NOTE ONE

主类结构

```
public class hello {
    public static void main(String[] args) {
        String s1 = "hello"; //定义变量字符值
        String s2 = "Java";
        String s3 = "is soooooooo hard";

        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
}
```

- 开始需定义public 类(class)

[!Note]class 不等同于Class (严格区分大小写)

- public class 定义初始类, 继续class可继续定义新类

```
public static void main
```

- 定义类: main
- 括号后面的内容进行内容定义(args)
- double(用处不明)
- System.out.println()和print一样, system在terminal输出print字符串

```
public class BMI{
    public static void main(String[] args) {
        double height = 180;
        int weight = 65;
        double exponent = weight/(height*height);
        System.out.println(
                "您的身高是: " + height
        );
        System.out.println(
                "您的体重是: " + weight
        );
        System.out.println("您的BMI指数为");
        if (exponent < 18.5) {
            System.out.println("您体重太轻了");
        }
        if (exponent >= 18.5 && exponent <29.9) {
            System.out.println("体重过重");
        }
        if (exponent >= 29.9){
            System.out.println("肥胖");
```

```
        }
    }
}
```

基本数据类型

整数数据类型:

int 64位 (***默认类型***)

声明:

```
int a = 15;
int b = 30;
int c = a-b;
System.out.printIn(c);
```

- byte 8位
- 声明: 和int一样

```
byte a, b, c;
```

- short 16位
    - 声明: 和int相同

```
short a = 90;
```

- long 32位
    - 声明: 结尾加L 或者l 其他和int一样

```
long number;
long number = 30L;
```

- float 单精度浮点类型
    - 声明: 小数后面+ F or f

```
float f1 = 2.2132132f;
```

- Double 双精度浮点类型:
    - 声明: 默认都是double数据类型, 小数后面+ D 或者d.
    - 但是没有特定要求

```
double f2 = 2.2323423;
double f3 = 2.3242234d;
```

字符类型:

- char 型:
    - 用于储存单个字符, 占用16位内存空间.
    - 用单引号表示, 's'

注意! 双引号就是字符串了

```
char a = 'b';
```

可以用unicode 来表示, 所以上面的也可以用

```
char a = 98;
```

转义字符:

- \123 八进制
- \u0052 四位16进制
- \' 单引号
- \\ 反斜杠
- \t 垂直制表符
- \r 回车
- \n 换行
- \b back space
- \f 换页

布尔类型:

```
boolean b;
boolean b1, b2;
boolean b True;
```

- 逻辑类型, 简称布尔型.
    - Boolean只有 True 或者是 False, 且不可以和整数类型转换.

一般是在流程判断中做为条件变量以及条件判断 来定义布尔类型.

变量与常量

声明变量

```
int age = 33;
double number3 = 2.3333d;
char word = 'w';
```

系统内存 ⇢ program区 ⇢ Data 区

写程序的时候在系统内存, 执行时进入内存中的program区, 数据暂存数据(Data)区.

声明常量(一直不会改变的量, 也叫 final 常量)

- 声明要指定数据类型 + final关键字
- final 数据类型 常量类型 [= 值]
- 常量一般全拼用大写字母, 如:

```
final float PI = 3.1415926f;
```

变量的有效范围

- 变量定义出后都暂存在内存中, 等程序执行到某个点, 变量会随内存释放, 就失去有效范围.
- 因此, 分出了"*成员变量*" 和 "*局部变量*".
- 在变量名前加上 **static** 就是静态变量, 可以跨类和整个应用程序进行应用(类似python的function "def")

Java 数据类型 (背过)

- 基本数据类型
    - 数值型 (int)
        - 整数类型, 存放整数 (byte[1], short [2], int [4], long[8])
        - 浮点(小数)类型 (float[4], double[8])
    - 字符型 (char[2]), 存放单个字符 '*a*'
    - 布尔型 (boolean[1]), 存放 true, false
- 引用数据类型
    - 类 (class)
    - 接口 (interface)
    - 数组 ([])

> [!NOTE]
> ([] 内的是字节)
> 小数和整数完全不同, 如果选择较大的, 精度大的小数, 就用double.
> 布尔类型为判断类型, 真假
> *引用数据类型, 在面向对象编程中*
> `string` 是一个类, 不是数据类型

