Java程序设计

2019春季

彭启民

pengqm@aliyun.com

变量

- 变量是用来存储数据的。在计算机中所有的数据都是存储在内存中的，形象地说，变量就是在内存中预分配一个空间，这个空间里可以根据需要存放不同的数据。变量就是数据的容器，在Java语言中，它也被称为域。
- 定义变量
  - 在内存中拿到一块空间，然后标上自己的名字，这个名字就是变量名。
- 给变量赋值
  - 将一个数、或一个字符，放到刚才分配的空间中去。以后访问A就可以得到5这个数。
- 修改变量的值
  - 修改变量赋值，与给变量赋值类似，只过是这个变量里本身就存在一个值，新的一个值是取而代之。

- 在Java语言中，只要在同一个代码段没有同名的变量，就可以在程序中的任何一个地方定义变量。一个代码块就是两个相对的“{”、“}”之间的部分。
- 建议
  - 做为一个良好的编程习惯，您应该在使用变量的代码块的开始处定义变量。这样利用程序的维护。

- 声明一个变量包括两个步骤：指定存储该变量的数据类型，以及给该变量起一个名称。

- 声明变量的格式：

  类型　变量名1[，变量名2，……，变量名n][=初值]；

变量的赋值

- Java使用赋值操作符“＝”给一个变量赋值例如

```
int age;
age= 18;
```

- 变量可以在声明时直接指定它的值。可以用下面这条语句来替代上面的两行语句。

```
int x = 18;
```

系统默认初始化值

数据类型	初始值
byte	0
short	0
int	0
long	0L
char	'\u0000'
float	0.0f
double	0
boolean	false
所有引用类型	null（不引用任何对象）

- 变量的作用域
  - 每个变量只在定义它的代码块中（包括这个代码块包含的代码块）生效。
- 全局变量在类的整个范围之内都有效
  - 无需初始化，系统自动给变量赋值。
- 局部变量在类中某个方法函数内或某个子类内有效
  - 必须要进行初始化赋值。
- 在Java语言中，即使作用域不同，其定义的变量名字也不能够相同。
  - 提高Java代码的可读性，防止混淆。

- 在Java中，变量的作用域分为四个级别
  - 类级：类级变量又称全局级变量或静态变量，需要使用static关键字修饰，在类定义后就已经存在，占用内存空间，可以通过类名来访问，不需要实例化。
  - 对象实例级：就是成员变量，实例化后才会分配内存空间，才能访问。
  - 方法级：在方法内部定义的变量，即局部变量。
  - 块级：定义在一个块（大括号包围的代码）内部的变量，变量的生存周期就是这个块，出了这个块就消失了，比如 if、for 语句的块。

```
public class TestVar{

   int a=20;

    public static void main(String[] args){
        TestVar var=new TestVar();
        var.print();
    }
    void print(){
        System.out.println("变量 a="+a);
    }
}
```

运行结果：
变量 a=20

```
public class VarDemo{
    public static String name = “变量作用域例程”;  // 类级变量
    public int i; // 对象实例级变量

    // 初始化块
    {
        int j = 2;// 块级变量
    }

    public void method1() {
        int j = 3;  // 方法级变量
        if(j == 3) {
            int k = 5;  // 块级变量
        }
        // 块级变量只能在块内部访问
        System.out.println("name=" + name + ", i=" + i + ", j=" + j);
    }

    public static void main(String[] args) {
        // 不创建对象，直接通过类名访问类级变量
        System.out.println(VarDemo.name);

        // 创建对象并访问它的方法
        VarDemo demo= new VarDemo();
        demo.method1();
    }
}
```

常量

需要声明某个变量是一个常量时，使用final关键字来限定。

```
final double pi = 3.14159;
pi = -5.0; //编译时出错!
final int x //空常量值
x = 12; //只能初始化一次
x = 100 //编译时出错!
```

注意：试图修改一个常量的值会造成编译错误。同时，常量只能被初始化一次

- 枚举 （1.5版本）
  - 列出有穷序列集的所有成员。
  - 继承自java.lang.Enum类(EnumSet、EnumMap)
  - 枚举类型对象之间可用“==”进行比较取值 //equals方法
- 例：

构建一个表示色彩的枚举，并赋值：

```
public enum MyColor{ Red, Yellow, Blue }
    MyColor color = MyColor.Red;

    for ( MyColor mycolor : MyColor.values() )
        System.out.println( mycolor );
//name()
//ordinal()
//toString()….
```

- 表达式
  - 表达式是运算符与操作数的结合，它是Java语言重要的组成部分。
  - 　一个表达式是可以由多个运算符构成的，它就象我们在儿时就学到过的“四则计算”一样。

- 例如：

  y=x-z*5/2

  3*（2+6/3）

运算符

- 算术运算符
  - 用来进行算术运算的符号。这类运算符是最基本、最常见的，它包括：
    - + 执行加法运算 5+5 （等于10）
    - - 执行减法运算 6-2 （等于4）
    - * 执行乘法运算 5*7 （等于35）
    - / 执行除法运算 8/2 （等于4）
    - % 执行模运算（就是求余数） 7%2 （等于1）
    - 幂：Java没有乘幂运算符，必须使用Math.pow(x,a)方法，表示x的a次幂。pow方法的两个参数都属于double类型，返回的值也是double类型。

