常用类

> Author：zhangzhang
>
> Version：1.0.0

一、内部类

1.1 内部类的分类

> - 成员内部类
> - 静态内部类
> - 局部内部类
> - 匿名内部类

1.2 什么是内部类

> 概念：在一个类的内部再定义一个完整的类。

> 特点：
>
> - 编译之后可生成独立的字节码文件。
> - 内部类可直接访问外部类的私有成员，而不破坏封装。
> - 可为外部类提供必要的内部功能组件。

1.3 成员内部类

> - 在类的内部定义，与实例变量、实例方法同级别的类。
> - 外部类的一个实例部分，创建内部类对象时，必须依赖外部类对象。
>
>   Outer out = new Outer();
>
>   Outer.Inner in = out.new Inner();
> - 当外部类、内部类存在重名属性时，会优先访问内部类属性。
> - 成员内部类不能定义静态成员。

> 案例演示：成员内部类使用。

```java
public class Outer {
  //实例变量
  private String name="张三";
  private int age=20;

  //内部类
  class Inner{
    private String address="北京";
    private String phone="110";
    //
    private String name="李四";

    //private static final String country="中国";
    //方法
    public void show() {
      //打印外部类的属性，内部类属性和外部类的属性名字相同Outer.this
      System.out.println(Outer.this.name);
      System.out.println(Outer.this.age);
      //打印内部类中的属性
      System.out.println(this.address);
      System.out.println(this.phone);
    }
  }

}
```

```java
public class TestOuter {
  public static void main(String[] args) {
    //1创建外部类对象
//    Outer outer=new Outer();
//    //2创建内部类对象
//    Inner inner=outer.new Inner();

    //3一步到位
    Inner inner=new Outer().new Inner();
    inner.show();
  }
}
```

注：打印外部类的属性，内部类属性和外部类的属性名字相同Outer.this。

1.4 静态内部类

> - 不依赖外部类对象，可直接创建或通过类名访问，可声明静态成员。
> - 只能直接访问外部类的静态成员（实例成员需实例化外部类对象）。
>
> Outer.Inner inner = new Outer.Inner();
> Outer.Inner.show();

> 案例演示：静态内部类使用。

```java
public class Outer {
  private String name="xxx";
  private int age=18;

  //静态内部类：和外部类相同
  static class Inner{
    private String address="上海";
    private String phone="111";
    //静态成员
    private static int count=1000;

    public void show() {
      //调用外部类的属性呢
      //1先创建外部类对象
      Outer outer=new Outer();
      //2调用外部类对象的属性
      System.out.println(outer.name);
      System.out.println(outer.age);
      //调用静态内部类的属性和方法
      System.out.println(address);
      System.out.println(phone);
      //调用静态内部类的静态属性
      System.out.println(Inner.count);
    }
  }
}
```

```
public class TestOuter {
  public static void main(String[] args) {
    //直接创建静态内部类对象
    Outer.Inner inner=new Outer.Inner();
    //调用方法
    inner.show();

  }
}
```

1.5 局部内部类

> - 定义在外部类方法中，作用范围和创建对象范围仅限于当前方法。
> - 局部内部类访问外部类当前方法中的局部变量时，因无法保障变量的生命周期与自身相同，变量必须修饰为final。
> - 限制类的使用范围。

> 案例演示：局部内部类使用。

```
public class Outer {
  private String name="刘德华";
  private int age=35;

  public  void show(final int i) {
    //定义局部变量
    String address="深圳";
    //局部内部类 :注意不能加任何访问修饰符
    class Inner{
      //局部内部类的属性
      private String phone="15588888888";
      private String email="liudehua@qq.com";
      //private final static int count=2000;

      public void show2() {
        //访问外部类的属性
        System.out.println(Outer.this.name);
        System.out.println(Outer.this.age);
        //访问内部类的属性
        System.out.println(this.phone);
        System.out.println(this.email);
        //访问局部变量,jdk1.7要求：变量必须是常量 final, jdk1.8 自动添加final
        System.out.println("深圳");
        System.out.println(i);
      }
    }
    //创建局部内部类对象
    Inner inner=new Inner();
    inner.show2();

  }
}
```

```
public class TestOuter {
  public static void main(String[] args) {
    Outer outer=new Outer();
    outer.show(120);
  }
}
```

