Java语言基础

孙聪

网络与信息安全学院

2020-09-21

课程内容

- Java概述
- 面向对象程序设计概念
- Java语言基础
- Java面向对象特性
- Java高级特征
- 容器类
- 常用预定义类
- 异常处理
- 输入输出
- 线程

提要

提要

标识符

- 标识符用来对变量、类和方法进行命名

标识符

- 标识符用来对变量、类和方法进行命名

命名规则

- 一个由字母、“_”（下划线）、“$”和数字组成的不限长度的序列
- 以字母、“_”或“$”起始，不能以数字起始
- 不能是Java关键字
- 不能是true、false或null
- 大小写敏感
- 不能包含空白

- 例：username，user_name，_sys_var1，$change

标识符

下列标识符哪些是合法的？

- $88
- #67
- num
- applet
- Applet
- 7#T
- b++
- --b
- class
- public

标识符

风格约定

- 不使用“_”、“$”作为标识符的第一个字符
- 类名、接口名：所有单词首字母大写，其余字母小写，如HelloWorld
- 变量名、方法名：首单词小写，其余单词的首字母大写，如someVariable
- 常量名：完全大写，单词间由“_”分隔，如CONST_POOL_NUM
- 方法名使用动词，类名和接口名使用名词，变量名尽量有含义
- 定义类名时不要使用$，否则该类的.class文件可能与编译器生成的内部类.class文件冲突

- 建议风格：课本附录1“Java代码风格”

注释

- // 注释一行
- /* 注释一行或多行 */
- /** 可用javadoc命令转化为HTML文件 */

注释

- // 注释一行
- /* 注释一行或多行 */
- /** 可用javadoc命令转化为HTML文件 */
- 注释不支持嵌套
  - 在开始于//的注释中，/*和*/没有特殊的意义
  - 在开始于/*或/**的注释中，//没有特殊的意义
- 注释不能出现在字符常量和字符串常量之中

关键字

指程序语言中预先定义的有特殊意义的标识符

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

- const和goto是C++的关键字，Java虽然将其保留为关键字，但Java尚未使用它们
- true，false和null虽然不是关键字，但也被Java保留，不能用作标识符

数据类型

Java数据类型分类

- 基本数据类型
  - 布尔型：boolean
  - 整型：byte，short，int，long，char
  - 浮点型：double，float
- 引用数据类型
  - 类
  - 接口
  - 数组
  - 枚举类型（J2SE 5.0）

基本数据类型-布尔型

布尔型：boolean

- 取值：
  - true
  - false（默认值）
- 例：boolean truth = true;
- Java中布尔值与整型值之间不能相互转换

基本数据类型-整型

整型：byte，short，int，long，char

- 有符号整型：byte，short，int，long
- 无符号整型（又称文本型）：char

类型	长度(bit)	取值范围
byte	8	$-2^7 \sim 2^7-1$
short	16	$-2^{15} \sim 2^{15}-1$
int	32	$-2^{31} \sim 2^{31}-1$
long	64	$-2^{63} \sim 2^{63}-1$
char	16	'\u0000'~'\uFFFF'

基本数据类型-整型

整型常量

- 整型常量具有三种进制表示
  - 十进制
  - 十六进制（以0x或0X起始）
  - 八进制（以0起始）
- 整型常量默认为int类型
  - byte或short类型的常量与int类型常量的形式相同
  - long类型的常量后应加L或l

	十进制	八进制	十六进制
int	24	030	0x18
long	24L	030L	0x18L

基本数据类型-文本型

文本型：char

- 16位的Unicode字符
- 例：char mychar='Q';

基本数据类型-文本型

Unicode：国际标准字符集

- 是一套字符编码系统，可支持各类文字的字符
- 支持的字符编码范围为0x0000至0x10FFFF

	ASCII	Unicode
A-Z	0x41-0x5A	0x0041-0x005A
a-z	0x61-0x7A	0x0061-0x007A
0-9	0x30-0x39	0x0030-0x0039
汉字，日文...	——	0x4E00-0x9FFF

- 一个字符的Unicode编码固定，但对Unicode编码的实现方式可能不同
- Unicode的实现方式：Unicode转换格式(UTF)，即Unicode值如何以bit表示
  - 是Java字符在文件中存储和在网络上传输时的表示形式
  - 目前通用的实现方式：UTF-32，UTF-16和UTF-8

基本数据类型-文本型

Java中的转义字符序列表示

转义字符序列	描述
\ddd	3位八进制数所表示的Unicode字符
\uxxxx	4位十六进制数所表示的Unicode字符
\'	单引号字符\u0027
\"	双引号字符\u0022
\\	反斜杠字符\u005C
\r	回车\u000D
\n	换行\u000A
\f	换页\u000C
\b	退格\u0008
\t	水平制表符（Tab键）\u0009

