Java

一、变量与常量

1、Java中的关键字

Java 语言中有一些具有特殊用途的词被称为关键字。关键字对 Java 的编译器有着特殊的意义，在程序中应用时一定要慎重哦！！

Java 中常用关键字：

abstract	boolean	break	byte	case	catch
char	class	continue	default	do	double
else	extends	false	final	finally	float
for	if	implements	import	instanceof	int
interface	long	native	new	null	package
private	protected	public	return	short	static
super	switch	synchronized	this	throw	throws
transient	true	try	void	volatile	while

2、认识Java标识符

问：标识符是神马？

答：标识符就是用于给 Java 程序中变量、类、方法等命名的符号。

```
package com.imooc;
public class Demo02 {
    public static void main(String[] args) {
        System.out.println("hello imooc");
    }
}
```

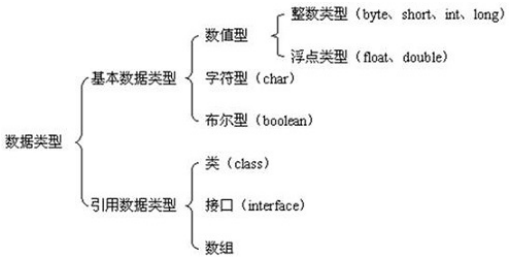
标识符

使用标识符时，需要遵守几条规则：

1. 标识符**可以**由**字母、数字、下划线（_）、美元符（$）**组成，但不能包含 @、%、空格等其它特殊字符，**不能以数字开头**。譬如：123name 就是不合法滴
2. 标识符**不能是 Java 关键字和保留字**（ Java 预留的关键字，以后的升级版本中有可能作为关键字），但可以包含关键字和保留字。如：不可以使用 void 作为标识符，但是 Myvoid 可以
3. 标识符是**严格区分大小写**的。 所以涅，一定要分清楚 imooc 和 IMooc 是两个不同的标识符哦！
4. 标识符的命名最好能反映出其作用，做到**见名知意**。

3、Java中的数据类型

Java 语言是一种**强类型**语言。通俗点说就是，在 Java 中存储的数据都是有类型的，而且必须在编译时就确定其类型。 Java 中有两类数据类型：



在 Java 的领域里，**基本数据类型变量**存的是**数据本身**，而**引用类型变量**存的是保存**数据的空间地址**。说白了，基本数据类型变量里存储的是直接放在抽屉里的东西，而引用数据类型变量里存储的是这个抽屉的钥匙，钥匙和抽屉一一对应。

常用的基本数据类型有：

数据类型	说　明	字节	应用	示例
int	整型	4	用于存储整数，如年龄、个数	int age=21;
double	双精度浮点型	8	用于存储小数，如商品价格、成绩平均分	double price=15.5;
float	单精度浮点型	4	用于存储小数,如身高	float height=175.2f;
char	字符型	2	用于存储单个字符，如性别‘男’、‘女’	char sex='男';
boolean	布尔型	1	表示“真”或“假”，取值只能为true或false	boolean flag=true;

你可能已经注意到了：

```
float height=175.2f;
char level='A';
```

为float变量赋值时在数值后添加字母f

为char变量赋值时使用单引号(')引起来

String 是一种常见的**引用数据类型**，用来表示字符串。在程序开发中，很多操作都要使用字符串来完成，例如系统中的用户名、密码、电子邮箱等

4、Java中变量的使用规则

1、Java 中的变量需要先声明后使用

```
3    public static void main(String[] args) {
4        System.out.println("变量love的内容是："+love);
5    }
```

变量未声明

2、变量使用时，可以声明变量的同时进行初始化

```
String love="imooc";
```

，

也可以先声明后赋值

```
String love;
love="i love imooc";
```

3、变量中每次只能赋一个值，但可以修改多次

```
String love="imooc","i love imooc";
```
变量中只能赋一个值

4、main 方法中定义的变量必须先赋值，然后才能输出

```
String love;
System.out.println("变量love的内容是："+love);
```
变量输出前必须先赋值

5、虽然语法中没有提示错误，但在实际开发中，变量名不建议使用中文，容易产生安全隐患，譬如后期跨平台操作时出现乱码等等

```
String name = "张三";
String 姓名 = "张三";
```
不建议在程序中应用中文变量名，切记，切记

