Day 04

2018年5月3日 8:36

课程回顾

1.类型转换

大类型 字节大/多 表达的数据大

小类型 字节小/少 表达的数据小

相对而言

byte -->short --->int---->long

float----> double

char---> int--->long

```
long l=100l;
int i=(int)l;//强制转换
```

极特殊的情况:

```
short s='a';

char c='a';
short s1=c;//错的
```

short 范围 -32768----32767

char范围 0-------65535

2.算数运算符

3.赋值运算符

4.逻辑运算符

5.关系运算符

6.位运算符

7.三元运算符

运算符的优先级:从高到低

() ++ -- ~ ！ 算数运算符 << >> >>> 关系运算符 && || ^ 三元 赋值

8.程序的执行流程

-顺序结构

-选择分支结构

-循环结构

9.if结构 选择分支结构

```
if(条件){
}

if(条件){
}else{
}

if(条件){
    if(条件){
    }else{}
}else{
     if(条件){
    }else{}
}
```

多重if结构

```
    If(条件1){
        //代码块
    }else if(条件2){
        //代码块
    }else if(条件3){
        //代码块
    }else if....
```

实际上是if嵌套的一个特例

说明:多重if结构,一般情况下应用场景,在一整段中分阶梯,一大段分若干小段

switch分支结构:

语法:

```
  switch(表达式){
  case 常量1:
        //代码块
        break;//可以省略
   case 常量2:
        //代码块
        break;//可以省略
```

case 常量n:
//代码块
break;//可以省略
default:
//代码块
break;
}
说明:
-switch结构是分支结构,多分支机构,处理固定值的分支
-switch(表达式) 表达式结果可以是byte,char,shot,int,枚举enum
还可以是String类型,jdk版本1.7及以上
-表达式的结果是一个固定的值,
-这个固定的值会逐一匹配case后面的常量值,
如果匹配上了,会执行该case后面的代码块,
如果有break语句就直接跳出switch结构,执行switch后面代码语句
如果没有break语句就继续执行当前case后面的所有的case后的代码语句
如果没有匹配上,就会执行default后面的语句块
-default是所有的case都不满足情况下执行default后的语句
default块可以放在所有case的前面,建议加上break;

switch和if的区别
if既可以做范围判断,也可以做定值判断
switch只能做定值判断
能用switch写的代码,就一定可以用if来写
能用if写的代码,不一定能用switch写

循环结构:while循环,do while循环,for循环
循环一定是有规律的,没有规律,绝对不能做成循环
如果想推断出循环的规律,需要推出前三次循环,就可以找出规律
建议用文本写出前三次循环
循环慎用,原因,循环及其浪费cpu

while循环:
常规:-循环变量初始值

-循环的条件

-循环变量的增量或减量

-循环体

不常规:while适合与写死循环,一定要一个if,根据if条件 break出循环

在不知道循环次数的前提下可以用while(true)

```
while(true){
    if(条件){
        break;
    }
}
```

语法:

```
循环变量的初始值//可以是省略
while(条件){
    //循环体代码块
    // 循环变量的增量或减量//可以省略
}
```

说明:

while(条件) 条件表达式结果一定布尔值,逻辑运算符,关系运算符

当条件为真值时,会进入循环体,指行循环体代码

指行完循环体代码后,再次回到条件,判断条件是否为真值

如果为真值则继续循环

当条件为假值false,就跳出循环,执行while(){}后面的代码块语句

先判断条件,后循环

do-while循环:

语法:

```
do{
    //循环体代码块
}while(条件);
```

说明:

也一样需要有规律

一定是先执行一次循环,然后再判断条件

不适合做死循环

for循环:常用的循环

for循环适合有次数的循环

无次数的循环用while(true)

语法:

```
for(循环变量初值;条件;增量或减量表达式){
    //循环体代码块
}
```

说明:

-必须有两个分号,不能缺失

- 循环变量初值,可以省略不写,可以写多个变量定义且赋初值,多个变量用逗号间隔

```
for(int i=0;;)
for(int i=0,j=10;;)
```

-条件 返回结果是布尔类型 关系表达式,逻辑表达式

条件可以不写,不写就是默认true

for(;;) 等同于 for(;true;) 但是不建议,应该用while(true)

条件为true 执行循环体

条件为false 执行for(){}后面的语句

- 增量或减量表达式,可以不写,可以写一个或多个表达式,多个用逗号间隔

```
for(int i=0,j=10;;i++,j--)
```

-for循环的执行流程

执行for循环,

首先执行"循环变量初值",且执行一次,在执行第一次循环前做相关初始化工作

然后判断条件,

条件为true 进入循环体

循环体执行完毕,

然后执行"增量或减量表达式"

