

计算机的数据存储存储

```
public class Test {
    public static void main(String[] args) {
        /* TIP 十进制 数值即可 */
        System.out.println(17);

        // 八进制 0开头
        System.out.println(017);

        // 二进制 0b开头
        System.out.println(0b1);
        // 十六进制一般以0x开头
        System.out.println(0x123);
    }
}
```

在计算机中，任意数据都是以二进制的形式来存储的

存储规则

- GB2312编码：发布简体中文汉字编码国家标准
- GBK:包含国家标准的全部中日韩汉字
- Unicode:国际标准字符集，将世界各种语言的每个字符的定义一个唯一的编码。

数据类型

数据类型	关键字
整数	byte (-128~127) \ short (+-3万) \ int (10位数) \ long (19位数)
浮点数	float (3.4^38) \ double (1.7^308)
字符	char
布尔	boolean

```
public class Test {
    public static void main(String[] args) {
        // long 需要加上后缀L
        long b = 9999999L;

        // float 需要加后缀F\ f
        float v = 1100102.0f;
        float q = 1100102.0F;
    }
}
```

整数和小数取值范围大小关系 double > float > long > int > short > byte

标识符命名规则 --硬性要求

- 由数字、字母、下划线、美元符号组成
- 不能以数字开头
- 不能是关键字
- 区分大小写

标识符命名规则 --软性要求

- 标识符是单词的时候全部小写
- 两个单词前面小写后面大写

键盘录入

Scanner 类

```
//TODO:引入库包
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        //TODO: 创建Scanner对象
        Scanner sc = new Scanner(System.in);

        //TODO:接收Scanner数据
        int n = sc.nextInt();

        //TODO:打印
        System.out.println(n);
    }
}
```

运算符

浮点数计算有可能不精确

```
public class Test {
    public static void main(String[] args) {
        float a = 1.1f;
        float b = 1.01f;

        //TODO:2.1100001 有小数计算, 结果有可能不精确
        System.out.println(a + b);
    }
}
```

```
}  
}
```

- 整数参与计算 结果只能得到整数
- 小数参与计算 可能不精确 但得到小数结果
- 取模，取余其实是除法运算，只不过得到是余数而已

算术运算符

数字进行计算时候，数据类型不一样的不能进行运算，需要转成一样的才能计算。

类型的转换方式

1. 隐式转换: 取值范围小 --> 取值范围大
2. 强制转换: 取值范围大 --> 取值范围小

隐式转换

```
public class Test {  
    public static void main(String[] args) {  
        int a = 10 ;  
  
        double b = 20.0;  
  
        /*TODO: 隐式转换（自动）有两个规则：  
        1.取值范围小的、和取值范围大的进行运算，小的会先提升为大的，在进行运算 a+b  
== double  
        2.byte short char 三种类型进行运算的时候，都会先提升为int 再运算  
  
        */  
        System.out.println(a+b);  
    }  
}
```

强制转换

```
public class Test{  
    public static void main(String[] args)  
    {  
        int a = 300;  
  
        byte b = (byte) a;// byte ~ +-127  
    }  
}
```

如果把一个取值范围大的数值、赋值给取值范围小的变量，可能会报错（超过取值范围）

```
public class Test{
    public static void main(String[] args)
    {
        byte b1 = 10;
        byte b2 = 20;

        byte a = (byte) (b1+b2) //TODO:(b1+b2)->int    (byte)(b1+b2)->byte
    }
}
```

字符串+

```
public class Test {
    public static void main(String[] args) {
        int a = 300;

        String b = "xxx123";
        //TODO:当+操纵符出现字符串时候, +就是字符串连接符 而不是算术运算符
        System.out.println(a);
        //TODO:先1+a 再 字符串拼接
        System.out.println(1+a+b);
    }
}
```

自增自减运算符

两种情况

1. 单独使用: ++和-- 无论是放在变量的前边还是后边, 单独写一行的结果是一样的
2. 参与计算;

```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = a++;//TODO:先用后加 (先赋予b再自身加)
        System.out.println(b); //10
        System.out.println(a); //11
    }
}
```

```
public class Test {
    public static void main(String[] args) {
```

```
        int a = 10;
        int b = ++a; //TODO:先加后用 (先自身+再赋予b)
        System.out.println(b); //11
        System.out.println(a); //11
    }
}
```