5、Java中的自动类型转换

在 Java 程序中，不同的基本数据类型的数据之间经常需要进行相互转换。

当然自动类型转换是需要**满足特定的条件**的：

1. 目标类型能与源类型兼容，如 double 型兼容 int 型，但是 char 型不能兼容 int 型

```
int age = 19;
char sex = '女';
char result = age + sex;
```
int类型不能自动转换为char类型

2. 目标类型**大于**源类型，如 double 类型长度为 8 字节， int 类型为 4 字节，因此 double 类型的变量里直接可以存放 int 类型的数据，但反过来就不可以了

```
double avg1 = 75.5;
int avg2 = avg1;
```
double类型不能自动转换为int类型

6、Java中的强制类型转换

相信小伙伴们也发现了，尽管自动类型转换是很方便的，但并不能满足所有的编程需要。

例如，当程序中需要将 double 型变量的值赋给一个 int 型变量，该如何实现呢？

显然，这种转换是不会自动进行的！因为 int 型的存储范围比 double 型的小。此时就需要通过**强制类型转换**来实现了。

语法：**(数据类型) 数值**

```
double avg1=75.8;
int avg2=(int)avg1;
System.out.println(avg1);
System.out.println(avg2);
```
将double类型强制转换为int类型

运行结果：

```
75.8
75
```

可以看到，通过强制类型转换将 75.8 赋值给 int 型变量后，结果为 75，数值上并未进行四舍五入，而是直接将小数位截断。

明白了吧，**强制类型转换可能会造成数据的丢失哦**，小伙伴们在应用时一定要慎重哦！

7、Java常量的应用

所谓常量，我们可以理解为是一种特殊的变量，它的值被设定后，在**程序运行过程中不允许改变**。

语法：**final 常量名 = 值;**

```
final String LOVE="IMOOC";
final double PI=3.14;
```
使用final关键字　常量名　常量值

程序中使用常量可以提高代码的**可维护性**。例如，在项目开发时，我们需要指定用户的性别，此时可以定义一个常量 SEX，赋值为 "男"，在需要指定用户性别的地方直接调用此常量即可，避免了由于用户的不规范赋值导致程序出错的情况。

伙计们注意啦：常量名一般使用**大写字符**

8、如何在Java中使用注释

在编写程序时，经常需要添加一些注释，用以描述某段代码的作用。

一般来说，对于一份规范的程序源代码而言，注释应该占到源代码的 1/3 以上。因此，注释是程序源代码的重要组成部分，一定要加以重视哦！

Java 中注释有三种类型：**单行注释、多行注释、文档注释**

```
package com.imooc;
/**
 * 这是文档注释
 * @author laurenyang
 * @version v1.0
 */
public class Demo03 {
    /*
     * 这是多行注释
     * 可以包括多行内容哦
     */
    public static void main(String[] args) {
        //这是单行注释
        System.out.println("Hello Imooc!");
        //System.out.println("Hello World!");
    }
}
```
文档注释以/**开头，以*/结尾

多行注释以/*开头，以*/结尾

单行注释以//开头，行末结尾

运行结果： **Hello Imooc!**

看：被注释的代码块在程序运行时是不会被执行的~~

我们可以通过 **javadoc** 命令从文档注释中提取内容，生成程序的 API 帮助文档。

```
E:\>javadoc -d doc Demo03.java
正在创建目标目录: "doc\"
正在加载源文件Demo03.java...
正在构造 Javadoc 信息...
标准 Doclet 版本 1.7.0_13
正在构建所有程序包和类的树...
正在生成doc\com\imooc\Demo03.html...
正在生成doc\com\imooc\package-frame.html...
正在生成doc\com\imooc\package-summary.html...
正在生成doc\com\imooc\package-tree.html...
正在生成doc\constant-values.html...
正在构建所有程序包和类的索引...
正在生成doc\overview-tree.html...
正在生成doc\index-all.html...
正在生成doc\deprecated-list.html...
正在构建所有类的索引...
正在生成doc\allclasses-frame.html...
正在生成doc\allclasses-noframe.html...
正在生成doc\index.html...
正在生成doc\help-doc.html...
```

