# Java语法

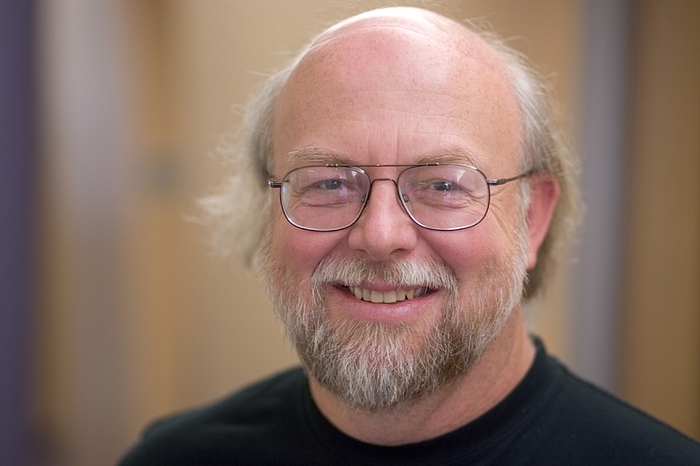
## 第一章 Java语言简介

### 第一节 Java语言概述

#### 什么是Java语言

**Java语言**是美国Sun公司（Stanford University Network），在1995年推出的高级的编程语言。所谓编程语言，是计算机的语言，人们可以使用编程语言对计算机下达命令，让计算机完成人们需要的功能。

下图是java语言之父 **James Gosling（詹姆斯·高斯林）**



#### Java语言发展历史

1995 年Sun公司发布Java1.0版本

1997 年发布Java 1.1版本

1998 年发布Java 1.2版本

2000 年发布Java 1.3版本

2002 年发布Java 1.4版本

2004 年发布Java 1.5版本

2006 年发布Java 1.6版本

2009 年Oracle甲骨文公司收购Sun公司，并于2011发布Java 1.7版本

2014 年发布Java 1.8版本

2017 年发布Java 9.0版本

#### Java语言能做什么

Java语言主要应用在互联网程序的开发领域。常见的互联网程序比如天猫、京东、物流系统、网银系统等，以及服

务器后台处理大数据的存储、查询、数据挖掘等也有很多应用。

### 第二节 Java原生开发环境

#### 1. Java虚拟机(JVM)

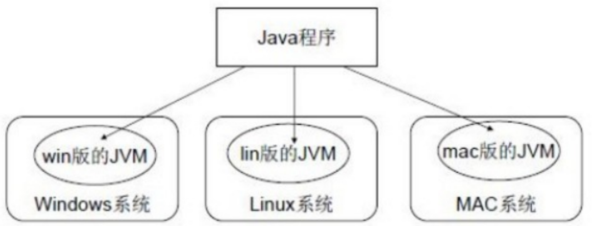
**JVM**（Java Virtual Machine ）：Java虚拟机，简称JVM，是运行所有Java程序的假想计算机，是Java程序的

运行环境，是Java 最具吸引力的特性之一。我们编写的Java代码，都运行在 JVM 之上。

**跨平台** ：任何软件的运行，都必须要运行在操作系统之上，而我们用Java编写的软件可以运行在任何的操作系

统上，这个特性称为Java语言的跨平台特性。该特性是由JVM实现的，我们编写的程序运行在JVM上，而JVM

运行在操作系统上。



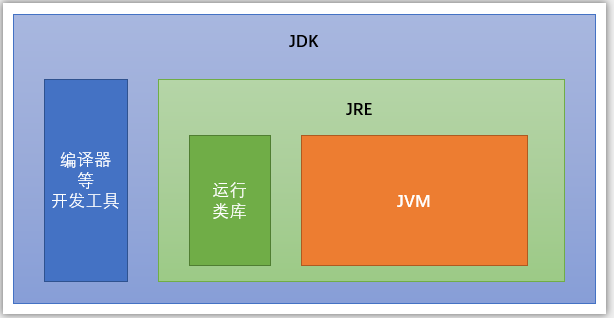
如图所示，Java的虚拟机本身不具备跨平台功能的，每个操作系统下都有不同版本的虚拟机。