基本数据类型-文本型

String

- 不是基本数据类型，是类
- 例：String greeting="Good Morning!";
- String对象表示的字符串不能修改
- 字符串、字符的连接运算：+

```
String  salutation = "Dr. ";
String name = "Pete " + "Seymour" ;
String title = salutation + name ;
```

  - 则title值：Dr. Pete Seymour

基本数据类型-文本型

运行并观察以下程序的输出

```
public class CharConst{
  public static void main(String args[]){
    char c1='Q';
    char c2='\u0051';
    char c3=0x0051;
    System.out.println(c1 + ";" + c2 + ";" + c3 + ";" + '\121');
  }
}
```

基本数据类型-浮点型

浮点型：float和double

类型	长度(bit)	正数范围
float	32	1.4e-45 ～ 3.4028235e+38 （有效位约6~7位）
double	64	4.9e-324 ～ 1.7976931348623157e+308 （有效位约15位）

基本数据类型-浮点型

浮点型常量

- 浮点型常量默认为double类型
- float类型的常量后必须加F或f，例：2.718F
- double类型的常量后可以加D或d，例：2.718D，2.718
- 浮点型常量的表达可以与科学计数法叠加，例：6.02E23，6.02e23F

基本数据类型-浮点型

下列哪些是正确的浮点数常量

- 12.3
- 12.3e+2
- 23.4e-2
- -334.4
- 20
- 39F
- 40D

特殊的浮点型值

- NaN
  - NaN是无序的
  - 除了NaN之外的浮点数集合中的所有元素都是有序的
- 正/负无穷大
  - 例：Double.POSITIVE_INFINITY, Float.NEGATIVE_INFINITY
- 0.0和-0.0
- 例：PrimitiveConst.java

特殊的浮点型值

习题2-1：写出以下表达式的运算结果

- 0.0 == -0.0
- 0.0 > -0.0
- 1.0 < Double.NaN
- 1.0 > Double.NaN
- 1.0 == Double.NaN
- 1.0 != Double.NaN
- 0.0 / 0.0
- 1.0 / 0.0
- 1.0 / -0.0

提要

变量

定义

一种与特定类型关联的存储位置

分类

- 从语言机制上划分
  - 数据类型分基本类型和引用类型，
    相应地，变量类型也分基本类型和引用类型
- 根据作用域划分
  - 成员变量
  - 局部变量
  - 方法参数
  - catch语句块入口参数（异常处理参数）

变量

观察不同类型变量的声明和赋值方法

```
public class Assign {
  public static void main(String args[]) {
    int x, y;
    float z = 3.414f;
    double w = 3.1415;
    boolean truth = true;
    char c;
    String str;
    String str1 = "bye";
    x = 6;
    y = 1000;
    c = 'A';
    str = "Hi out there";
  }
}
```

引用变量的声明、引用与赋值

- 基本类型变量在声明时，直接分配数据空间，如

```
int a;
a = 12;
```

- 引用类型变量在声明时，不直接分配数据空间，而仅仅分配引用空间，必须经过实例化，才能开辟数据空间

```
Date today;
today = new Date(); //实例化
```

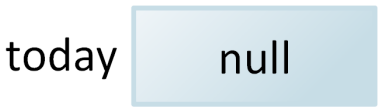
引用变量的声明、引用与赋值

- 基本类型变量在声明时，直接分配数据空间，如

```
int a;
a = 12;
```

- 引用类型变量在声明时，不直接分配数据空间，而仅仅分配引用空间，必须经过实例化，才能开辟数据空间

```
Date today;
today = new Date(); //实例化
```



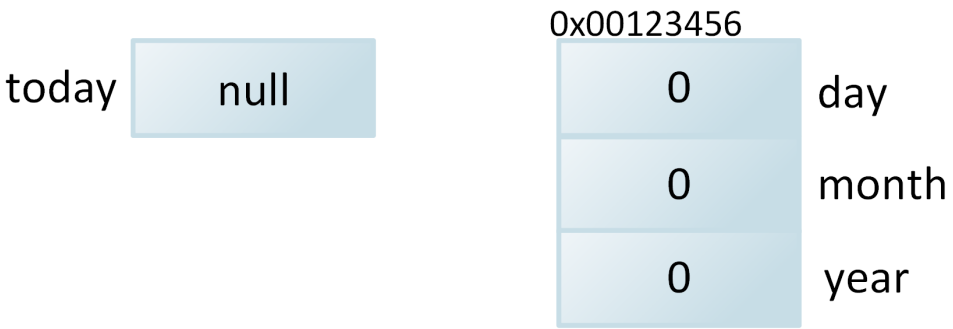
引用变量的声明、引用与赋值

- 基本类型变量在声明时，直接分配数据空间，如

```
int a;
a = 12;
```

- 引用类型变量在声明时，不直接分配数据空间，而仅仅分配引用空间，必须经过实例化，才能开辟数据空间

```
Date today;
today = new Date(); //实例化
```