1.6 匿名内部类

> - 没有类名的局部内部类（一切特征都与局部内部类相同）。
> - 必须继承一个父类或者实现一个接口。
> - 定义类、实现类、创建对象的语法合并，只能创建一个该类的对象。
> - 优点：减少代码量。
> - 缺点：可读性较差。

> 案例演示：匿名内部类使用。

```
public interface Usb {
  //服务
  void service();
}
```

```
public class TestUsb {
  public static void main(String[] args) {
    //局部内部类
//    class Fan implements Usb{
//
//      @Override
//      public void service() {
//        System.out.println("连接电脑成功，风扇开始工作了....");
```

```
//
//      }
//    }

    //使用局部内部类创建对象
//    Usb usb=new Fan();
//    usb.service();

    //使用匿名内部类优化(相当于创建了一个局部内部类)
    Usb usb=new Usb() {
      @Override
      public void service() {
        System.out.println("连接电脑成功，风扇开始工作了....");

      }
    };
    usb.service();
  }
}
```

二、Object类【 重点 】

2.1 概述

> - 超类、基类，所有类的直接或间接父类，位于继承树的最顶层。
> - 任何类，如没有书写extends显示继承某个类，都默认直接继承Object类，否则为间接继承。
> - Object类中所定义的方法，是所有对象都具备的方法。
> - Object类型可以存储任何对象。
> - 作为参数，可接受任何对象。
> - 作为返回值，可返回任何对象。

2.2 常用方法

2.2.1 getClass()

> public final Class<?> getClass(){...}
>
> - 返回引用中存储的实际对象类型。
> - 应用：通常用于判断两个引用中实际存储对象类型是否一致。

2.2.2 hashCode()方法

> public int hashCode(){...}
>
> - 返回该对象的十进制的哈希码值。
> - 哈希算法根据对象的地址或字符串或数字计算出来的int类型的数值。
> - 哈希码并不唯一，可保证相同对象返回相同哈希码，尽量保证不同对象返回不同哈希码。

2.2.3 toString()方法

> public String toString(){...}
>
> - 返回该对象的字符串表示（表现形式）。
> - 可以根据程序需求覆盖该方法，如：展示对象各个属性值。

2.2.4 equals()方法

> public boolean equals(Object obj){...}
>
> - 默认实现为(this == obj)，比较两个对象地址是否相同。
> - 可进行覆盖，比较两个对象的内容是否相同。

