemo{
    public static void main(String[] args) throws Exception {
        System.out.println(    Double.MAX_VALUE*Double.MAX_VALUE);
    }
}
```

现在发现此时的计算结果并不存在，因为已经超过了double的范畴。

面试题：请问当前假设有两个很大的数字要进行数学计算*（超过了double范围），你该怎么做？

如果真的超过了double的范围，那么肯定无法使用double进行保存，只有String才可以准确的保存好这个数据，如果真的数据很大的数字要进行数学计算，只能够将其变为String型，而后按位取出每一个字符保存的数据，进行手工的计算。

所以在java里面考虑到了此类情况，专门提供了大数字的操作类，其中就有BigInteger、BigDecimal两种。

BigInteger类的构造方法：public BigInteger(String val)，它接收的是String型。

```
package cn.mldn.demo;
import java.math.BigInteger;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        BigInteger bigA=new BigInteger("234123941932409181");
        BigInteger bigB=new BigInteger("234123941932");
        System.out.println("加法操作："+bigA.add(bigB));
        System.out.println("减法操作："+bigA.subtract(bigB));
        System.out.println("乘法操作："+bigA.multiply(bigB));
        System.out.println("除法操作："+bigA.divide(bigB));
        BigInteger result[]=bigA.divideAndRemainder(bigB);//result[]数组里面只有两个值，一个保存商，一个保存余数。
        System.out.println("商："+result[0]+"，余数："+result[1]);
    }
}
```

在Java里面虽然提供了大数字的操作类，但是很多的时候，我们项目开发可能对于数字要求会更加敏感，而这个时候Java本身所提供的数字类是帮助不大的。

章节42 课时 166 04022_数字操作类（BigDecimal）（重点）

大浮点数：BigDecimal

BigInteger不能够保存小数，而BigDecimal可以保存小数数据。在BigDecimal类里面提供有如下的构造：

构造一：public BigDecimal(String val)

构造二：public BigDecimal(double val)

与BigInteger一样，BigDecimal本身也支持基础的数学计算。可是使用BigDecimal还有一个非常重要的目的，就是可以利用它来实现准确的四舍五入操作。

之前使用过Math.round()实现过四舍五入操作，但是这种操作有一个问题，所有的小数位都四舍五入了。假设有一家公司的年收入按照亿进行计算，今年收入：3.45678，按照Math.round()的做法，相当于只有3亿。

遗憾的是BigDecimal类里面没有直接提供有四舍五入的操作支持，可以可以利用我们的除法操作实现。

除法操作：public BigDecimal divide(BigDecimal divisor,int scale,int roundingMode)

BigDecimal divisor：被除数

int scale：保留的小数位

int roundingMode：进位模式(public static final int ROUND_HALF_UP)

范例：实现准确的四舍五入
```
package cn.mldn.demo;
import java.math.BigDecimal;
import java.math.BigInteger;
class MyMath{
    /**
     * 实现准确位数的四舍五入操作
     * @param num 要进行四舍五入操作的数字
     * @param scale 要保留的小数位
     * @return 处理后的四舍五入数据
     */
    public static double round(double num,int scale){
        BigDecimal bigA=new BigDecimal(num);
        BigDecimal bigB=new BigDecimal(1);
        return bigA.divide(bigB,scale,BigDecimal.ROUND_HALF_UP).doubleValue();
    }
}
public class TestDemo{
    public static void main(String[] args) throws Exception {
        System.out.println(MyMath.round(19.123342532, 2));
        System.out.println(MyMath.round(-15.5, 0));
        System.out.println(MyMath.round(15.5, 0));
    }
}
```
此类操作的功能在日后的开发之中一定要会使用，属于工具类的支持范畴。
总结
1、Math类重点要清楚round()方法的坑；
2、Random类生成随机数
3、如果数据量大就使用BigInteger或BigDecimal，这两个类是Number的子类。

章节43 课时 167 04023_日期处理类（Date）
本次预计讲解的知识点：
1、Date类的使用；
2、Calendar类的使用；
3、SimpleDateFormat类的使用；
具体内容：
3.1Date类
在之前一直在编写简单java类，但是所编写的数据表与简单Java类的转换里面缺少了Date数据类型，所以本部分就属于简单Java类的最后一块重要的拼版。
在Java里面提供有一个java.util.Date的类，它直接就表示当前的日期时间。
范例：取得当前的日期时间
```
package cn.mldn.demo;

import java.util.Date;

public class TestDemo{
    public static void main(String[] args) throws Exception {
        Date date=new Date();
        System.out.println(date);
    }
}
```

这个时候的确是输出了当前的日期时间，只不过格式实在是难看。
一直以来强调过一个概念：long可以描述出日期时间数据，那么这一点在Date类的方法上也是可以看见的；在Date类里面定义了如下的几个重要的方法：
无参构造：public Date()；
有参构造：public Date(long date)，接收long型数据；
转换为long型：public long getTime()。
范例：Date与long的转换
```
package cn.mldn.demo;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        long cur=System.currentTimeMillis();//取得当前的日期时间以long型返回
        Date date=new Date(cur);
        System.out.println(date);
        System.out.println(date.getTime());//Date转换为long
    }
```

```
}
```

以后的代码编写过程之中，依然需要以上的转换操作，尤其是getTime()方法。

章节43 课时 168 04024_日期处理类（SimpleDateFormat）

日期格式化：SimpleDateFormat（核心）

java.text是一个专门实现国际化程序的开发包，而SimpleDateFormat是一个专门处理格式化日期的工具类：即可以将Date型的对象转换为String型的形式显示。而主要使用的是以下方法：

构造方法：public SimpleDateFormat(String pattern)，需要传递转换格式；

将Date转换为String：public final String format(Date date)

将String转换为Date：public Date parse(String source)throws ParseException

throws ParseException

现在的关键就在于转换格式上，对于常见的转换单位：年（yyyy）、月（MM）、日（dd）、时（HH）、分（mm）、秒（ss）、毫秒（SSS）。

范例：将日期格式化显示（Date型数据变为了String型数据）

```
package cn.mldn.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Date date=new Date();
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
        String str=sdf.format(date);//将Date型变为了String型
        System.out.println(str);
    }
}
```

除了可以将日期变为字符串之外，也可以将字符串转换为日期。

范例：将字符串转换为日期

```
package cn.mldn.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="2010-10-1 12:12:12.121";//将Date型变为了String型
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
        Date date=sdf.parse(str);//将字符串变为日期型数据
        System.out.println(date);
    }
}
```

在将字符串变为日期型数据的时候，如果日期型数据给出的月不对，那么会自动进行进位。如果格式与给出的转换格式不符合，则会出异常。

总结：关于数据类型的转换

在数据表的操作里面重点说过了有一个常用类型：VARCHAR2(String)、CLOB(String)、Number(Double、int)、Date(java.util.Date)。

Date与String类之间的转换依靠的是SimpleDateFormat；

String与基本类型之间的转换依靠的是包装类与String.valueOf()方法；

long与Date转换依靠的是Date类提供的构造以及getTime()方法。

章节43 课时 169 04025_日期处理类（Calendar）

Date类和SimpleDate类两个往往是一起使用的，但是Calendar这个类主要是进行一些简单的日期计算的。

Calendar类定义：

```
public abstract class Calendar
extends Object
implements Serializable, Cloneable, Comparable<Calendar>
```

这是一个抽象类，那么应该依靠子类进行对象的实例化操作。但是在这个类里面它提供有一个static方法，此方法返回的正式本类对象。

public static Calendar getInstance()

范例：取得当前的日期时间

```
package cn.mldn.demo;
import java.util.Calendar;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Calendar cal=Calendar.getInstance();//取得本类对象
        StringBuffer buf=new StringBuffer();
        buf.append(cal.get(Calendar.YEAR)).append("-");
        buf.append(cal.get(Calendar.MONTH)+1).append("-");
        buf.append(cal.get(Calendar.DAY_OF_MONTH)).append(" ");
        buf.append(cal.get(Calendar.HOUR_OF_DAY)).append(":");
        buf.append(cal.get(Calendar.MINUTE)).append(":");
```

```
        buf.append(cal.get(Calendar.SECOND));
        System.out.println(buf);
    }
}
```
但是这个类在取得的时候可以进行一些简单的计算，例如：若干天之后的日期。
```
package cn.mldn.demo;
import java.util.Calendar;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Calendar cal=Calendar.getInstance();//取得本类对象
        StringBuffer buf=new StringBuffer();
        buf.append(cal.get(Calendar.YEAR)).append("-");
        buf.append(cal.get(Calendar.MONTH)+1).append("-");
        buf.append(cal.get(Calendar.DAY_OF_MONTH)+5).append(" ");
        buf.append(cal.get(Calendar.HOUR_OF_DAY)).append(":");
        buf.append(cal.get(Calendar.MINUTE)).append(":");
        buf.append(cal.get(Calendar.SECOND));
        System.out.println(buf);
    }
}
```
如果是日期计算，要比Date容易，如果使用Date进行天的计算，那么就需要使用long完成了。
总结：
以后数据库中的日期型就使用java.util.Date表示；
2、代码模型：SimpleDateFormat类实现String与Date间的互相转换。

章节44 课时 170 04026_比较器（Arrays类）

预计讲解的知识点：
1、重新认识一下Arrays类：
2、两种比较器的使用；
3、数据结构——二叉树（Binary Tree）
具体内容
Arrays类（了解）
在之前一直使用的“java.util.Arrays.sort()”可以实现数组的排序，而Arrays类就是java.util保重提供的一个工具类，这个工具类主要是完成所有与数组有关的操作功能。
在这个类里面存在有二分查找法：
public static int binarySearch(数据类型[] a,数据类型 key)
二分查找有一个前提，那么就是数组必须是排序后的内容。
范例：实现二分查找
```
package cn.mldn.demo;
import java.util.Arrays;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        int data[]=new int[]{1,5,6,2,3,4,9,8,7,10};
        java.util.Arrays.sort(data);
        System.out.println(Arrays.binarySearch(data, 9));
    }
}
```
//先排序，9排在第9个，脚标从0开始，9的脚标就是8.其实只要数字大于0就证明要找的数字存在。
Arrays类提供了数组比较：public **static** boolean equals(数据类型[] a,数据类型[] a2)，与Object类的equals()屁关系都没有。

要想判断数组是否相同，需要顺序完全一致。
```
package cn.mldn.demo;
import java.util.Arrays;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        int dataA[]=new int[]{1,2,3};
        int dataB[]=new int[]{2,1,3};
        System.out.println(Arrays.equals(dataA,dataB));
    }
}
```
其他操作：
填充数组：public static void fill(数据类型[] a, 数据类型 val)//以一个指定的内容将数组填满
将数组变为字符串输出：public static String toString(数据类型[] a)
```
package cn.mldn.demo;
import java.util.Arrays;
public class TestDemo{
```