整数类型的使用

- Java的整数类型就是用来存放整数数值的
- 整数的类型:

类型	占用储存空间	范围
byte[字节]	1字节	-128~127
short[短整型]	2字节	-2^{15} ~ $2^{15}-1$ -32768 32767
int[整形]	4字节	-2^{31} ~ $2^{31}-1$ -2147483647 ~ 2147483647
long[长整型]8字节	-2^{63} ~ $2^{63}-1$	

> [!NOTE]
> 具体为社么涉及到二进制, 四种整数类型

案例演示:

```
byte n1 = 10; //这样分配的是一个字节
// n1数值指向内存, n1 = 10, 占用一个字节大小
short n2 = 10; //这样分配的是两个字节
```

> [!NOTE]
> 虽然数值一样, 但是由于所指数据类型不一样,所以所占空间不同
> 以此类推, 和上章的表格是一模一样的

整数类型的细节

1. Java各整数类型有固定的范围和字段成都, 不受具体OS[操作系统]的影响 以保证java程序的可移植性.
2. Java的整数型常量(具体型)默认为int型(不指示数据类型的话), 声明long型的话必须后面加 'l' 或 'L'
3. java程序中木哦人为int型, 除非不足以表示大数, 才使用long
4. `bit`: 计算机中的**最小储存单位**; `byte`: 计算机中**基本储存单元**, `1byte = 8 bit`

> [!Note]
> **尽量选择小的数据类型, 保小不保大**
> 如果能确认, 那就小, 但是不确认(如回文报), 那就保险用long

实例演示

```
public class Int Detail {
  public static void main(String[] args){
    int n1 = 1; // 4个字节
    int n2 = 1L; // 变成long型, 编译报错, 原因是数据类型不同
    long n3 = 1L // 正确long
  }
}
```

> [!NOTE] byte 和 bit 怎么在计算机内存储?
> 一个字节为基本单位, 一个字节内包含8个bit.

```
byte n1 = 3
short n2 = 3
```

byte	short
00000011	0000000000000011
一个字节	两个字节

↑*一个 '0/1' 就是一个bit*

> 思考: long n3 = 5 在内存中怎么画出来

浮点类型 (float)

- **基本介绍:**
  - Java的浮点类型可以表示一个小数, 比如 123.4, 7.8, 0.12等等

类型	占用内存空间	范围
单精度 float	4字节	-3.403E38 ~ 3.403E38
双精度 double	8字节	-1.798E308 ~ 1.798E308

> [!NOTE]
>
> 1. 关于浮点数在机器中存放形式的简单说明, 浮点数 = 符号位+指数位+尾数位
> 2. 尾数可能丢失 造成精度损失 (小数都是近似值)

- **案例演示:**

```
double n1 = 88.9
```

浮点型使用细节

- **浮点型使用细节**
  - 与整数型类似, Java浮点类型也有固定的范围和字段长度, 不受OS的影响 [float4个字节, double 8个字节]
  - Java的浮点型常量 (具体值) 默认为`double`型, 声明`float`型常量, 后面必须加 'f' 或者 'F'
  - **浮点型有两种表现形式**:
    - **十进制**: 5.12, 521.0f, 0.521
    - **科学计数法**: 5.12e2[5.12e*10的二次方*, *5.12e-2*[*5.12e*10负二次方]
  - 通常情况下, 应该使用double型, 因为更精确
    - `double num9 = 2.1234567851;` 输出结果为原数
    - `float num10 = 2.1234567851F;` 输出结果为2.1234567, 保留了7位小数

案例演示

```
float num1 = 1.1; //报错，因为float为4字节，而默认为double
float num2 = 1.1F; // 这是正确的
double num3 = 1.1; //这也是正确的
double  num4 = 1.1F; // 可以，把4字节塞进8字节
//十进制
double num5 = .123; //等价0.123
//科学计数法
double num6 = 5.12e-2; //0.0512
```

陷阱

```
double num11 = 2.7; //输出为2.7
double num12 = 8.1 / 3; // 输出为接近2.7的一个小数，而不是2.7
```