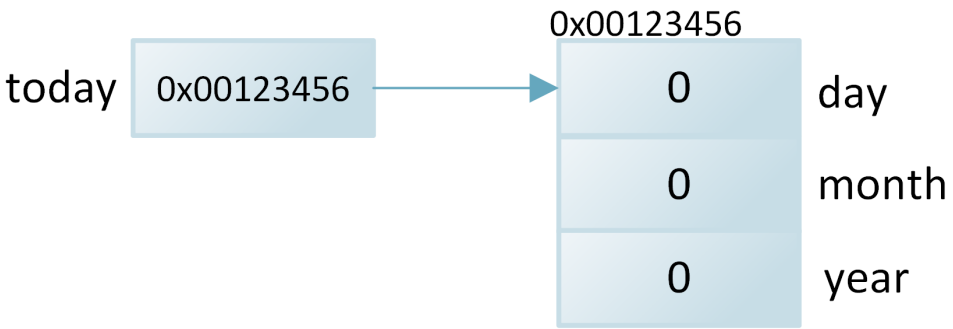
引用变量的声明、引用与赋值

- 基本类型变量在声明时，直接分配数据空间，如

```
int a;
a = 12;
```

- 引用类型变量在声明时，不直接分配数据空间，而仅仅分配引用空间，必须经过实例化，才能开辟数据空间

```
Date today;
today = new Date(); //实例化
```



引用变量的声明、引用与赋值

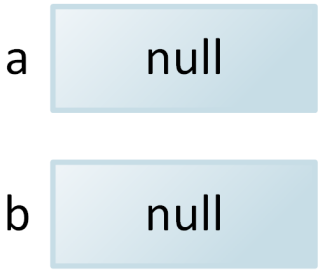
- 引用类型变量之间的赋值

```
Date a, b;
a = new Date();
b = a;
```

引用变量的声明、引用与赋值

- 引用类型变量之间的赋值

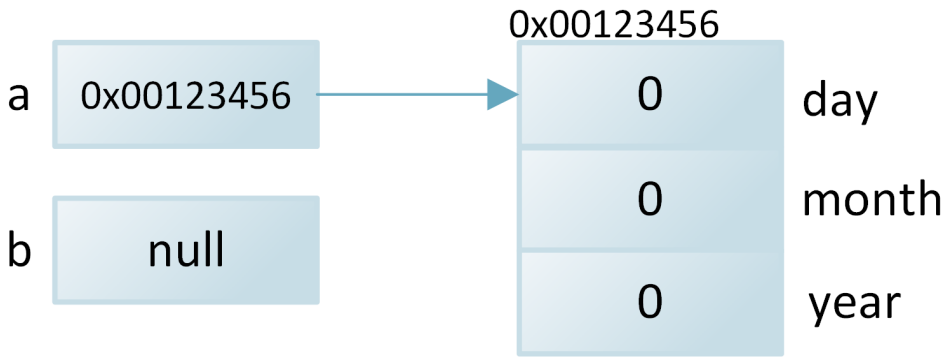
```
Date a, b;
a = new Date();
b = a;
```



引用变量的声明、引用与赋值

- 引用类型变量之间的赋值

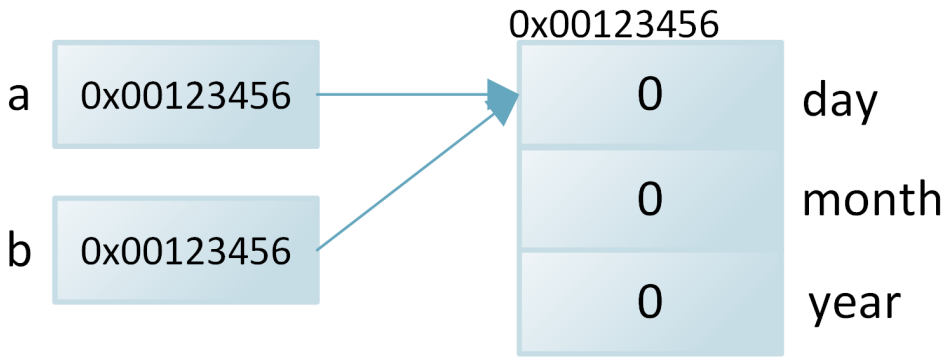
```
Date a, b;
a = new Date();
b = a;
```