```
    public static void main(String[] args) throws Exception {
        int data[]=new int[10];
        Arrays.fill(data,3);
        System.out.println(Arrays.toString(data));
    }
}
```

以上实际上就属于数组的基本操作，只不过这样的操作在实际的开发里很少出现。

章节44 课时 171 04027_比较器（Comparable接口）（核心）

```
package cn.mldn.demo;
import java.util.Arrays;
class Book{
    private String title;
    private double price;
    public Book(String title,double price){
        this.title=title;
        this.price=price;
    }
    @Override
    public String toString() {
        return "书名："+this.title+"，价格"+this.price;
    }
}
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Book books[]=new Book[]{
                new Book("Java开发",89.9),
                new Book("Android开发",99.9),
                new Book("JSP开发",69.9),
                new Book("Oracle开发",79.9)
        };
        Arrays.sort(books);//对象数组排序
        System.out.println(Arrays.toString(books));

    }
}
```

```
Exception in thread "main" java.lang.ClassCastException: cn.mldn.demo.Book cannot be cast to java.lang.Comparable
    at java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:320)
    at java.util.ComparableTimSort.sort(ComparableTimSort.java:188)
    at java.util.Arrays.sort(Arrays.java:1246)
    at cn.mldn.demo.TestDemo.main(TestDemo.java:25)
```

造成此类异常只有一个原因，两个没有关系的对象发生了强制性的转换。每一个对象实际上只保留有地址信息，地址里面是有内容的，所以如果是普通的int型数组要进行比较，只要判断大小就够了。但是如果是对象数组，里面包含的如果只是编码（地址）比较是没有意义的，就拿上面的代码来讲，应该按照价格排序才是有意义的，所以此处必须要明确的设置出比较规则：

比较的规则就是由Comparable接口定义的，此接口定义如下：

```
public interface Comparable<T>{
  public int compareTo();
}
```

实际上String类就是Comparable接口的子类，之前使用的comparaTo()方法就是比较的操作功能，如果用户要针对于对象进行比较，建议compareTo()方法返回三类数据：1（大于）、0（等于）、-1（小于）。

范例：使用比较器

```
package cn.mldn.demo;
import java.util.Arrays;
class Book implements Comparable<Book>{//实现比较
    private String title;
    private double price;
    public Book(String title,double price){
        this.title=title;
        this.price=price;
    }
    @Override
    public String toString() {
        return "书名："+this.title+"，价格"+this.price+"\n";
    }
    @Override
```

```
    public int compareTo(Book o) {//Arrays.sort会自动调用
        if(this.price>o.price){
            return 1;
        }else if(this.price<o.price){
            return -1;
        }else{
            return 0;
        }
    }
}
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Book books[]=new Book[]{
                new Book("Java开发",89.9),
                new Book("Android开发",99.9),
                new Book("JSP开发",69.9),
                new Book("Oracle开发",79.9)
        };
        Arrays.sort(books);//对象数组排序
        System.out.println(Arrays.toString(books));

    }
}
```

compareTo()方法有Arrays.sort()自动进行调用。

总结：以后不管何种情况下，只要是一组对象要排序，对象所在的类一定要实现Comparable接口

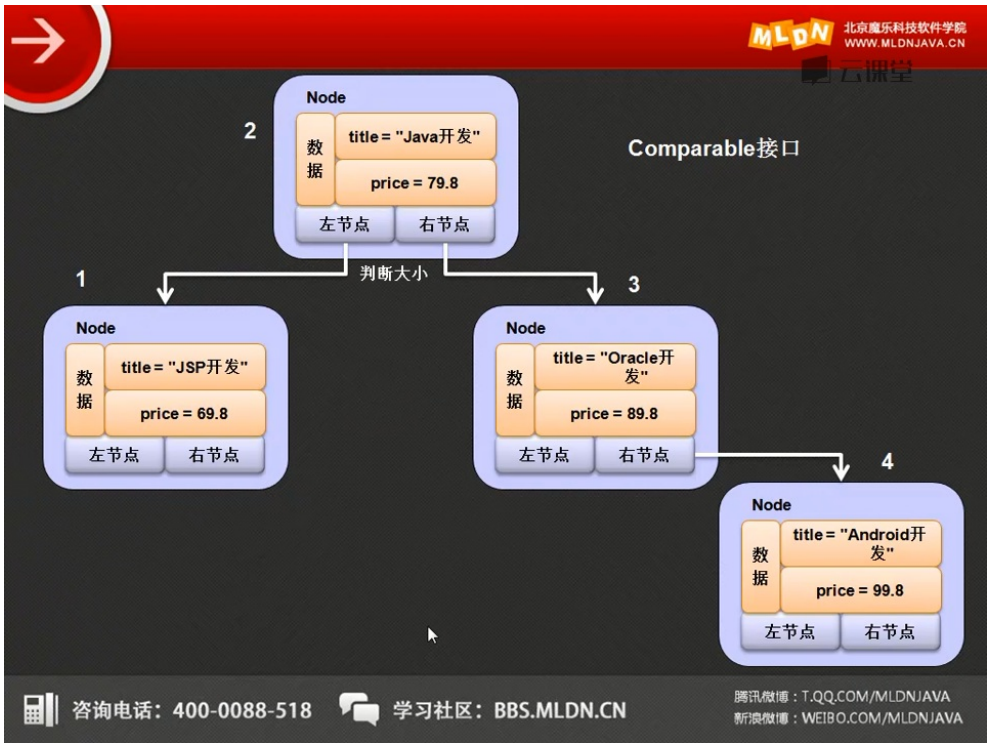
章节44 课时 172 04028_比较器（二叉树实现）

数据结构——BinaryTree（了解）

树是一种比链表更为复杂的概念应用，本质也属于动态对象数组，但是与链表不同在于，树的最大特种是可以针对于数据进行排序。

树的操作原理：选择第一个数据作为根节点，而后比根节点晓得放在根节点的左子树（左节点），比根节点大的放在有字数（右节点），取得的时候按照中序遍历的方式取出（左-中-右）。

在任何数据结构里面Node类的核心功能是保存真实数据以及配置节点关系。



范例：实现二叉树

定义出要使用的数据，数据所在的类要实现Comparable接口。class Book implements Comparable<Book>{//实现比较

```
    private String title;
    private double price;
    public Book(String title,double price){
        this.title=title;
        this.price=price;
    }
    @Override
```

```
    public String toString() {
        return "书名："+this.title+"，价格"+this.price+"\n";
    }
    @Override
    public int compareTo(Book o) {//Arrays.sort会自动调用
        if(this.price>o.price){
            return 1;
        }else if(this.price<o.price){
            return -1;
        }else{
            return 0;
        }
    }
}
```

定义二叉树，所有的数据结构都需要有Node类的支持。

```
package cn.mldn.demo;

import java.util.Arrays;

import javax.xml.crypto.Data;

class Book implements Comparable<Book>{//实现比较
    private String title;
    private double price;
    public Book(String title,double price){
        this.title=title;
        this.price=price;
    }
    @Override
    public String toString() {
        return "书名："+this.title+"，价格"+this.price+"\n";
    }
    @Override
    public int compareTo(Book o) {//Arrays.sort会自动调用
        if(this.price>o.price){
            return 1;
        }else if(this.price<o.price){
            return -1;
        }else{
            return 0;
        }
    }
}
class BinaryTree{
    private class Node{
        private Comparable data;//排序的依据就是Comparable
        private Node left;//保存左节点
        private Node right;//保存右节点
        public Node(Comparable data) {
            this.data=data;
        }
        public void addNode(Node newNode) {
            if(this.data.compareTo(newNode.data)<0){//如果newNode.data>0，结果将以升序排列
                if(this.left==null){
                    this.left=newNode;
                }else{
                    this.left.addNode(newNode);
                }
            }else {
                if(this.right==null){
                    this.right=newNode;
                }else{
                    this.right.addNode(newNode);
                }
            }
        }
        public void toArrayNode(){
            if(this.left!=null){//表示有左节点
```