然后判断条件

条件为true,进入循环体

条件为false就跳出循环体

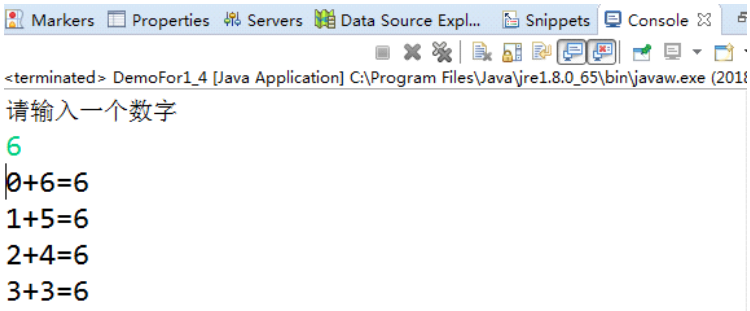
循环往复...

条件为false　跳出循环执行for(){}后面的语句

如何推循环规律:

推前三次循环就一定能找出规律

```
public static void main(String[] args) {
        // TODO Auto-generated method stub
    Scanner  input =new Scanner(System.in);
    System.out.println("请输入一个数字");
    int num=input.nextInt();
  for(int i=0,j=num;i<=j;i++,j--){
     System.out.println(i+"+"+j+"="+(i+j));
  }
  }
```



第一次循环:

当 i=0,j=6　i<=j　0<=6　true　0+6=6　i++,j--　i=1,j=5

第二次循环

当i=1,j=5　i<=j　1<=5　true　1+5=6　i++,j--　i=2,j=4

第三次循环

当 i=2,j=4　i<=j　2<=4　true　2+4=6　i++ ,j--　i=3,j=3

最后找一次临界值

break:应用switch中用于终止switch

应用循环(for,while,dowhile)中,用于终止循环

break　终止当前循环(常用)

break 标识符　终止标识符指定的循环(不建议使用)

continue :应用在循环(for,while,dowhile)中,用于结束本次循环,继续下一次循环

continue　继续下一次循环(常用)

continue 标识符 继续下一次标识符指定的循环(不建议使用)

循环嵌套:循环中中嵌套其他的循环(for while dowhile)

外层循环一次,内层循环可能循环多次,内层循环完毕,外层循环进行下一次循环,建议循环嵌套不要超过4层

语法:以for为例子

```
for(){
    for(){
    }
}
```

```
public static void main(String[] args) {
    Scanner input =new Scanner(System.in);
    for(int i=0;i<3;i++){//外层循环控制班级个数
        double sum=0;
        System.out.println("第"+(i+1)+"个班级");
        for(int j=0;j<4;j++){//内层循环控制学生数
            System.out.println("请输入第"+(i+1)+"个班级的第"+(j+1)+"个学生成绩");
            int score=input.nextInt();
            sum+=score;
        }
        System.out.println("第"+(i+1)+"平均分:"+(sum/4));
    }
}
```

第一次外层循环:

当 i=0 i<3 0<3 true i++ i=1

sum=0;

内层循环第一次

当j=0 j<4 0<4 true sum+=score j++ j=1

内层循环第二次

当 j=1 j<4 1<4 true sum+=score j++ j=2

内层循环第三次

当 j=2 j<4 2<4 true sum+=score j++ j=3

第二次外层循环:

当 i=1 i<3 1<3 true i++ i=2

sum=0;

内层循环第一次

当j=0 j<4 0<4 true sum+=score j++ j=1

内层循环第二次

当 j=1 j<4 1<4 true sum+=score j++ j=2

内层循环第三次

当 j=2 j<4 2<4 true sum+=score j++ j=3

第三次外层循环:

当 i=2 i<3 2<3 true i++ i=3

sum=0;

内层循环第一次

当j=0 j<4 0<4 true sum+=score j++ j=1

内层循环第二次

当 j=1 j<4 1<4 true sum+=score j++ j=2

内层循环第三次

当 j=2 j<4 2<4 true sum+=score j++ j=3

小总结:

做程序主要思路

1.获取数据

2.基于数据处理业务

3.显示输出结果

整体是顺序结构

第二步骤处理业务

用到了分支,用到循环,来实现具体业务

现在可以处理复杂的业务,(前提,程序员必须了解业务,不了解业务是写不出代码)

突然发现数据不够用了,一个变量只能存一个数据,如果数据多了,

程序员需要做好多的变量名,名多了,无法记忆变量代表什么数据

java中提出一个数据结构,叫做数组

数组就是一组相同数据类型的数据的集合,用一个数组名来代表

用面向对象的类来实例化对象中可以放置不同类型的数据

用集合来存储若干对象的数据

有了分支和循环可以实现业务,业务需要数据

补充内容:

评判一个程序的的执行效率,时间复杂度, 循环次数

debug跟踪代码的执行流程,程序高手,不是写代码的高手

是调错的高手

如何写一个程序:

遵循先获取数据,处理业务逻辑,输出业务的结果

先尽量分析需求,能分析多少就分析多少,尽量用文字表达出来

根据先期分析的需求写代码,先写着看,不要怕写错,写错了可以debug调试

有可能写完了,调试完了,之后发现思路错了,重新分析需求换一个思路

来实现,如果又错了,劝你买个彩票.