引用变量的声明、引用与赋值

- 引用类型变量之间的赋值

```
Date a, b;
a = new Date();
b = a;
```



变量的作用域

- 局部变量：从声明变量的位置开始，到包含该变量的代码块结束为止
- 类成员变量：至少包括整个类
- 方法参数：所在方法的内部
- catch语句块入口参数：catch语句块

变量的默认初始值

- 对象的成员变量有默认初始值（系统自动初始化）

变量类型	初始值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	'\u0000'
boolean	false
所有引用类型	null

- 局部变量没有默认初始值
  - 必须在使用前手工赋初始值，若局部变量未初始化就使用，编译器报错

变量的默认初始值

以下程序是否有错，如果有错请改正

```
public class TestInit {
    int x;
    public static void main(String args[]) {
        TestInit init = new TestInit();
        int x = (int) (Math.random() * 100);
        int z;
        int y;
        if (x > 50) {
            y = 9;
        }
        z = x + y + init.x;
        System.out.println("x=" + x + ",y=" + y + ",z=" + z +
                           ",init.x=" + init.x);
    }
}
```

- 又例：VariablesAndLocalVarInit.java

操作符与表达式

关系

将操作符与操作数按照特定规则连接起来，就构成了通常意义的表达式

操作符与表达式

关系

将操作符与操作数按照特定规则连接起来，就构成了通常意义的表达式

Java操作符类别

操作符类别	操作符列表
算术操作符	+, -, *, /, %, ++, --
关系操作符	>, >=, <, <=, ==, !=
位操作符	>>, <<, >>>, &, |, ^(逐位异或), ~(逐位取反)
逻辑操作符	&, |, !, ^(异或), &&, ||
赋值操作符	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=
其他操作符	?:, [], ., (), (type), new, instanceof

算术操作符

- 一元算术操作符：+，-，++，--
  - 能够将byte/short/char类型的操作数自动提升为int型
  - 位操作符~也具有类似的类型转换效果

一元操作符的类型提升效果

```
public class UnaryConversion{
    public static void main(String[] args){
        byte b=2;
        char c='\u1234';
        //byte b2 = -b;                    //int值不能直接赋给byte类型变量b2
        //char c2 = +c;                    //int值不能直接赋给char类型变量c2
        System.out.println((-b) + ";" + (+c));
        int i= b;                                          //byte转换为int
        System.out.println(Integer.toHexString(i));
    }
}
```

算术操作符

二元算术运算中的自动类型转换

- 检查两个操作数的类型(type1, type2):
- if (double, *)或(*, double), then * → double
- else if (float, *)或(*, float), then * → float
- else if (long, *)或(*, long), then * → long
- else (*, *) → (int, int)

- 例: BinaryConversion.java

算术操作符

二元算术运算中的自动类型转换

- 检查两个操作数的类型(type1, type2):
- if (double, *)或(*, double), then * → double
- else if (float, *)或(*, float), then * → float
- else if (long, *)或(*, long), then * → long
- else (*, *) → (int, int)

- 例: BinaryConversion.java
- 注意
  - 即使两个操作数全是byte型或short型, 运算结果也是int型
  - 整型之间的/运算和%运算中除数为0时会产生异常
  - %的操作数可以为浮点数: 9.5%3=0.5
  - 只有当被除数为负数时, 余数才是负数: -7%3=-1, 20%-13=7
  - +运算可以连接字符串

算术操作符

习题2-4：假设int a=1和double d=1.0，且每个表达式均独立，以下表达式的结果分别是什么

- a = 46 / 9;
- a = 46 % 9 + 4 * 4 - 2;
- a = 45 + 43 % 5 * (23 * 3 % 2);
- d = 4 + d * d + 4;
- d += 1.5 * 3 + (++a);
- d -= 1.5 * 3 + a++;

关系操作符

- 计算操作数之间的关系并得到boolean结果
- 数值型的关系操作符遵循与二元算术操作符相同的自动类型转换规则

逻辑操作符

- &，| 称为不短路与、或
- &&,|| 称为短路与、或
- 短路：
  - 在逻辑表达式计算过程中，一旦部分运算结果能够确定整个表达式的值，则不再计算表达式的余下部分