```
            this.left.toArrayNode();
        }
        BinaryTree.this.retData[BinaryTree.this.foot++]=this.data;
        if(this.right!=null){
            this.right.toArrayNode();//右子树输出
        }
      }
   }
   private Node root;//定义根节点
   private int count;//保存元素个数
   private Object reData;
   private int foot;
   private Object[] retData;

   public void add(Object obj){//进行数据的追加
      Comparable com=(Comparable) obj;//必须变为Comparable才可以实现Node的保存
      Node newNode=new Node(com);//创建新的节点
      if(this.root==null){//表示现在不存在根节点
         this.root=newNode;//保存根节点
      }else{
         this.root.addNode(newNode);//交给Node类处理
      }
      this.count++;
   }
   public Object [] toArray(){
      if(this.root==null){
         return null;
      }
      this.foot=0;
      this.retData=new Object[this.count];
      this.root.toArrayNode();
      return this.retData;
   }
}
public class TestDemo{
   public static void main(String[] args) throws Exception {
      BinaryTree bt=new BinaryTree();
      bt.add(new Book("Java开发",89.9));
      bt.add(new Book("Android开发",99.9));
      bt.add(new Book("JSP开发",69.9));
      bt.add(new Book("Oracle开发",79.9));
      Object obj[]=bt.toArray();
      System.out.println(Arrays.toString(obj));
  }
}
```

可是这些内容，Java的类库都有了自己的实现。

章节44 课时 173 04029_比较器（Comparator接口）（了解）

挽救的Comparator

Comparable接口的主要特征是在类定义的时候就默认实现好的接口，那么如果说现在有一个类已经开发完善了。

```
class Book{//实现比较
   private String title;
   private double price;
   public Book(){}
   public Book(String title,double price){
      this.title=title;
      this.price=price;
   }
   @Override
   public String toString() {
      return "书名："+this.title+"，价格"+this.price+"\n";
   }
   public void setTitle(String title) {
      this.title = title;
   }
   public void setPrice(double price) {
      this.price = price;
```

```
    }
    public String getTitle() {
        return title;
    }
    public double getPrice() {
        return price;
    }
}
```

但是由于初期的设计并没有安排此类对象数组的排序。而后又突然需要实现对象数组的排序，那么在这个时候在不能修改Book类定义的情况下是不可能使用Comparable接口的，为此，在Java里面为了解决这样的问题，又出现了另外一个比较器：java.util.Comparator，原本Comparator接口下有2个方法：

```
@FunctionalInterface
public interface Comparator<T>{
  public int compare(T o1,T o2);
  public boolean equals(Object obj);
}
```

而真正要实现的只有Compare()方法，需要单独准备出一个类来实现Comparator接口

范例：定义排序的工具类

```
class BookComparator implements Comparator<Book>{
    @Override
    public int compare(Book o1, Book o2) {
        if(o1.getPrice()>o2.getPrice()){
            return 1;
        }else if(o1.getPrice()<o2.getPrice()){
            return -1;
        }else{
            return 0;
        }
    }
}
```

之前使用Comparable接口的时候利用的是Arrays类中的sort()方法，可是现在更换了一个接口之后，那么也可以使用另外一个被重载的sort()方法：

```
public static <T> void sort(T[] a,Comparator<? super T> c)
```

范例：实现排序

```
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Book books[]=new Book[]{
                new Book("Java开发",89.9),
                new Book("Android开发",99.9),
                new Book("JSP开发",69.9),
                new Book("Oracle开发",79.9)
        };
        Arrays.sort(books,new BookComparator());
        System.out.println(Arrays.toString(books));
    }
}
```

使用Comparator比较麻烦，以为要定义一个专门的排序类，而且要用排序的时候也要明确的指明一个排序规则类。

面试题：请解释Comparable和Comparator的区别？（请解释两种比较器的区别）

如果对象数组要进行排序那么必须设置排序规则，可以使用Comparable或Comparator接口实现

java.lang.Comparable是在一个类定义的时候实现好的接口，这样本类的对象数组就可以进行排序，在Comparable接口下定义有一个public int comparaTo()方法；

java.utilComparable是专门定义一个指定类的比较规则，属于挽救的比较操作，里面有两个方法，public int compare()、public boolean equals()

总结：

1、以后不管何种情况下只要是牵扯到对象数组的排序一定使用Comparable接口；

2、根据自己的实际情况来决定是否要熟练编写链表。

57% 20:12 63% 20:22 70% 20:32 79% 20:42 94% 21:24

章节45 课时 174 04030_正则表达式（正则引出）

具体内容

问题引出

为了更好的说明正则的应用，下面要求编写一个程序：判断一个字符串是否由数字所组成。

实现原理：

将字符串变为字符数组；

而后判断每一个字符是否在0~9的范围之间。

```
package cn.mldn.demo;
```

```
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="123";
        System.out.println(isNumber(str));
    }
    public static boolean isNumber(String temp){
        char data[]=temp.toCharArray();//将字符串变为字符数组，目的是循环取出
        for(int x=0;x<data.length;x++){
            if(data[x]>'9'||data[x]<'0'){
                return false;
            }
        }return true;
    }
}
```

此时判断字符串是否由数组所组成，是一个很容易实现的功能，但是就这样一个简短的操作，却写出了8行代码，那么如果现在是更加复杂的操作呢?

范例：更简单的做法

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="123";
        System.out.println(str.matches("\\d+"));
    }
}
```

一个写了很多行的代码，最后只是写了一行简单的操作就实现了，而其中出现的“\\d+”就是政治表达式。

正则是从jdk1.4的时候正式引入到java中的工具类，所有正则支持的类都定义在java.util.regex包里面。在jdk1.4之前，如果要想使用正则，则需要单独下载一个政治表达式的开发包之后才可以使用。

在java.util.regex包里面定义了两个主要的类：

Pattern类：此类对象如果要想取得必须使用compile（ ）方法，方法的功能是编译正则；

Matcger类：通过Pattern类取得。

章节45 课时 175 04031_正则表达式（正则标记）

正则标记**(背、死了都要会)**

所有的正则可以使用的标记都在java.util.regex.Pattern类里面定义。

1、单个字符(数量:1)

字符:表示由一位字符所组成；

\\:表示转义字符"\"移位；

\t:表示一个"\t"符合；

\n:匹配换行(\n)符号；

2、字符集(数量:1)

[abc]:表示字符a或者字符b或者字符c中的任意一位。

[^abc]:表示不是abc中的任意一位。

[a-z]:所有的小写字母。

[a-zA-Z]:表示任意的一位字母，不区分大小写。

[0-9]:表示任意的一位数字；

3、简化的字符集表达式(数量:1)

.:表示任意的一位字符；

\d:表示一位数字，等价于"[0-9]"，属于简化写法，写代码时需要写\\d

\D:等价于"[^0-9]"，属于简化写法；

\s:表示任意的空白字符，例如"\t""\n";

\S:表示任意的非空白字符

\w:等价于"[a-zA-Z_0-9]",表示由任意的字母、数字、_所组成；

\W:等价于"[^a-zA-Z_0-9]",表示不是由任意的字母、数字、_所组成；

4、边界匹配(不要在java中使用，在JavaScript中使用)

^:正则的开始

$:正则的结束

5、数量表达

正则?:表示此正则可以出现0次或1次。

正则+:表示此正则可以出现1次或1次以上。

正则*:表示此正则可以出现0次1次或多次。

正则{n}:表示此正则正好出现n次。例如"\\d{3}"

正则{n,}:表示此正则出现n次以上(包含n次)。例如"\\d{3,}"

正则{n,m}:表示正则出现n~m次。

6、逻辑运算:

正则1正则2:正则1判断完成之后继续判断正则2；

正则|正则2:正则1或者正则2有一组满足即可；

(正则):将多个正则作为一组，可以为这一组单独设置出现的次数。

章节45 课时 176 04032_正则表达式（String类对正则的支持）

String类对正则的支持（重点）
在JDK1.4之后，由于正则的引入，所以在String类里面也相应增加了新的操作方法支持。

No	方法名称	类型	
1	public boolean matches(String regex)	普通	正则验证，使用指定的字符串判断是否符合给定的正则表达式
2	public String replaceAll(String regex, String replacement)	普通	全部替换
3	public String replaceFirst(String regex, String replacement)	普通	替换首个
4	public String[] split(String regex)	普通	全部拆分
5	public String[] split(String regex, int limit)	普通	部分拆分

给出的几个方法里面对于替换和拆分实际上难度不高，最关键就是正则匹配，在验证上使用特别多。
范例：实现字符串的替换。
```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="wukun&*（@JPOIJPIJ）（*PI12341）（*Uo70982340i";
        String regex="[^a-z]";//此处编写正则
        System.out.println(str.replaceAll(regex,""));
    }
}
```
范例：字符串拆分，按照数字拆
```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="123jkj12hk123hljkfy1837ads";
        String regex="[\\d+]";//此处编写正则
        String result[]=str.split(regex);
        for(int x=0;x<result.length;x++){
            System.out.println(result[x]);
        }
    }
}
```
所有正则之中最应该引起我们兴奋的事情是因为可以使用它进行验证。
范例：验证一个字符串是否是数字，如果是则将其变为double型
数字可能是整数（10）也可能是小数（10.2）；
```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="10";
        String regex="\\d+(\\.\\d+)?";
        System.out.println(str.matches(regex));
        if(str.matches(regex)){//转型之前要进行验证
            System.out.println(Double.parseDouble(str));
        }
    }
}
```
范例：判断给定的字符串是否是一个IP地址（IPV4）
IP地址：192.168.1.1 每一个端最大是三个长度（1~3个长度都可以）
```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="192.168.1.1";
        String regex="(\\d{1,3}\\.){3}\\d{1,3}";
        System.out.println(str.matches(regex));
```

```
        if(str.matches(regex)){//转型之前要进行验证
            System.out.println(str);
        }
    }
}
```

范例：给定一个字符串，要求判断其实日期格式，如果是则将其转换为Date型数据。

```
package cn.mldn.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="2016-08-19";
        String regex="\\d{4}-\\d{2}-\\d{2}";
        System.out.println(str.matches(regex));
        if(str.matches(regex)){//转型之前要进行验证
            Date date=new SimpleDateFormat("yyyy-MM-dd").parse(str);
            System.out.println(date);
        }
    }
}
```

