

1.2 Java 语言基础

- 变量和数据类型
- 字符串和数组
- 运算符

目录

1

变量和数据类型

2

字符串和数组

3

运算符

Java 变量

- **变量**[变数]是程序中保存数据值的容器。
- 在 Java 中，有不同类型的变量，例如：
 - **String**: 存储字符串，例如 “Hello world”。
 - **int**: 存储整数，例如 123 或 -123。
 - **double**: 存储浮点数，即小数，例如 19.99 或 -19.99。
 - **boolean**: 存储具有两种状态的值：true 或 false.

Note



Java 中，每个变量只能存储一个类型的值。

变量的声明

- 要**声明**[宣言]（创建）一个变量，必须指定其**类型**和**名称**：

```
type name;
```

- 其中 **type** 是 Java 的一个类型（例如 **int** 或 **String**），而 **name** 是变量的名称（例如 **x** 或 **name**）。

Example ✓

以下代码创建了一个存储字符串的变量 **str**：

```
String str;
```

- 同一个变量只能被声明一次。

变量的赋值

- **赋值**_[代入]即将具体的数据值存入变量中。Java 中使用赋值符号 “=” 实现。

```
variable = value;
```

- 赋值符号会将其右边的值存入左边的变量。右边可以是一个现成的值，也可以是演算式、其他变量或方法。

Example ✓

以下代码创建了一个存储字符串的变量 str，并在其中存入值 “Hello world!”：

```
1 String str;  
2 str = "Hello world!";
```


变量的使用

- 当我们声明一个变量并为其赋值后，就可以随时通过变量名来使用它的值。

Example ✓

以下代码创建了一个存储字符串的变量 `str`，在其中存入值 “Hello world!”，并在屏幕上输出它的值：

```
1 String str;  
2 str = "Hello world!";  
3 System.out.println(str); // => Hello world!
```

Tips

你可以在声明变量的同时为其赋值（初始化）：

```
1 String str = "Hello world!";  
2 System.out.println(str); // => Hello world!
```

变量被赋值时，其原本存储的值会被覆盖。

```
1 int num = 1;  
2 num = 2;  
3 System.out.println(num); // => 2
```

构造名称的规则

- 在 Java 中，类、变量等的名称（**标识符**）可以由开发者自行决定，但须遵守以下基本规则：

1. 名称只能包含英文大小写字母（a – z, A – Z）、阿拉伯数字（0 – 9）、下划线（_）和美元符号（\$）。
2. 名称**不能**以**数字****开头**。
3. 名称不能和 Java 自带的**关键字**[キーワード]相同。

关键字一览：

 https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

- Java 对大小写敏感。“myVar” 和 “myvar” 是不同的名称。
- 在 Java 代码风格上，建议使用“**驼峰形命名**”。变量名应以小写字母，类名应以大写字母开头。变量名应以简洁明了的英文单词构成。

标识符

Example ✓

- `i`
- `string1`
- `studentNameList`
- `SomE$oDd_name` (合法, 但不推荐)

Example ✗

- `int` (与关键字重复)
- `1stVar` (以数字开头)
- `var#1` (包含非法字符)
- `two words` (包含空格)

数据类型

- Java 中的**数据类型**[データタイプ]分为两类：
 - **原始数据类型**[プリミティブ型・基本型]（Primitive Data Types），共 9 种。
存储一些较为基础、简单的数据，如数字。
 - **非原始数据类型**[参照型]，如 String、数组和自定义的类。
存储一些复杂数据，如字符串。

原始数据类型

- **原始数据类型**，或**基本类型**，用于存储一些简单的数据。
- 所有基本类型以小写字母开头，是 Java 中的关键字。

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

- 还有一种特殊的基本类型 `void`，我们暂时不做介绍。

整数类型

- `byte`, `short`, `char`, `int`, `long` 类型可用于存储**整数**[整数]。
- 不同类型的存储范围和所占内存空间有所不同。