#### 2. JRE和JDK

**JRE** (Java Runtime Environment) ：是Java程序的运行时环境，包含 JVM 和运行时所需要的 核心类库 。

**JDK** (Java Development Kit)：是Java程序开发工具包，包含 JRE 和开发人员使用的工具。

它们之间的关系如下图所示：



我们想要运行一个已有的Java程序，那么只需安装 JRE 即可。

我们想要开发一个全新的Java程序，那么必须安装 JDK 。

注意：JDK安装后还需配置环境变量

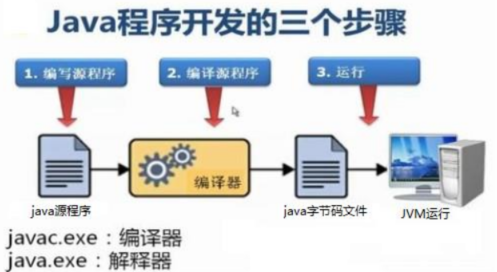
**JAVA\_HOME** = JDK安装路径(如D:\java\jdk-9.0.1)

**PATH** += %JAVA\_HOME%\bin

#### 3. Java程序开发三步骤

开发环境已经搭建完毕，可以开发我们第一个Java程序了。

Java程序开发三步骤：**编写 -> 编译 -> 运行**。



#### 4. 第一个Java程序

**(1) 编写源文件**

在一个方便的目录新建后缀名为.java的HellWorld文件，输入以下代码：

public class **HelloWorld** {

       public static void **main**(String[] args) {

         System.out.**println**("Hello World!");

       }

    }

注意：文件名必须是 HelloWorld ，保证文件名和类的名字是一致的，注意大小写。

附HelloWorld.java文件如下：



**(2) 编译Java源文件**

在DOS命令行中，**进入Java源文件的目录**，使用 javac 命令进行编译。

命令格式如下：

javac Java源文件名.后缀名

例如：

javac HelloWorld.java

编译成功后，命令行没有任何提示。打开文件目录，发现产生了一个新的文件 HelloWorld.class ，该文件

就是编译后的文件，是Java的可运行文件，称为**字节码文件**，有了字节码文件，就可以运行程序了。

注意：Java源文件的编译工具 javac.exe ，在JDK安装目录的bin目录下。但是由于配置了环境变量，可以再任意目录下使用。

**(3) 运行java程序(字节码)**

在DOS命令行中，进入Java源文件的目录，使用 java 命令进行运行。

命令格式如下：

java 类名字

例如：

java HelloWorld

注意：java HelloWord 不要写 不要写 不要写 .class

#### 5. 关于第一个程序的几点说明

**(1) 关于main方法**

**main 方法**：称为主方法。写法是**固定格式**不可以更改。main方法是程序的入口点或起始点，无论我们编写多

少程序，JVM在运行的时候，都会从main方法这里开始执行。

**(2) 关于java中的注释**

**注释 ：**就是对代码的解释和说明。其目的是让人们能够更加轻松地了解代码。为代码添加注释，是十分必须

要的，它不影响程序的编译和运行。

Java 中有单行注释和多行注释

单行注释以 **//** 开头 **换行** 结束

多行注释以 **/\*** 开头  以 **\*/** 结束

**(3) 关于关键字KeyWords**

**关键字 ：**是指在程序中，Java已经定义好的单词，具有特殊含义。

HelloWorld 案例中，出现的关键字有  **public** 、 **class** 、  **static** 、 **void** 等，这些单词已经被