赋值运算符

分类

符号	作用	说明
=	赋值	int a=10, 将10赋值给变量a
+=	加后赋值	a+=b, 将a+b的值给a
-=	减后赋值	a-=b, 将a-b的值给a
=	乘后赋值	a=b, 将a*b的值给a
/=	除后赋值	a/=b, 将a÷b的商给a
%=	取余后赋值	a%=b, 将a÷b的余数给a

+= -= *= /= %= 都隐藏了一个强制类型转换。

关系（比较）运算符

符号	说明
==	a==b, 判断a和b的值是否相等, 成立为true, 不成立为false
!=	a!=b, 判断a和b的值是否不相等, 成立为true, 不成立为false
>	a>b, 判断a是否大于b, 成立为true, 不成立为false
>=	a>=b, 判断a是否大于等于b, 成立为true, 不成立为false
<	a<b, 判断a是否小于b, 成立为true, 不成立为false
<=	a<=b, 判断a是否小于等于b, 成立为true, 不成立为false

```
public class Test {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int c = a + b;

        System.out.println(a==b); //false
    }
}
```

```
        System.out.println(c==a);//false
    }
}
```

逻辑运算符

符号	作用	说明
&	逻辑与(且)	并且，两边都为真，结果才是真
		逻辑或
^	逻辑异或	相同为 false，不同为 true
!	逻辑非	取反

短路逻辑运算符

符号	作用	说明
&&	短路与	结果和&相同，但是有短路效果
\	短路或	结果和\相同，但是有短路效果

1. &| 无论左边是true false 右边都要执行

2. &&|| 如果左边能确定整个表达式的结果，右边不执行。（提高执行效率）

三元运算符

1. 格式：关系表达式？表达式1：表达式2.

2. 求两个数的最大值

```
System.out.println(a>b ? a:b);
```

运算符优先级

()优先级最高

原码补码反码

原码


十进制数据的最近之表现形式，最左边是符号位，0正1负：只能计算正数计算

反码

为了解决源码不能计算负数的问题而出现 计算规则：正数的反码不变，负数的反码在原码的基础上，符号位不变，数值取反，0变1 1变0

补码

在反码的基础上加上1。解决正负相加（跨0）补码还能多记录一个数值-128。-128无源码和反码

img_2.png

```
public class Test {  
    public static void main(String[] args) {  
        int a = 300; //0000 0000 0000 0000 0000 0001 **0010 1100**  
        byte b = (byte) a; // 0010 1100  
        System.out.println(b); //44  
    }  
}
```

流程控制语句

if 语句

和C++一样

```
if (true)  
{  
    System.out.println("true");  
}
```

```
if (true)  
{  
    System.out.println("true");  
}  
else  
{  
    System.out.println("false");  
}
```

Switch语句

```
switch (表达式){  
    case 表达式值1:  
        语句体1;
```

```
        break;
    case xxxxxx:
        语句体2;
        break;
    case xxxx -> 语句;// -> 不需要break
    default:
        语句体n;
        break;
}
```

for语句

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

while语句

```
while(true)
{
    System.out.println("Hello World");
}
```

```
do{
    System.out.println("Hello World");
}
while (条件判断语句)
```

无限循环

```
for(;;)
{
    System.out.println("Hello World");
}
```



```
while (true)
{
    System.out.println("Hello World");
}
```

循环关键字

continue:跳出本次循环，继续执行下一次循环

break:结束整个循环

练习题

1.平方根

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        /*
            键盘录入一个大于等于2 的整数x 计算并返回x的平方根 。 结果只保留整数部分 ， 小数
            部分被舍去。
        */

        // 10
        Scanner sc = new Scanner(System.in);
        int num = sc.nextInt();

        for (int i = 1; i <= num; i++) {
            if ((i * i < num && (i+1)*(i+1)>num) || i*i == num) {
                System.out.println(i);
            }
        }
    }
}
```

2.质数

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        // 输入一个整数 判断是否为质数 ： 只有被1或者本身整除 才叫质数 否者为合数
        Scanner sc = new Scanner(System.in);
```

```
        System.out.println("请输入一个正整数");

        int number = sc.nextInt();
        boolean flag = true;
        for (int i = 2; i < number; i++) {
            if (number % i == 0) {
                flag = false;
                break;
            }
        }

        if (flag) {
            System.out.println(number + "是质数");
        }
        else{
            System.out.println(number + "不是质数");
        }
    }
}
```