打开首页，查看下生成的 API 文档

PS: 使用文档注释时还可以使用 **javadoc 标记**，生成更详细的文档信息：

@author 标明开发该类模块的作者

@version 标明该类模块的版本

@see 参考转向，也就是相关主题

@param 对方法中某参数的说明

@return 对方法返回值的说明

@exception 对方法可能抛出的异常进行说明

二、运算符

1、Java中的条件运算符

条件运算符（ ?: ）也称为“三元运算符”。

语法形式：**布尔表达式 ？ 表达式1 ：表达式2**

运算过程：如果布尔表达式的值为 **true** ，则返回 **表达式1** 的值，否则返回 **表达式2** 的值

例如：

```
String str = (8>5) ? "8大于5" : "8不大于5" ;
System.out.println( str );
```

因为，表达式 8>5 的值为 true，所以，返回： **8大于5**

三、数组

1、使用 Arrays 类操作 Java 中的数组

Arrays 类是 Java 中提供的一个工具类，在 java.util 包中。该类中包含了一些方法用来直接操作数组，比如可直接实现数组的排序、搜索等。

Arrays 中常用的方法：

1、 排序

语法： **Arrays.sort(数组名);**

可以使用 sort( ) 方法实现对数组的排序，只要将数组名放在 sort( ) 方法的括号中，就可以完成对该数组的排序（按升序排列），如：

```
1 import java.util.Arrays;→使用Arrays类时必须添加此行，导入包
2 public class HelloWorld {
3     public static void main(String[] args) {
4         // 定义一个整型数组
5         int[] scores = { 78, 93, 97, 84, 63 };
6         // 使用Arrays类的sort()方法对数组进行排序
7         Arrays.sort(scores);
8         System.out.println("排序后数组中元素的值：");
9         // 循环遍历输出数组中元素的值
10        for (int i = 0; i < scores.length; i++) {
11            System.out.println(scores[i]);
12        }
13    }
14 }
```

运行结果：

排序后数组中元素的值：

63
78
84
93
97

2、 将数组转换为字符串

语法： **Arrays.toString(数组名);**

可以使用 toString( ) 方法将一个数组转换成字符串，该方法按顺序把多个数组元素连接在一起，多个元素之间使用逗号和空格隔开，如：

```
import java.util.Arrays;——▶使用Arrays类时必须添加此行，导入包
public static void main(String[] args) {
    // 定义一个整型数组
    int[] nums = new int[] { 25, 7, 126, 53, 14, 86 };
    // 使用Arrays类的toString()方法将数组转换为字符串并输出
    System.out.println("输出数组nums中的元素：" + Arrays.toString(nums));
}
```

运行结果为：

输出数组nums中的元素：[25，7，126，53，14，86]

2、使用 foreach 操作数组

foreach 并不是 Java 中的关键字，是 for 语句的特殊简化版本，在遍历数组、集合时， foreach 更简单便捷。从英文字面意思理解 foreach 也就是“ for 每一个”的意思，那么到底怎么使用 foreach 语句呢？

语法：

```
for(元素类型  元素变量 : 遍历对象){
    执行的代码
}
```

我们分别使用 for 和 foreach 语句来遍历数组

```
public static void main(String[] args) {
    // 定义一个字符串数组
    String[] hobbys = { "imooc", "爱慕课", "www.imooc.com" };
    System.out.println("***使用for循环输出数组中的元素");
    for (int i = 0; i < hobbys.length; i++) {
        System.out.println(hobbys[i]);
    }

    System.out.println();
    System.out.println("***使用foreach循环输出数组中的元素");
    for (String hobby : hobbys) {
        System.out.println(hobby);
    }
}
```

运行结果：

```
***使用for循环输出数组中的元素
imooc
爱慕课
www.imooc.com

***使用foreach循环输出数组中的元素
imooc
爱慕课
www.imooc.com
```

四、方法

1、如何定义 Java 中的方法

所谓方法，就是用来解决一类问题的代码的有序组合，是一个功能模块。

一般情况下，定义一个方法的语法是：

```
访问修饰符  返回值类型  方法名(参数列表){
    方法体
}
```

其中：

1、 访问修饰符：方法允许被访问的权限范围， 可以是 public、protected、private 甚至可以省略 ，其中 public 表示该方法可以被其他任何代码调用，其他几种修饰符的使用在后面章节中会详细讲解滴

2、 返回值类型：方法返回值的类型，如果方法不返回任何值，则返回值类型指定为 void ；如果方法具有返回值，则需要指定返回值的类型，并且在方法体中使用 return 语句返回值