Java定义好，**全部都是小写字母**，notepad++中颜色特殊。

**(4) 关于标识符**

**标识符 ：**是指在程序中，我们自己定义内容。比如类的名字、方法的名字和变量的名字等等，都是标识符。

HelloWorld 案例中，出现的标识符有类名字 **HelloWorld** 。

**命名规则： 硬性要求**

① 标识符可以包含 **英文字母 26个(区分大小写)** 、 **0-9数字** 、 **$（美元符号）** 和 **\_（下划线）** 。

② 标识符不能以数字开头。

③ 标识符不能是关键字。··

**命名规范： 软性建议**

① 类名规范：首字母大写，后面每个单词首字母大写（大驼峰式）。

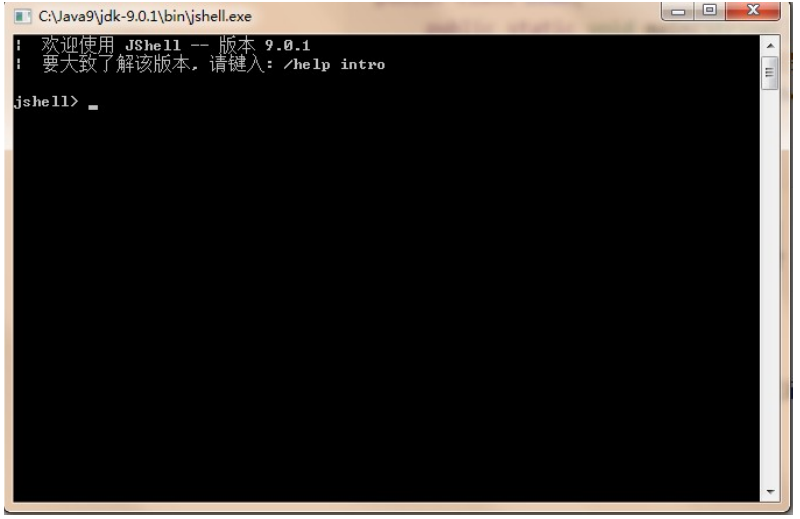
② 方法名规范： 首字母小写，后面每个单词首字母大写（小驼峰式）。

③ 变量名规范：全部小写。

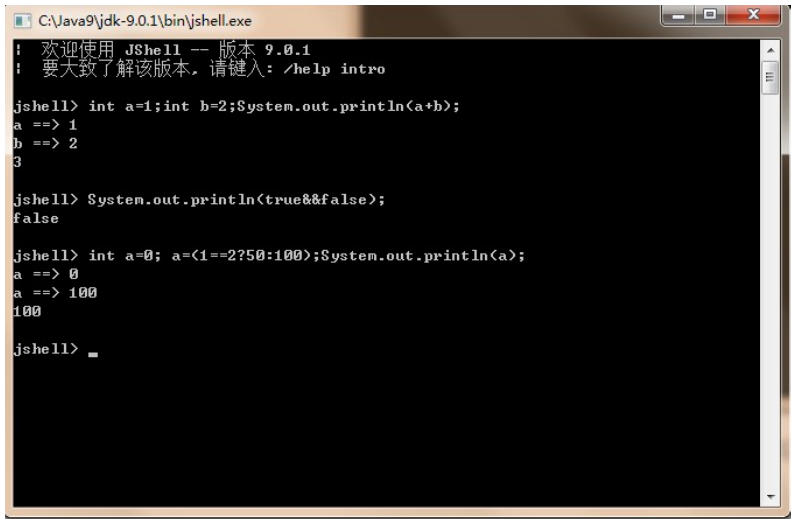
#### 6. Jshell脚本

什么时候会用到 JShell 工具呢，当我们编写的代码非常少的时候，而又不愿意编写类，main方法，也不愿意去编译和运行，这个时候可以使用JShell工具。

启动JShell工具，在DOS命令行直接输入JShell命令。



接下来可以编写Java代码，无需写类和方法，直接写方法中的代码即可，同时无需编译和运行，直接回车即可



### 第三节 Java集成开发环境

1. IntelliJ IDEA概述

2. 使用IDEA运行HelloWorld程序

3. IDEA 的一些常用设置

4. IDEA 的一些常用快捷键

## 第二章 Java语言基础

### 第一节 常量与变量

#### 1. java中的六类常量

**常量：**是指在Java程序中固定不变的数据。

Java中常量分类如下：

|  |  |  |
| --- | --- | --- |
| 类型 | 含义 | 数据举例 |
| 整数常量 | 所有的整数 | 0，1， 567， -9 |
| 小数常量 | 所有的小数 | 0.0， -0.1， 2.55 |
| 字符常量 | 单引号引起来,只能写一个字符,必须有内容 | 'a' ， ' '， '好' |
| 字符串常量 | 双引号引起来,可以写多个字符,也可以不写 | "A" ，"Hello" ，"你好" ，"" |
| 布尔常量 | 只有两个值（流程控制中讲解） | true ， false |
| 空常量 | 只有一个值（引用数据类型中讲解） | null |

