1.数据类型

1.1基本数据类型

基本类型	大小	最小值	最大值	包装器类型
boolean	-	-	-	Booolean
char	16 bits	Unicode 0	Unicode $2^{16}-1$	Character
byte	8 bits	-128	+127	Byte
short	16 bits	-2^{15}	$+2^{15}-1$	Short
int	32 bits	-2^{31}	$+2^{31}-1$	Integer
long	64 bits	-2^{63}	$+2^{63}-1$	Long
float	32 bits	IEEE754	IEEE754	Float
double	64 bits	IEEE754	IEEE754	Double
void	-	-	-	void

```
boolean flag = true;
char c = 'j';
byte b = 8;
short s = 16;
int size = 32;
long l = 100;
double d = 456.789;
```

1.2类型转换

数值范围较小的赋值给数值范围较大的，类型自动提升

```
int x = Integer.MAX_VALUE;
long l = x;
System.out.println(l);    //2147483647
```

数值范围较大的赋值给数值范围较小的，需要强制转换，可能会丢失精度

```
long l = Integer.MAX_VALUE +1;
int i  = (int) l;
System.out.println(i);      //-2147483648
```

2运算符

2.1赋值

赋值使用操作符“=”。它的意思是“取右边的值（即右值），把它复制给左边（即左值）”。右值可以是任何常数、变量或者表达式（只要它能生成一个值）。但左值必须是一个明确的、已命名的变量。

基本类型赋值

```
int a = 1;
int b = a;
System.out.println("a = " + a + " b = " + b);
//a = 1 b = 1

a = 2;
System.out.println("a = " + a + " b = " + b);
//a = 2 b = 1
```

对象赋值

```
 static class Tank {
        int level;
    }

    public static void main(String[] args) {
        Tank t1 = new Tank();
        Tank t2 = new Tank();
        t1.level = 11;
        t2.level = 22;
        System.out.println("t1.level = " + t1.level + " b = " + t2.level);
        //t1.level = 11 b = 22

        t2 = t1;
        t1.level = 33;
        System.out.println("t1.level = " + t1.level + " b = " + t2.level);
        //t1.level = 33 b = 33
    }
```

2.2算术操作符

Java的基本算术操作符与其他大多数程序设计语言是相同的。其中包括加好（+）、减号（-）、除号（/）、乘号（*）以及取模操作符（%，它从整数除法中产生余数）。整数除法会直接去掉结果的小数位，而不是四舍五入地结果

```
        int a = 5;
        int b = 2;
        int x = a + b;
        System.out.println("a + b = " + x);
        //a + b = 7

        x = a - b;
        System.out.println("a - b = " + x);
        //a - b = 3
```

```
        x = a * b;
        System.out.println("a * b = " + x);
        //a * b = 10

        x = a / b;
        System.out.println("a / b = " + x);
        //a / b = 2

        x = a % b;
        System.out.println("a % b = " + x);
        //a % b = 1


        //简化写法
        a = a + b;
        a += b;
```

2.3自动递增与递减

递增与递减运算符是两种快捷运算。其中，递减操作符是“--”，表示“减少一个单位”，递增操作符是“++”，表示“增加一个单位”。举个例子，假设a是一个int值，则表达式++a就等价于（a = a +1)。

对每种类型的运算符，都有两个版本可供选用；通常将其称为“前缀版”和“后缀版”。“前递增”表示++ 运算符位于变量或表达式的前面；而“后递增”表示++运算符位于变量或表达式的后面。类似地，“前递 减”意味着--运算符位于变量或表达式的前面；而“后递减”意味着--运算符位于变量或表达式的后面。对 于前递增和前递减（如++A 或--A），会先执行运算，再生成值。而对于后递增和后递减（如A++或A--），会先生成值，再执行运算。

```
        int i = 1;
        System.out.println("i : " + i);
        //i : 1
        System.out.println("++i : " + ++i);
        //++i : 2
        System.out.println("i++ : " + i++);
        i++ : 2
        System.out.println("i : " + i);
        i : 3
        System.out.println("--i : " + --i);
        --i : 2
        System.out.println("i-- : " + i--);
        i-- : 2
        System.out.println("i : " + i);
        i : 1
```

2.4关系运算符

关系运算符生成的是一个“布尔”（Boolean）结果。它们评价的是运算对象值之间的关系。若关系是真实的，关系表达式会生成true（真）；若关系不真实，则生成false（假）。关系运算符包括小于（<）、大于（>）、小于或等于（<=）、大于或等于（>=）、等于（==）以及不等于（!=）。等于和不等于适用于所有内建的数据类型，但其他比较不适用于boolean 类型。

