1 Java注释

1、单行注释

```
//这是单行注释
```

2、多行注释

```
/*
这是
多行注释
 */
```

3、文档注释

```
/**
 * 这是文档注释
 * @param args
 */
```

- @author：作者。
- @version：版本。
- @param：方法的参数类型。
- @return：方法的返回类型。
- @exception：抛出的异常。
- @throws：抛出的异常，和exception同义

2 变量

2.1 常量

Java语言中，主要是利用final关键字来定义常量。

例如：

```
final String name = "Tom";
```

2.2 变量

在JAVA中通过三个元素来描述变量：变量类型，变量名以及变量值；变量类型可以是八种基本数据类型，也可以是引用数据类型。

例如：

```
String name = "Tom";
int age = 20;
```

变量名： 1、字母，数字，下划线和美元符号$； 2、首字母只能是字母，下划线和美元符号，不能是数字； 3、不能是关键字； 4、变量名称严格区分大小写； 5、变量名可以但不要使用中文。

3 基本数据类型

Java中基本数据类型有八种，分为四类，简称为四类八种。

数据种类	数据类型	存储空间
byte	整数	1字节（8位）
short	整数	2字节（16位）
int	整数	4字节（32位）
long	整数	8字节（64位）
float	浮点数	4字节
double	浮点数	8字节
char	字符	2字节
boolean	布尔	1字节

```
//1、整型   : byte short int long
//2、浮点型 : float double
//3、字符型 : char
//4、布尔型 : boolean
//数字取值范围从小到大
//byte < short  < int < long < float < double
```

3.1 整数类型

1）整数的直接量默认类型为int类型 ，如果直接写出的整数超过了int类型的取值范围，将会出现编译错误。

2）除了通常的十进制书写形式，整数直接量也经常写成十六进制的形式（以0x开头）或八进制的形式（以0开头）。

3）如果表示long直接量，需要使用 L 或者 l 结尾，一般建议 L 做结尾。

```
//声明并赋值变量： 关键词  变量名 = 变量值 ;
byte b = 1;   //1个字节
short s = 1;  //2个字节
int i = 1;    //4个字节
long l = 1;   //8个字节
```

3.2 浮点数类型

```
float f ;
f = 1.1f;   //float后面需要写一个f
double d = 1.1;
```

3.3 字符类型

```
//只能保存一个字
char c = 'A';
c = '中';
```

3.4 布尔类型

```
boolean flag1 = true;   //真的 对的
boolean flag2 = false;  //假的 错的
```

4 输入与输出

4.1 输入

```
//键盘输入
Scanner sc = new Scanner(System.in);
byte b = sc.nextByte();  //开辟一个byte类型的内存空间，将用户输入的数据保存在该空间中，注意用户输入不能超过byte的范围
```

注意：

键盘输入需要导入" java.util.Scanner "。

```
import java.util.Scanner;
```

4.2 输出

```
//会自动换行
System.out.println("请输入：");
//不会换行
System.out.print("请输入：");
//输出是红色的
System.err.println("请输入：");
System.err.print("请输入：");
```

5 数据类型转换

5.1 自动类型转换

从小类型到大类型可以自动完成 ，称为自动类型转换，又称为隐式类型转换。

类型的大小关系如下 ：

byte => short => int => long => float => double

char

```
//自动类型转换：将小的（取值范围小的）数据类型赋值给大的数据类型
//byte < short  < int < long < float < double
int i = 10;
System.out.println(i);
//大数据类型 = 小数据类型;  自动类型转换
double d = 10;    //  浮点型 = 整型;   正确的
System.out.println(d);
//int j = 1.0;     //   整型 = 浮点型;   错误的
```

5.2 强制类型转换

从大类型到小类型需要强制类型转换，称为强制类型转换。强制类型转换时，需注意，可能会造成精度缺失。

```
//大数据类型赋值给小数据类型
//   小数据 = 大数据;
//byte < short  < int < long < float < double
double d = 3.1415926;
int i = (int) d;
//以下符合自动类型转换
int j = (short) d;
int k = (byte) d;
```

6 运算符及表达式

6.1 赋值运算符

```
int money = 100;
```

将100的字面直接量通过 “=” 放入变量存储空间中，“=” 就称为赋值运算符。

作用：将 “=” 右边的结果赋值给左边的变量（将右边的值，放入左边变量的存储空间中）。

6.2 算术运算符

Java运算符除了通常的+（加）、-（减）、*（乘）、/（除）以外，还包括%（取模运算，取余数）和++（自增运算）及--自减运算。

在进行运算时，需要数据类型统一，它们会自动向大类型转换。

6.2.1 加减乘除

```
int a = 1 + 2;
int b = 2 - 1;
int c = 2 * 3;
int d = 1 / 2;
double d1 = 1 / 2.0;
int result = d + c - b * a / 2 ;
```