> equals重写步骤：
>
> - 比较两个引用是否指向同一个对象。
> - 判断obj是否为null。
> - 判断两个引用指向的实际对象类型是否一致。
> - 强制类型转换。
> - 依次比较各个属性值是否相同。

```
public class Student  {
  private String name;
  private int age;
  public Student() {
    // TODO Auto-generated constructor stub
  }


  public Student(String name, int age) {
    super();
    this.name = name;
```

```
    this.age = age;
  }


  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }


  @Override
  public String toString() {
    return "Student [name=" + name + ", age=" + age + "]";
  }


  @Override
  public boolean equals(Object obj) {
    //1判断两个对象是否是同一个引用
    if(this==obj) {
      return true;
    }
    //2判断obj是否null
    if(obj==null) {
      return false;
    }
    //3判断是否是同一个类型
//    if(this.getClass()==obj.getClass()) {
//
//    }
    //intanceof 判断对象是否是某种类型
    if(obj instanceof Student) {
      //4强制类型转换
      Student s=(Student)obj;
      //5比较熟悉
      if(this.name.equals(s.getName())&&this.age==s.getAge()) {
        return true;
      }


    }
    return false;
  }

  @Override
  protected void finalize() throws Throwable {
    System.out.println(this.name+"对象被回收了");
  }

}
```

```
public class TestStudent {
  public static void main(String[] args) {
    //1getClass方法
    System.out.println("------------1getClass---------");
    Student s1=new Student("aaa", 20);
    Student s2=new Student("bbb",22);
    //判断s1和s2是不是同一个类型
    Class class1=s1.getClass();
    Class class2=s2.getClass();
    if(class1==class2) {
      System.out.println("s1和s2属于同一个类型");
    }else {
      System.out.println("s1和s2不属于同一个类型");
    }
    System.out.println("-----------2hashCode------------");
    //2hashCode方法
    System.out.println(s1.hashCode());

    System.out.println(s2.hashCode());
    Student s3=s1;
    System.out.println(s3.hashCode());

    //3toString方法
    System.out.println("-----------3toString------------");
    System.out.println(s1.toString());
    System.out.println(s2.toString());

    //4equals方法：判断两个对象是否相等
    System.out.println("-----------4equals------------");
```

```
    System.out.println(s1.equals(s2));

    Student s4=new Student("小明", 17);
    Student s5=new Student("小明",17);
    System.out.println(s4.equals(s5));


  }
}
```

2.2.5 finalize()方法

- 当对象被判定为垃圾对象时，由JVM自动调用此方法，用以标记垃圾对象，进入回收队列。
- 垃圾对象：没有有效引用指向此对象时，为垃圾对象。
- 垃圾回收： 由GC销毁垃圾对象，释放数据存储空间。
- 自动回收机制：JVM的内存耗尽，一次性回收所有垃圾对象。
- 手动回收机制：使用System.gc(); 通知JVM执行垃圾回收。

```
public class TestFinalize {
  public static void main(String[] args) {
//    Student s1=new Student("aaa", 20);
//    Student s2=new Student("bbb", 20);
//    Student s3=new Student("ccc", 20);
//    Student s4=new Student("ddd", 20);
//    Student s5=new Student("eee", 20);
    new Student("aaa", 20);
    new Student("bbb", 20);
    new Student("ccc", 20);
    new Student("ddd", 20);
    new Student("eee", 20);
    //回收垃圾
    System.gc();
    System.out.println("回收垃圾");

  }
}
```

三、包装类

3.1 概述

- 基本数据类型所对应的引用数据类型。
- Object可统一所有数据，包装类的默认值是null。

基本类型	包装类
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

3.2 装箱、拆箱

- 八种包装类提供不同类型间的转换方式。
- Number父类中提供的6个共性方法。
  - parseXXX()静态方法（除了Character）。
  - valueOf()静态方法。
- 注意：需保证类型兼容，否则抛出NumberFormatException异常。
- JDK 5.0之后，自动装箱、拆箱。基本数据类型和包装类自动转换。

案例演示：包装类使用。

```
public class TestInteger {
  public static void main(String[] args) {
//    int num=10;
    //类型转换:装箱,基本类型转成引用类型的过程
    //基本类型
    int num1=18;
    //使用Integer类创建对象
```

```
        Integer integer1=new Integer(num1);
        Integer integer2=Integer.valueOf(num1);
        System.out.println("装箱");
        System.out.println(integer1);
        System.out.println(integer2);

        //类型转型:拆箱,引用类型转成基本类型
        Integer integer3=new Integer(100);
        int num2=integer3.intValue();
        System.out.println("拆箱");
        System.out.println(num2);


        //JDK1.5之后,提供自动装箱和拆箱
            System.out.println("-------------JDK1.5之后,提供自动装箱和拆箱----------------");
        int age=30;
        //自动装箱
        Integer integer4=age;
        System.out.println("自动装箱");
        System.out.println(integer4);
        //自动拆箱
        int age2=integer4;
        System.out.println("自动拆箱");
        System.out.println(age2);

        System.out.println("-------------基本类型和字符串之间转换----------------");
        //基本类型和字符串之间转换
        //1 基本类型转成字符串
        int n1=255;
        //1.1使用+号
        String s1=n1+"";
        //1.2使用Integer中的toString()方法
        String s2=Integer.toString(n1, 16);//f
        System.out.println(s1);
        System.out.println(s2);


        //2字符串转成基本类型
        String str="150";
        //使用Integer.parseXXX();
        int n2=Integer.parseInt(str);
        System.out.println(n2);

        //boolean字符串形式转成基本类型, "true"--->true  非"true"---->false
        String str2="false";
        boolean b1=Boolean.parseBoolean(str2);
        System.out.println(b1);

    }
}
```