      例如：

      double a = Math.pow(10,3); //定义了一个变量a，它的值是10的3次幂。

- 赋值运算符
  - 赋值运算符就是用来为变量赋值的。最基本的赋值运算符就是等号“=”。我们在变量那一个小节就已经使用过了：“变量名=值”，这里的“=”就是赋值运算符。
  - 在Java语言中，还提供了一种叫算术赋值运算符：
    - +=　　X += 5 等价于 X = X+5
    - -=　　X -= 5 等价于 X = X-5
    - *=　　X *= 5 等价于 X = X*5
    - /=　　X /= 5 等价于 X = X/5
    - %=　　X %= 5 等价于 X = X%5

- 自增/减运算符
  - 自增运算符“++”，使变量加1
    - X ++ 等价于 X+=1、X=X+1；
  - 自减运算符“--”，使变量减1
    - X-- 等价于 X-=1、X=X-1；
  - 自增/减运算符放在变量的左边或者右边将使得到值完全不同，完成的功能也完成不同。

自增自减运算符

- 自增（++）和自减（--）操作符都是单目运算符，即只有一个操作数的运算符。
- 两种用法：
  - 前置运算——运算符放在变量之前。
    - 格式：＋＋变量；－－变量。
    - 前置运算先使变量的值增（或减）１，然后再以变化后的值参与其它运算，即先增减、后运算。
  - 后置运算——运算符放在变量之后。
    - 格式：变量＋＋、变量－－
    - 后置运算时，变量先参与其它运算，然后再使变量的值增（或减）１，即先运算、后增减。

关系运算符

- 用于大小关系比较的运算符有4个：

  < 、<=、>、>=

- 用于相等关系的运算符有2个：

  ==、!=

- 大小关系运算符的优先级别高于相等关系运算符。

- 例如：

```
int a=4,b=6;
boolean y,c=true;
y=a>b!=c;
```

- 用来连接两个或两个以上布尔型变量或表达式。
- 布尔运算的操作数和运算结果都是布尔型的量

- 关系运算符
  - “= =”：比较两个对象是否相等；
  - “>”：大于
  - “<”：小于
  - “>=”：大于等于
  - “<=”：小于等于

- 逻辑运算符
  - 又称为布尔运算符，是用来处理一些逻辑关系的运算符，它最经常应用于流程控制。

- 在Java语言中提供了四种逻辑运算符：
  - 与运算符：“&&”
  - 或运算符：“||”
  - 非运算符：“！”
  - 异或运算符：“^”

- 或运算符“||”

X	Y	X||Y
true（真）	true（真）	true（真）
true（真）	false（假）	true（真）
false（假）	true（真）	true（真）
false（假）	false（假）	false（假）

- 与运算符“&&”

X	Y	X&&Y
true（真）	true（真）	true（真）
true（真）	false（假）	false（假）
false（假）	true（真）	false（假）
false（假）	false（假）	false（假）

- 非运算符“！”

X	!X
true（真）	false（假）
false（假）	true（真）

- 异或运算符“^”

X	Y	X^Y
true（真）	true（真）	false（假）
true（真）	false（假）	true（真）
false（假）	true（真）	true（真）
false（假）	false（假）	false（假）

- 按位运算符
  - &（加）
  - |（或）
  - ^（异或）
  - ~（否）
  - >>向右移位
  - <<（向左移位）
  - > >>（用零来填充位于顶部的位）

- &&
  - 如果第一个操作数被判定为“假”，系统不再判定或求解第二个操作数。因为对于与运算，只要有一个操作数为假（false），其结果即为假（false）。
- &
  - 无论第一个操作数是真还是假，都会求解第二个操作数，然后根据两个操作数的值计算最终的结果。
- ||
  - 如果第一个操作数被判定为“真”，系统不再判定或求解第二个操作数。因为对于或运算，只要有一个操作数为真（true），其结果即为真（true）。
- |
  - 无论第一个操作数是真还是假，都会求解第二个操作数。

例 “&&”和“&”的区别

```
public class AndDemo {
   public static void main(String[] args) {
        int n1=1,n2=2,n3=3,n4=4;
          boolean x=true,y=true;
          boolean z=(x=n1>n2)&&(y=n3>n4);
          System.out.println("&&:
   x="+x+",y="+y+",z="+z);
          x=true;y=true;
          z=(x=n1>n2)&(y=n3>n4);
          System.out.println("&:
   x="+x+",y="+y+",z="+z);
   }
}
```

- 运行结果如下：

```
&&: x=false,y=true,z=false
&: x=false,y=false,z=false
```

移位操作符

- 在Java中有三种移位操作：一个左移位(<<)和两个右移位(>>和>>>)。移位操作是通过对一个和整型数相对应的二进制数值的移动来实现的。
- 将一个整型数向左移位(<<)将使二进制的最低位补上一个0
- 向右移动操作符将二进制数据向右移动。“>>”是带符号右移，而“>>>”是不带符号右移。不带符号右移在数值向右移动的时候，无论符号位是什么，总是以0来填补。

三目运算符

- 三目运算符在使用时需要三个操作数，它是一个if /else控制结构的简写方式。格式如下：

(表达式1)？(表达式2)：(表达式3)