运气好可能思路是对的,但是有小问题,可以通过debug跟踪调试,

来完善代码

1.设置断点 双击要设置断点的行头

程序执行到断点语句就停下来,等待用户逐行执行,以便观察执行的

流程或执行中数据变化是否正确

原则:

首先要猜测可能出现问题的代码范围,在这个范围前设置断点

猜测的原则:

根据控制台的错误提示

根据输出的结果

如果没有任何的提示,只能把断点设置程序入口的第一行

其次,循环往复,用猜测的方式设置断点,逐渐缩小代码的范围

直到找到具体错误行,改正即可

注意:初次学习使用debug,设置断点可能不会猜测的很准

2.程序运行后(debug as),会停止到断点的位置,等待程序员进一步的指令

F5:碰到函数,就进入函数内部

F6:单步执行,如果碰到函数,就直接把函数执行完毕

F8:从执行到当前的行开始"继续执行"

如果后面还有断点,就停到断点处(快速执行当前行到断点处的所有代码)

如果没有断点,就直接执行到程序结束

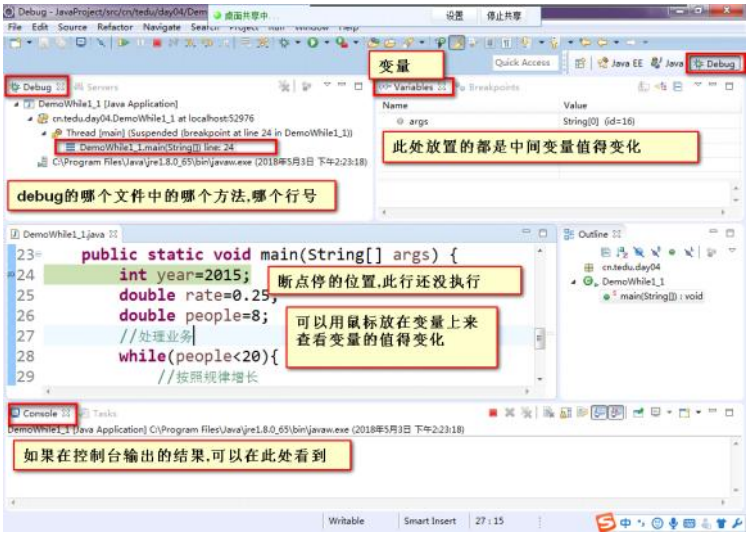
如果想终止程序的执行,点击 "红方块"按钮 ctrl+F2

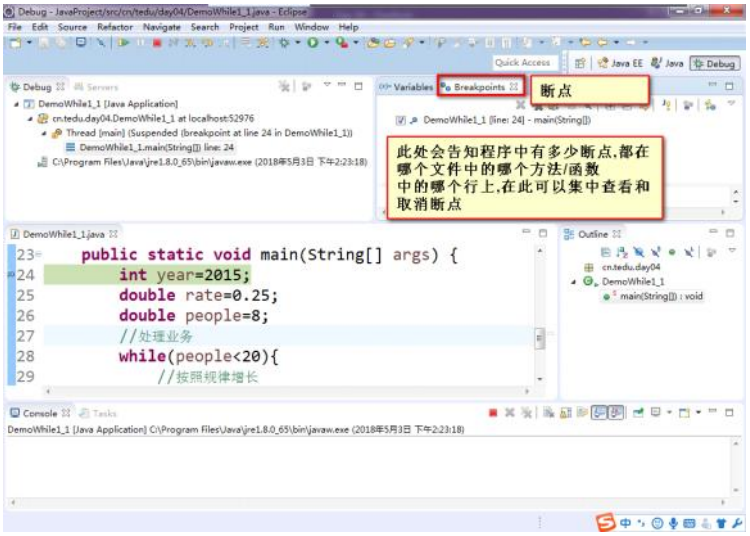
建议:用左手按F5,F6,F8,有右手放在鼠标上,此时鼠标放在变量上,会显示变量的值

3.程序运行后,如果有断点,会提示用户切换debug的视图,

建议在debug的视图下,做debug跟踪调试

如果想切换回JavaEE,在窗口的右上角点击 javaEE图标即可





无论多复杂的程序都可以用debug跟踪调试清楚