3.3 整数缓冲区

> - Java预先创建了256个常用的整数包装类型对象。
> - 在实际应用当中，对已创建的对象进行复用。

> 面试题：分析以下输出结果的原因。

```
public class TestInteger2 {
    public static void main(String[] args) {
        //面试题
        Integer integer1=new Integer(100);
        Integer integer2=new Integer(100);
        System.out.println(integer1==integer2);

        Integer integer3=Integer.valueOf(100);//自动装箱Integer.valueOf
        Integer integer4=Integer.valueOf(100);
        System.out.println(integer3==integer4);//true

        Integer integer5=Integer.valueOf(200);//自动装箱
        Integer integer6=Integer.valueOf(200);
        System.out.println(integer5==integer6);//false

    }
}
```

四、String类【 重点 】

4.1 概述

> - Java程序中的所有字符串文本（例如“abc”）都是此类的实例。
> - 字符串字面值是常量，创建之后不可改变。
> - 常用创建方式：

- String str1 = “Hello”;
- String str2 = new String(“World”);

4.2 常用方法

方法名	描述
public char charAt(int index)	根据下标获取字符
public boolean contains(String str)	判断当前字符串中是否包含str
public char[] toCharArray()	将字符串转换成数组。
public int indexOf(String str)	查找str首次出现的下标，存在，则返回该下标；不存在，则返回-1
public int length()	返回字符串的长度
public String trim()	去掉字符串前后的空格
public String toUpperCase()	将小写转成大写
public boolean endsWith(String str)	判断字符串是否以str结尾
public String replace(char oldChar,char newChar)	将旧字符串替换成新字符串
public String[] split(String str)	根据str做拆分
public String subString(int beginIndex,int endIndex)	在字符串中截取出一个子字符串

案例演示：String的使用。

```
public class TestString1 {
  public static void main(String[] args) {
    String name="hello";//"hello" 常量存储在字符串池中,
    name="zhangsan";//“张三”赋值给name变量，给字符串赋值时，并没有修改数据，而是重新开辟一个空间
    String name2="zhangsan";

    //演示字符串的另一种创建方式，new String();
    String str=new String("java");
    String str2=new String("java");
    System.out.println(str==str2);
    System.out.println(str.equals(str2));

    System.out.println("------------字符串方法的使用 1-----------------");
    //字符串方法的使用
    //1、length();返回字符串的长度
    //2、charAt(int index);返回某个位置的字符
    //3、contains(String str);判断是否包含某个子字符串

    String content="java是世界上最好的java编程语言,java真香";
    System.out.println(content.length());
    System.out.println(content.charAt(content.length()-1));
    System.out.println(content.contains("java"));
    System.out.println(content.contains("php"));

    System.out.println("------------字符串方法的使用 2-----------------");
    //字符串方法的使用
    //4、toCharArray();返回字符串对应的数组
    //5、indexOf();返回子字符串首次出现的位置
    //6、lastIndexOf();返回字符串最后一次出现的位置
    System.out.println(Arrays.toString(content.toCharArray()));
    System.out.println(content.indexOf("java"));
    System.out.println(content.indexOf("java", 4));
    System.out.println(content.lastIndexOf("java"));

    System.out.println("------------字符串方法的使用3-----------------");

    //7、trim();去掉字符串前后的空格
    //8、toUpperCase();//把小写转成大写 toLowerCase();把大写转成小写
    //9、endWith(str);判断是否已str结尾,startWith(str);判断是否已str开头

    String content2="   hello World   ";
    System.out.println(content2.trim());
    System.out.println(content2.toUpperCase());
    System.out.println(content2.toLowerCase());
    String filename="hello.java";
    System.out.println(filename.endsWith(".java"));
    System.out.println(filename.startsWith("hello"));


    System.out.println("------------字符串方法的使用4-----------------");

    //10、replace(char old,char new); 用新的字符或字符串替换旧的字符或字符串
    //11、split();对字符串进行拆分

    System.out.println(content.replace("java", "php"));

    String say="java is the best   programing language,java xiang";
```

```
    String[] arr=say.split("[ ,]+");
    System.out.println(arr.length);
    for (String string : arr) {
      System.out.println(string);
    }

    //补充两个方法equals 、compareTo();比较大小
    System.out.println("---------补充---------");
    String s1="hello";
    String s2="HELLO";
    System.out.println(s1.equalsIgnoreCase(s2));

    String s3="abc";//97
    String s4="ayzawe";//120
    System.out.println(s3.compareTo(s4));

    String s5="abc";
    String s6="abc";
    System.out.println(s5.compareTo(s6));
  }
}
```