> [!ATTENTION]
> 当我们对运算结果是小数的进行相等判断时，要小心.
> **应该是以两个数的差值的绝对值，在某个精度范围类判断**
> 如果是直接查询出的小数或直接赋值，那可以判断相等

```
if(num11 == num 12){
  System.out.printIn("equal")
};
//结果不输出
//正确写法：可以通过Java的API来进行判断
if (Math.abs(num11 - num12) < 0.00001 ){
  System.out.printIn("equal")
};
// 差值非常小，到所规定的规定精度，就认为相等
```

字符类型(char)

字符类型可以表示单个字符，字符类型为char，占用内存两个字节，多个字符用string

```
char c1 = 97; //字符类型可以直接存放一个数字，97的对应就是a
char c2 = '\t'; // 两个合起来就是一个转义字符
char c3 = '啊啊啊'
char c4 = 'a'
```

字符类型可以直接存放，根源是对应不同的数字.
也就是说，如果char直接打数字，就会输出对应97的字符

字符类型使用细节

1. 字符常量是使用 `('')`，必须使用单引号，双引号就是字符串了
2. Java中还允许使用转义字符 `\` 来将其后面的字符转变为特殊字符型常量，例如: `char c3 = '\n';`
3. Java中，它的本质是一个整数，输出时所对应的是unicode码所对应的数字

```
public class CharDetail{
  public static void main(String[]args){
    char c1 = 97
    System.out.printIn(c1); //输出为a

    char c2 = 'a'
    System.out.printIn((int)c2); //输出为97
  }
}
```

4. 可以给 `char` 赋一个整数, 输出时就会按照对应的unicode字符输出
5. `char` 类型是可以进行运算的, 因为其本质是数字

```
system.out.printIn('a'+10) // 输出为107
```

字符类型的本质

1. 字符型存储到计算机中, 需要将字符对应发码值找出,
    - 比如a => 97 => 转成二进制(110 0001) => 进行存储
    - 读取: 二进制(110 0001) => 97 => a
2. 字符编码表
    1. ASCII: 一个字节表示, 一个128个字节
    2. Unicode: Unicode无论字母汉字都是两个字节
    3. UTF-8: 大小可变, 字母一个字节, 汉字3个字节
    4. GBK: 表示汉字, 范围广, 字母一个, 汉字2个
    5. Big5: 繁体中文

布尔类型: Boolean

1. 布尔类型只允许取值true或false, 无null
2. 只占一个字节
3. 适用于逻辑运算, 一般用于程序流程控制
    - if条件控制语句
    - while循环控制
    - do-while循环控制
    - for循环控制
4. 不可以用0或者非0的整数代替false和true, 没有其他的值

案例演示

```
//前面就还是那些...
boolean passExam = true;{
  if (passExam == true) {
  System.out.printIn("1")
  } else {
    system.out.printIn("2")
  }
}
```

类型转换

基本数据类型转换

- **类型转换**
    - 介绍: 当java程序在进行复制或者运算的时候, 精度小的类型会自动转换为精度大的数据类型
    - 数据类型按照精度(容量大小)排序为:
      -
- **第一条**

> [!NOTE] `char` => `int` => `long` => `float` => `double`

- **第二条**

> [!NOTE] `byte` => `short` => `int` => `long` => `float` => `double`

- **案例分析**

```
int a = 'c'; //没问题
double d = 80; //没问题
```

- **自动转换**

```
int num = 'a'; // char -> int
double d1 = 80; // int -> double
```