范例：判断电话号码，一般要编写电话号码一下几种格式都是满足的：

格式一：51283346，一般长度是7~8位的数字是电话号码；

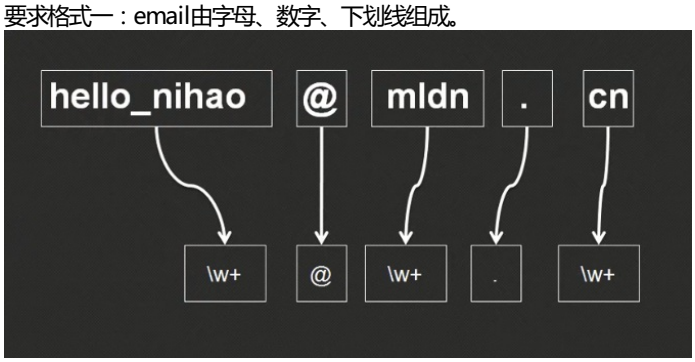
格式二：010-51283346，区号一般是3~4位，而且区号和电话之间的“-”只有在出现区号的时候才出现；

格式三：(010)-51283346

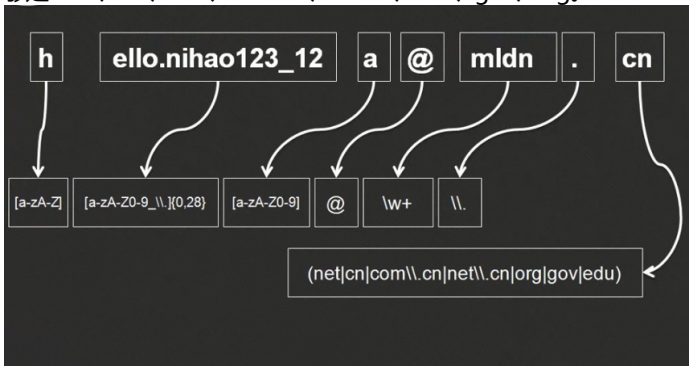
```
package cn.mldn.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="(010)-51283346";
        String regex="((\\d{3,4}-)|(\\(\\d{3,4}\\)-))?\\d{7,8}";
        System.out.println(str.matches(regex));
    }
}
```

范例：验证email地址

要求格式一：email由字母、数字、下划线组成。



要求格式二：用户名要求由字母、数字、_、.组成，其中必须以字母开头，字母和数字和结尾，用户名长度不超过30，最后的根域名只能够是.com、.cn、.net、.com.cn、net.cn、edu、gov、org。




```
package cn.mldn.demo;
import java.text.SimpleDateFormat;
import java.util.Date;
public class TestDemo{
```

```
    public static void main(String[] args) throws Exception {
        String str="1hello_nihao@mldn.net.cn";
        String regex="[a-zA-Z0-9][a-zA-Z0-9_\\.]{0,28}[a-zA-Z0-9]@\\w+\\.(com|cn|net|com\\.cn|net\\.cn|edu|gov|org)";
        System.out.println(str.matches(regex));
    }
}
```

章节45 课时 177 04033_正则表达式（java.util.regex包支持）

在大多数情况下使用正则的时候都会采用String类完成，但是正则最原始的开发包是java.util.regex,这个包里面也提供有两个类。
范例：Pattern类

```
package cn.mldn.demo;
import java.util.Arrays;
import java.util.regex.Pattern;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="a1b22c333d4444e55555";
        String regex="\\d+";
        Pattern pattern=Pattern.compile(regex);//编译正则
        String result[]=pattern.split(str);//拆分字符串
        System.out.println(Arrays.toString(result));
    }
}
```

范例：字符串验证

```
package cn.mldn.demo;
import java.util.Arrays;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        String str="123123123";
        String regex="\\d+";
        Pattern pattern=Pattern.compile(regex);
        Matcher mat=pattern.matcher(str);
        System.out.println(str.matches(regex));//匹配结果
    }
}
```

正式因为String类本身就已经支持这样两种操作了，所以对于String类而言由于所有接收的数据也都是在付出，所以很少会再去利用Pattern与Matcher两个类进行操作了。
总结
1、利用正则实现验证代码可以最少化
2、一定要清楚String类对正则支持的几个方法，以及所有讲解过的相关程序。

章节46 课时 178 04034_反射机制（认识反射）

反射的话先通过“反”来理解，既然有“反”就一定有“正”，在正常情况下，一定是先有类而后再产生对象。

```
package cn.mldn.demo;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Date date=new Date();
        System.out.println(date);
    }
}
```

所谓的“反”就是指可以利用对象找到对象的出处，在Object类里面提供有一个方法：
取得Class对象：public final Class<?> getClass()
范例：观察反

```
package cn.mldn.demo;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Date date=new Date();
        System.out.println(date.getClass());
    }
}
```

发现调用了getClass()方法后的输出就输出了类的完整名称，等于是找到了对象的出处。

章节46 课时 179 04035_反射机制（实例化Class类对象）

java.lang.Class是一个类，这个类是反射操作的源头，即：所有的反射都要从此类开始进行，而最关键的是这个类有三种实例化方式：
第一种：调用Object类中的getClass()方法

```
package cn.mldn.demo;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Date date=new Date();
        Class <?> cls=date.getClass();
        System.out.println(cls);
    }
}
```

第二种：使用“类.class”取得；以后讲解Hibemate、MyBatis、Spring；

```
package cn.mldn.demo;
import java.util.Date;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Class <?> cls=Date.class;
        System.out.println(cls);
    }
}
```

之前是在产生类的实例化对象之后取得Class类对象但是此时并没有实例化对象的产生。
第三种：调用Class类提供的一个方法；
实例化Class类对象public static Class<?> forName(String className)throws ClassNotFoundException

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Class <?> cls=Class.forName("java.util.Date");//相当于起了个类名称为java.util.Date
        System.out.println(cls);
    }
}
```

此时可以不适用import语句导入一个明确的类，而类名称是采用字符串的形式进行描述的。

章节46 课时 180 04036_反射机制（反射实例化对象）
当拿到一个类的时候，肯定要直接使用关键字new进行对象的实例化操作，这属于习惯性的做法，但是如果有了Class类对象，那么就可以做到，利用反射来实现对象实例化操作。
public T newInstance()throws InstantiationException,IllegalAccessException
范例：利用反射实例化对象

```
package cn.mldn.demo;
class Book{
    public Book(){
        System.out.println("*********Book的无参构造方法*********");
    }
    public String toString(){
        return "这是一本书！";
    }
}
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Book b=new Book();
        System.out.println(b);
    }
}
```

```
package cn.mldn.demo;
class Book{
    public Book(){
        System.out.println("*********Book的无参构造方法*********");
    }
    public String toString(){
        return "这是一本书！";
    }
}
```

```
public class TestDemo{
    public static void main(String[] args) throws Exception {
        //Book类的完整名称，此行代码作用取得Class类对象
        Class <?> cls=Class.forName("cn.mldn.demo.Book");
        Object obj=cls.newInstance();
        Book book=(Book) obj;//向下转型
        System.out.println(obj);
        System.out.println(book);
    }
}
```

有了反射之后，以后进行对象实例化的操作不再只是单独的依靠关键字new完成了，反射也同样可以完成，但是这并不表示new就被完全取代了。

在任何的开发之中，new是造成耦合的最大元凶，一切的耦合都起源于new。

范例：观察工厂设计模式

```
package cn.mldn.test;
interface Fruit{
    public void eat();
}
class Apple implements Fruit{
    @Override
    public void eat() {
        System.out.println("*****吃苹果******");
    }
}
class Factory{
    public static Fruit getInstance(String className){
        if("apple".equals(className)){
            return new Apple();
        }
        return null;
    }
}
public class TestFactory {
    public static void main(String[] args) {
        Fruit f=Factory.getInstance("apple");
        f.eat();
    }
}
```

如果此时增加了Fruit接口子类，那么就表示程序要修改工厂类。

```
package cn.mldn.test;
interface Fruit{
    public void eat();
}
class Apple implements Fruit{
    @Override
    public void eat() {
        System.out.println("*****吃苹果******");
    }
}
class Orange implements Fruit{
    @Override
    public void eat() {
        System.out.println("*****吃橘子******");
    }
}
class Factory{
    public static Fruit getInstance(String className){
        if("apple".equals(className)){
            return new Apple();
        }else if("orange".equals(className)){
            return new Orange();
        }
        return null;
    }
}
public class TestFactory {
    public static void main(String[] args) {
```

```
        Fruit f=Factory.getInstance("orange");
        f.eat();
    }
}
```

每增加一个类就要去修改工厂类，那么如果随时都可能增加子类呢?工厂类就要一直被修改，因为现在工程类中的对象都是通过关键字new直接实例化的，而new就成了所有问题的关键点。要想解决这一问题，就只能依靠反射完成。

```
package cn.mldn.test;
interface Fruit{
    public void eat();
}
class Apple implements Fruit{
    @Override
    public void eat() {
        System.out.println("*****吃苹果******");
    }
}
class Orange implements Fruit{
    @Override
    public void eat() {
        System.out.println("*****吃橘子******");
    }
}
class Factory{
    public static Fruit getInstance(String className){
        Fruit f=null;
        try{
            f=(Fruit)Class.forName(className).newInstance();//返回Object变为Fruit
        }catch(Exception e){
        }
        return f;
    }
}
public class TestFactory {
    public static void main(String[] args) {
        Fruit f=Factory.getInstance("cn.mldn.test.Apple");
        f.eat();
    }
}
```

此时的程序就真正完成了解耦合的目的，而且可扩展性非常的强。

章节46 课时 181 04037_反射机制（调用构造方法）

在之前所编写的代码实际上发现了都默认使用了类之中的无参构造方法，可是类中还有可能不提供无参构造呢：

范例：观察当前程序问题

由于此时Book类没有提供无参构造方法，所以以下的代码时错误的。