类型	范围	所占空间
byte	-128 至 127	1 Byte
short	-32,768 至 32,767	2 Byte
char	0 至 65,535	2 Byte
int	-2,147,483,648 至 2,147,483,647	4 Byte
long	-9,223,372,036,854,775,808 至 9,223,372,036,854,775,807	8 Byte

小数类型

- `float` 和 `double` 类型可用于存储小数（**浮点数**[浮動小数点数]）。
- 不同类型的精度，存储范围和所占内存空间有所不同。


类型	最高精度	最大范围	所占空间
<code>float</code>	约小数点后7位	正负至约 $3e+38$	4 Byte
<code>double</code>	约小数点后15位	正负至约 $2e+308$	8 Byte

Note

`float` 类型的数值结尾须加上 “f”：

```
float a = 1.5f;
```

布尔类型

- 有时在编程中，需要用到一个二元的数据类型，例如：
 - 是 / 否
 - 开 / 关
 - 真 / 假
- 为此，Java 有一个 **boolean** 数据类型，该数据类型可以存储 **true** 或 **false**。这两个值也被称为**布尔值**[ブール値]。
- 在之后学习的控制语句（ § 1.3.1）中，我们需要计算一个布尔值以判断是否满足某个条件。

类型转换

- 如果我们想将原始数据类型的值赋给另一原始数据类型，需要进行**类型转换****[キャスト]**。
- Java 中的类型转换分为两种：
 - **扩大转换**：将较小的类型转换为较大的类型。
由于不会丢失信息，因此 Java 可以自动进行此种转换。
 - **缩小转换**：将较大的类型转换为较小的类型。
由于会丢失信息，因此需要**手动标明**转换。
- 类型的“大小”：较大的类型“包含”较小的类型。
 - `byte < short < int < long < float < double`
 - `char < int`
- 手动转换的方式：

```
1 int a = 100;  
2 char b = (char) a;
```



Coffee ☕ Break

char 类型与编码 (1)

Java 中的 **char** 类型不仅可以表达一个整数，同时也可以用来表达一个字符。实际上，char 就是 Character（字符）一词的缩写。

在计算机中，每一个数据都是以二进制数字形式存储及传递的。因此，文字无法直接被存储在内存中。我们需要为每一个文字确定一个编号，将这个编号作为实际的数据存储。char 中存储的便是这个编号的数值。

这个编号到文字的对应关系被称为**编码**（encoding）或**字符集**（charset）。不同语言可能用到不同的字符集，这也是文件出现乱码的原因之一。

Coffee ☕ Break

char 类型与编码 (2)

大部分的字符集中英文字母和基本标点符号的编码方式都是一样的，满足“**ASCII**”编码标准。如，编号 65 代表了字母“A”。

Try 

ASCII.md

Try 

Char.java

同时，为了解决不同编码之间的转换问题，也存在一些包好了所有语言文字的通用编码集。常见的通用编码集包括 UTF-8, UTF-16 等。

我们建议所有代码文件都以 **UTF-8** 编码存储。

Q & A

Question and answer

目录

1

变量和数据类型

2

字符串和数组

3

运算符

引用类型

- 除了刚刚介绍的原始类型以外的类型被称为**非原始数据类型**，或**引用类型**[参照型]。它和原始类型之间的主要区别包括：
 - 原始类型在 Java 无法由程序员定义。
 - 引用类型可用于调用**方法**[メソッド]以执行某些操作，而原始类型则不能。
 - 所有引用类型变量都可以存储一个特殊的值 **null**。
 - 为了区别于基本类型，引用类型的名称一般都以**大写字母**开头。
 - 原始类型的存储大小取决于数据类型，而引用类型的大小则取决于操作系统（在 32 位系统上为 4 Byte，在 64 位系统上为 8 Byte）。