6.2.2 取模（求余数）

```
int number = 1234;
int qian =  number / 1000;
int bai = number % 1000 / 100;  //求模  余数
int shi = number % 100 / 10;
int ge = number % 10;
```

6.2.3 自运算

```
int num = 0;
num += 100;  //num = num + 100;
num -= 100;  //num = num - 100;
num *= 100;  //num = num * 100;
num /= 100;  //num = num / 100;
num %= 100;  //num = num % 100;
```

6.2.4 自增自减

++（--）：

（1）++在前，先自增，再运算

（2）++在后，先运算，在自增

（3）++只能 作用于变量，不能作用字面直接量

（4）num++ 等价于 num = num+1

```
int numA = 1;
numA++;  // numA += 1;  numA = numA + 1;
numA--;
```

```
int numB = 1;
System.out.println(numB++);//先输出再运算
System.out.println(++numB);//先运算再输出
```

6.3 字符串连接符

+ :

（1）若两端都为数字，则做加法运算

（2）若一边为字符串，则做字符串拼接

```
int numA = 1;
int numB = 2;

System.out.println("结果："+numA);
//当+左右两边均为数字时，java会自动进行加法运算，如果有任意一方不为数字，进行拼接运算
int numC = numA+ numB;
System.out.println("结果："+numC);
System.out.println("结果："+(numA+numB));

String name1 = "王小杨";
int age1 = 18;
char sex1 = '男';

System.out.println(name1+"    "+age1+"    "+sex1);
System.out.println("name1"+"    "); //错误写法
```

6.4 关系运算符

关系运算符用于比较两个数值的大小，运算的结果是一个布尔值（true或false）。

关系运算符	名称
>	大于
>=	大于等于
<	小于
<=	小于等于
==	等于
!=	不等于

6.5 逻辑运算符

逻辑运算符用于两个布尔型的变量或常量的运算。逻辑运算符常见的主要有6个：

逻辑运算符	名称	作用
&&	与	前后两个操作数必须都是true，才返回true，否则返回false
&	不短路与	作用与&&相同 ，但没有短路效果
||	或	只要两个操作数中有一个是true，就可以返回true，否则返回false
|	不短路或	作用与||相同，但没有短路效果
!	非	只需要一个操作数，如果操作数为true，则返回false； 如果操作数为false，则返回true
^	异或	当两个操作数不同时，返回true，否则返回false

优先级：

! > && > ||

6.6 三目运算符

语法：boolean ? 数1：数2;

执行过程：

计算boolean的值，

若为true，则整个的结果为数1；

若为false，则整个的结果为数2。

```
String s = !true ? "数1" : "数2";
System.out.println(s);
```

7 分支结构

7.1 if分支结构

7.1.1 if分支语句

```
boolean flag = true;
if(flag){
    System.out.println("语句1");
}
```

判断逻辑表达式的值：

若为true，则执行if语句块中的语句；

若为false，则不执行if语句块中的语句。

7.1.2 if-else分支语句

```
boolean flag = true;
if(flag){
    System.out.println("语句1");
}else{
    System.out.println("语句2");
}
```

判断逻辑表达式的值：

若为true，则执行if语句块中的语句；

若为false，则执行else语句块中的语句。

7.3 if-elae-if分支语句

```
boolean flag = true;
if(flag == false){
    System.out.println("语句1");
}else if(flag == true){
    System.out.println("语句2");
}else{
    System.out.println("语句3");
}
```

判断逻辑表达式1的值：

若为true，则执行语句1；

若为false，则判断逻辑表达式2。

若为true，则执行语句2；

若为false，则执行else中的语句3。

7.2 switch-case分支语句

switch-case分支语句是一个特殊的分支结构，可以根据一个整数表达式的不同取值，从不同的程序入口开始执行。

```
//switch 表达式只能是 byte short int char String
//switch只能单值匹配，无法范围匹配，if可以单值也可以范围
int i = 3;
switch (i)
{
    case 1:
        System.out.println("语句1");
        break;  //结束 switch会一直执行到碰见第一个break停止，如果真个switch都没有break,系统会全部执行！！
    case 2:
        System.out.println("语句2");
        break;
    default:
        System.out.println("如果条件都不匹配，执行default");
}
```

8 循环结构

8.1 for循环

语法：

for(1.声明并赋值; 2.布尔值; 4.自增或自减){

3.循环体;

}

循环顺序：

1 => 2 => 3

4 => 2 => 3

4 => 2 => 3

```
int count = 5;
//循环5次  变量i也是在循环内部声明的
for(int i = 0 ; i < count ; i++)
{
    System.out.println("语句1");
}
```