```
package cn.mldn.demo;
public class TestDemo{
    public static void main(String[] args) throws Exception {
        Class <?> cls=Class.forName("cn.mldn.po.Book");
        Object obj=cls.newInstance();//实例化对象
        System.out.println(obj);
    }
}
Exception in thread "main" java.lang.InstantiationException: cn.mldn.po.Book
    at java.lang.Class.newInstance(Class.java:427)
    at cn.mldn.demo.TestDemo.main(TestDemo.java:5)
Caused by: java.lang.NoSuchMethodException: cn.mldn.po.Book.<init>()//没有合适的方法，没有无参构造
    at java.lang.Class.getConstructor0(Class.java:3082)
    at java.lang.Class.newInstance(Class.java:412)
    ... 1 more
```

以上所出现的错误指的就是因为当前Book类里面并没有无参构造方法，所以程序无法进行对象的实例化。在这种情况下，只能够明确的调用有参构造方法。

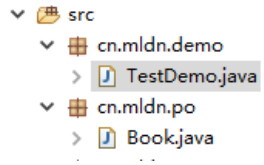
在Class类里面提供有一个方法可以取得构造：

取得全部构造：public T newInstance()throws InstantiationException,IllegalAccessException

取得一个指定参数顺序的构造：public Constructor<T> getConstructor(Class<?>... parameterTypes)throws NoSuchMethodException,SecurityException

以上两个方法返回的都是“java.lang.reflect.Constructor”类的对象。在这个类中提供有一个明确传递有参构造内容的实例化对象的方

法：public T newInstance(Object... initargs)
        throws InstantiationException,
            IllegalAccessException,
            IllegalArgumentException,
            InvocationTargetException
范例：明确调用类中的有参构造，红字部分背诵。




```
package cn.mldn.po;
public class Book{
   private String title;
   private double price;
   public Book(String title,double price){
      this.title=title;
      this.price=price;
   }
   public String toString(){//覆写toString
      return "书名："+this.title+"，价格："+this.price;
   }
}
package cn.mldn.demo;
import java.lang.reflect.Constructor;
public class TestDemo{
   public static void main(String[] args) throws Exception {
      Class <?> cls=Class.forName("cn.mldn.po.Book");
      Constructor <?> con=cls.getConstructor(String.class,double.class);
      Object obj=con.newInstance("java开发",89.8);
      System.out.println(obj);
  }
}
```

避免出现以上麻烦：简单java类的开发之中不管提供有多少个构造方法，请至少保留有无参构造。

章节46 课时 182 04038_反射机制（调用普通方法）

类中的普通方法只有在一个类产生实例化对象之后才可以调用，并且实例化对象的方式有三种。
范例：定义一个类

```
package cn.mldn.po;
public class Book{
   private String title;
   public void setTitle(String title) {
      this.title = title;
   }
   public String getTitle() {
      return title;
   }
}
```

这个类有无参构造方法，所以实例化对象的时候可以直接利用Class类中提供的newInstance()方法完成。
在Class类里面提供有一下取得类中Method的操作：
 取得一个类中的全部方法：public Method[] getMethods()throws SecurityException
 取得指定方法：public Method getMethod(String name, Class<?>... parameterTypes)throws NoSuchMethodException,SecurityException
以上的两个操作返回的是java.lang.reflect.Method类的对象，在这个类里面重点关注一个方法。
 调用方法：public Object invoke(Object obj, Object... args)throws IllegalAccessException,IllegalArgumentException,InvocationTargetException
范例：反射调用方法
类中的普通方法只有在一个类产生实例化对象之后才可以调用，并且实例化对象的方式有三种。
范例：定义一个类

```
package cn.mldn.po;
public class Book{
 private String title;
 public void setTitle(String title) {
  this.title = title;
 }
```

```
 public String getTitle() {
  return title;
 }
}
```
这个类有无参构造方法，所以实例化对象的时候可以直接利用Class类中提供的newInstance()方法完成。
在Class类里面提供有一下取得类中Method的操作：
 取得一个类中的全部方法：public Method[] getMethods()throws SecurityException
 取得指定方法：public Method getMethod(String name, Class<?>... parameterTypes)throws NoSuchMethodException,SecurityException
以上的两个操作返回的是java.lang.reflect.Method类的对象，在这个类里面重点关注一个方法。
```
package cn.mldn.demo;
import java.lang.reflect.Method;
public class TestDemo{
  public static void main(String args[])throws Exception {
     String fieldName="title";
     Class <?> cls=Class.forName("cn.mldn.po.Book");
     Object obj=cls.newInstance();//必须给出实例化对象
     Method setMet=cls.getMethod("set"+initcap(fieldName),String.class);
     Method getMet=cls.getMethod("get"+ initcap(fieldName));
     setMet.invoke(obj, "Java开发");//等价于Book类对象.setTitle("Java开发");
     System.out.println(getMet.invoke(obj));
  }

  private static String initcap(String str) {
    return str.substring(0,1).toUpperCase()+str.substring(1);
  }
}
```

章节46 课时 183 04039_反射机制（调用成员）

类中的属性一定要在本类实例化对象产生之后才可以分配内存空间。在Class类里面提有取得成员的方法：
 取得全部成员：public Field[] getDeclaredFields()throws SecurityException
 取得指定成员：public Field getDeclaredField(String name)throws NoSuchFieldException,SecurityException
返回的类型是java.lang.reflect.Field类，在这个类里面有两个重要的方法
 取得属性内容：public Object get(Object obj)throws IllegalArgumentException,IllegalAccessException
 设置属性内容：public void set(Object obj,Object value)throws IllegalArgumentException,IllegalAccessException
在java.lang.reflect.AccessibleObject类下面（JDK1.8之后修改）：
 Executable：下面继续继承了Constructor、Method；
 Field：在这个类里面提供有一个方法：public void setAccessible(boolean flag)throws SecurityException，设置是否封装；

范例：现在提供有如下的类
```
package cn.mldn.po;
public class Book{
   private String title;
}
```
 这个类里面之定义了一个私有属性，按照原始的做法，此时它一定无法被外部所使用。
范例：反射调用
```
package cn.mldn.demo;

import java.lang.reflect.Field;

public class TestDemo{
  public static void main(String args[])throws Exception {
     String fieldName="title";
     Class <?> cls=Class.forName("cn.mldn.po.Book");
     Object obj=cls.newInstance();//必须给出实例化对象
     Field titleField=cls.getDeclaredField("title");
     titleField.setAccessible(true);//封装取消了
     titleField.set(obj,"java开发");//相当于Book类对象.title="java开发";
     System.out.println(titleField.get(obj));//相当于直接输出Book类对象.title
  }
  private static String initcap(String str) {
    return str.substring(0,1).toUpperCase()+str.substring(1);
  }
}
```
 构造方法与普通方法同样也可以取消封装，只不过很少这样去做，而且对于数学的访问还是建议使用setter和getter方法完成。
1、实例化对象的方式又增加了一种反射；
2、对于简单Java类的定义应该更加清晰了；

3、反射调用类结果只是一个开始。

章节47 课时 184 04040_国际化程序实现

如果说现在有一套程序，可能中国要使用、美国要使用、俄罗斯也要使用。很明显，不管哪里使用，程序的核心功能是不会改变的，唯一改变的就是语言。
如果要想实现语言的统一，那么唯一能够做的方式就是将所有需要显示的语言定义在各自的资源文件里面。
所谓的国际化应用指的就是根据当前的语言环境读取指定的预压资源文件。
如果要想实现国际化的操作，那么首先要解决的问题就是如何读取资源文件的问题。
所谓的资源文件指的是后缀名称为“*.properties”里面保存的内容按照“key=vlaue”的形式保存，而且资源文件的命名标准与Java类完全一样。
范例：定义一个Message.properties
如果保存的是中文信息，那么必须将其变为UNICODE编码
info=中华人民共和国
这里面保存的info是这个信息的key,以后要根据这个key取得对应的value。
如果要读取资源文件的信息使用java.util.ResourceBundle类，这是一个抽象类，但是这个类的内部也提供有一个static方法用于取得本类对象。
根据当前语言环境取出:public static final ResourceBundle getBundle(String baseName)//就是Message
设置指定语言环境：public static final ResourceBundle getBundle(String baseName,Locale locale)
当取得了ResourceBundle类对象之后可以通过以下的方法读取数据：
简单读取：public final String getString(String key)
java.text是专门负责国际化处理的程序包，在这个包里面有一个专门处理站位数据的操作类MessageFormat
格式化文本：public static String format(String pattern,Object... arguments)
范例：读取普通文本

```
package cn.mldn.demo;
import java.util.ResourceBundle;
public class TestDemo{
  public static void main(String args[])throws Exception {
     //访问的时候一定不要加上后缀，因为默认找到的后缀就是“*.properties”
     //此时Message.properties文件一定要放在CLASSPATH路径下
     ResourceBundle rb=ResourceBundle.getBundle("Messages");
     System.out.println(rb.getString("info"));
  }
}
```

很多时候数据是会被改变的，例如：
范例：修改Message.properties文件

```
wel.msg=欢迎{0}光临，现在时间是：{1}!
```

范例：设置读取的可变内容

```
package cn.mldn.demo;
import java.text.MessageFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.ResourceBundle;
public class TestDemo{
  public static void main(String args[])throws Exception {
     //访问的时候一定不要加上后缀，因为默认找到的后缀就是“*.properties”
     //此时Message.properties文件一定要放在CLASSPATH路径下
     ResourceBundle rb=ResourceBundle.getBundle("Messages");
     String str=rb.getString("wel.msg");//具备有占位符的内容
     System.out.println(MessageFormat.format(str, "阿佑",new SimpleDateFormat("yyyy-MM-dd").format(new Date())));
  }
}
```

如果你只是从事应用开发，那么不需要编写以上的代码，你所要编写的只是一个资源文件而已。
国际化程序应该要根据所在的国家不同可以显示不同的内容，可是你只是提供了一个资源文件，那么怎么进行不同语言内容的显示呢：于是这个时候就需要Locale类帮忙了。
Locale保存的是一个国家的区域和语言编码
中国：zh_CN;
美国: en_US;
可以在定义资源文件的时候加上指定的语言编码，例如：
范例：定义中文的资源文件——Messages_zh_CN.properties