3、 方法名：定义的方法的名字，必须使用合法的标识符

4、 参数列表：传递给方法的参数列表，参数可以有多个，多个参数间以逗号隔开，每个参数由参数类型和参数名组成，以空格隔开

根据方法是否带参、是否带返回值，可将方法分为四类：

Ø 无参无返回值方法

Ø 无参带返回值方法

Ø 带参无返回值方法

Ø 带参带返回值方法

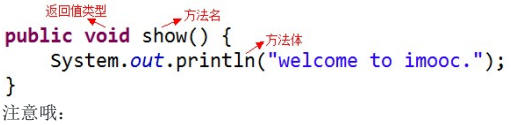
2、Java 中无参无返回值方法的使用

如果方法不包含参数，且没有返回值，我们称为无参无返回值的方法。

方法的使用分两步：

第一步，定义方法

例如：下面代码定义了一个方法名为 show，没有参数，且没有返回值的方法，执行的操作为输出“ welcome to imooc. ”




```
public void show() {
    System.out.println("welcome to imooc.");
}
```

注意哦：

1、 方法体放在一对大括号中，实现特定的操作

2、 方法名主要在调用这个方法时使用，需要注意命名的规范，一般采用第一个单词首字母小写，其它单词首字母大写的形式

第二步，调用方法

当需要调用方法执行某个操作时，可以先创建类的对象，然后通过 **对象名.方法名();** 来实现（关于类和对象的概念在后面章节中会详细讲解滴，先熟悉语法，表着急哦~~）

例如：在下面的代码中，我们创建了一个名为 hello 的对象，然后通过调用该对象的 show( ) 方法输出信息

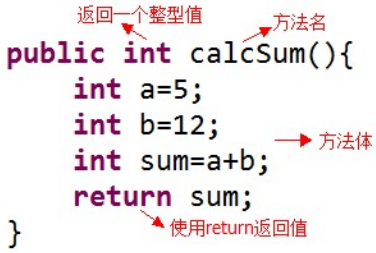
```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         // 创建对象，对象名为hello
4         HelloWorld hello = new HelloWorld();
5         // 通过对象名.方法名()的形式调用方法
6         hello.show();
7     }
8
9     // 定义方法
10    public void show() {
11        System.out.println("welcome to imooc.");
12    }
13 }
```

运行结果为： **welcome to imooc.**

3、Java 中无参带返回值方法的使用

如果方法不包含参数，但有返回值，我们称为无参带返回值的方法。

例如：下面的代码，定义了一个方法名为 calSum，无参数，但返回值为 int 类型的方法，执行的操作为计算两数之和，并返回结果




```
public int calcSum(){
    int a=5;
    int b=12;
    int sum=a+b;
    return sum;
}
```

在 calSum( ) 方法中，返回值类型为 int 类型，因此在方法体中必须使用 return 返回一个整数值。

调用带返回值的方法时需要注意，由于方法执行后会返回一个结果，因此在调用带返回值方法时一般都会接收其返回值并进行处理。如：

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         // 创建对象，对象名为hello
4         HelloWorld hello = new HelloWorld();
5         // 调用方法并接收方法的返回值，保存在变量sum中
6         int sum = hello.calcSum();
7         System.out.println("两数之和为：" + sum);
8     }
9
10    // 定义无参带返回值的方法
11    public int calcSum() {
12        int a = 5;
13        int b = 12;
14        int sum = a + b;
15        return sum;
16    }
17 }
```

运行结果为： **两数之和为：17**

不容忽视的“小陷阱”：

1、 如果方法的返回类型为 void，则方法中不能使用 return 返回值！

```
27    public void showInfo() {
28        return "Java";
29    }
```

2、 方法的返回值最多只能有一个，不能返回多个值

```
31      public int getInfo(){
32          int score1=94;
33          int score2=87;
34          return score1, score2;
35      }
```

3、方法返回值的类型必须兼容，例如，如果返回值类型为 int，则不能返回 String 型值

```
41      public int getInfo(){
42          String name="imooc";
43          return name;
44      }
```

4、Java 中带参无返回值方法的使用

有时方法的执行需要依赖于某些条件，换句话说，要想通过方法完成特定的功能，需要为其提供额外的信息才行。例如，现实生活中电饭锅可以实现“煮饭”的功能，但前提是我们必须提供食材，如果我们什么都不提供，那就真是的“巧妇难为无米之炊”了。我们可以通过在方法中加入参数列表接收外部传入的数据信息，参数可以是任意的基本类型数据或引用类型数据。

我们先来看一个带参数，但没有返回值的方法：

带有一个String类型的参数，参数名为name