2.5逻辑运算符

逻辑运算符AND（&&）、OR（||）以及NOT（!）能生成一个布尔值（true 或false）

```
        Random rand = new Random();
        int i = rand.nextInt() % 100;
        int j = rand.nextInt() % 100;
        System.out.println("i = " + i);
        System.out.println("j = " + j);
        System.out.println("i > j is " + (i > j));
        System.out.println("i < j is " + (i < j));
        System.out.println("i >= j is " + (i >= j));
        System.out.println("i <= j is " + (i <= j));
        System.out.println("i == j is " + (i == j));
        System.out.println("i != j is " + (i != j));

        System.out.println("(i < 10) && (j < 10) is "
                + ((i < 10) && (j < 10)));
        System.out.println("(i < 10) || (j < 10) is "
                + ((i < 10) || (j < 10)));
/**
i = -25
j = 85
i > j is false
i < j is true
i >= j is false
i <= j is true
i == j is false
i != j is true
(i < 10) && (j < 10) is false
(i < 10) || (j < 10) is true
**/
```

短路

```
        static boolean test1(int val) {
        System.out.println("test1(" + val + ")");
        System.out.println("result: " + (val < 1));
        return val < 1;
    }
    static boolean test2(int val) {
        System.out.println("test2(" + val + ")");
        System.out.println("result: " + (val < 2));
```

```
        return val < 2;
    }
    static boolean test3(int val) {
        System.out.println("test3(" + val + ")");
        System.out.println("result: " + (val < 3));
        return val < 3;
    }
    public static void main(String[] args) {
        if(test1(0) && test2(2) && test3(2))
            System.out.println("expression is true");
        else
            System.out.println("expression is false");
    }
/**
test1(0)
result: true
test2(2)
result: false
expression is false
**/
```

2.6按位运算符

按位运算符允许我们操作一个整数主数据类型中的单个“比特”，即二进制位。按位运算符会对两个自变量中对应的位执行布尔代数，并最终生成一个结果。

a	b	a|b	a&b	~a	a^b
0	0	0	0	1	0
0	1	1	0	1	1
1	0	1	0	0	1
1	1	1	1	0	0

2.7位移运算

移位运算符面向的运算对象也是二进制的“位”。可单独用它们处理整数类型（主类型的一种）。左移位运 算符（<<）能将运算符左边的运算对象向左移动运算符右侧指定的位数（在低位补0）。“有符号”右移位 运算符（>>）则将运算符左边的运算对象向右移动运算符右侧指定的位数。“有符号”右移位运算符使用了 “符号扩展”：若值为正，则在高位插入0；若值为负，则在高位插入1。Java 也添加了一种“无符号”右 移位运算符（>>>），它使用了“零扩展”：无论正负，都在高位插入0。

~x + x +1 = 0

```
    public static void main(String[] args) {
        Random rand = new Random();
```

```
        int i = rand.nextInt();
        int j = rand.nextInt();
        pBinInt("-1", -1);
        pBinInt("+1", +1);
        int maxpos = 2147483647;
        pBinInt("maxpos", maxpos);
        int maxneg = -2147483648;
        pBinInt("maxneg", maxneg);
        pBinInt("i", i);
        pBinInt("~i", ~i);
        pBinInt("-i", -i);
        pBinInt("j", j);
        pBinInt("i & j", i & j);
        pBinInt("i | j", i | j);
        pBinInt("i ^ j", i ^ j);
        pBinInt("i << 5", i << 5);
        pBinInt("i >> 5", i >> 5);
        pBinInt("(~i) >> 5", (~i) >> 5);
        pBinInt("i >>> 5", i >>> 5);
        pBinInt("(~i) >>> 5", (~i) >>> 5);
    }

    static void pBinInt(String s, int i) {
        System.out.println(
                s + ", int: " + i + ", binary: ");
        System.out.print(" ");
        for(int j = 31; j >=0; j--)
            if(((1 << j) & i) != 0)
                System.out.print("1");
            else
                System.out.print("0");
        System.out.println();
    }
    /**
    -1, int: -1, binary:
 11111111111111111111111111111111
+1, int: 1, binary:
 00000000000000000000000000000001
maxpos, int: 2147483647, binary:
 01111111111111111111111111111111
maxneg, int: -2147483648, binary:
 10000000000000000000000000000000
i, int: 1995942729, binary:
 01110110111101111010101101001001
~i, int: -1995942730, binary:
 10001001000010000101010010110110
-i, int: -1995942729, binary:
 10001001000010000101010010110111
j, int: -969616149, binary:
 11000110001101001101010011101011
i & j, int: 1177845833, binary:
 01000110001101001000000001001001
i | j, int: -151519253, binary:
```

```
 11110110111101111111111111101011
i ^ j, int: -1329365086, binary:
 10110000110000110111111110100010
i << 5, int: -554342112, binary:
 11011110111101010110100100100000
i >> 5, int: 62373210, binary:
 00000011101101111011110101011010
(~i) >> 5, int: -62373211, binary:
 11111100010010000100001010100101
i >>> 5, int: 62373210, binary:
 00000011101101111011110101011010
(~i) >>> 5, int: 71844517, binary:
 00000100010010000100001010100101
 **/
```