逻辑操作符

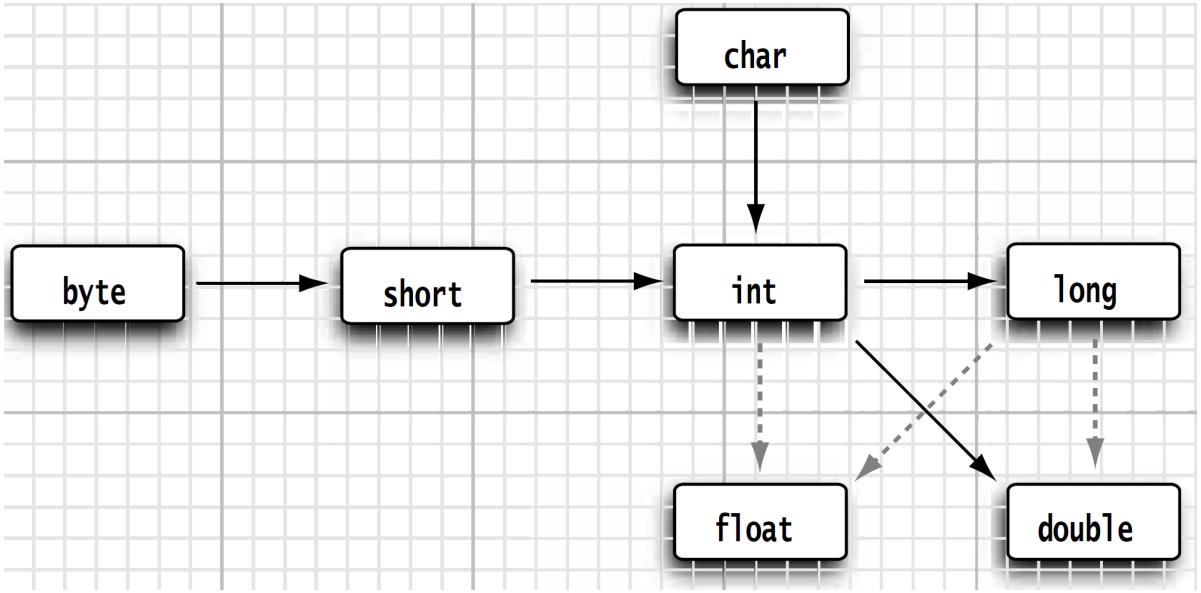
```
public class ShortCircuit {
    static boolean bLessThan1(int n) {
        System.out.println("(n<1)? " + (n < 1));
        return (n < 1);
    }
    static boolean bLessThan2(int n) {
        System.out.println("(n<2)? " + (n < 2));
        return (n < 2);
    }
    public static void main(String[] args) {
        System.out.println("Short Circuit:");
        System.out.println( bLessThan1(1) && bLessThan2(1) );
        System.out.println("Not Short Circuit:");
        System.out.println( bLessThan1(1) & bLessThan2(1) );
    }
}
```

位操作符

- 右移操作符
  - 带符号右移>>:
    1010... >> 2 → 111010...
  - 无符号右移>>>: 以0填充
    1010... >>> 2 → 001010...
- 当逻辑操作符与位操作符相同时，根据操作数判定是何种运算（&, |, ^）
- 例：ShiftRight.java

赋值操作符及赋值中的类型转换

- 如果赋值a=b中，a的类型为A，b的类型为B，则该赋值合法的前提是在下图中能够找到一条从B到A的路径



- 虚箭头表示有可能损失精度

```
int n = 123456789;
float f = n; // f is 1.23456792E8
```

- 又例：AssignConversion.java

强制类型转换

- 将较大范围类型的操作数值转换为较小范围类型的操作数值时，需要强制类型转换
- 一般形式：(type) expression
  - 例：(float)x/2

强制类型转换

- 将较大范围类型的操作数值转换为较小范围类型的操作数值时，需要强制类型转换
- 一般形式：(type) expression
  - 例：(float)x/2
- 对强制类型转换的限制：
  - 整型与浮点型可以相互转换
  - 基本类型和数组、对象等引用类型之间不能互相转换
  - 布尔型与整型之间不能互相转换
  - 布尔型与浮点型之间不能互相转换

强制类型转换

- 将较大范围类型的操作数值转换为较小范围类型的操作数值时，需要强制类型转换
- 一般形式：(type) expression
  - 例：(float)x/2
- 对强制类型转换的限制：
  - 整型与浮点型可以相互转换
  - 基本类型和数组、对象等引用类型之间不能互相转换
  - 布尔型与整型之间不能互相转换
  - 布尔型与浮点型之间不能互相转换
- 截尾和舍入
  - 由float/double类型向int/long类型强制转换时，执行截尾操作
  - 要得到舍入结果，需使用java.lang.Math的round()方法

强制类型转换

截尾与舍入示例

```
public class CastRoundingNum {
    public static void main(String[] args) {
        double above = 0.7;
        double below = 0.4;
        float fabove = 0.7f;
        float fbelow = 0.4f;
        System.out.println("(int)above: " + (int) above);
        System.out.println("(int)below: " + (int) below);
        System.out.println("(int)fabove: " + (int) fabove);
        System.out.println("(int)fbelow: " + (int) fbelow);
        System.out.println("Math.round(above): " + Math.round(above));
        System.out.println("Math.round(below): " + Math.round(below));
        System.out.println("Math.round(fabove): " + Math.round(fabove));
        System.out.println("Math.round(fbelow): " + Math.round(fbelow));
    }
}
```