注意：

① Java中，字符常量的单引号内只能有一个字符，中英文皆可，但不能写两个及以上的字符，如~~’AB’~~是错误的。

② 空常量不能直接用来打印输出，如·~~System.out.println(null)~~是错误的。

一些常量用法的例子如下：



#### 2. 变量与数据类型

**变量：**常量是固定不变的数据，那么在程序中可以变化的量称为变量。

Java中要求一个变量每次只能保存一个数据，必须要明确保存的数据类型。Java中数据类型分为两大类：

**基本数据类型 ：**包括 整数 、 浮点数 、 字符 、 布尔 。

**引用数据类型 ：**包括 类 、 数组 、 接口 、**字符串** 、lamda表达式等

在Java(jdk9.0)中，基本数据类型有且只有**四类八种**，分别如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 关键字 | 内存占用 | 取值范围 |
| 字节型 | byte | 1个字节 | -128~127 |
| 短整型 | short | 2个字节 | -32768~32767 |
| 整型 | int（默认） | 4个字节 | -231次方~2的31次方-1 |
| 长整型 | long | 8个字节 | -2的63次方~2的63次方-1 |
| 单精度浮点数 | float | 4个字节 | 1.4013E-45~3.4028E+38 |
| 双精度浮点数 | double（默认） | 8个字节 | 4.9E-324~1.7977E+308 |
| 字符型 | char | 2个字节 | 0-65535 |
| 布尔类型 | boolean | 1个字节 | true，fals |

注意几点：

① Java中的默认类型：整数类型是 int 、浮点类型是 double 。

特别的，long类型：建议数据后加**L**表示。float类型：建议数据后加**F**表示。

② 数据范围与字节数不一定相关，例如float数据范围比long更加广泛，但是float是4字节，long是8字节。

③ 在java中，字符型占用的是两个字节

#### 3. 变量的定义

变量定义的格式包括三个要素： **数据类型** 、 **变量名** 、 **数据值** 。

格式

数据类型 变量名 = 数据值;

练习

定义所有基本数据类型的变量，代码如下：

public class **Variable** {

    public static void **main**(String[] args){

*//定义字节型变量*

            byte b = 100;

            System.out.**println**(b);

*//定义短整型变量*

            short s = 1000;

            System.out.**println**(s);

*//定义整型变量*

            int i = 123456;

            System.out.**println**(i);

*//定义长整型变量*

            long l = 12345678900L;

            System.out.**println**(l);

*//定义单精度浮点型变量*

            float f = 5.5F;

            System.out.**println**(f);

*//定义双精度浮点型变量*

            double d = 8.5;

            System.out.**println**(d);

*//定义布尔型变量*

            boolean bool = false;

            System.out.**println**(bool);

*//定义字符型变量*

            char c = 'A';

            System.out.**println**(c);

    }

}

其它一些变量定义的例子：



注意：

① 对于float和long类型来说，字母后缀F和L不要丢掉。

② 在java中，未初始化的变量不能使用。

③ 如果使用byte或者short类型的变量，那么右侧的数据值不能超过左侧类型的范围。

④ 可以通过一个语句来创建多个变量，但是一般情况不推荐这么写。

具体可看下面示例：



### 第二节 数据类型转换

Java程序中要求参与的计算的数据，必须要保证数据类型的一致性，如果数据类型不一致将发生类型的转换。

#### 1. 自动类型转换

**自动转换 ：**将 取值范围小的类型 自动提升为 取值范围大的类型 。如以下示例：

public static void **main**(String[] args) {

        int i = 1;

        byte b = 2;