字符串

- **字符串** [文字列] 是存储一段文本的数据类型，在 Java 中由 **String** 类型存储。
- 字符串的内容需要用双引号括起。
- 作为一个引用类型，**String** 自带许多方便的方法。比如，可以使用以下 **length** 方法得到字符串的长度：

```
1 String str = "abcdefghijklm";  
2 System.out.println(str.length()); // => 13
```

- 更多方法会在之后（ § 3.3.2）介绍。

打印变量

- 我们之前学过使用 `System.out.println` 方法可以打印一个字符串。
- 当我们想打印多个字符串时，可以用 “+” 把它们连在一起：

```
System.out.println("Hello" + ", " + "World"); // => Hello, World
```

- 也可以用 “+” 连接字符串和其他类型的数据。这些数据会被转换成字符串：

```
1 int id = 100;  
2 System.out.println("Hello, " + id + "!"); // => Hello, 100!
```

转义字符

- 某些特殊字符无法被直接写在代码中（例如：换行符），我们需要通过特别的方法表达它们。**转义序列**（escape sequence）通过书写多个字符的组合来表达这些特殊字符。Java 中的转义序列都以反斜杠“\”开头。因此，反斜杠也被称为“**转义字符**”。
- 下面列出一些常用的需要转义的字符（想一想，它们为什么需要被转义？）：

字符	书写方法	字符	书写方法
换行符	\n	backspace	\b
Tab	\t	反斜杠 \	\\
双引号 “	\"	单引号 ‘	\'

数组

- **数组**_[配列] (Array) 用于将多个值存储在单个变量中，从而避免为每个值声明单独的变量。

- 在 Java 中，要声明数组，使用方括号 “[]” 定义变量类型：

```
int[] arr;
```

- 可以使用大括号 “{}” 初始化数组 (*只能在声明时使用!*)：

```
String[] texts = {"hello", "world", "!"};
```

- 任何类型都有对应的数组类型，无论是否是基本类型。
- 同一个数组中的每个值都是相同类型。

数组的创建

- 除了直接使用大括号外，还可以通过 **new** 指令创建数组。你可以：

- 创建一个指定长度的数组：

```
texts = new String[3];
```

 - 注意：这种创建方法会使数组内所有元素自动被设定为“默认值”。
 - 数字的默认值为 0，布尔值为 false，而引用类型为 null。

- 或创建一个包含指定元素的数组：

```
texts = new String[] {"a", "b", "c"};
```

- 不要混用这两种语法！

Example ×

```
int[] arr = new int[2] {1, 2}; // error
```

数组的使用

- 数组中的每一个数据称作它的**元素**[要素]。
- 数组中的元素按顺序排列，每个元素的编号（位置）称作它的**索引**[要素]。在数组名称后用“**[]**”符号即可按索引访问某个元素：

```
1 int[] arr = new int[] {1, 2, 3};  
2 arr[1] = 100;  
3 System.out.println(arr[1]); // => 100
```

Note



数组索引以 **0** 开始：[0] 是第一个元素。[1] 是第二个元素，依此类推。这一特性在很多计算机语言里都有。

数组的使用

- 可以通过数组的 **length** 属性获得其大小（注意到和 String 的 **length** 方法有什么区别了吗？）：

```
1 int[] arr = new int[] {1, 2, 3};  
2 System.out.println(arr.length); // => 3
```

- 想一想，数组的最后一个元素的索引和数组的大小满足什么关系？

Try

01011
11010
01011

Array.java

多维数组

- 我们说过，任何类型都有对应的数组类型，包括数组。我们称一个数组的数组为 **2 维数组**。一个 2 维数组的数组为 3 维数组，以此类推。这些数组都是 **多维数组** [多次元配列]。多维数组可以用来表达图片，网络等结构复杂的数据。
- 多维数组的声明方法与普通数组类似，只需把类型（“[]” 前的部分）写成 “数组类型”：

```
int[][] array2D;
```

- 你同样可以使用 “{}” 进行初始化：

```
int[][] array2D = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

接次页 ➞



- 在使用指定大小的 `new` 方法创建数组时，可以指定多个维度的大小：