随机数 猜数字

```
/**
 * 导入包
 */
import java.util.Random;

// 创建对象 ---表示我们要用Random 这个类了
Random random = new Random();

//生成随机数 --- 真正开始干活了
int number = random.nextInt(随机数取值范围)
```

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

public class Test {
    public static void main(String[] args) {
        Random rand = new Random();
        rand.setSeed(System.currentTimeMillis());
        for (int i = 0; i < 100; i++) {
            int r = rand.nextInt(100); // 0-99
            System.out.println(r);
        }
    }
}
```

数组

数组是一个容器 用来存储同种数据类型的多个值

但是符合隐式转换，小范围 转 大范围

- 如：int类型的数组容器(byte short int)
- double类型的数组容器(byte short int long float double)
- 建议：容器的类型和存储的数据类型保持一致

数组的定义

两种格式

1. 数据类型[]数组名 : int []array;
2. 数据类型 数组名[] : int array[];

数组的初始化

初始化：就是在内存中，为数组容器开辟空间，并将数据存入容器的过程

数组的静态初始化

```
//静态初始化 《完整格式》
int []array = new int[]{11 ,22, 33};

double []array2 = new double[]{1.1 , 2.2 ,3.3};

// 简写模式
int []array3 = {11,22,33};

double []array4 = {1.1 , 2.2 ,3.3};
```

```
public class Test {
    public static void main(String[] args) {
        //存储学生的年龄
        int []StudentAges = new int[]{10,20};

        //存储学生的姓名
        String []StudentName = new String[]{"张三" , "李四"};

        //定义学生的身高
        double []StudentHeight = new double[]{1.6,1.90};
        double []StudentHeight2 ={1.6,1.90};
    }
}
```

```
}  
}
```

[I@723279cf 数组的地址值表示数组在内存中的位置

- []:表示一个数组
- D:表示数组的类型
- @:表示间隔符号（固定格式）
- 723279cf:真正地址值（16进制）

数组访问/遍历

索引：下角标/编号

```
public class Test {  
    public static void main(String[] args) {  
        //存储学生的年龄  
        int []StudentAges = new int[]{10,20};  
  
        //存储学生的姓名  
        String []StudentName = new String[]{"张三" , "李四"};  
  
        //定义学生的身高  
        double []StudentHeight = new double[]{1.6,1.90};  
        double []StudentHeight2 ={1.6,1.90};  
        System.out.println(StudentAges[0]);  
        for (String v : StudentName) System.out.println(v);  
    }  
}
```

数组动态初始化

动态初始化：初始化时只指定数组的长度，由系统为数组分配初始值

```
//格式： 数据类型[]数组名 = new 数据类型[数组长度]  
  
int[]arr = new int[3]
```

```
import java.util.Random;  
  
public class Test {  
    public static void main(String[] args) {  
        //存储学生的年龄
```

```
int []StudentAges = new int[]{10,20};

//存储学生的姓名
String []StudentName = new String[50];

Random rand = new Random();

for(int i=0;i<StudentName.length;i++){
    int age = rand.nextInt(50) +1; //1-50
    StudentName[i] = "学生"+age;
}

for(String s:StudentName){
    System.out.println(s);
}
}
```

数组默认初始化值的规律

- 整数类型：默认初始化为0
- 小数类型：默认初始化为0.0
- 字符类型：默认初始化为'\u0000' 空格
- 布尔类型：默认初始化为 false
- 引用数据类型：默认初始化为Null (4类基本类型之外 如 String)

数组动态初始化和静态初始化的区别

动态初始化：手动指定数组的长度，由系统给出默认初始化值

- 只明确元素的个数，不明确具体的数值，推荐使用动态初始化。

静态初始化：手动指定数组的元素，由系统计算元素的个数，计算出数组的长度

- 明确具体的数值，推荐使用静态初始化。

数据的问题

超出索引范围会报错

数组的常见操作

1. 求最值
2. 求和
3. 交换数据
4. 打乱数据

1.最大值

```
import java.util.Arrays;
import java.util.Random;

public class Test {
    public static void main(String[] args) {
        // 自然数对象
        Random rand = new Random();
        // 数组动态初始化
        int[] arr = new int[10];
        // 遍历赋值自然值
        for (int i = 0; i < arr.length; i++) arr[i] = rand.nextInt(100);
        // 初始化最大值, 不建议使用0 直接使用数组索引0
        int max = arr[0];
        // 遍历数组寻找最大值
        for (int v : arr) {
            if (v > max) max = v;
        }
        // 输出最大值
        System.out.println(max);
        System.out.println(Arrays.stream(arr).max());
    }
}
```

2.数组求和