- 执行顺序：先求解表达式1，如果为真，则表达式2的值就作为整个表达式的值；如果表达式1的值为假，则表达式3的值就是整个表达式的值。

- 例如：

```
max = (a>b) ? a : b;
```

按照运算符的优先级，也可写作：

```
max = a>b ? a : b;
```

如果a>b，则max=a；否则max=b。执行结果是把a和b中的较大者赋值给max。

```
int a=3,b=4;
int max=a>b?a:b;
```

max的值为4。

- 三目条件运算可以嵌套
- 例：
  - 如有以下的语句，则max表示的是a、b、c三个数中的最大值，其值为5。

```
int a=3,b=4,c=5;
int max=(a>b ? a:b)>c ? (a>b?a:b):c;
```

其他运算符

- **对象运算符（instanceof)**
  - 对象运算符instanceof用来判断一个对象是否是某一个类或者其子类的实例。如果对象是该类或者其子类的实例，返回true；否则返回false。
- **()**
  - 括号运算符（）的优先级是所有运算符中最高的，所以它可以改变表达式运算的先后顺序。在有些情况下，它可以表示方法或函数的调用。
- [ ]
  - 方括号运算符[ ]是数组运算符。
- **．运算符**
  - ．运算符用于访问对象实例或者类的类成员函数。
- new**运算符**
  - new运算符用于创建一个新的对象或者新的数组。

运算符的优先级与结合性

- 优先级决定了同一表达式中多个运算符被执行的先后次序，如乘除运算优先于加减运算，同一级里的运算符具有相同的优先级。运算符的结合性则决定了相同优先级的运算符的执行顺序。

优先级	运算符	结合性
高 ↓ 低	. [] ()	从左至右
	++ −− + − ~ !	从右至左
	* / %	从左至右
	+ −	从左至右
	<< >> >>>	从左至右
	< > <= >= instanceof	从左至右
	== !=	从左至右
	&	从左至右
	|	从左至右
	&&	从左至右
	||	从左至右
	? :	从右至左
	= *= /= %= += −= <<= >>= >>>= &= ^= |=	从右至左

结构化程序设计

- Java语言是面向对象的语言，但在局部的语句块内部，仍需要借助于结构化程序设计的基本流程结构来组织语句。结构化程序设计的基本原则是：任何程序都可以且只能由三种基本流程结构构成，即顺序结构、分支结构和循环结构。

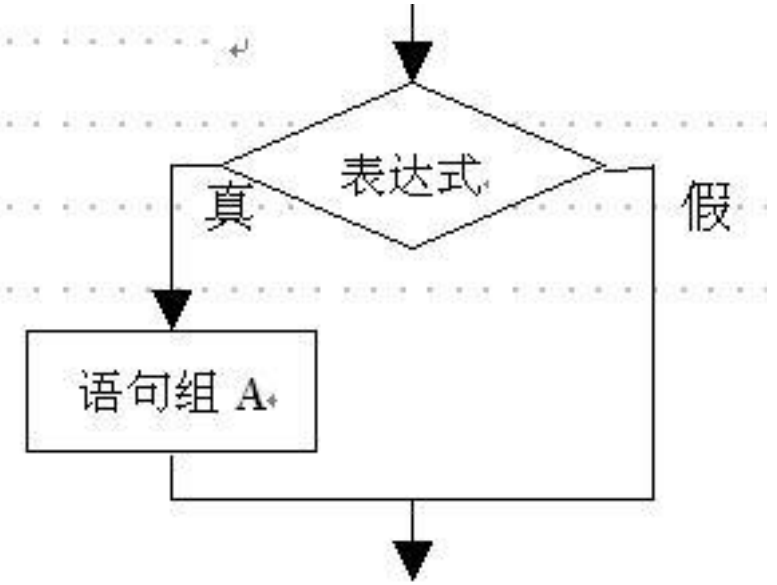
控制流程

- 控制流程是任何一种程序语言都要提供的，Java提供的控制流程和许多其他的程序语言基本上是相同的。
- 条件语句：
  - if (条件) 语句;
  - if (条件) { 代码块 }
  - if (条件) 语句; else 语句;
  - if (条件) { 代码块 } else { 代码块 }