- **自动类型转换注意和细节**
    1. 有多种数据混合运算的时候, 系统首先自动将所有数据转换成容量最大的那种数据类型, 然后再进行计算
    2. 当我们把精度大的数据类型赋值给精度小的数据类型的时候, 就会报错, 反之, 就会自动进行数据类型转换
    3. *(byte, short)和char之间不会发生自动转换*
    4. byte, short, char 可以计算, 在进行运算的时候会首先转换成int类型
    5. boolean不参与转换
    6. 自动提升原则: 表达式的结果的类型自动提升为操作数中最大的类型\

```
//自动类型转换细节
public class AutoConvertDetail {
    public static void main(String[] args){
        //有多种数据混合运算的时候， 系统首先自动将所有数据转换成容量最大的那种数据类型，然后再进行计算
        int n1 = 10;
        float d1 = n1 +1.1; // 错，因为1.1为double类型，所以应该是↓
        double d1 = n1+ 1.1 // 对
        float d1 = n1 +1.1F // 对，因为F告诉编译器为float类型，float比int大，所以自动转换为float进行计算
        //细节: 当把精度大的数据类型赋值给精度小的数据类型的时候，就会报错，反之，就会自动进行数据类型转换
        int n2 = 1.1; // 错，double -> int
        //细节: (byte, short)和char之间不会发生自动转换
        // 当把一个具体的数赋给byte的时候，(1)先判断该数是否再byte范围内，如果是，就可以
        byte n3 = 10; //对，-128 ~ +127
        //
```

```
        int n1 = 1;
        byte b2 = n2; // 错，int比byte大
        // 如果是变量赋值，先判断变量类型，此处为int型
        char c1 = b1; // 错，原因就是因为byte不能自动转换
        //
        //byte，short，char 可以计算，在进行运算的时候会首先转换成int类型
        //
        byte b2 = 1;
        short s1 = 1;
        short s2 = b2 + s1; // 错. b2 +s1 => int
        int s2 = b2 + s1; // 对
        //
        byte b3 = 1
        byte b4 = b2 + b3 // 错，两者相加等于int
        // byte，short，char，但凡两者运算就成int

        // 布尔类型不参与运算
        boolean pass = true;
        int num100 = pass; //boolean不参与运算
        byte b4 = 1;
        short s3 = 100;
        int num200 = 1;
        double num300 = 1.11;
        double num 333 = b4 +s3 + num200 +num300 // 因为double最大，所以转换成
double


    }
}
```

- **强制类型转换**
    - 自动类型转换的逆过程，将容量大的数据类型转换成容量小的树蕨理想，使用的时候需要加上强制转换符`()`，但是可能会造成精度降低或者溢出，需要小心注意
- **案例演示**

```
int i = (int)1.9;
System.out.printIn(i);
\\ 结果为1，造成精度损失

int j = 2000;
byte b1 = (byte)n2;
\\结果为-48，造成数据溢出
```

- **强制类型转换细节说明**
    1. 当进行数据的大小从大 --> 小，就需要用到强制转换
    2. 强转符号只针对于最近的操作数有效，往往会使用小括号来提升优先级
    3. char类型可以报错int的常量值，但是不能报错int的变量值，需要强转
    4. byte和short类型在进行运算的时候，当作int类型来处理

```java
int x = (int) 10*3.5 + 6*1.5; // 报错, double=>int不行
int x = (int) (10*3.5 + 6*1.5); // 小括号提升优先级, 成功编译. 44.0 -> 44
//
char c1 = 100;
int m =100;
char c2 = m; //错误
char c3 = (char)m; // 100对应的字符
```

基本数据类型和String的转换

- **介绍:**
  - 在程序开发的时候, 我们经常需要把基本数据类型转成String类型, 或者把String类型转换成基本数据类型.
  - 基本数据类型转String类型语法: 基本类的值+ `""` 即可

```java
int n1 = 100;
String s1 = n1 + "";
```

- String类型转基本数据类型, 通过包装类调用`parseXX`即可

```java
//使用基本数据类型对应的包装类的相应方法, 得到基本数据类型
//详解在对象和方法的时候
String s5 = "123";
int num1 = Integer.parseInt(s5);\\123
double num2 = Double.parseDouble(s5);\\123.0
float num3 = Float.parseFloat(s5);\\123.0
long num4 = Long.parseLong(s5);\\123
byte num5 = Byte.parseByte(s5);\\123
boolean b = Boolean.parseBoolean("True";)\\True
short num6 = Short.parseBoolean(s5);\\123
//怎么把字符串转成char? 含义是把字符串的第一个字符得到
// s5.charAt(0) 得到s5字符串的第一个字符 '1'
System.out.printIn(s5.charAt)(0)
```

- **注意事项**
  - 在将String转换成基本数据类型的时候, 要确保试String类型能够转成有效的数据, 比如我们可以把123转成一个整数, 但是不能不能把hello转成整数
  - 如果格式不正确, 就会抛出异常

Java运算符

运算符介绍

运算符是一种特殊的符号, 用于表示数据的运算, 赋值和表示等.