8.2 while循环

语法（先判断，再执行）：

while(布尔值){

循环体;

}

```
boolean flag = true;
while(flag)
{
    System.out.println("语句1");
}
```

8.3 do-while循环

语法（先执行，后判断，如果条件不成立，也最少执行一次）：

do{

循环体;

}while(布尔值);

```
int i = 0 ;
do{
    System.out.println("do-while");
    i++;
}while(i<1);
```

8.4 break语句

break用于循环，可使程序终止循环后面的语句，常与分支语句一起使用。

```
//break只能出现在循环或者switch中，表示结束
for(int i = 0 ; i < 10; i++)
{
    System.out.println("语句1");
    break;

}
```

```
while(true)
{
    System.out.println("语句2");
    break;
}
switch (1)
{
    case 1:
        System.out.println("语句3");
        break;
}
```

8.5 continue语句

continue用于循环中，其作用为跳过循环体中剩余语句而执行下一次循环，常与分支语句一起使用。

```
//continue:跳过当前循环，执行下一次循环
for(int i = 0 ; i < 10 ; i++)
{
    if(i == 7)
    {
        continue;
    }
    System.out.println(i);
}
```

9 数组

9.1 数组的定义

数组：一堆相同数据类型的有序集合

- 一堆： n >= 0
- 相同数据类型：数据类型一样，字面意思
- 有序：先进先出（first in first out = FIFO）
- 集合：相对集中

数组长度一旦声明，在运行期间不能改变。

9.2 数组的声明

数据类型[] 数组名 = new 数据类型[大小]；

```
//声明数组
int[] array = new int[5];
//int[]: 表示数组中的每一个元素都是int类型。
//array: 数组类型变量（引用类型）。
//[5]: 数组长度。
```

注意：

（1）执行new语句才使得数组分配到指定大小的空间。

（2）int[] arr 与int arr[]两种写法均可。

（3）声明数组时不规定数组长度，new关键字分配空间时需要指定分配的空间大小。

```
//java声明
int[] a1 = new int[10];
//c++ c#
int a2[] = new int[10];
//声明并赋值
int[] a3 = new int[]{1,2,3,4,5,6,7};
int[] a4 = {1,2,3,4,5,6};

//整型类数组
byte[] bytes = new byte[3];
short[] shorts = new short[3];
int[] ints = new int[3]; //推荐使用
long[] longs = new long[3];

//浮点型（小数类型）
float[] floats = new float[5];
double[] doubles = new double[5];

//字符
char[] chars = new char[6];

//布尔类型
boolean[] booleans = new boolean[5];

//字符串数组  引用类型数组
String[] strings = new String[10];
```

9.3 数组的初始化

基本类型的数组创建后，其元素的初始值：

- byte、short、int、long为0；
- char:\u0000;
- float和double为0.0；
- boolean为flase。

可以在数组声明的同时，对数组元素进行初始化，例如：

```
//元素的个数即为数组的长度。
int[] arr = {10, 20, 30, -10};
```

此种写法只能用于声明的同时初始化，不能用于赋值，如下面代码会有编译错误：

```
//错误的初始化
int[] arr;
arr = {1, 2, 3, 4};
```

可以通过下面的方式给已经声明的数组类型变量进行初始化：

```
int[] arr;
arr = new int[]{1, 2, 3, 4};
```

注意：此时[]不可写长度，元素的个数就是数组的长度。

```
int[] ints = new int[3];
double[] doubles = new double[3];
String[] strings = new String[3];

//向数组中存储数据
//使用下标
ints[0] = 100;
ints[1] = 100;
ints[2] = 300;
//ArrayIndexOutOfBoundsException:数组下标越界异常
//ints[3] = 400; error

doubles[0] = 1.5;
doubles[1] = 1;
doubles[2] = 9;

strings[0] = "赵光辉";
strings[1] = "王见光";
strings[2] = "项复玉";

//循环录入数据
Scanner sc = new Scanner(System.in);

for(int i = 0 ; i < 3  ; i++)
{
    System.out.println("请输入学号：");
    ints[i] = sc.nextInt();

    System.out.println("请输入学生姓名：");
    strings[i] = sc.next();

    System.out.println("请输入学生成绩：");
    doubles[i] = sc.nextDouble();
}
```

9.4 数组的访问

9.4.1 获取数组的长度

调用数组的length属性可以获取数组的长度：

```
int[] arr = new int[]{1, 2, 3, 4, 5, 6};
int len = arr.length;
```

上述代码中len的值为6。

9.4.2 查看数组元素

Arrays.toString() 方法可以查看数组元素。

```
int[] array = {1, 2, 3, 4, 5, 6, 7};
System.out.println(Arrays.toString(array));
```