if条件语句

- if语句的一般格式：

```
if(表达式)
        {语句组A;}
    [else
        {语句组B;} ]
```



- if语句的执行过程：

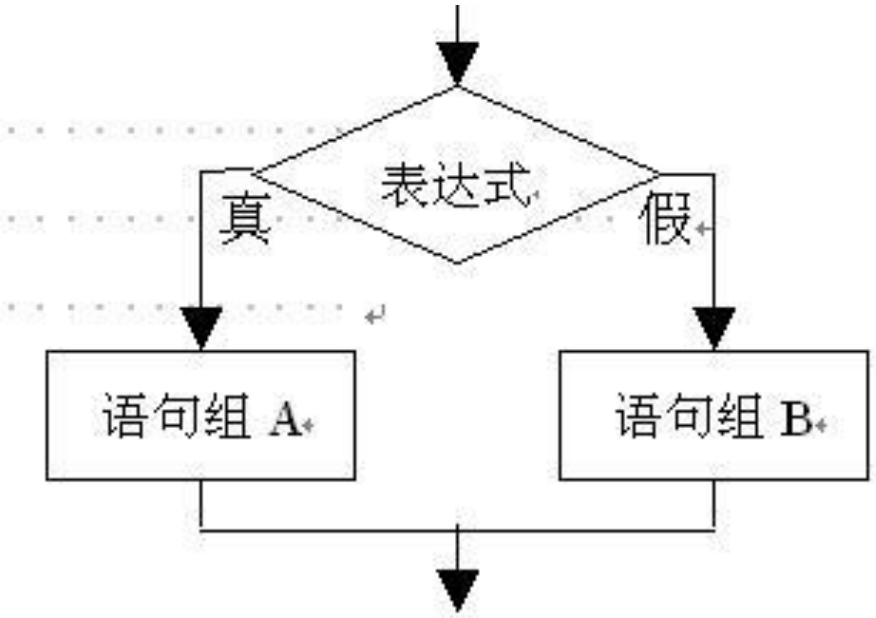
（1）缺省else子句时

例如：if(a>b) System.out.println(“max=“+a);

if条件语句

（2）指定else子句时

例如：
```
if(a>b)    System.out.println("max="+a);
else       System.out.println("max="+b);
```



if语句的嵌套

一般形式：

```
if（表达式1 ）
    if（表达式2 ）
    {语句组A;}
  else
    {语句组B;}
else  if（表达式3)
        {语句组C;}
        else
        {语句D;}
```

例：输入学生的分数score（0～100），如果90≤score≤100，输出“优秀”，如果80≤score<90，输出“良好”，如果70≤score<80，输出“中等”，如果60≤score<70，输出“及格”，如果0≤score<60，输出“不及格”，如果输入分数超出范围，输出“输入错误”。试编写Java应用程序。

```
public class Score{
        public static void main(String[] args) throws
    java.io.IOException{
            String str="";    //声明字符串
        char c=' ';       //声明字符型变量
        int score;        //声明整型变量
            System.out.println("请输入分数:\n");
        while(c!='\n')     //如果按回车键，则退出循环
        {
                c=(char)System.in.read(); //从键盘输入一个字符赋
    值给c
                str=str+c;     //将字符添加到str的末尾
                str=str.trim();//除去字符串开始和尾部空格
        }
                score=Integer.parseInt(str); //将字符串转换为整型
    数
```

```
if(score>=90 & score<=100) System.out.println("优
    秀");
        else if(score>=80 & score<90)
    System.out.println("良好");
            else if(score>=70 & score<80)
    System.out.println("中等");
                else if(score>=60 & score<70)
    System.out.println("及格");
                    else if(score>=0 & score<60)
    System.out.println("不及格");
                        else  System.out.println("输入错误
    ！ ");
    }
}
```

运行结果如下：

请输入分数:

78

中等

- 多重分支：

```
switch(choice)
    {
        case 1:
            ...
            break;
            case 2:
            ...
            break;
            default:
            ...
            break;
    }
```

- default从句是可选的。switch语句只能对char类型或除了long之外的其他整数类型进行测试。

- switch语句后面的控制表达式的数据类型
  - byte、short、char、int
  - 枚举类型enum
  - Java.lang.String类型（Java 7）
    - 不能是StringBuffer或StringBuilder
  - 不能是boolean类型。

switch语句

witch语句格式：

```
switch（表达式）
{
  case 常量1: 语句组1; break;  //分支1
  case 常量2: 语句组2; break;  //分支2
  ……
  case 常量n: 语句组n; break;  //分支n
  [default: 语句组n+1; ]       //分支n+1
}
```

switch、case、default为关键字，break语句是switch语句的出口。

例：用switch语句实现

```
public class Sample1_13 {
        public static void main(String[] args) throws
    java.io.IOException{
            String str="";    //声明字符串
        char c=' ';       //声明字符型变量
        int score;        //声明整型变量
            System.out.println("请输入分数(0-100):");
        while(c!='\n')    //如果按回车键，则退出循环
        {
             c=(char)System.in.read(); //从键盘输入一个字符赋值给
    c
              str=str+c;     //将字符添加到str的末尾
             str=str.trim();//除去字符串开始和尾部空格
        }
```

```
        score=Integer.parseInt(str); //将字符串转换为整型数
                switch(score/10){
                  case 10:
                  case 9: System.out.println("优秀");break;
                  case 8: System.out.println("良好");break;
                  case 7: System.out.println("中等");break;
                  case 6: System.out.println("及格");break;
                  case 5:
                  case 4:
                  case 3:
                  case 2:
                  case 1:
                  case 0: System.out.println("不及格");break;
                  default : System.out.println("输入错误");
                }
    }
}
```

运行结果如下：

请输入分数:

87

良好

注:

switch的执行是按照从小到大的顺序执行的，最后执行default语句。

不管default放在什么 位置，它总是在最后一个处理，然后继续向下处理！

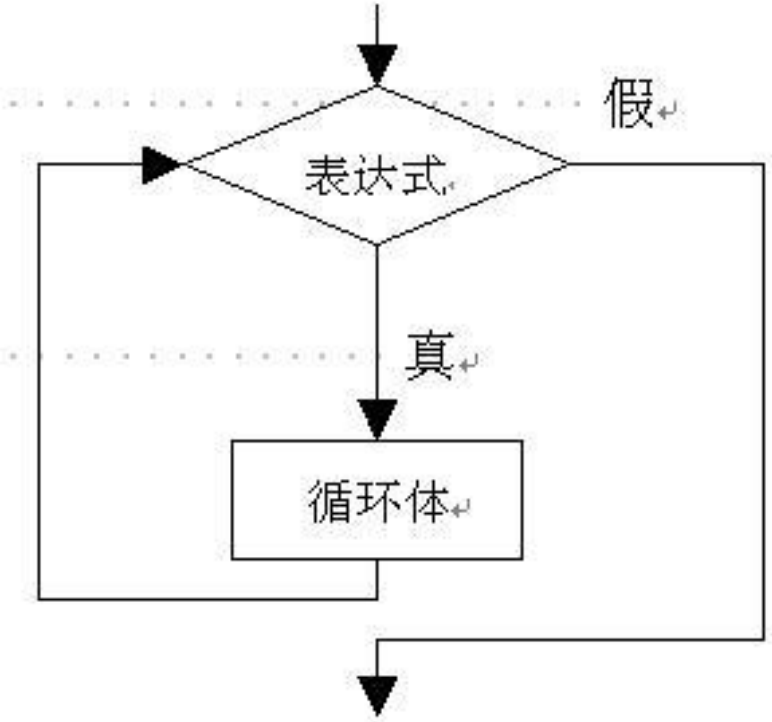
- 不确定循环：
  - while (条件) { 代码块 }
  - do { 代码块 } while (条件);

while循环

while循环语句的基本结构如下：

```
while(表达式)
{
    循环体
  }
```

表达式返回的值为布尔型。



例：求 1+2+3+4+5+……+100。

```
public class Sum {
    public static void main(String[] args) {
        int n=100,sum=0,i=1;   //变量定义及初始化
        while(i<=n)
        {              //循环体
                sum=sum+i; //求和
                i++;       //循环变量更新
        }
        System.out.println("sum="+sum);   //输出运算结果
    }
}
```

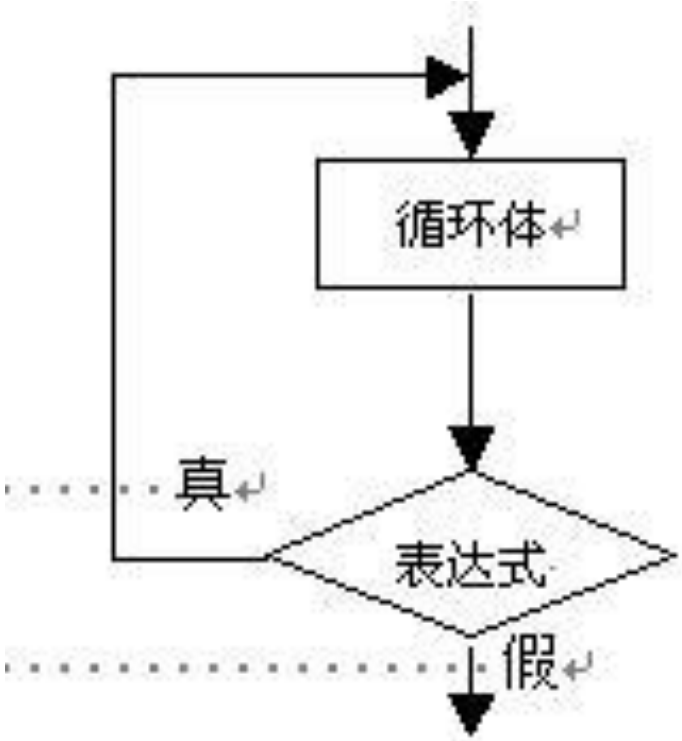
运行结果如下：

sum=5050

do-while循环

- 语法结构如下：

```
do
{
循环体
}while(表达式);
```



- 确定循环：
  - for(int i=1; i <= 10; i++) { 代码块 }
  - 在for语句的第一个位置声明了一个变量后，这个变量的作用域会扩展到for循环主体结束。但循环之外是不能使用这个变量的。

例 嵌套for循环示例()

```
char current ='A';          //声明并初始化变量
  current
      for(int row = 1; row <= 3; row++)  //外循环
      {
          for(int column = 1; column <= 10;
  column++)  //内嵌循环
          {
              System.out.print(current + " ");
              current++;          //当前字符值自增1
              if(current>'Z') break;
              //当前字符值 大于Z时，退出循环
          }
      System.out.println();     //换行
      }
```

- Java1.5、1.6提供了增强的for循环
  - 简化迭代器的书写格式。
    - 底层使用了迭代器(iterator)遍历。
  - 适用范围
    - 实现了Iterable接口的对象
    - 数组对象
  - 格式：

for(数据类型 变量名 :遍历的目标){ //数据类型 变量名:声明一个变量用来接收遍历目标遍历后的元素

}

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
/**
 * 增强for循环
*/
public class NewForTest {

    public static void main(String[] args) {
        List<Integer> list = new LinkedList<Integer>();
        for (int i = 0; i < 100; i++) {
            list.add(i);
        }

        int sum1 = 0;
        for (int i = 0; i < list.size(); i++) {
            sum1 = sum1 + list.get(i);
        }
        System.out.println(“普通循环计算结果”+sum1);
        int sum2 = 0;
        for (int c : list) {
            sum2 = sum2 + c;
        }
        System.out.println(“增强for循环计算结果”+sum2");
    }