1. 算数运算符
2. 赋值运算符
3. 关系运算符
4. 逻辑元素符
5. 位运算符
6. 三元运算符

算数运算符

- 介绍
    - 算数运算符是对数值类型的变量进行计算的, 在Java程序中使用的非常多.

符号	运算	Example	结果
+	正号	+7	7
-	负号	b=11;-b	-11
+	加	9+9	18
-	减	9-9	0
*	乘	7*8	56
/	除	9/9	1
%	取模(取余)	11%9	2
++	自增(前): 先运算后取值 自增(后): 先取值后运算	a=2; b=++a; a=2; b=a++;	a=3;b=3 a=3;b=2
--	自减(前): 先运算后取值 自减(后): 先取值后运算	a=2; b=--a; a=2; b=a--;	a=1;b=1 a=1;b=2
+	字符串相加	"aaa" + "bbb"	aaa bbb

算数运算符举例

```java
public class ArithmeticOperator {

    //编写一个main方法
    public static void main(String[] args) {
        // /使用
        System.out.println(10 / 4); //人算是2.5, java中是2, 详情见上一章
        System.out.println(10.0 / 4); //java是2.5
        double d = 10 / 4;//10/4=2.0, 但是先计算再赋值, 把2赋给double, 成2.0
        System.out.println(d);// 是2.0

        // % 取模 ,取余
        // 取模的本质: a % b = a - a / b * b

        System.out.println(10 % 3); //1


        // -10 % 3 => -10 - (-10) / 3 * 3 = -10 + 9 = -1
        System.out.println(-10 % 3); // -1


        // 10 % -3 = 10 - 10 / (-3) * (-3) = 10 - 9 = 1
        System.out.println(10 % -3); //1


```

```
        // -10 % -3 =  (-10) - (-10) / (-3) * (-3) = -10 + 9 = -1
        System.out.println(-10 % -3);//-1



        //++的使用
        // 当++在单独使用的时候，那么前++和后++是完全一样的
        int i = 10;
        i++;//自增 等价于 i = i + 1; => i = 11
        ++i;//自增 等价于 i = i + 1; => i = 12
        System.out.println("i=" + i);//12

        /*
        作为表达式使用
        前++：++i先自增后赋值
        后++：i++先赋值后自增
         */
        int j = 8;
        int k = ++j; // 这样写等价于两个语句，先自增j=j+1，后赋值k=j
        // 答案为9，k和j都是9
        int k = j++; // 先执行k=j(先把j的值赋给k)，后面再自增.
        // 答案为9，但是j为8
    }
}
```

算术运算符实战

> [!NOTE]
> 注意事项:
>
> 1. `System`, 不是 `system`, 大小写
> 2. 老问题... 分号忘记加

1. 59天放假, 问还有几个星期和几天

```
public class work1 {
    public static void main(String[] args) {
        int dayleft = 59;
        int weekNumber = dayleft/7;
        int restDay = dayleft%7;
        System.out.print("weeksleft" + weekNumber);
        System.out.print("daysleft" + restDay);
    }
}
```

2. 定义一个变量保存华氏温度，华氏温度转换摄氏温度的公式为5/9*(华氏温度-100),请求出华氏温度对应的摄氏温度

```java
public class work2{
    public static void main(String[] args) {
        float huaTemp = 1234.5F;
        float tempature = (huaTemp-100) * 5/9;

        System.out.println("huatemp is" + huaTemp);
        System.out.println("tempature is" + tempature);
    }
}
```

关系运算符(RelationalOperator)

- 介绍

    1. 关系运算符的结果都是boolean型号, 也就是说, 它只有两个类型, true和false
    2. 关系表达式经常用在if结构的条件中或者循环结果的条件中