*// byte x = b + i; // 报错*

*//int类型和byte类型运算，结果是int类型*

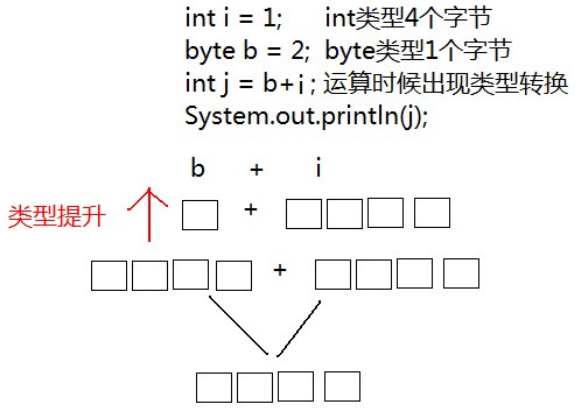
        int j = b + i;

        System.out.**println**(j);

    }

byte 类型内存占有1个字节，在和 int 类型运算时会提升为 int 类型 ，自动补充3个字节，因此计算后的结果还是 int 类型。

图解如下：



同样道理，当一个 int 类型变量和一个 double 变量运算时， int 类型将会自动提升为 double 类型进行运算。

public static void **main**(String[] args) {

        int i = 1;

        double d = 2.5;

*//int类型和double类型运算，结果是double类型*

*//int类型会提升为double类型*

        double e = d+i;

        System.out.**println**(e);

    }

**自动类型转换规则**

范围小的类型向范围大的类型提升， byte 、short、char 运算时直接提升为 int 。

更多例子如下：



#### 2. 强制转换转换

将 1.5 赋值到 int 类型变量会发生什么？产生编译失败，肯定无法赋值。

int i = 1.5; *// 错误*

double 类型内存8个字节， int 类型内存4个字节。 1.5 是 double 类型，取值范围大于 int 。可以理解为 double 是8升的水壶， int 是4升的水壶，不能把大水壶中的水直接放进小水壶去。

想要赋值成功，只有通过强制类型转换，将 double 类型强制转换成 int 类型才能赋值。

**强制类型转换 ：**将 取值范围大的类型 强制转换成 取值范围小的类型 。

比较而言，自动转换是Java自动执行的，而强制转换需要我们自己手动执行。

强制类型转换的格式如下：

数据类型 变量名 = （数据类型）被转数据值；

将 1.5 赋值到 int 类型，代码修改为：

*// double类型数据强制转成int类型，直接去掉小数点。*

int i = (int)1.5;

同样道理，当一个 short 类型与 1 相加，我们知道会类型提升，但是还想给结果赋值给short类型变量，就需要强制转换。

public static void **main**(String[] args) {

*//short类型变量，内存中2个字节*

         short s = 1;