```
public void show(String name) {
    System.out.println("欢迎您，" + name + "！");
}
```

上面的代码定义了一个 show 方法，带有一个参数 name，实现输出欢迎消息。

调用带参方法与调用无参方法的语法类似，但在调用时必须传入实际的参数值

对象名.方法名(实参 1，实参 2，……，实参 n)

例如：

创建对象，对象名为hello

调用带参方法时必须传入参数值

```
HelloWorld hello = new HelloWorld();
hello.show("爱慕课");
```

运行结果为：**欢迎您，爱慕课！**

很多时候，我们把定义方法时的参数称为**形参**，目的是用来定义方法需要传入的参数的个数和类型；把调用方法时的参数称为**实参**，是传递给方法真正被处理的值。

一定不可忽视的问题：

1、调用带参方法时，必须保证实参的数量、类型、顺序与形参一一对应

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         HelloWorld hello = new HelloWorld();
4         hello.show(25.6);
5     }
6     public String show(String name) {
7         return "欢迎您，" + name + "！";
8     }
9 }
```

实参与形参类型不一致

2、调用方法时，实参不需要指定数据类型，如

```
hello.show("爱慕课");
```

3、方法的参数可以是基本数据类型，如 int、double 等，也可以是引用数据类型，如 String、数组等

```
1 import java.util.Arrays;
2 public class HelloWorld {
3     public static void main(String[] args) {
4         HelloWorld hello = new HelloWorld();
5         int[] scores={84,91,74,62};
6         hello.print(scores);
7     }
8     public void print(int[] scores){
9         System.out.println(Arrays.toString(scores));
10    }
11 }
```

实参为整型数组　　参数为整型数组

使用Arrays类的toString()方法将数组转换为字符串并输出

4、当方法参数有多个时，多个参数间以逗号分隔

多个参数间以逗号分隔

```
public int calc(int num1,int num2){
    int num3=num1+num2;
    return num3;
}
```

5、Java 中带参带返回值方法的使用

如果方法既包含参数，又带有返回值，我们称为带参带返回值的方法。

例如：下面的代码，定义了一个 show 方法，带有一个参数 name，方法执行后返回一个 String 类型的结果

返回值为String类型　　带有一个String类型的参数

```
public String show(String name) {
    return "欢迎您，" + name + "！";
}
```

调用带参带返回值的方法：

创建对象，对象名为hello

调用带参带返回值的方法，将返回值保存在变量welcome中

```
HelloWorld hello = new HelloWorld();
String welcome=hello.show("爱慕课");
System.out.println(welcome);
```

运行结果为：**欢迎您，爱慕课！**

6、Java 中方法的重载

问： 什么是方法的重载呢？

答： 如果同一个类中包含了两个或两个以上方法名相同、方法参数的个数、顺序或类型不同的方法，则称为方法的重载，也可称该方法被重载了。如下所示 4 个方法名称都为 show，但方法的参数有所不同，因此都属于方法的重载：

```
无参方法
public void show() {
    System.out.println("welcome");
}
重载show方法，一个字符串参数
public void show(String name) {
    System.out.println("welcome:" + name);
}
重载show方法，两个参数
public void show(String name, int age) {
    System.out.println("welcome:" + name);
    System.out.println("age:" + age);
}
重载show方法，两个参数顺序不同
public void show(int age ,String name ) {
     System.out.println("welcome 2 :" + name);
     System.out.println("age 2 :" + age);
 }
```

问： 如何区分调用的是哪个重载方法呢？

答： 当调用被重载的方法时， Java 会根据参数的个数和类型来判断应该调用哪个重载方法，参数完全匹配的方法将被执行。如：

```
public static void main(String[] args) {
    HelloWorld hello = new HelloWorld();
    hello.show(); // 调用无参的show方法
    hello.show("tom"); // 调用带有一个字符串参数的show方法
    hello.show("jack", 20); // 调用带有字符串参数和整型参数的show方法
}
```

运行结果：

```
welcome
welcome：tom
welcome：jack
age：20
```

判断方法重载的依据：

1、 必须是在同一个类中

2、 方法名相同

3、 方法参数的个数、顺序或类型不同

4、 与方法的修饰符或返回值没有关系