}
```

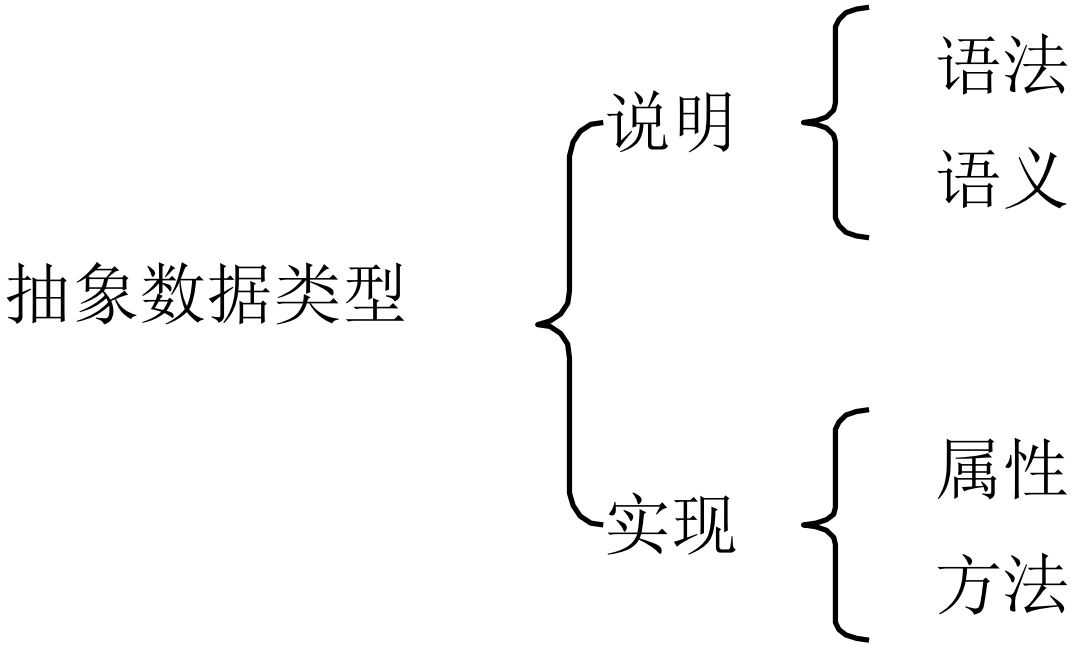
- 增强的for循环
  - 注意事项：
    - 在使用增强for循环变量元素的过程中不准对所用集合对象的元素个数进行修改。
      - 获取迭代器由jvm完成
    - 迭代器遍历元素与增强for循环变量元素不同
      - 使用迭代器遍历集合的元素时可以删除集合的元素
      - 增强for循环变量集合的元素时不能调用迭代器的remove方法删除元素
    - 普通for循环与增强for循环的区别
      - 普通for循环可以没有遍历的目标
      - 增强for循环一定要有遍历的目标

类、对象与方法

所有的物体都可以看作对象，对象有一定的框架结构，具有一定的功能，完成一定的任务，而且这些对象之间可以建立起联系，从而就像人类社会那样处理各种各样的事务。

抽象

- 一个抽象数据类型（Abstract Data Type，ADT）可分成4个部分。



类的三个基本特性：

封装性

继承性

多态性

类包括两个成员要素

- 字段/域(field)
  - 类中的数据成员,定义了类所需要的数据,**描述类是什么**。
- 方法(method)
  - 类中的成员函数,方法定义了类的功能,**描述类做什么。**

对象是类的一个实例

- 在程序中创建—个对象将在内存中开辟一块空间，其中包括该对象的属性和方法。
- 创建对象使用关键字运算符**new**。
- 而相对于简单变量，对象就是复杂的既包括数据又包括方法代码的复杂数据单位。

- 类是同种对象的抽象综合，是创建对象的模板
  - 创建对象以某个类为模板，这个类里定义了哪些属性和方法，新建的对象就必然在内存里拥有相应的属性和方法。
  - 相对于简单数据类型，类就是用户自己定义的复杂的抽象数据类型

Java程序的基本结构

```
Package xxxx
Import yy.zz;
[修饰符]class 类名1
{
零个到多个构造器定义..
零个到多个属性…
零个到多个方法…
}

[修饰符]class 类名n
{
零个到多个构造器定义..
零个到多个属性…
零个到多个方法…
}
```

Java程序是由一个个类定义组成。

类的基本定义方法：

```
[修饰符] class 类名 [extends 父类] [implements 接口名]
{
    类成员变量声明
    类方法声明
}
```

类里各成员之间的定义顺序没有任何影响，各成员之间可以相互调用。
static修饰的成员不能访问没有static修饰的成员。

- class 定义数据类型
- new 创建类的一个对象

```
class Student {
  String id;
  String firstname;
  String lastname;
  Date birthday;
  byte gender;
}