五、可变字符串

> - 概念：可在内存中创建可变的缓冲空间，存储频繁改变的字符串。

> - Java中提供了两个可变字符串类：
>   - StringBuilder：可变长字符串，JDK5.0提供，运行效率快、线程不安全。
>   - StringBuffer：可变长字符串，JDK1.0提供，运行效率慢、线程安全。
>   - 这两个类中方法和属性完全一致。

> 常用方法：

方法名	属性
public StringBuilder append(String str)	追加内容。
public StringBuilder insert(int dstOffset, CharSequence s)	将指定 字符串插入此序列中。
public StringBuilder delete(int start, int end)	移除此序列的子字符串中的字符。
public StringBuilder replace(int start, int end, String str)	使用给定字符串替换此序列的子字符串中的字符。start开始位置、end结束位置。
public int length()	返回长度（字符数）。

> 案例演示：StringBuilder的使用。

```
public class TestStringBuilder {
  public static void main(String[] args) {
    //StringBuffer sb=new StringBuffer();
    StringBuilder sb=new StringBuilder();
    //1 append();追加
    sb.append("java世界第一");
    System.out.println(sb.toString());
    sb.append("java真香");
    System.out.println(sb.toString());
    sb.append("java不错");
    System.out.println(sb.toString());
    //2 insert();添加
    sb.insert(0, "我在最前面");
    System.out.println(sb.toString());
    //3 replace();替换
    sb.replace(0, 5, "hello");
    System.out.println(sb.toString());
    //4 delete();删除
    sb.delete(0, 5);
    System.out.println(sb.toString());
    //清空
    sb.delete(0, sb.length());
    System.out.println(sb.length());


  }
}
```

> 案例演示：验证StringBuilder效率高于String。

```
public class TestStringBuilder2 {
  public static void main(String[] args) {
    //开始时间
    long start=System.currentTimeMillis();
    String string="";
    for(int i=0;i<99999;i++) {
      string+=i;
    }
    System.out.println(string);
//    StringBuilder sb=new StringBuilder();
//    for(int i=0;i<99999;i++) {
//      sb.append(i);
//    }
//    System.out.println(sb.toString());
    long end=System.currentTimeMillis();
    System.out.println("用时:"+(end-start));
  }
}
```

六、BigDecimal

6.1 为什么使用BigDecimal?