*/\**

*出现编译失败*

*s和1做运算的时候，1是int类型，s会被提升为int类型*

*s+1后的结果是int类型，将结果在赋值会short类型时发生错误*

*short内存2个字节，int类型4个字节*

*必须将int强制转成short才能完成赋值*

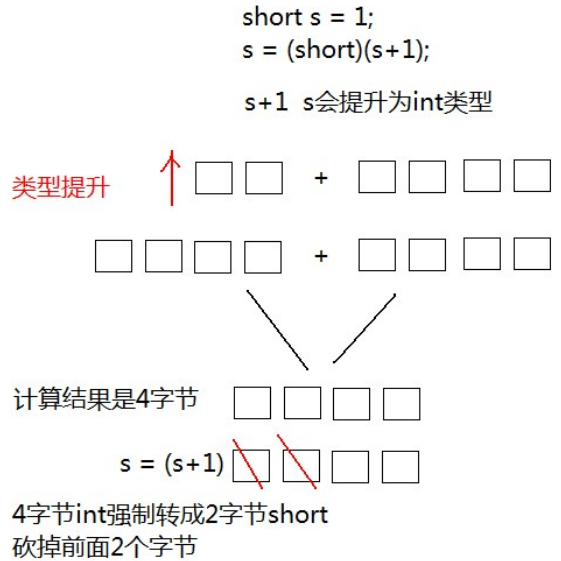
*\*/*

         s = s + 1；*//编译失败*

         s = (short)(s+1);*//编译成功*

    }

图解如下：



更多强制类型转换例子和注意事项如下：



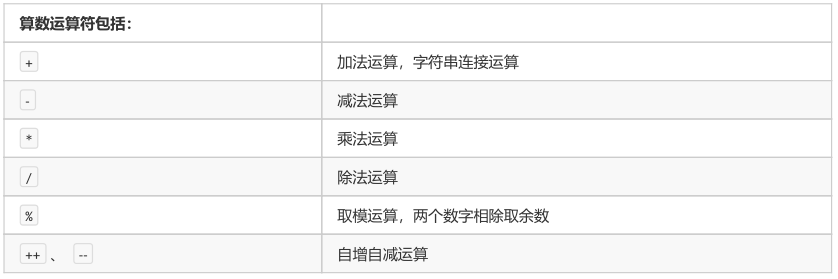
#### 3. 类型转换注意

(1) 浮点转成整数，直接取消小数点，可能造成数据损失精度。

(2) int 强制转成 short 砍掉2个字节，可能造成数据丢失

### 第三节 运算符

#### 1. 算术运算符



Java中，整数使用以上运算符，无论怎么计算，也不会得到小数。

如下：

public static void **main**(String[] args) {

       int i = 1234;

       System.out.**println**(i/1000\*1000);*//计算结果是1000*

    }

更多示例：

 ………………

特别说明：

**++ 运算，变量自己增长1。**反之， -- 运算，变量自己减少1，用法与 ++ 一致。有两种用法：

**(1) 独立运算：**

变量在独立运算时， 前 ++ 和 后 ++ 没有区别 。

变量 前 ++ ：例如 ++i 。

变量 后 ++ ：例如 i++

**(2) 混合运算：**

和其他变量放在一起， 前 ++ 和 后 ++ 就产生了不同。

变量 前 ++ ：变量a自己加1，将加1后的结果赋值给b，也就是说a先计算。a和b的结果都是2。

public static void **main**(String[] args) {

        int a = 1;

        int b = ++a;

        System.out.**println**(a);*//计算结果是2*

        System.out.**println**(b);*//计算结果是2*

    }

变量 后 ++ ：变量a先把自己的值1，赋值给变量b，此时变量b的值就是1，变量a自己再加1。a的结果是2，b的结果是1。

public static void **main**(String[] args) {

        int a = 1;

        int b = a++;

        System.out.**println**(a);*//计算结果是2*

        System.out.**println**(b);*//计算结果是1*

    }

**+ 符号在字符串中的操作：**

+ 符号在遇到字符串的时候，表示连接、拼接的含义。

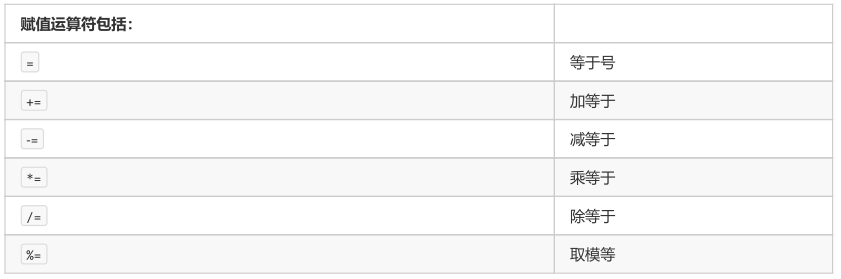
"a"+"b" 的结果是“ab”，连接含义

public static void **main**(String[] args){

      System.out.**println**("5+5="+5+5);*//输出5+5=55*

    }

#### 2. 赋值运算符



赋值运算符，就是将符号右边的值，赋给左边的变量。

public static void **main**(String[] args){

        int i = 5;

        i+=5;*//计算方式 i=i+5 变量i先加5，再赋值变量i*

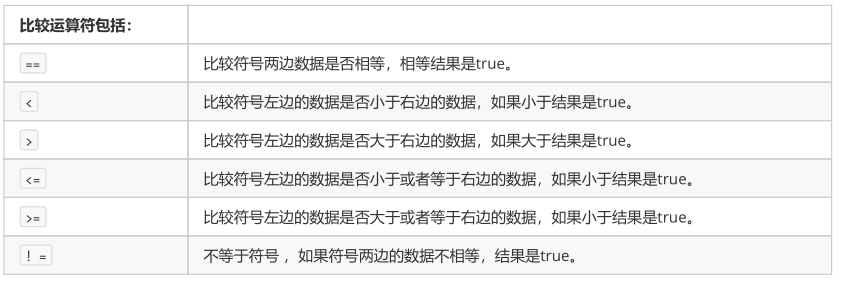
        System.out.**println**(i); *//输出结果是10*