```
wel.msg=欢迎{0}光临！
```

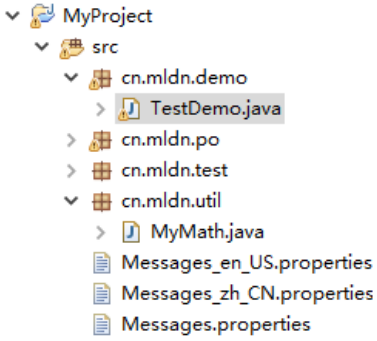
定义英文的资源文件——Messages_en_US.propreties

```
wel.msg=WELCOME{0}!
```

设置的baseName设置的一定是Messages,所有的语言代码又Locale类设置，在java.util.Locale类里面提供有以下的方法：
构造方法：public Locale(String language,String country)
取得当前语言环境：public static Locale getDefault()

范例：读取中文文件

```
package cn.mldn.demo;
import java.text.MessageFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.ResourceBundle;
public class TestDemo{
  public static void main(String args[])throws Exception {
     Locale loc=new Locale("zh","CN");
     ResourceBundle rb=ResourceBundle.getBundle("Messages",loc);
     String str=rb.getString("wel.msg");//具备有占位符的内容
     System.out.println(MessageFormat.format(str, "阿佑"));
     System.out.println(Locale.getDefault());
  }
}
```



范例：读取英文

```
package cn.mldn.demo;
import java.text.MessageFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.ResourceBundle;
public class TestDemo{
  public static void main(String args[])throws Exception {
     Locale loc=new Locale("en","US");
     ResourceBundle rb=ResourceBundle.getBundle("Messages",loc);
     String str=rb.getString("wel.msg");//具备有占位符的内容
     System.out.println(MessageFormat.format(str, "阿佑"));
     System.out.println(Locale.getDefault());
  }
}
```

如果已经存在有特定的语言资源文件，那么就不会读取其他不设置语言的资源文件了。
特定语言的资源文件读取的优先级会高于公共语言资源文件读取的优先级。
总结：
1、资源文件：文件名称每个单词首字母大写，而后后缀必须是*.properties
2、通过ResourceBundle类可以读取在指定的CLASSPATH下的资源文件，读取时不需要输入文件后缀，
 |-动态的占位文本格式化：MessageFormat。
3、Locale类用于指定读取的资源文件的语言环境

章节48 课时 185 04041_文件操作类（基本操作）

在java编程利民，对于所有初学者而言，最麻烦的步伐可能就是Java IO了，因为这里面牵扯到的父类与子类实在是太多了，学习原则：抽象类中定义的抽象方法会根据实例化其子类的不同，也会完成不同的功能。
使用File类进行文件的操作。
如果要进行所有的文件以及文件内容的开发操作，应该使用java.io包完成，而在java.io包里面一共有五个核心类和一个核心接口：
五个核心类：File、InputStream、OutputStream、Reader、Writer
一个核心接口：Serializable
在整个java.io包里面，File类是为唯一一个与文件本身操作有关的类，但是不涉及到文件的具体内容。文件本身指的是文件的创建、删除等操作。
如果要想使用File类，那么首先就需要通过它提供的构造方法定义一个要操作文件的路径：
设置完整路径：public File(String pathname)大部分情况下使用此类操作
设置父路径与子文件路径：public File(String parent,String child)在安卓上使用的比较多
常见文件：public boolean createNewFile()throws IOException;
  如果目录不能访问；
  如果现在文件重名，或者是文件名称错误；

删除文件：public boolean delete();
判断文件是否存在：public boolean exists();

```
package cn.mldn.demo;
import java.io.File;
import java.text.MessageFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.ResourceBundle;
public class TestDemo{
  public static void main(String args[])throws Exception {
    File file=new File("d:\\test.txt");//设置文件的路径,此处\test中的\t是制表符,会出现错误
    if(file.exists()){
      file.delete();
    }else{
      System.out.println(file.createNewFile());
    }
  }
}
```

以上的程序已经完成了具体的文件创建于删除操作，但是此时的程序会存在有两个问题。
在Windows系统里面支持的是“\”路径分隔符，在Linux下使用的是“/”路径分隔符；
在File类里面提供有一个常量public static final String separator
全局常量应该是全大写的，但是这个由于出现的过早，是历史遗留问题

```
File file=new File("d:"+File.separator+"test.txt");//表示分隔符
```

在进行java.io操作的过程之中，会出现有延迟情况，因为现在的问题是Java程序是通过JVM间接的调用操作系统的文件处理函数进行的文件处理操作，所以中间会出现延迟情况。

章节48 课时 186 04042_文件操作类（File类操作方法）

以上已经实现了文件的创建操作，但是这个时候是直接创建在了根路径下，下面来创建包含有子目录的文件。

```
File file=new File("d:"+File.separator+"demo"+File.separator+"test.txt");
```

那么现在又要“demo”目录不存在，所以系统会认为此时的路径是不能够使用的，那么就会出现创建错误。
现在必须要想办法判断要创建的父路径是否存在。
找到父路径：public File getParentFile()
创建目录：public boolean mkdirs()

```
package cn.mldn.demo;
import java.io.File;
import java.text.MessageFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.ResourceBundle;
public class TestDemo{
  public static void main(String args[])throws Exception {
    File file=new File("d:"+File.separator+"demo"+File.separator+"hello"+File.separator+"nihao"+File.separator+"test.txt");//设置文件的路径,此处\test中的\t是制表符,会出现错误
    if(!file.getParentFile().exists()){//如果父路径不存在
      file.getParentFile().mkdirs();
    }
    if(file.exists()){
      file.delete();
    }else{
      System.out.println(file.createNewFile());
    }
  }
}
```

在File类里面还提供有一系列的取得文件信息内容的操作功能：
取得文件大小：public long length()按照字节返回；
判断是否是文件：public boolean isFile();
判断是否是目录：public boolean isDirectory()
最近一次修改时间：public long lastModified()

```
package cn.mldn.demo;
import java.io.File;
import java.math.BigDecimal;
import java.text.SimpleDateFormat;
import java.util.Date;
public class TestDemo{
  public static void main(String args[])throws Exception {
```

```
    File file=new File("d:"+File.separator+"68031-106.jpg");
    if(file.exists()){
        System.out.println("是否是文件"+file.isFile());
        System.out.println("是否是目录"+file.isDirectory());
        System.out.println("文件大小"+(new BigDecimal((double)file.length()/1024/1024).divide(new
BigDecimal(1),2,BigDecimal.ROUND_HALF_UP))+"M");
        System.out.println("上次修改时间"+new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new
Date(file.lastModified())));
    }
  }
}
```

整个取得过程里面都是取得文件的相关信息，但是不包含有文件的内容。

章节48 课时 187 04043_文件操作类（操作目录）

以上的所以操作都是围绕着文件进行的，但是在整个磁盘上除了文件之外，还会包含有使用的目录，那么对于目录而言，最为常用的功能应该就是列出目录组成，在File类里面定义由如下的两个列出目录的方法:

列出目录下的信息：public String[] list()

列出所有的信息以File类对象包装：public File[] listFiles()

范例：列出信息

```
package cn.mldn.demo;
import java.io.File;
public class TestDemo{
  public static void main(String args[])throws Exception {
    File file=new File("d:"+File.separator);
    if(file.isDirectory()){
        String result[]=file.list();
        for(int x=0;x<result.length;x++){
            System.out.println(result[x]);
        }
    }
  }
}
```

此时的确是已经可以列出目录中的内容了，但是所列出来的是目录下的子目录或文件的名字。

范例：列出全部的File类对象

```
package cn.mldn.demo;
import java.io.File;
public class TestDemo{
  public static void main(String args[])throws Exception {
    File file=new File("d:"+File.separator);
    if(file.isDirectory()){
        File result[]=file.listFiles();
        for(int x=0;x<result.length;x++){
            System.out.println(result[x]);
        }
    }
  }
}
```

为了更好的体验出以上操作的好处，下面输出一个类似于资源管理器的界面。

```
package cn.mldn.demo;
import java.io.File;
import java.math.BigDecimal;
import java.sql.Date;
import java.text.SimpleDateFormat;
public class TestDemo{
  public static void main(String args[])throws Exception {
    File file=new File("d:"+File.separator);
    if(file.isDirectory()){
        File result[]=file.listFiles();
        for(int x=0;x<result.length;x++){
            System.out.println(result[x].getName()+"\t\t\t"+new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new
Date(result[x].lastModified()))+"\t\t"+(result[x].isDirectory()?"文件夹":"文件")+"\t\t"+(result[x].isFile()?(new
BigDecimal((double)file.length()/1024/1024).divide(new BigDecimal(1),2,BigDecimal.ROUND_HALF_UP))+"M":""));
        }
    }
  }
}
```

通过一系列验证可以发现，取得文件对象列表会更加的方便，因为可以继续取出很多的内容。
思考题：列出指定目录下的所有子路径
原则：如果现在给定的路径依然是一个目录，则应该向里面继续列出所有的组成，应该使用递归的方式完成。

