layout	title	categories	tags	avatarimg	author
post	Java环境与语法	train	how language java	/img/head.jpg	wangyifan

简介

本文是对如何学习一门编程语言的具体实践-Java语言第一篇。

前面说了基础语法是很简单的，如果有语言基础，比较好的方法是过一遍Learn X in Y minutes。

下文是在公司给毕业生做培训时的PPT，可做参考！

Java环境搭建

- 到Oracle官方网站下载相应操作系统的JDK进行安装
- 打开命令行,输入

```
java -version
```

得到如下信息，则表示搭建成功!

```
java version "1.8.0"
Java(TM) SE Runtime Environment (build 1.8.0-b132)
Java HotSpot(TM) 64-Bit Server VM (build 25.0-b70, mixed mode)
```

Java中需要理解两个路径：

- path是系统查找程序的路径
- classpath是Java查找类的路径

第一个程序

- 创建文件Hello.java
- 输入如下代码

```
//例子代码
```

```
public class Hello{
    public static void main(String[] args){
        int a = 1 + 1;
        int b = 1 + 2;
        System.out.println("(1 + 1) * 3 + (1 + 2) = " + (a * 3 + b));
    }
}
```

- 打开命令行,切换到文件所在路径
- 输入javac Hello.java
- 输入java Hello

代码组成

```
//例子代码
public class Hello{
    public static void main(String[] args){
        int a = 1 + 1;
        int b = 1 + 2;
        System.out.println("(1 + 1) * 3 + (1 + 2) = " + (a * 3 + b));
    }
}
```

- 注释
- 标识符(Hello,args)
- 关键字(public,class,static,void)
- 字面量(字符串) ("(1 + 1) * 3 + (1 + 2) = ")
- 变量(args,a,b)
- 类
- 方法

注释

```
//单行注释
```

```
/*
多行注释
*/
```

```
/**
 * javadoc注释
 */
```

标识符

- 标识符由字母（A-Z或者a-z）,美元符（\$）、下划线（_）或数字组成
- 标识符都应该以字母（A-Z或者a-z）,美元符（\$）、或者下划线（_）开始
- 关键字不能用作标识符
- 标识符是大小写敏感的
- 合法标识符举例：age、\$salary、~value~、_~1value~
- 非法标识符举例：123abc、-salary

关键字

字面量

- 字符串:"Hello"
- 字符:'H','e','l','l','o'
- 布尔值:true,false
- 数值:1,2.4

变量

- 变量就是一个命名的内存块
- 变量只能存储一种具体类型的数据
- 使用变量之前必须声明(变量名和类型)
- 使用前需要初始化
- 局部变量:方法或语句块内部
- 成员变量:方法外,类内部

类和对象

- 类是对一个特定类型对象的描述,它定义了一种新的类型

- 类定义中的成员变量可以是任意类型
- 对象是类的实体,使用关键字new来创建

方法

- 实例方法
- 静态方法(static)
- 方法调用[递归调用]

```
public class Hello{
    public static void main(String[] args){
        System.out.println("Hello");
        Hello.main(null);
    }
}
```

基本类型

使用递归来计算基本类型的范围

```
public class Test{
    public static long p(int num){
        if(num == 0) return 1;
        return 2 * p(--num);  //--num 与 num--
    }

    public static long caculateRange(int num){
        if(num == 0) return 1;
        return p(num) + caculateRange(--num);
    }
}
```

数组

数组是相同类型的,用一个标示符名称封装到一起的一个对象序列或基本类型数据序列

一维数组

```
int[] arr;
int arr[];
//初始化
arr = {1,2,3};
arr = new int[3];  //默认初始化
//使用,下标从0开始
arr[0];
```

多维数组

```
int[][] arr = { {1,2,3},{4,5,6}};
int[][][] arr = new int[2][3][4];
//每一维长度可以不同
```

运算符(操作符)

运算符指明对操作数的运算方式

- 算术运算符
- 关系运算符
- 位运算符
- 逻辑运算符
- 赋值运算符
- 其他运算符

算术运算符

```
+     加法 - 相加运算符两侧的值
-     减法 - 左操作数减去右操作数
*     乘法 - 相乘操作符两侧的值
/     除法 - 左操作数除以右操作数
%     取模 - 右操作数除左操作数的余数
++    自增 - 操作数的值增加1
-     自减 - 操作数的值减少1
```

实例

```
public class Test {
  public static void main(String args[]) {
     int a = 10;
     int b = 20;
     int c = 25;
     int d = 25;
     System.out.println("a + b = " + (a + b) );
     System.out.println("a - b = " + (a - b) );
     System.out.println("a * b = " + (a * b) );
     System.out.println("b / a = " + (b / a) );
     System.out.println("b % a = " + (b % a) );
     System.out.println("c % a = " + (c % a) );
     System.out.println("a++   = " +  (a++) );
     System.out.println("b--   = " +  (a--) );
     System.out.println("d++   = " +  (d++) );
     System.out.println("++d   = " +  (++d) );
  }
}
```