操作符的优先级与结合方向

操作符	结合性
[] . ()（方法调用）	从左向右
++ --（后置增量）	从右向左
! ~ ++ --（前置增量） + -（一元操作）	从右向左
(type)（强制类型转换） new	从右向左
* / %	从左向右
+ -（二元操作）	从左向右
<< >> >>>	从左向右
< <= > >= instanceof	从左向右
== !=	从左向右
&	从左向右
^	从左向右
|	从左向右
&&	从左向右
||	从左向右
?:	从右向左
=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=	从右向左

习题2-10：x和y都是int类型，以下哪些表达式合法？

- x > y > 0
- x = y && y
- x /= y
- x or y
- x and y
- (x != 0) || (x = 0)

习题2-9：假设a，b，c初始值均为1，下面语句执行后a，b，c的值分别是多少？

- a = b += c = 5;

提要

程序流控制

语句类型	语句样式

单向if语句	if(布尔表达式) {
	  语句(块);
	}

双向if语句	if(布尔表达式) {
	  语句(块)1;
	} else {
	  语句(块)2;
	}

switch语句	switch(switch表达式) {
	  case 值1: 语句(块)1; break;
	  case 值2: 语句(块)2; break;
	  ...
	  case 值N: 语句(块)N; break;
	  default: 默认情况下执行的语句(块);
	}

程序流控制

语句类型	语句样式

while循环	while(循环继续条件){
	　　语句(块);
	}

do-while循环	do{ 　　语句(块); } while(循环继续条件);

for循环	for(初始操作; 循环继续条件; 每次迭代后操作){
	　　语句(块);
	}

增强的for循环	for(标识符: 可迭代类型的表达式){
	　　语句(块);
	}

break语句	break [label];
continue语句	continue [label];
label语句	label: 语句(块);
return语句	return [表达式];

循环语句

- Java SE 5.0后的for循环新形式：
  - 将一个集合作为一个整体放入for循环中，
    在for循环中可将集合中的元素进行逐个处理，例：

```
String[] names = {"Wang","Zhang","Li","Wu"};
for(String option: names) {
    System.out.println(option);
}
```

  - 又例：EnhancedFor.java

循环语句

- 在循环条件中不要使用浮点值来比较值是否相等，否则可能导致不精确的循环次数和结果

```
double item=1, sum=0;
while(item!=0){
    sum+=item;
    item-=0.2;
}
System.out.println(sum);
```

分支语句

- if-else
  - 布尔型与整型之间不能相互转换，循环与分支条件必须得到一个布尔型的值，不能像C那样接受整型：
    - C:

      ```
      int x=3; if(x){…}
      ```

    - Java:

      ```
      int x=3; if(x!=0){…}
      ```

  - 条件操作符?:是if-else的一种紧缩格式表达

分支语句

- if-else
  - 布尔型与整型之间不能相互转换，循环与分支条件必须得到一个布尔型的值，不能像C那样接受整型：
    - C:

      ```
      int x=3; if(x){…}
      ```

    - Java:

      ```
      int x=3; if(x!=0){…}
      ```

  - 条件操作符?:是if-else的一种紧缩格式表达
- switch
  - switch整型表达式的值必须是int兼容的类型，即byte，short，char或int
  - 不论执行哪个分支，程序都会继续执行直至遇到break语句或到达整个switch语句末尾才结束switch语句

分支语句

```
public class SwitchDemo2 {
    public static void main(String[] args) {
        int month = 2; int year = 2000; int numDays = 0;
        switch (month) {
        case 1: case 3: case 5: case 7: case 8: case 10: case 12:
            numDays = 31; break;
        case 4: case 6: case 9: case 11:
            numDays = 30; break;
        case 2:
            if (((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0))
                numDays = 29;
            else
                numDays = 28;
            break;
        }
        System.out.println("The number of Days = " + numDays);
    }
}
```

跳转语句

```
1 loop:  while(true){
2     for(...){
3         switch(...){
4             case -1:
5             case '\n':
6                 break loop;                    //跳出while到11行
7             ...
8         }
9     }
10 }
11 test:  for(...){
12     while(...){
13         if(...){
14             ...
15             continue test;                    //跳到11行
16         }
17     }
18 }
```