```
int[][] array2D = new int[3][3];
```

- 你也可以只指定第 1 个维度的大小（此时数组中的每一个数组均为 *null*）：

```
int[][] array2D = new int[3][];
```

- 使用多维数组时，需要使用多个 “`[]`” 运算符以获得数据。下例中，前面的 `[2]` 表示“第 2 个数组”，后面的 `[0]` 表示“该数组的第 0 个元素”。

```
System.out.println(array2D[3][0]);
```



Q & A

Question and answer

目录

1

变量和数据类型

2

字符串和数组

3

运算符

Java 运算符

- **运算符**[运算符]用于对已有数据执行运算，产生新的数据。
- 根据功能，Java 的运算符可以分为以下几组：
 - 算术运算符
 - 赋值运算符
 - 比较运算符
 - 逻辑运算符
 - 按位运算符

算术运算符

- **算术运算符**[算術演算子] 用于执行常见的数学运算。

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	$x \% y$
++	Increment	Increases the value of a variable by 1	$++x$
--	Decrement	Decreases the value of a variable by 1	$--x$

Tips

$++(--x)$ 也可以写作 $x++(--)$ 。你能想出它们的区别吗？

比较运算符

- **比较运算符**[比較演算子]用于比较两个值：

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Note  等于写作“==”，而“=”是赋值命令。

逻辑运算符

- 有时，我们需要判断的条件比较复杂，无法用一个比较符号计算出。比如“x 或者 y 比 3 大”“x 和 y 都是奇数”等等。此时，我们就需要用到逻辑运算符来构建布尔表达式。
- **逻辑运算符**[論理演算子]用于确定变量或值之间的逻辑：

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	<code>x < 5 && x < 10</code>
	Logical or	Returns true if one of the statements is true	<code>x < 5 x < 4</code>
!	Logical not	Reverse the result, returns false if the result is true	<code>!(x < 5 && x < 10)</code>

布尔表达式

- **布尔表达式**[ブール式]通常由一些逻辑运算组合而成，是返回一个布尔值的表达式。

Example ✓

要想表达“x 和 y 都比 3 大”，我们只需要写成：

```
x > 3 && y > 3
```

- 练习：以下代码输出什么？

```
1 int x = 1;  
2 int y = 2;  
3 System.out.println((x > y) || x > (y + 1) && x < 0);
```

- 实际书写并运行以下，结果和你的预期相符吗？

优先顺序

- 就像在数学中乘除法总是先于加减法进行, Java 中的运算符也都具有**优先顺序**[優先順位] (precedence)。优先顺序高的运算符会先被计算。

Tips

忘了优先顺序?
使用括号 “**()**” 总是没错的!

Operator Precedence

Operators	Precedence
postfix	<code>expr++ expr--</code>
unary	<code>++expr --expr +expr -expr ~ !</code>
multiplicative	<code>* / %</code>
additive	<code>+ -</code>
shift	<code><< >> >>></code>
relational	<code>< > <= >= instanceof</code>
equality	<code>== !=</code>
bitwise AND	<code>&</code>
bitwise exclusive OR	<code>^</code>
bitwise inclusive OR	<code> </code>
logical AND	<code>&&</code>
logical OR	<code> </code>
ternary	<code>? :</code>
assignment	<code>= += -= *= /= %= &= ^= = <<= >>= >>>=</code>

赋值运算符

- **赋值运算符**[代入演算子]用于为变量赋值。

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Try
Operator.java

Q & A

Question and answer

总结

Sum Up

1. 数据类型:

- ① 基本类型: 整数、小数、布尔值、字符。
- ② 非基本类型: 字符串、数组。
- ③ 类型转换。

2. 运算符:

- ① 算术运算符。
- ② 比较运算符。
- ③ 逻辑运算符。
- ④ 赋值运算符。
- ⑤ 运算符的优先顺序。

THANK YOU!