关系运算符

```
==    检查如果两个操作数的值是否相等，如果相等则条件为真
=!    检查如果两个操作数的值是否相等，如果值不相等则条件为真
>     检查左操作数的值是否大于右操作数的值，如果是那么条件为真
<     检查左操作数的值是否小于右操作数的值，如果是那么条件为真
>=    检查左操作数的值是否大于或等于右操作数的值，如果是那么条件为真
<=    检查左操作数的值是否小于或等于右操作数的值，如果是那么条件为真
```

实例

```
public class Test {
  public static void main(String args[]) {
     int a = 10;
     int b = 20;
     System.out.println("a == b = " + (a == b) );
     System.out.println("a != b = " + (a != b) );
     System.out.println("a > b = " + (a > b) );
     System.out.println("a < b = " + (a < b) );
     System.out.println("b >= a = " + (b >= a) );
     System.out.println("b <= a = " + (b <= a) );
  }
}
```

位运算符

```
&     按位与操作符，当且仅当两个操作数的某一位都非0时候结果的该位才为1
|     按位或操作符，只要两个操作数的某一位有一个非0时候结果的该位就为1
^     按位异或操作符，两个操作数的某一位不相同时候结果的该位就为1
~     按位补运算符翻转操作数的每一位
<<    按位左移运算符。左操作数按位左移右操作数指定的位数
>>    按位右移运算符。左操作数按位右移右操作数指定的位数
>>>       按位右移补零操作符。左操作数的值按右操作数指定的位数右移，移动得到的空位以零填充。
```

实例

```
public class Test {
  public static void main(String args[]) {
     int a = 60; /* 60 = 0011 1100 */
     int b = 13; /* 13 = 0000 1101 */
     int c = 0;
     c = a & b;        /* 12 = 0000 1100 */
     System.out.println("a & b = " + c );
     c = a | b;        /* 61 = 0011 1101 */
     System.out.println("a | b = " + c );
     c = a ^ b;        /* 49 = 0011 0001 */
     System.out.println("a ^ b = " + c );
     c = ~a;           /*-61 = 1100 0011 */
     System.out.println("~a = " + c );
     c = a << 2;     /* 240 = 1111 0000 */
     System.out.println("a << 2 = " + c );
     c = a >> 2;     /* 215 = 1111 */
     System.out.println("a >> 2  = " + c );
     c = a >>> 2;     /* 215 = 0000 1111 */
     System.out.println("a >>> 2 = " + c );
  }
}
```

逻辑运算符

```
&&    称为逻辑与运算符。当且仅当两个操作数都为真，条件才为真
||    称为逻辑或操作符。如果任何两个操作数任何一个为真，条件为真
!     称为逻辑非运算符。用来反转操作数的逻辑状态。如果条件为true，则逻辑非运算符将得到false。
```

实例

```
public class Test {
  public static void main(String args[]) {
     boolean a = true;
     boolean b = false;
     System.out.println("a && b = " + (a&&b));
     System.out.println("a || b = " + (a||b) );
     System.out.println("!(a && b) = " + !(a && b));
  }
}
```

赋值运算符

```
=     简单的赋值运算符，将右操作数的值赋给左侧操作数
+=    加和赋值操作符，它把左操作数和右操作数相加赋值给左操作数
-=    减和赋值操作符，它把左操作数和右操作数相减赋值给左操作数
*=    乘和赋值操作符，它把左操作数和右操作数相乘赋值给左操作数
/=    除和赋值操作符，它把左操作数和右操作数相除赋值给左操作数
%=    取模和赋值操作符，它把左操作数和右操作数取模后赋值给左操作数
<<=   左移位赋值运算符
>>=   右移位赋值运算符
&=    按位与赋值运算符
^=    按位异或赋值操作符
|=    按位或赋值操作符
```

实例

```
public class Test {
  public static void main(String args[]) {
     int a = 10; int b = 20; int c = 0; c = a + b;
     System.out.println("c = a + b = " + c );
     c += a ;
     System.out.println("c += a  = " + c );
     c -= a ;
     System.out.println("c -= a = " + c );
     c *= a ;
     System.out.println("c *= a = " + c );
     a = 10; c = 15; c /= a ;
  }
}
```

三目运算符

```
variable x = (expression) ? value if true : value if false
```

```
public class Test {
   public static void main(String args[]){
        int a , b;
        a = 10;
        b = (a == 1) ? 20: 30;
        System.out.println( "Value of b is : " +  b );
        b = (a == 10) ? 20: 30;
        System.out.println( "Value of b is : " + b );
   }
}
```

instanceof

```
( Object reference variable ) instanceof  (class/interface type)
```

```
"James" instanceof String;
```

运算符优先级

```
后缀      () [] . (点操作符)
一元    ++ - ! ~
乘性    * / %
加性    + -
移位    >> >>>  <<
关系    >> = << =
相等    ==  !=
按位与  &
按位异或     ^
按位或  |
逻辑与  &&
逻辑或  ||
```

```
条件    ? :
赋值    = += -= *= /= %= >>= <<= &= ^= |=
逗号    ,
```