跳转语句

```
public class BreakAndContinueWithLabel{
    public static void main(String[] args){
        String searchMe = "Look for a substring in me";
        String substring = "sub";
        boolean found = false;
        int max = searchMe.length() - substring.length();
  test: for (int i = 0; i <= max; i++) {                    // i:子串查找开始位置
            int n = substring.length();
            int j = i;                                      // j:当前在主串中的查找位置
            int k = 0;                                      // k:当前在子串中的查找位置
            while (n-- != 0) {
                if (searchMe.charAt(j++) != substring.charAt(k++)) {
                    continue test;
                }
            }
            found = true;
            break test;                                             // = break;
        }
        System.out.println(found ? "Found!" : "Didn't find!");
    }
}
```

提要

数组声明

- 数组用于存储元素个数固定、元素类型相同的有序集
  - 长度在创建时确定，在创建后不变
  - 元素都是同一种类型

数组声明

- 数组用于存储元素个数固定、元素类型相同的有序集
  - 长度在创建时确定，在创建后不变
  - 元素都是同一种类型
- 定义数组的要素
  - 数组长度
  - 元素类型
  - 数组变量的名称
  - 数组维度（支持多维数组的语言需要提供）
  - 其中，数组长度在数组创建时指定，其余三要素在数组声明时指定

数组声明

- 声明包含三部分
  - 元素类型：基本类型或引用类型
  - 数组变量名称
  - 数组维度
- 一维数组声明的格式：
  - 元素类型[] 数组变量名称;
  - 元素类型 数组变量名称[];
  - 例：
    - `char s[]; char[] s;` //基本类型数组
    - `Point p[]; Point[] p;` //引用类型数组
    - `char[] s,m,n;` //三个变量都是字符数组变量

数组创建

- 数组类型是一种引用类型，因此
  - 数组声明不分配数组的数据空间，而是创建一个对该类型数组的引用
  - 数组的数据空间可以看作一个对象实例，需要用new关键字创建
  - 数组变量与数组对象不同，数组变量存储了对数组对象的引用
- 数组实例化：创建数组对象，并建立数组变量与数组对象之间的引用关系
  - 数组变量名称=new 元素类型[数组长度];
  - 例：
    - `s = new char[20];` //创建有20个字符的数组
    - `p = new Point[100];` //创建有100个Point引用的数组

数组初始化

- 在创建数组对象时，使用元素类型的默认值对每个元素进行初始化
- 进一步初始化
  - 直接对数组的各个元素赋值：

```
Point[ ] p = new Point[100];
p[0] = new Point();
p[1] = new Point();
...
```

  - 使用数组初始化器：

```
String names[] = {"Jack", "Wang", "Lee"};
int  a[] = {1, 2, 3};
Date d[] = {new Date(), new Date(), new Date()};
```

习题2-14：下列数组声明与创建语句是否合法？

- `long j=new long(20);`
- `Integer[] ilist=new Integer(1..20);`
- `float f[]={1.0, 2.1, 3.2};`
- `Point[][] r=new Point[2];`

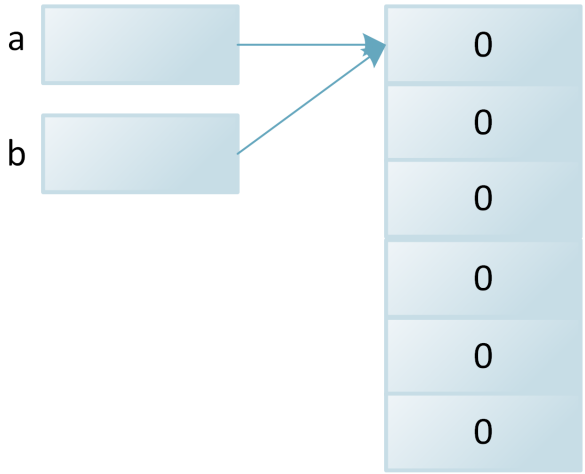
数组元素的访问

- 数组对象中的元素可以看作数组对象的成员变量
  - 这些成员没有名称
  - 必须通过“数组变量名称[索引值]”的方式访问
  - 例：p[1]表示访问由数组变量p指向的数组对象中索引为1的元素
  - 索引值必须为int类型： 对于short、byte、char类型的索引值，编译器会自动将其转换为int类型
  - 例：UnaryConversion2.java
- 所有数组都包含表示元素个数的成员变量length