9.4.3 通过下标访问数组元素

数组中的元素通过下标的方式进行访问：

注意：下标从0开始，最大到length-1

例如：

```
int[] arr = new int[]{1, 2, 3, 4, 5, 6};
int temp = arr[2];//获取第3个元素—3
//交换数组下标为2和3的两个相邻元素的值
int t = arr[2];
arr[2] = arr[3];
arr[3] = t;
//交换后结果为1，2，4，3，5，6
```

9.4.4 遍历（循环）数组元素

遍历数组元素通常选用for循环语句，循环变量作为访问数组元素的下标，即可访问数组中的每一个元素。

```
int[] arr = new int[10];
for (int i = 0; i < arr.length; i++)
{
    arr[i] = 100;
}
```

9.5 数组的复制

1）可以使用for循环复制数组：

```
int[] arr = new int[]{1, 32, 23, 231, 432, 123, 12};
int[] arr1 = new int[arr.length];
for (int i = 0; i < arr.length; i++)
{
    arr1[i] = arr[i];
}
System.out.println(Arrays.toString(arr1));
```

2 ） System.arraycopy() (源数组, 源数组中的起始位置, 目标数组, 目标数组中的起始位置, 要复制的数组元素的数量)；该方法属于JDK提供的。

```
int[] arr = new int[]{1, 32, 23, 231, 432, 123, 12};
int[] arr1 = new int[arr.length];
System.arraycopy(arr, 0, arr1, 0, arr.length);
System.out.println(Arrays.toString(arr1));
```

3 ） Arrays.copyOf() 方法也可以用来复制数组：

```
int[] arr = new int[]{1, 32, 23, 231, 432, 123, 12};
//                        原数组   新数组长度
int[] arr1 = Arrays.copyOf(arr, arr.length);
System.out.println(Arrays.toString(arr1));
```

特点：生成的新数组是原始数组的副本。

9.6 数组的扩容

数组的长度在创建以后是不可改变的，所谓扩容是指创建一个更大的新数组，并将原有的数组的内容复制到其中。

可以通过Arrays.copyOf() 方法，简便实现数组的扩展。

```
int[] arr = new int[]{1, 32, 23, 12};
arr = Arrays.copyOf(arr, arr.length + 1);
```

9.7 数组排序

排序是对数组施加的最常用的算法。所谓的排序，是指将数组元素按照从小到大的顺序重新排列。

一般情况下，通过排序过程中数组元素的交换次数来衡量排序算法的优劣。

常用的排序算法有：插入排序、冒泡排序、快速排序等。

9.7.1 冒泡排序

经典冒泡排序的原则：比较相邻的2个元素，如果违反最后的顺序准则，则交换。可以简单理解为：

第1次找到所有元素中最大的放在最后一个位置上，不再变动；

第2次找到剩余所有元素中最大的放在最后一个位置上，不再变动；

以此类推，直到排序完成。比较时，即可采用下沉的方式，也可以使用上浮的方式实现。

```
int[] arr = new int[]{3, 9, 1, 4, 2, 7};
//外层循环-1  内存循环-i-1
for (int i = 0; i < arr.length - 1; i++)
{
    for (int j = 0; j < arr.length - 1 - i; j++)
    {
        //前面数字 > 后面的数字，  让大的去后面，小的去前面
        //按照升序排列
        if (arr[j] > arr[j + 1])
        {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}
System.out.println(Arrays.toString(arr));
```

9.7.2 自动排序

使用JDK提供的Arrays.sort()方法进行数组排序。

```
int[] arr = new int[]{3, 9, 1, 6, 5};
Arrays.sort(arr);
System.out.println(Arrays.toString(arr));
```

9.8 二维数组

数组的元素可以是任意类型，所以也可以是数组类型。

9.8.1 声明与赋值

```
//声明int类型的数组arr，包含3个元素
//每个元素都是int[]类型，默认值为null
//默认值为null，是因为它是引用类型
int[][] arr = new int[3][];
//arr[0]是int[]类型
arr[0] = new int[2];
arr[1] = new int[3];
arr[2] = new int[1];
```

```
int[][] ints = new int[][]
        {
                {1, 2, 3, 4, 5},
                {5, 4, 3},
                {},
                {1}
        };
```

9.8.2 二维数组的访问

```
String[][] classes =
        {
                {"Z","R","R","S"},
                {"Z","W","X","C","L"},
                {"R","Z","L","Y","Q"},
                {"L","W","L","W","W"},
                {"W","H","W","F","L"},
                {"L","Y"}
        };

for(int i = 0 ; i < classes.length;i++)
{
    for(int j = 0 ; j < classes[i].length;j++)
    {
        System.out.print(classes[i][j]+"    ");
    }
    System.out.println();
}
```