```
package cn.mldn.demo;
import java.io.File;
public class TestDemo{
  public static void main(String args[])throws Exception {
    File file=new File("d:"+File.separator+"BaiduYunDownload");
    print(file);
  }
  public static void print(File file){
    if(file.isDirectory()){//如果现在给定的是一个路径
      File result[]=file.listFiles();//列出子目录
      if(result!=null){//已经可以列出目录
        for(int x=0;x<result.length;x++){
          print(result[x]);//继续列出
        }
      }
    }
    System.out.println(file);
  }
}
```

如果将以上的输出操作更换为了删除呢？这就成为了一个恶性程序了。

```
package cn.mldn.demo;
import java.io.File;
public class TestDemo{
  public static void main(String args[])throws Exception {
    File file=new File("d:"+File.separator+"mydemo");
    print(file);
  }
  public static void print(File file){
    if(file.isDirectory()){//如果现在给定的是一个路径
      File result[]=file.listFiles();//列出子目录
      if(result!=null){//已经可以列出目录
        for(int x=0;x<result.length;x++){
          print(result[x]);//继续列出
        }
      }
    }
    file.delete();
  }
}
```

1、File类本身只是操作文件的，不涉及到内容；
2、File类中的重要方法：
设置完整路径：public File(String pathname);大部分情况下使用此类操作
删除文件：public boolean delete();
判断文件是否存在：public boolean exists();
找到父路径：public File getParentFile()
创建目录：public boolean mkdirs()
3、在使用File类操作的时候路径的分隔符使用：File.separator;

章节49 课时 188 04044_字节流与字符流（简介）

File类虽然可以操作文件，但是并不是操作文件的内容，如果要进行内容的操作只能够通过两种途径完成：字节流，字符流。
如果要进行输入、输出操作一般都会按照如下的步骤进行（以文件操作为例）：
通过File类定义一个要操作文件的路径；
通过字节流或字符流的子类对象为父类对象实例化；
进行数据的读（输入）、写（输出）操作；
数据流属于资源操作，资源操作必须关闭。
对于java.io包而言它定义了两类流：
字节流（JDK1.0）：InputStream、OutputStream
字符流（JDK1.1）：Reader、Writer

章节49 课时 189 04045_字节流与字符流（输出流：OutputStream）

字节输出流：OutputStream(重点)
OutputStream类是一个专门进行字节数据输出的一个类，这个类定义如下：
public **abstract** class OutputStream extends Object implements Closeable, Flushable
首先可以发现OutputStream类实现了两个接口：Closeable、Flushable，这两个接口定义如下：

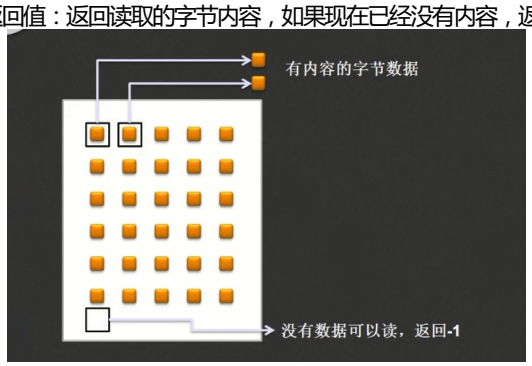
Closeable接口：JDK1.5开始提供的	Flushable接口：JDK1.5开始提供的

public interface Closeable
extends AutoCloseable{
public void close()throws IOException;
}
	public interface Flushable{
	public void flush()throws IOException
	}

最有意思的是在JDK1.7的时候引入了一个非常神奇的自动关闭机制
，所以让Closeable又多继承了一个AutoCloseable接口，这个父接口定义如下：
```
public interface AutoCloseable{
    public void close()throws Exception;
}
```
但是OutputStream类在JDK1.0的时候就提供的，这个类原本就定义了close()与flush()两个操作方法，所以现在以上的两个接口就几乎可以忽略了。
在OutputStream类里面一共提供有三个输出的方法：
1、输出单个字节：public abstract void write(int b)throws IOException
//虽然括号里的是int型 但是传的一定是byte
举栗：byte d=120;只要不超过128就是可以的
但是 int d=120; byte b=d;就不行，因为强制转换了。
2、输出全部字节数组：public void write(byte[] b)throws IOException
3、输出部分字节数组：**public void write(byte[] b,int off,int len)throws IOException**
OutputStream本身是属于抽象类，如果要想为抽象类进行对象的实例化操作，那么一定要使用抽象类的子类，本次由于要进行的是文件操作所以可以使用FileOutputStream子类。在这个子类里面定义由如下的构造方法：
创建或覆盖已有文件：**public FileOutputStream(File file)throws FileNotFoundException**
文件内容追加：public FileOutputStream(File file,boolean append)throws FileNotFoundException
范例：文件内容的输出
```
package cn.mldn.demo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
public class TestDemo{
    public static void main(String args[])throws Exception{
        //1、定义要输出文件的路径
        File file=new File("d:"+File.separator+"demo"+File.separator+"my.txt");
        //1、此时由于目录不存在，所以文件不能够输出，那么应该首先创建文件的目录
        if(!file.getParentFile().exists()){//文件目录不存在
            file.getParentFile().mkdirs();//创建目录
        }
        //2、应该使用OutputStream和其子类进行对象的实例化,此时目录存在，文件还不存在
        OutputStream output=new FileOutputStream(file);
        //3、进行文件内容的输出
        String str="好好学习，天天向上。";
        byte data[]=str.getBytes();//将字符串变为字节数组
        output.write(data);//将内容输出
        //4、资源操作的最后一定要进行关闭
        output.close();
    }
}
```
以上试讲整个字节数组的内容进行了输出，并且发现，如果此时要输出的文件不存在，那么会自动进行创建，可是对于输出操作在整个OutputStream类里面一共定义有三个方法：
范例：采用单个字节进行输出：
```
for(int x=0;x<data.length;x++){
    output.write(data[x]);//将内容输出
}
```
范例：输出部分字节数组内容
```
output.write(data,4,6);//将内容输出
```
但是每一次都是覆盖不是很好，希望可以实现内容的追加，那么只需要更好FileOutputStream类的构造方法即可。
```
//2、应该使用OutputStream和其子类进行对象的实例化,此时目录存在，文件还不存在
OutputStream output=new FileOutputStream(file,true);
//3、进行文件内容的输出
String str="好好学习，天天向上!\r\n";
```
只要是程序要输出内容，都可以利用OutputStream类完成。

章节49 课时 190 04046_字节流与字符流（输入流：InputStream）

如果程序需要进行数据的读取操作，可以利用InputStream类实现功能，此类定义如下：
public abstract class InputStream extends Object implements Closeable
虽然此类实现了Closeable接口但是与OutputStream类一样，不用考虑此接口的存在，在InputStream里面也定义有数据读取的方法：
读取单个字节：public abstract int read()throws IOException
|-返回值：返回读取的字节内容，如果现在已经没有内容，返回-1



将读取的数据保存在字节数组里：public int read(byte[] b)throws IOException
|-返回值：返回读取的数据长度，但是如果已经读取的结尾了，返回-1
将读取的数据保存在部分字节数组里：public int read(byte[] b,int off,int len)throws IOException
|-返回值：读取部分数据的长度，如果已经到结尾了，返回-1

InputStream是一个抽象类，所以如果要想进行文件读取使用FileInputStream子类，而这个子类的构造
构造方法：public FileInputStream(File file)throws FileNotFoundException
范例：向数组里面读取数据

```
package cn.mldn.demo;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class TestDemo{
    public static void main(String args[])throws Exception{
        //1、定义要输入文件的路径
        File file=new File("d:"+File.separator+"demo"+File.separator+"my.txt");
        //需要判断文件是否存在后才可以进行读取
        if(file.exists()){
            //2、使用InputStream进行读取
            InputStream input=new FileInputStream(file);
            //3、进行数据读取
            byte data[]=new byte[1024];//准备出一个1024的数组
            int len=input.read(data);
            input.read(data);//将内容保存到字节数组之中
            //4、关闭输入流
            input.close();
            System.out.println("【"+new String(data,0,len)+"】");
        }
    }
}
```

范例：单个字节数据读取
由于一个文件有很多的字节数据，所以如果要读取肯定采用循环的方式完成，由于不确定循环的次数，所以应该使用while循环完成，下面为了更好地演示出问题，使用do..while与while两种方式实现。
实现一（不用）：使用do..while

```
package cn.mldn.demo;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class TestDemo{
    public static void main(String args[])throws Exception{
        //1、定义要输入文件的路径
        File file=new File("d:"+File.separator+"demo"+File.separator+"my.txt");
        //需要判断文件是否存在后才可以进行读取
        if(file.exists()){
            //2、使用InputStream进行读取
            InputStream input=new FileInputStream(file);
            //3、进行数据读取
```

```
            byte data[]=new byte[1024];//准备出一个1024的数组
            int foot=0;//表示字节数组的操作脚标
            int temp=0;//表示接收每次读取的字节数据
            do{
               temp=input.read();//读取一个字节
               if(temp !=-1){//现在是真实的内容
                  data[foot++]=(byte)temp;//保存读取的字节到数组之中
               }
            }while(temp !=-1);//如果现在读取的temp的字节数据不是-1，表示还有内容
            //4、关闭输入流
            input.close();
            System.out.println("【"+new String(data,0,foot)+"】");
         }
      }
   }
```

实现二（重点）：利用while循环读取

```
package cn.mldn.demo;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class TestDemo{
   public static void main(String args[])throws Exception{
      //1、定义要输入文件的路径
      File file=new File("d:"+File.separator+"demo"+File.separator+"my.txt");
      //需要判断文件是否存在后才可以进行读取
      if(file.exists()){
         //2、使用InputStream进行读取
         InputStream input=new FileInputStream(file);
         //3、进行数据读取
         byte data[]=new byte[1024];//准备出一个1024的数组
         int foot=0;//表示字节数组的操作脚标
         int temp=0;//表示接收每次读取的字节数据
         //第一部分：(temp=input.read()),表示read()方法将读取的字节内容给了temp变量
         //第二部分：temp=input.read())!=-1，判断读取出来的temp内容是不是-1;
         while((temp=input.read())!=-1){
            data[foot++]=(byte)temp;//有内容进行保存
         }
         //4、关闭输入流
         input.close();
         System.out.println("【"+new String(data,0,foot)+"】");
      }
   }
}
```

此类读取数据的方式是指以后的开发之中使用最多的情况。

章节49 课时 191 04047_字节流与字符流（字符输出流：Writer）

字符输出流：