数组复制

- 数组变量之间赋值是引用赋值，无法通过变量赋值复制数组数据

```
int a[] = new int [6];
int b[];
b = a ;
```

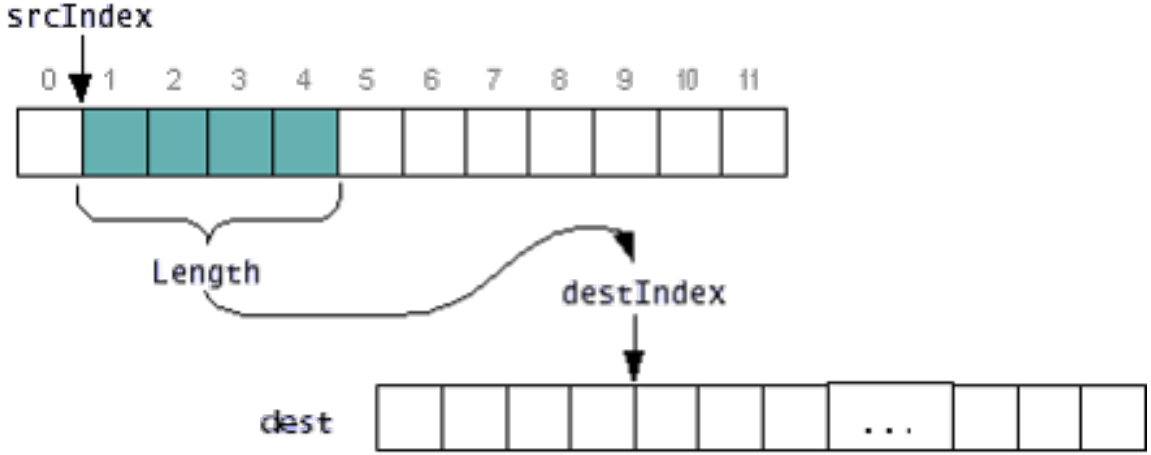


- 复制方法一：使用循环语句逐个复制

```
int[] target = new int[source.length];
for(int i = 0; i < source.length; i++)
    target[i] = source[i];
```

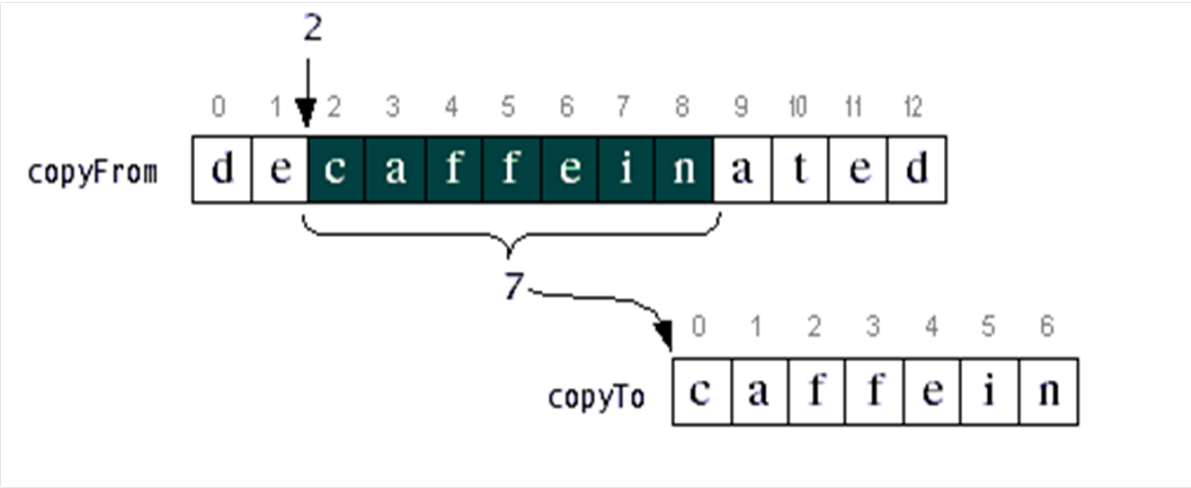
数组复制

- 复制方法二：java.lang.System的静态arraycopy()方法
  - System.arraycopy(Object source, int srcIndex,
    Object dest, int destIndex,
    int length);
  - 注意：dest数组空间必须预先分配好



数组复制

```
public class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                            'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```



多维数组

- 声明：使用连续的[]说明数组维度
  - 例：int a[][]; 或 int[][] a;
- 创建与初始化
  - 需指明至少一个嵌套层的数组长度
  - 必须从最高维指定数组长度
  - 例：

```
a = new int[4][4];                    //为每一维分配内存

a = new int[4][];          //为第一维分配内存，生成不规则数组
a[0] = new int[10];
a[1] = new int[5];
...

a = new int[][4];                     //不合法
```

多维数组

- 多维数组中，每一维的长度均可通过length变量获取

```
public class ArrayLength{
    public static void main(String[] args){
        int[][] a={{1,2,3,4,5},{1,2,3}};
        System.out.println(a.length + ";" + a[0].length + ";"
                                          + a[1].length);
    }
}
```

习题2-7：以下程序执行结束后，i和j的值分别是多少

```
int i = 5;
int j = (i++) + (i++);
```

习题2-11：实现程序，在字符串“Java is my favorate programming language”中查找字符a的出现次数