```
System.out.println(Arrays.stream(arr).sum());
```

3.数组打乱 使用自然数随机取数组索引，并使用交换数据temp

Java 内存分配

Java内存分配	解释	应用
栈	方法运行时使用的内存，比如main方法运行，进入方法栈中执行	方法运行时使用的内存，比如main方法运行，进入方法栈中执行
堆	存储对象或者数组，new来创建的，都存储在堆内存	new来创建的，都储存在堆内存，new出来的东西会在这块内存中开辟空间并产生地址
方法区	存储可以运行的class文件	
本地方法栈	JVM在使用操作系统功能的时候使用，和我们开发无关	
寄存器	给CPU使用，和我们开发无关	

数组的内存图

1. 只要new出来的一定在堆里面开辟了一个小空间
2. 如果new了多次，那么在堆里面由多个小空间，每个小空间都有各自的数据
3. 两个数组指向同一个空间的内存图 ---> 它们地址一样，存储内存一样

方法(函数)

方法的介绍

方法 (method)是程序中最小的执行单元。

```
public class Test {  
    //TODO:main方法/主方法  
    public static void main(String[] args) {  
  
    }  
}
```

方法的使用时机

- 重复的代码、具有独立功能的代码可以抽取到方法中

方法好处

1. 提高代码的复用性
2. 提高代码的可维护性

方法的格式

方法的定义

1. 最简单的方法定义
2. 带参数的方法定义
3. 带返回值的方法定义

1.1 简单方法的定义

```
/*  
    格式:  
        public static void 方法名(){  
            方法体(打包的代码)  
        }  
*/  
  
public static void playGame(){  
    System.out.println("Hello World");  
}
```

```
}
```

1.2 简单方法的调用

```
playGame();
```

值得注意的是 **方法必须先定义后调用，否则程序会报错**

```
public class Test {  
    //定义PrintHello方法  
    public static void PrintHello() {  
        System.out.println("Hello");  
    }  
    public static void main(String[] args) {  
        //调用方法  
        PrintHello();  
    }  
    /**  
     *      public static void PrintHello() {  
     *      System.out.println("Hello");  
     *      }  
     *      放在main方法前后都行  
     */  
}
```

2.1 带参数方法的定义

```
public class Test {  
  
    public static void main(String[] args) {  
        //调用方法  
        PrintHello(10);  
    }  
    //定义PrintHello方法  
    public static void PrintHello(int num1) {  
        System.out.println(num1);  
    }  
}
```

2.2 带参数的方法 形参与实参

- 形参：全称形式参数，是指方法定义中的参数
- 实参：全称实际参数，方法调用中的参数

```
public class Test {  
  
    public static void main(String[] args) {  
        //调用方法  
        int num = 10;    //TODO:num为实参  
        PrintHello(num);  
    }  
    //定义PrintHello方法  
    public static void PrintHello(int num1) { //TODO:num1为形参  
        System.out.println(num1);  
    }  
}
```

3.1带返回值的定义

```
public static 返回值类型 方法名 （参数） {  
    方法体;  
    return 返回值;  
}
```

示例

```
public class Test {  
  
    public static void main(String[] args) {  
        //调用方法  
        int num = 10;    //TODO:num为实参  
        num = PrintHello(num);  
        System.out.println(num);  
    }  
    //定义PrintHello方法  
    public static int PrintHello(int num1) { //TODO:num1为形参  
        return num1*num1;  
    }  
}
```

方法的注意事项

- 1、方法不调用不执行
 - 2、方法与方法之间是平级关系，不能互相嵌套定义
 - 3、方法的编写顺序和执行顺序无关
 - 4、方法的返回值类型为void，表示该方法没有返回值没有返回值的方法可以省略return语句不写如果要编写return，后面不能跟具体的数据，
 - 5、return语句下面，不能编写代码，因为永远执行不到，属于无效的代码
-

return关键字

1. 方法没有返回值:可以省略不写。如果书写，表示结束方法
 2. 方法有返回值:必须要写。表示结束方法和返回结果
-

方法的重载

- 1.1 函数名相同、形参类型或者个数不同
- 1.2 重载必须都在一个类才叫重载