控制执行流程

- if-else
- switch
- while
- do-while
- for
- foreach
- break
- continue
- return

if

```
if(布尔表达式) {
   //如果布尔表达式为true将执行的语句
}
```

```
public class Test {
   public static void main(String args[]){
      int x = 10;
      if( x < 20 ){
         System.out.print("这是 if 语句");
      }
   }
}
```

if...else

```
if(布尔表达式){
   //如果布尔表达式的值为true
}else{
```

```
    //如果布尔表达式的值为false
}
```

```
public class Test {
   public static void main(String args[]){
      int x = 30;
      if( x < 20 ){
         System.out.print("这是 if 语句");
      }else{
         System.out.print("这是 else 语句");
      }
   }
}
```

if...else if...else

```
if(布尔表达式 1){
   //如果布尔表达式 1的值为true执行代码
}else if(布尔表达式 2){
   //如果布尔表达式 2的值为true执行代码
}else if(布尔表达式 3){
   //如果布尔表达式 3的值为true执行代码
}else {
   //如果以上布尔表达式都不为true执行代码
}
```

实例

```
public class Test {
   public static void main(String args[]){
      int x = 30;
      if( x == 10 ){
         System.out.print("Value of X is 10");
      }else if( x == 20 ){
         System.out.print("Value of X is 20");
      }else if( x == 30 ){
         System.out.print("Value of X is 30");
      }else{
         System.out.print("This is else statement");
      }
   }
}
```

if嵌套

```
public class Test {
   public static void main(String args[]){
      int x = 30;
      int y = 10;
      if( x == 30 ){
         if( y == 10 ){
             System.out.print("X = 30 and Y = 10");
          }
       }
    }
}
```

switch

```
switch(expression){
    case value :
       //语句
       break; //可选
    case value :
       //语句
       break; //可选
    //你可以有任意数量的case语句
    default : //可选
       //语句
}
```

注意点

- switch语句中的变量类型只能为byte、short、int或者char,enum. **JDK7开始支持String**
- switch语句可以拥有多个case语句。每个case后面跟一个要比较的值和冒号。
- case语句中的值的数据类型必须与变量的数据类型相同，而且只能是常量或者字面常量。
- 当变量的值与case语句的值相等时，那么case语句之后的语句开始执行，直到break语句出现才会跳出switch语句。
- 当遇到break语句时，switch语句终止。程序跳转到switch语句后面的语句执行。
- switch语句可以包含一个default分支，该分支必须是switch语句的最后一个分支。default在没有case语句的值和变量值相等的时候执行。

实例

```
public class Test {
   public static void main(String args[]){
      char grade = 'C';
      switch(grade) {
         case 'A' :
            System.out.println("Excellent!");
            break;
         case 'B' :
         case 'C' :
            System.out.println("Well done");
            break;
         case 'D' :
            System.out.println("You passed");
         case 'F' :
            System.out.println("Better try again");
            break;
         default :
            System.out.println("Invalid grade");
      }
      System.out.println("Your grade is " + grade);
   }
}
```

while

```
while( 布尔表达式 ) {
    //循环内容
}
```

```
public class Test {
   public static void main(String args[]) {
      int x = 10;
      while( x < 20 ) {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }
}
```

do-while

```
do {
        //代码语句
}while(布尔表达式);
```

```
public class Test {
   public static void main(String args[]){
      int x = 10;
      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
   }
}
```

for

```
for(初始化; 布尔表达式; 更新) {
    //代码语句
}
```

- 最先执行初始化步骤。可以声明并初始化一个或多个循环控制变量，也可以是空语句。
- 然后，检测布尔表达式的值。如果为true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句。
- 执行一次循环后，更新循环控制变量。
- 再次检测布尔表达式。循环执行上面的过程。

实例

```
public class Test {
   public static void main(String args[]) {
      for(int x = 10; x < 20; x = x+1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
```

```
}
```

foreach

```
for(声明语句 : 表达式) {
   //代码句子
}
```

```
public class Test {
   public static void main(String args[]){
      int[] numbers = {10, 20, 30, 40, 50};
      for(int x : numbers){
         System.out.print( x );
         System.out.print(",");
      }
      System.out.print("\n");

      String[] names ={"James", "Larry", "Tom", "Lacy"};
      for(String name : names) {
         System.out.print( name );
         System.out.print(",");
      }
   }
}
```

break

```
public class Test {
   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers) {
         if(x == 30) {
          break;
         }
         System.out.print(x);
         System.out.print("\n");
      }
   }
}
```

continue

```
public class Test {
   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};
      for(int x : numbers) {
         if(x == 30) {
          continue;
         }
         System.out.print(x);
         System.out.print("\n");
      }
   }
}
```

labeled break与labeled continue

```
public class Test {
   public static void main(String args[]) {
        int [] numbers = {10, 20, 30, 40, 50};
        outer: for(int x : numbers) {
            for(int i = 1;i < 3;i++) {
                if (x == 30) {
                    break outer;
                }
                System.out.print(x);
                System.out.print("\n");
            }
            System.out.println("outer" + x);
        }
    }
}
```

return

- 跳出当前方法,并返回方法所需要的值(如果有返回值的话)