    }

更多示例如下：



#### 3. 比较运算符



比较运算符，是两个数据之间进行比较的运算，运算结果都是布尔值 true 或者 false 。

public static void **main**(String[] args) {

        System.out.**println**(1==1);*//true*

        System.out.**println**(1<2);*//true*

        System.out.**println**(3>4);*//false*

        System.out.**println**(3<=4);*//true*

        System.out.**println**(3>=4);*//false*

        System.out.**println**(3!=4);*//true*

    }

更多示例：



#### 4. 逻辑运算符



逻辑运算符，是用来连接两个布尔类型结果的运算符，运算结果都是布尔值 true 或者 false

public static void **main**(String[] args)  {

        System.out.**println**(true && true);*//true*

        System.out.**println**(true && false);*//false*

        System.out.**println**(false && true);*//false，右边不计算*

        System.out.**println**(false || false);*//falase*

        System.out.**println**(false || true);*//true*

        System.out.**println**(true || false);*//true，右边不计算*

        System.out.**println**(!false);*//true*

    }

更多示例：



#### 5. 三元运算符

**三元运算符格式：**

数据类型 变量名 = 布尔类型表达式？结果1：结果2

**三元运算符计算方式：**

布尔类型表达式结果是 true，三元运算符整体结果为结果1，赋值给变量。

布尔类型表达式结果是 false，三元运算符整体结果为结果2，赋值给变量。

public static void **main**(String[] args) {

        int i = (1==2 ? 100 : 200);

        System.out.**println**(i);*//200*

        int j = (3<=4 ? 500 : 600);

        System.out.**println**(j);*//500*

    }

更多示例：



#### 6. 一些注意事项

**(1) +=运算符的扩展**

下面的程序有问题吗？

public static void **main**(String[] args){

      short s = 1;

      s+=1;

      System.out.**println**(s);

    }

**分析：**  s += 1 逻辑上看作是 s = s + 1 计算结果被提升为int类型，再向short类型赋值时发生错误，因为不能将取值范围大的类型赋值到取值范围小的类型。但是， s=s+1 进行两次运算 ， += 是一个运算符，只运算一次，并带有强制转换的特点，

**也就是说 s += 1 就是 s = (short)(s + 1)** ，因此程序没有问题编译通过，运行结果是2.

**(2) 常量和变量的运算(编译器的常量优化)**

下面的程序有问题吗？

public static void **main**(String[] args){

      byte b1=1;

      byte b2=2;

      byte b3=1 + 2;

      byte b4=b1 + b2;

      System.out.**println**(b3);

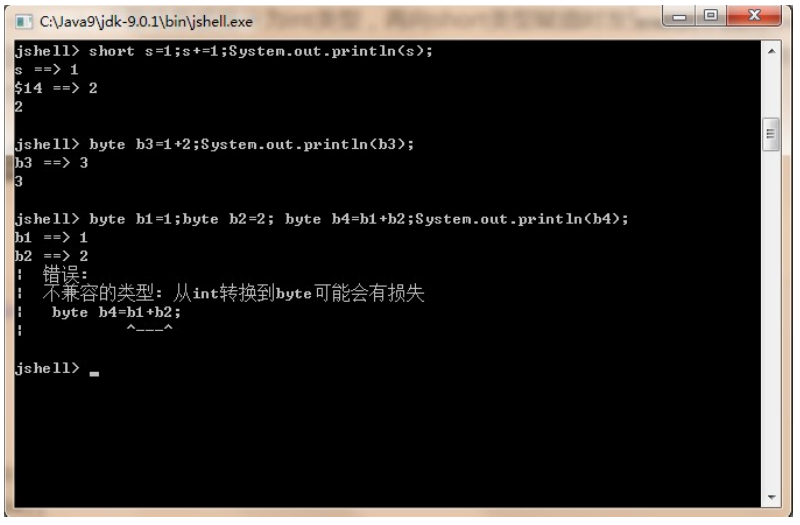
      System.out.**println**(b4);