运算符	运算	范例	结果
`==`	相等于	`8==7`	false
`!=`	不等于	`8!=7`	true
`<`	小于	`8<7`	false
`>`	大于	`8>7`	true
`<=`	小于等于	`8<=7`	false
`>=`	大于等于	`8>=7`	true
`instanceof`	检查是否是类的对象	`"javastudy" instanceof String`	true

> [!ATTENTION]
>
> 1. 关系运算符的结果都是boolean类型
> 2. 比较关系运算符的表达式, 我们称其为关系表达式, a>b
> 3. **比较关系符 `==` 不能写成 `=`**

- **案例演示**

```java
int a = 9;
int b = 8;
System.out.println(a > b); //T
System.out.println(a >= b); //T
System.out.println(a <= b); //F
System.out.println(a < b); //F
System.out.println(a == b); //F
System.out.println(a != b); //T
boolean flag = a > b; //T
System.out.println("flag=" + flag);
```

逻辑运算符

- **用于连接多个条件, 最终的结果也是一个Boolean值**

1. 短路与`&&`, 短路或`||`, 取反
2. 逻辑与`&`, 逻辑或`|`, 取反

a	b	a&b	a&&b	a|b	!a	a^b
true	true	true	true	true	false	false
true	false	false	false	true	true	false	true
false	true	false	false	true	true	true	true
false	false	false	false	false	false	true	false

逻辑运算的规则

1. `a&b`: `&`逻辑与: 当a和b同时为true, 则结果为true, 否则为false
2. `a&&b`: `&&`短路与: 当a和b同时为true, 则结果为true, 否则为false
3. `a|b`: `|`逻辑或: 当a和b一个为true, 结果就是true, 否则为false
4. `a||b`: `||`短路或: 当a和b有一个为true, 结果就是true, 否则为false
5. `!a`: 叫取反, 或者非运算. 当a为true的时候, 结果就是false; 当结果为false的时候, 结果就是true
6. `a^b`: 逻辑或与, 当a和b不同的时候, 结果就是true, 否则为false

`||`和`|`基本规则

名称	语法	特点
`||`短路或	条件1`||`条件2	两个条件中只要有一个成立, 结果就是true, 否则为false
`|`逻辑或	条件1`|`条件2	只要有一个条件成立, 结果就是true

- **`||`和`|`的实例演示**
    - 首先是*短路或*, age大于20但是不小于30, 输出为true
    - *逻辑或*, age大于20但是不小于30, 输出仍旧为true

```java
int age = 50;
if(age > 20 || age < 30) {
    System.out.println("true");
}
//&逻辑或
if(age > 20 | age < 30) {
    System.out.println("ok200");
}
```

- 区别?
    - **短路或**: 如果第一个条件为true, 则第二个条件不会进行判断, 最终结果直接输出为true, *效率较高*
    - **逻辑或**: 不管第一个条件是否为true，第二个条件都要判断，*效率低*

对比代码:短路或

原因?

第二个条件不进行判断, 因此`||`后面的b++不会进行运算, 直接输出b=9

```java
int a = 4;
int b = 9;
if( a > 1 || ++b > 4) {
    System.out.println("true");
}
System.out.println("a=" + a + " b=" + b);
// 输出为a=4, b=9
```

对比代码:逻辑或

原因?

第二个条件在逻辑或下, 无论第一个怎么样, 在`|`的后面的b++会进行运算, 会输出10

```java
int a = 4;
int b = 9;
if( a > 1 | ++b > 4) {
    System.out.println("true");
}
System.out.println("a=" + a + " b=" + b);
// 输出为a=4, b=10
```

`!`取反 基本规则

名称	语法	特点
`!`非(取反)	!条件	如果条件本身成立, 结果为false,否则为true

案例演示`!`取反

```java
//! 操作是取反 T->F  , F -> T
System.out.println(60 > 20); //T
System.out.println(!(60 > 20)); //F
```

案例演示`^`逻辑异或

a为true,b为false, 二者不同, 所以为true

```java
//a^b: 叫逻辑异或，当 a 和 b 不同时，则结果为true, 否则为false
boolean b = (10 > 1) ^ (3 > 5);
System.out.println("b=" + b);//T
```

该赋值运算符了