```
public abstract class Writer extends Object implements Appendable, Closeable, Flushable
```

这个类除了实现Closeable与Flushable接口外，又多实现了一个Appendable，这个接口定义如下：

```
public interface Appendable{
   public Writer append(char c)
         throws IOException;
   public Writer append(CharSequence csq)
         throws IOException;
   public Writer append(CharSequence csq,int start,int end)
         throws IOException
}
```

在Appendable接口里面定义了追加的操作，而且追加的数据都是字符或者字符串。

在Writer类里面定义有一下的输出方法（部分）：

输出全部字符数组：public void write(char[] cbuf)throws IOException;

输出字符串：public void write(String str)throws IOException

Writer是一个抽象类，如果要想为这个类的对象实例化，应该使用FileWriter子类；

范例：使用Writer类实现内容输出

```
package cn.mldn.demo;
import java.io.File;
```

```
import java.io.FileWriter;
import java.io.Writer;
public class TestDemo{
    public static void main(String args[])throws Exception{
        //1、定义要输入文件的路径
        File file=new File("D:"+File.separator+"demo"+File.separator+"my.txt");
        if(file.getParentFile().exists()){
            file.getParentFile().mkdirs();
        }
        //2、实例化了Writer类对象
        Writer out=new FileWriter(file);
        //3、进行内容输出
        String str="千万不要被节后综合症给打败，你们已经开始有人挑战我了！";
        out.write(str);//输出字符串数据
        //4、关闭输出流
        out.close();
    }
}
```

可以发现Writer作为字符输出流，可以直接进行字符串的输出，这一点就比OutputStream强了。

章节49 课时 192 04048_字节流与字符流（字符输入流：Reader）

章节49 课时 193 04049_字节流与字符流（字节流与字符流区别）

章节49 课时 194 04050_转换流

章节49 课时 195 04051_综合实战：文件拷贝

先写一个主类，在写主方法，点击run as运行。
再点击run as进行配置，选择Arguments 在Program arguments框里写上d:\凤凰.jpg d:\phoenix.jpg
然后开始写程序

```
package cn.mldn.demo;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
public class CopyDemo {
 public static void main(String[] args) throws Exception{
  long start=System.currentTimeMillis();
  if(args.length!=2){//初始化参数不足2位
   System.out.println("命令执行错误");
   System.exit(1);//程序退出执行
  }
  File inFile=new File(args[0]);//第一个为源文件路径
  if(!inFile.exists()){//如果源文件不存在
   System.out.println("源文件不存在，请确认执行路径");
   System.exit(1);//程序退出执行
  }
  //如果此时源文件正确，那么就需要定义输出文件，同时要考虑到输出文件有目录
  File outFile=new File(args[1]);
  if(!outFile.getParentFile().exists()){//输出文件不存在
   outFile.getParentFile().mkdirs();//创建目录
  }
  //实现文件内容的拷贝
  InputStream input=new FileInputStream(inFile);
  OutputStream output=new FileOutputStream(outFile);
  int temp=0;//保存每次读取的内容
  while((temp = input.read())!=-1){
   output.write(temp);
  }
  input.close();
  output.close();
  long end=System.currentTimeMillis();
  System.out.println("拷贝所需时间:"+(end-start));
 }
}
```

以上代码虽然完成了复制，但是对于大文件复制过程是在太慢。下面进行优化

InputStream：public int read(byte []) throws IOException
　|- 将内容读取到字节数组中，如果没有数据了返回的是-1，否则就是读取长度（水瓢舀水概念）
OutputStream：public int write(byte[],int off,int len)throws IOException
　|- 要设置的数组实际上就是在read()方法里面使用的数组；
　|- 一定是从字节数组的第0个元素开始输出，输出读取的数据长度。
范例：改进拷贝操作
```
package cn.mldn.demo;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
public class CopyDemo {
    public static void main(String[] args) throws Exception{
        long start=System.currentTimeMillis();
        if(args.length!=2){//初始化参数不足2位
            System.out.println("命令执行错误");
            System.exit(1);//程序退出执行
        }
        File inFile=new File(args[0]);//第一个为源文件路径
        if(!inFile.exists()){//如果源文件不存在
            System.out.println("源文件不存在，请确认执行路径");
            System.exit(1);//程序退出执行
        }
        //如果此时源文件正确，那么就需要定义输出文件，同时要考虑到输出文件有目录
        File outFile=new File(args[1]);
        if(!outFile.getParentFile().exists()){//输出文件不存在
            outFile.getParentFile().mkdirs();//创建目录
        }
        //实现文件内容的拷贝
        InputStream input=new FileInputStream(inFile);
        OutputStream output=new FileOutputStream(outFile);
        int temp=0;//保存每次读取的内容
        byte data[]=new byte[1024];//每次读取1024字节
        //将每次读取的数据保存在字节数组里面，并且返回读取的个数
        while((temp = input.read(data))!=-1){
            output.write(data);//输出数组
        }
        input.close();
        output.close();
        long end=System.currentTimeMillis();
        System.out.println("拷贝所需时间:"+(end-start));
    }
}
```
在以后的代码编写过程中，一定会给你一个InputStream字节流对象，就需要按照以上的做法以同样的方式进行内容的输出。
本程序属于之前的总结程序，如果把本程序掌握清楚，就可以轻松掌握InputStream和OutputStream的核心操作，并且在以后的开发之中以上的代码100%会使用到，必须掌握。

章节50 课时 196 04052_字符编码

本次预计讲解的知识点：
　了解一下常见的额字符编码
　了解乱码的产生原因。
具体内容（了解）
　计算机中所有的信息组成都是二进制数据，那么所有能描述出的中文文字都是经过处理后的结果。在计算机的世界里，所有的语言文字都会使用编码来进行描述，例如：最常见的编码是ASCII码。在实际的工作里面最为常见的几种编码如下：
　GBK GB2312：中文的国标编码，其中GBK包含有简体中文和繁体中文两种，而GBK2312只包含简体中文。
　ISO8859-1：是国际编码，可以描述任何的文字信息；
　UNICODE：是十六进制编码，造成传输的无用数据过多；
　UTF编码（UTF-8）：融合了ISO8859-1和UNICODE编码的特点；
在以后的所有开发里面，使用的都是UTF-8编码。
所谓的乱码最本质的方式就是编码与解码的字符集不统一。
```
package cn.mldn.demo;
public class CopyDemo {
    public static void main(String[] args) throws Exception{
        System.getProperties().list(System.out);
    }
```

```
}
file.encoding=GBK
file.separator=\
```
发现默认的编码是GBK，那么也就是说默认输出的中文都是GBK的编码标准。

范例：
```
package cn.mldn.demo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
public class CopyDemo {
    public static void main(String[] args) throws Exception{
        File file=new File("d:"+File.separator+"my.txt");
        OutputStream output=new FileOutputStream(file);
        output.write("中国万岁，世界万岁，人民万岁。".getBytes());//没有转码
        output.close();
    }
}
```
范例：乱码
```
package cn.mldn.demo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
public class CopyDemo {
    public static void main(String[] args) throws Exception{
        File file=new File("d:"+File.separator+"my.txt");
        OutputStream output=new FileOutputStream(file);
        output.write("中国万岁，世界万岁，人民万岁。".getBytes("ISO8859-1"));//没有转码
        output.close();
    }
}
```

总结

1、以后开发的代码使用的都是UTF-8编码；

2、乱码的本质就是编码与解码不统一。

章节50 课时 197 04053_内存流

本次预计讲解的知识点

当我们学习到了AJAX+XML（JSON）应用的时候才会牵扯到此部分。

可以使用内存流实现IO操作

具体内容

在之前使用过了文件操作流实现了针对于文件数据的输入与输出操作，但是如果说现在某一种应用，需要进行IO操作，可是又不想产生文件的时候，就可以利用内存来实现输入与输出的操作。

针对于内粗来，java.io包里面提供了两组操作：

字节内存流：ByteArrayInputStream、ByteArrayOutputStream

字符内存流：CharArrayReader、CharArrayWriter

本次是以字节内存流操作为主。下面重点来看一下ByteArrayInputStream与ByteArrayOutStream的继承结构与构造方法。

名称：	ByteArrayInPutStream	ByteArrayOutputStream
继承结构：	java.lang.Object java.io.InputStream java.io.ByteArrayInputStream	java.lang.Object java.io.OutputStream java.io.ByteArrayOutputStream
构造方法：	public ByteArrayInputStream(byte[] buf)	public ByteArrayOutputStream()
	表示将要操作的数据设置到输入流	从内存输出数据

下面为了更好地说明出问题，特别做一个举例：

以文件操作为例：

|-输出（OutPutStream）：程序→OutPutStream→内存；

|-输入（InPutStream）：程序←InputStream←内存；

以内存操作为例：

|-输出（InputStream）：程序→InputStream→内存

|-输入（OutputStream）：程序←OutputStream←内存；

范例：实现一个小写字母转大写字母的操作

为了方便的实现字母的转大写转换（避免不必要的字符也被转换了）可以借助java.lang.Character
|-public static char toLowerCase(char ch)
|-public static int toLowerCase(int codePoint)
|-public static char toUpperCase(char ch)
|-public static int toUpperCase(int codePoint)