    }

**分析：** b3 = 1 + 2 ， 1 和 2 是常量，为固定不变的数据，在编译的时候（编译器javac），已经确定了 1+2 的结果并没有超过byte类型的取值范围，可以赋值给变量 b3 ，因此 b3=1 + 2 是正确的。

反之， b4 = b2 + b3 ， b2 和 b3 是变量，变量的值是可能变化的，在编译的时候，编译器javac不确定b2+b3的结果是什么，因此会将结果以int类型进行处理，所以int类型不能赋值给byte类型，因此编译失败。

在jshell中体现：



更多例子：

………………………

### 第四节 流程控制

#### 1. 流程控制与顺序结构

#### 2. 判断语句

#### 3. 选择语句

#### 4. 循环语句

### 第五节 数组

#### 1. 数组的定义和访问

#### 2. 数组的原理内存图

#### 3. 数组的常见操作

#### 4. 数组作为方法参数和返回值

### 第六节 方法

我们在学习运算符的时候，都为每个运算符单独的创建一个新的类和main方法，我们会发现这样编写代码非常的繁琐，而且重复的代码过多。能否避免这些重复的代码呢，就需要使用方法来实现。

**方法：** 就是将一个功能抽取出来，把代码单独定义在一个大括号内，形成一个单独的功能。

当我们需要这个功能的时候，就可以去调用。这样即实现了代码的复用性，也解决了代码冗余的现象。

#### 1. 方法的定义

**定义格式：**

修饰符 返回值类型 方法名 （参数列表）｛

     代码...

    return ;

｝

**定义格式解释：**

**修饰符：** 目前固定写法 public static 。

**返回值类型：** 目前固定写法 void ，其他返回值类型在后面的课程讲解。

**方法名：**为我们定义的方法起名，满足标识符的规范，用来调用方法。

**参数列表：** 目前无参数， 带有参数的方法在后面的课程讲解。

**return ：**方法结束。因为返回值类型是void，方法大括号内的return可以不写。

一个简单的例子：

public static void **methodName**() {

       System.out.**println**("这是一个方法");

    }

在定义方法时的一些注意事项：

(1) 方法必须定义在一类中方法外, 不能定义在另一个方法的里面

public class **Demo** {

        public static void **main**(String[] args){

        }

*//正确写法，类中，main方法外面可以定义方法*

        public static void **method**(){}

    }

public class **Demo** {

    public static void **main**(String[] args){

*//错误写法，一个方法不能定义在另一方法内部*

        public static void **method**(){}

    }

}

#### 2. 方法的调用

方法在定义完毕后，方法不会自己运行，必须被调用才能执行，我们可以在主方法main中来调用我们自己定义好的方法。在主方法中，直接写要调用的方法名字就可以调用了。

public static void **main**(String[] args) {

*//调用定义的方法method*

**method**();

    }

*//定义方法，被main方法调用*

    public static void **method**() {

       System.out.**println**("自己定义的方法，需要被main调用运行");

    }

**一个简单的例子：**

将三元运算符代码抽取到自定义的方法中，并调用。

public static void **main**(String[] args) {

*//调用定义的方法operator*

**operator**();

    }

*//定义方法，方法中定义三元运算符*

    public static void **operator**() {

        int i = 0;

        i = (1==2 ? 100:200);

        System.out.**println**(i);

        int j = 0 ;

        j = (3<=4 ? 500:600);

        System.out.**println**(j);

    }

更多简单例子：



使用方法时的一些注意事项

#### 3. 方法的重载

## 第三章 Java面向对象

### 第一节 面向对象思想

### 第二节 封装与常用的Java类

### 第三节 继承与抽象类

### 第四节 接口与多态

### 第五节 内部类与权限

## 第四章 Java泛型与容器

第一节

## 第五章 Java系统编程

## 第六章 Java网络编程

## 第七章 JDBC