Student a = new Student();
```

用引用操纵对象

- 一切都是对象，但操纵的标识符实际上是“引用”（reference）
- 一个引用并不一定需要有一个对象与它关联

  例：　Cat c;
- 可在创建一个引用的同时进行初始化

  例：　Cat c = new Cat(“kitty”);

- **关于类头---class**
  - **class**关键字，在它的后面应跟随新数据类型的名称。
  - 父类名跟在**extends** 关键字后面，说明当前类是哪个已经存在类的子类，存在**继承关系**。
    - 继承是类与类之间的一种非常重要的关系。
  - 接口名跟在**implements**关键字后面，用来说明当前类中实现了哪个**接口**定义的功能和方法。

- **关于类头---类的修饰符**
  - 类的修饰符用来说明类的特殊性质。
- 分为三种：
  - 访问控制符：**public 公共类**
  - 抽象类说明符：**abstract**
  - 最终类说明符：**final**

类成员(字段)的声明

［域修饰词］ 类型　变量名或带初始化的变量名列表；

类成员(字段)类型

- 基本类型
- References－引用数据类型变量，使用方法和基本数据类型变量类似，只不过其对应的存储单元内存放的是引用。
  - 指对象类型
  - 预定义
  - 应用定义

例:

```
class DataOnly {
  int i;
  float f;
  boolean b;
}

DataOnly d = new DataOnly();
d.i = 47;
d.f = 100.0f;
d.b = true;
```

类方法

- 类的方法，又称为**成员函数**，是JAVA创建的有名字的子程序。
- 类中定义的方法通常起到两种作用
  - 围绕着类的属性进行各种操作；
  - 与其他的类或对象进行数据交流、消息传递等操作。

- 与普通函数和过程不同
  - 方法只应用于特定类及其祖先类的对象。
  - 方法是类的成员，并有自己的可访问性。
- 在一个类中，程序的作用体现在方法中。
  - 一个主方法(public static void main(…))和若干个子方法构成。
  - 主方法调用其他方法，其他方法间也可互相调用，同一个方法可被一个或多个方法调用任意次。

方法

- 基本组成部分包括方法头与方法体两部分：

[**修饰符**]**返回类型**方法名（参数列表） **throws** 例外名1，例外名2……

{

方法体：

局部变量声明；

语句序列；

}

方法签名

- 方法签名也称为方法声明。例如main()方法的签名就是：

```
public static void main(String [ ] args)
```

- 访问控制符的值可以是：private,public,protected,和default
- 可选控制符的值可以是：static,final,abstract,native和synchronized
- 方法的返回值可以是基本数据类型，也可以是对象的引用。
- 方法名必须符合标识符的规定。
- 参数列表

- 返回值类型
  - 当一个方法被调用时，它可以返回一个值给该方法的调用者。我们可以通过定义方法的“返回值类型”来规定它返回的值的数据类型。这个数据类型可以是任何一个有效的Java数据类型，或者void。
  - 如果返回值类型为void，则说明这个方法调用后不返回任何值。

- 参数列表（Parameter list）
  - 是可选项，作用与函数的参数类似，利用它在调用该方法时就可以传递给该方法想要处理的数据。
  - Java的参数传递都是传值形式
  - 每一个参数的定义和变量的定义类似，即一个数据类型，加上一个标志符，多个参数之间用逗号隔开。

不定长度参数

- J2SE5.0之后开始支持
- 例:

```
public int sum(int… nums){
    int sum = 0;
    for(int num : nums) sum += num;
    return sum;
}
```

- 不定长参数必须为参数表的最后一个
- 每个方法只能有一个不定长参数

- 方法重载
  - 例：求两个整数的最大值方法

    MaxOfInt（int a ，int b）

    在编写一个求两个浮点型最大值的方法

    MaxOfFloating (float a，float b)

    ……

    希望的调用形式： Max(a, b)

方法

- 方法重载Method overloading
  - 许多方法具有相同的名称（方法名相同）
  - 系统如何区分这些名称相同的方法？
    - 根据参数的不同来决定调用哪个方法
      - 参数的个数
      - 参数的类型
- 方法重载使用原则
  - 把不同类型数据的类似操作可取相同名

编写方法体

- 一个完整的方法是由方法签名和方法体共同组成的，方法体以一对大括号表示。在方法体内，放置具体的实现该方法的代码。
  - 如果该方法有返回值，则必须在方法中书写return语句，返回某个值。

```
public String getName() //方法签名
{
  return name;        //方法体
}
```

例：声明方法体

```
class Department
{
    int m_ DeptNo;              //部门编号
    String  m_DeptName;  //部门名称
    int m_DeptTotalEmp;  //部门雇员数
    ManagerEmployee m_DeptMgr;  //部门主管
    int getDeptNo( )   //获取当前对象的部门号
    {   return _DeptNo;
     }   //返回这个对象的部门号
}
```

例

```
boolean setDeptNo(int NewNo)  //把当前部门号重新设置称参数值
{
    if( NewNo >0)                    //若给出的新部门号合法
  {
      m_DeptNo=NewNo；//修改类属性m_DeptNo
       return true；          //返回true，声明部门编号修改成功
  }
    else return false; //若形式参数给出新部门号非法，返回false
  }
```

方法调用

- 通过.(dot)来调用某个方法。
  - 调用其他类中的方法必须指定类名或对象名：

    类名（对象名）.**方法名（实际参数）**
  - 在同一个类中调用方法，可直接使用如下的格式来调用方法：

    方法名（实际参数）
  - **每一个方法都有一个隐含的参数，称为this，**它引用作为方法调用主体的对象。

方法调用

```
1.  public class Test {
2.      public static String name="张三";

3.      public static void main(String[]args)
4.      {
5.         this.getName();
6.      }

7.      public static String getName()
8.      {
9.         return name;
10.     }
11.}
```

```
1.  public class Test {
2.      public static String name="张三";

3.      public static void main(String[]args)
4.      {
5.         getName();