- **思考：以下程序输出结果是多少?**

```
public class TestBigDecimal {
    public static void main(String[] args) {
        double d1=1.0;
        double d2=0.9;
        System.out.println(d1-d2);
    }
}
```

输出结果：
0.09999999999999998

→ 很多实际应用中需要精确运算，而double是近似值存储，不再符合要求，需要借助BigDecimal。

6.2 BigDeicmal基本用法

> - 位置：java.math包中。
> - 作用：精确计算浮点数。
> - 创建方式：BigDecimal bd=new BigDecimal(“1.0”)。

> 常用方法：

方法名	描述
BigDecimal add(BigDecimal bd)	加
BigDecimal subtract(BigDecimal bd)	减
BigDecimal multiply(BigDecimal bd)	乘
BigDecimal divide(BigDecimal bd)	除

```
public class Demo7 {
  public static void main(String[] args) {
    double d1=1.0;
    double d2=0.9;
    System.out.println(d1-d2);

    //面试题
    double result=(1.4-0.5)/0.9;
    System.out.println(result);
    //BigDecimal,大的浮点数精确计算
    BigDecimal bd1=new BigDecimal("1.0");
    BigDecimal bd2=new BigDecimal("0.9");
    //减法
    BigDecimal r1=bd1.subtract(bd2);
    System.out.println(r1);
```

```
    //加法
    BigDecimal r2=bd1.add(bd2);
    System.out.println(r2);

    //乘法
    BigDecimal r3=bd1.multiply(bd2);
    System.out.println(r3);

    //除法
    BigDecimal r4=new BigDecimal("1.4")
      .subtract(new BigDecimal("0.5"))
      .divide(new BigDecimal("0.9"));
    System.out.println(r4);

    BigDecimal r5=new BigDecimal("20").divide(new BigDecimal("3"),2,BigDecimal.ROUND_HALF_UP);
    System.out.println(r5);

  }
}
```

- 除法：divide(BigDecimal bd,int scal,RoundingMode mode)。
- 参数scale： 指定精确到小数点后几位。
- 参数mode：
  - 指定小数部分的取舍模式，通常采用四舍五入的模式。
  - 取值为BigDecimal.ROUND_HALF_UP。

七、Date

- Date表示特定的瞬间，精确到毫秒。
- Date类中的大部分方法都已经被Calendar类中的方法所取代。
- 时间单位
  - 1秒=1000毫秒
  - 1毫秒=1000微秒
  - 1微秒=1000纳秒

案例演示：

```
public class TestDate {
  public static void main(String[] args) {
    //1创建Date对象
    //今天
    Date date1=new Date();
    System.out.println(date1.toString());
    System.out.println(date1.toLocaleString());
    //昨天
    Date date2=new Date(date1.getTime()-(60*60*24*1000));
    System.out.println(date2.toLocaleString());
    //2方法after before
    boolean b1=date1.after(date2);
    System.out.println(b1);
    boolean b2=date1.before(date2);
    System.out.println(b2);

    //比较 compareTo();
    int d=date2.compareTo(date1);
    System.out.println(d);
    //比较是否相等 equals()
    boolean b3=date1.equals(date2);
    System.out.println(b3);

  }
}
```

八、Calendar

- Calendar提供了获取或设置各种日历字段的方法。
- protected Calendar() 构造方法为protected修饰，无法直接创建该对象。

常用方法：

方法名	说明
static Calendar getInstance()	使用默认时区和区域获取日历
void set(int year,int month,int date,int hourofday,int minute,int second)	设置日历的年、月、日、时、分、秒。
int get(int field)	返回给定日历字段的值。字段比如年、月、日等
void setTime(Date date)	用给定的Date设置此日历的时间。Date-Calendar
Date getTime()	返回一个Date表示此日历的时间。Calendar-Date
void add(int field,int amount)	按照日历的规则，给指定字段添加或减少时间量
long getTimeInMillis()	毫秒为单位返回该日历的时间值