6.      }

7.      public static String getName()
8.      {
9.         return name;
10.     }
11.}
```

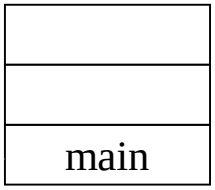
Java 8 允许使用 :: 关键字来传递方法或者构造函数引用
用 ClassName::new 来获取ClassName类构造函数的引用

```
detect = someth::firstLetter;
String fl = detector.detect(“Test");
System.out.println(fl);    // “T"
```

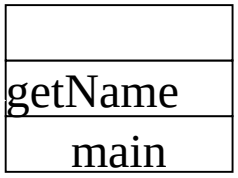
Java应用程序的执行从main函数开始，调用其他方法后又回到main函数，在main函数中结束整个程序的运行。

方法调用栈

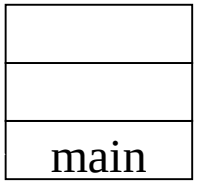
- 系统中存在一个专门用来维护方法调用的栈，称之为调用栈（call stack）
  - 当前在执行的方法是处于栈顶位置的方法。如果该方法完成了它的执行操作，那么它将从栈中删除，控制流就返回到栈中下一个方法的中去了。如果在执行该方法的过程中，调用了一个新方法，新方法就会被添加到栈中，成为栈顶元素



开始执行main方法



中断main方法调用getName方法



完成getName重新调用main方法

方法调用栈

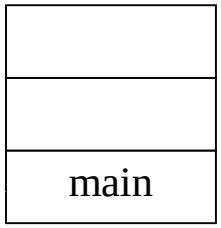
- 一个方法被调用执行，它就会添加到调用栈中成为栈顶元素，并开始执行，直到它完成。有三种会导致方法执行的完成
  - 方法执行时被return返回了，此时方法的执行就完成了，它向调用者返回一个基本类型数值或是一个引用
  - 当方法没有返回值，则在执行完方法体的最后一条语句时，该方法就执行完成，并从调用栈中删除，进而获取新的栈顶方法来处理
  - 方法抛出一个异常，该异常将告知调用者，该方法也执行完毕，

递归方法(Recursion)

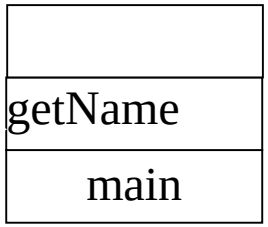
在getName方法中调用自己

```
public class Test {
    public static String name="张三";
    public static void main(String[]args)
    {
        getName();
    }
    public static String getName()
    {
        getName();
        return name;
    }
}
```

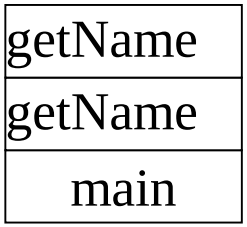
递归方法(Recursion)



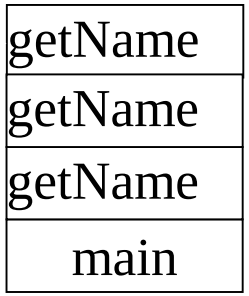
开始执行main方法



中断main方法调用getName方法



中断getName调用getName新副本

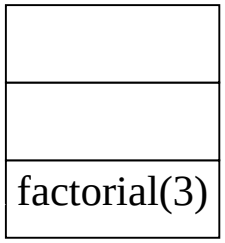


中断getName调用getName新副本

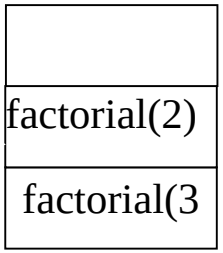
递归应用示例

```
public long factorial( long number )
{
     // 基本情况
     if ( number <= 1 )
        return 1;
     // 递归步骤
    else
        return number * factorial( number - 1 );
} // end method factorial
```

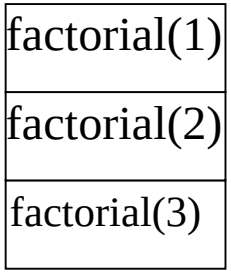
递归应用示例：factorial(3)



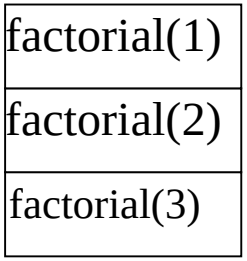
执行factorial(3)方法



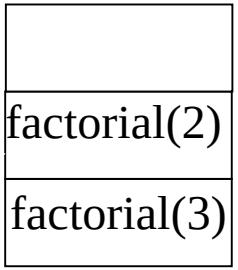
中断f(3)方法调用副本f(2)方法



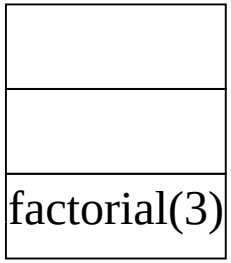
中断f(2)调用f(1)新副本



f(1)是基本情况，返回



根据f(1)的返回值，f(2)得到求解



根据f(2)的求解，f(3)可以计算结果

- 调用一个**实例方法**的语法格式是：

  对象名.方法名(实际参数);

- 还有一种通过类就可以直接访问的**静态字段**和**静态方法**，这种静态的字段和方法用static关键字标识，不需要创建实例就可以通过类直接访问。

  System.out.println(“Hellow, world!”);