> 案例演示：

```
public class TestCalendar {
  public static void main(String[] args) {
    //1创建Calendar对象
    Calendar calendar=Calendar.getInstance();
    System.out.println(calendar.getTime().toLocaleString());
    System.out.println(calendar.getTimeInMillis());
    //2获取时间信息
    //获取年
    int year=calendar.get(Calendar.YEAR);
    //月 从0-11
    int month=calendar.get(Calendar.MONTH);
    //日
    int day=calendar.get(Calendar.DAY_OF_MONTH);//Date
    //小时
    int hour=calendar.get(Calendar.HOUR_OF_DAY); //HOUR12小时 HOUR_OF_DAY24小时
    //分钟
    int minute=calendar.get(Calendar.MINUTE);
    //秒
    int second=calendar.get(Calendar.SECOND);
    System.out.println(year+"年"+(month+1)+"月"+day+"日"+hour+":"+minute+":"+second);
    //3修改时间
    Calendar calendar2=Calendar.getInstance();
    calendar2.set(Calendar.DAY_OF_MONTH, 5);
    System.out.println(calendar2.getTime().toLocaleString());

    //4add方法修改时间
    calendar2.add(Calendar.HOUR, -1);
    System.out.println(calendar2.getTime().toLocaleString());

    //5补充方法
    calendar2.add(Calendar.MONTH, 1);
    int max=calendar2.getActualMaximum(Calendar.DAY_OF_MONTH);
    int min=calendar2.getActualMinimum(Calendar.DAY_OF_MONTH);
    System.out.println(max);
    System.out.println(min);
  }
}
```

九、SimpleDateFormat

> - SimpleDateFormat是以与语言环境有关的方式来格式化和解析日期的类。
> - 进行格式化（日期 -> 文本）、解析（文本 -> 日期）。

> 常用时间模式字母：

字母	日期或时间	示例
y	年	2019
M	年中月份	08
d	月中天数	10
H	1天中小时数(0-23)	22
m	分钟	16
s	秒	59
S	毫秒	367

> 案例演示：

```java
public class TestSimpleDateFormat {
  public static void main(String[] args) throws Exception{
    //1创建SimpleDateFormat对象  y 年 M 月
    //SimpleDateFormat sdf=new SimpleDateFormat("yyyy/MM/dd HH-mm-ss");
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy/MM/dd");
    //2创建Date
    Date date=new Date();
    //格式化date(把日期转成字符串)
    String str=sdf.format(date);
    System.out.println(str);
    //解析  (把字符串转成日期)
    Date date2=sdf.parse("1990/05/01");
    System.out.println(date2);
  }
}
```

十、System

> System系统类，主要用于获取系统的属性数据和其他操作。

> 常用方法：

方法名	说明
static void arraycopy(...)	复制数组
static long currentTimeMillis()	获取当前系统时间，返回的是毫秒值
static void gc();	建议JVM赶快启动垃圾回收器回收垃圾
static void exit(int status)	退出jvm，如果参数是0表示正常退出jvm，非0表示异常退出jvm。

> 案例演示：

```java
public class TestSystem {
  public static void main(String[] args) {
    //1 arraycopy：数组的复制
    //src：源数组
    //srcPos:从那个位置开始复制 0
    //dest：目标数组
    //destPos:目标数组的位置
    //length：复制的长度
    int[] arr= {20,18,15,8,35,26,45,90};
    int[] dest=new int[8];
    System.arraycopy(arr, 4, dest, 4, 4);
    for(int i=0;i<dest.length;i++) {
      System.out.println(dest[i]);
    }
    //Arrays.copyOf(original, newLength)
    System.out.println(System.currentTimeMillis());

    long start=System.currentTimeMillis();
    for(int i=-9999999;i<99999999;i++) {
      for(int j=-999999;j<9999999;j++) {
        int result=i+j;
      }
    }
    //2 获取毫秒数
    long end=System.currentTimeMillis();
    System.out.println("用时:"+(end-start));


    new Student("aaa", 19);
    new Student("bbb", 19);
    new Student("ccc", 19);
    //3回收垃圾
    System.gc();//告诉垃圾回收期回收

    //4推出jvm
    System.exit(0);

    System.out.println("程序结束了....");
  }
}
```