2.8三元if- else 运算符

```
布尔表达式 ? 值0:值1
```

若“布尔表达式”的结果为true，就计算“值0”，而且它的结果成为最终由运算符产生的值。但若“布尔 表达式”的结果为false，计算的就是“值1”，而且它的结果成为最终由运算符产生的值。

```
    public static void main(String[] args) {
        int i = 6;
        int x = i < 10 ? i * 100 : i * 10;
        System.out.println("x = " + x);
    }
    //x = 600
```

2.9运算符优先级

运算符	结合性
[ ] . ( ) (方法调用)	从左向右
! ~ ++ -- +(一元运算) -(一元运算)	从右向左
* / %	从左向右
+ -	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

3.控制流

3.1 if-else

if-else 语句或许是控制程序流程最基本的形式。其中的else 是可选的，所以可按下述两种形式来使用if：

```
if(布尔表达式)
        语句

或者

if(布尔表达式)
        语句
else
        语句
```

作为if-else 的一个例子，下面这个test()方法可告诉我们猜测的一个数字位于目标数字之上、之下还是相 等：

```
        static int test(int testval) {
        int result = 0;
        int target = 10;
        if (testval > target) {
            result = -1;
        } else if (testval < target) {
            result = +1;
        } else {
            result = 0; // match
        }
        return result;
    }
```

3.2while循环

```
while(布尔表达式)
        语句
```

在循环刚开始时，会计算一次“布尔表达式”的值。而对于后来每一次额外的循环，都会在开始前重新计算 一次。 下面这个简单的例子可产生随机数，直到符合特定的条件为止：

```
double r = 0;
while(r < 0.99d) {
        r = Math.random();
        System.out.println(r);
}
```

3.3do-while循环

```
do
        语句
while(布尔表达式)
```

while 和do-while 唯一的区别就是do-while 肯定会至少执行一次；也就是说，至少会将其中的语句“过一 遍”——即便表达式第一次便计算为false。而在while 循环结构中，若条件第一次就为false，那么其中的 语句根本不会执行。在实际应用中，while 比do-while 更常用一些。

3.4 for循环

for 循环在第一次反复之前要进行初始化。随后，它会进行条件测试，而且在每一次反复的时候，进行某种 形式的“步进”（Stepping）。for 循环的形式如下：

```
for(初始表达式; 布尔表达式; 步进)
        语句
```

无论初始表达式，布尔表达式，还是步进，都可以置空。每次反复前，都要测试一下布尔表达式。若获得的 结果是false，就会继续执行紧跟在for 语句后面的那行代码。在每次循环的末尾，会计算一次步进。 for 循环通常用于执行“计数”任务：

```
    public static void main(String[] args) {
        for (char c = 0; c < 128; c++)
            if (c != 26) {
                System.out.println("value: " + (int) c +
                        " character: " + c);
            }

    }
```

3.5中断和继续

在任何循环语句的主体部分，亦可用break 和continue 控制循环的流程。其中，break 用于强行退出循环，不执行循环中剩余的语句。而continue 则停止执行当前的反复，然后退回循环起始和，开始新的反复。 下面这个程序向大家展示了break 和continue 在for 和while 循环中的例子：

```
    public static void main(String[] args) {
        for(int i = 0; i < 100; i++) {
            if(i == 74) break; // Out of for loop
            if(i % 9 != 0) continue; // Next iteration
            System.out.println(i);
        }
        int i = 0;
        // An "infinite loop":
        while(true) {
            i++;
            int j = i * 27;
            if(j == 1269) break; // Out of loop
            if(i % 10 != 0) continue; // Top of loop
            System.out.println(i);
        }
    }
```

3.6switch

Switch有时也被划分为一种“选择语句”。根据一个整数表达式的值，switch 语句可从一系列 代码选出一段执行。它的格式如下：

```
switch(整数选择因子) {
        case 整数值1 : 语句; break;
        case 整数值2 : 语句; break;
        case 整数值3 : 语句; break;
        case 整数值4 : 语句; break;
        case 整数值5 : 语句; break;
        //..
        default:语句;
}
```

例如：

```
int i = 2;
switch(i){
case 0:
        System.out.println("0");
case 1:
        System.out.println("1");
case 2:
        System.out.println("2");
default:
        System.out.println("default");
}
/**
2
default
**/
```

4.数组

Java 中定义数组的语法有两种：

```
type arrayName[];
type[] arrayName;
```

type 为Java中的任意数据类型，包括基本类型和组合类型，arrayName为数组名，必须是一个合法的标识符，[ ] 指明该变量是一个数组类型变量。例如：

```
int demoArray[];
int[] demoArray;
```

这两种形式没有区别，使用效果完全一样，读者可根据自己的编程习惯选择。

与C、C++不同，Java在定义数组时并不为数组元素分配内存，因此[ ]中无需指定数组元素的个数，即数组长度。而且对于如上定义的一个数组是不能访问它的任何元素的，我们必须要为它分配内存空间，这时要用到运算符new，其格式如下：

```
arrayName=new type[arraySize];
```

其中，arraySize 为数组的长度，type 为数组的类型。如：

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

为一个整型数组分配3个int 型整数所占据的内存空间。

通常，你可以在定义的同时分配空间，语法为：

```
 type arrayName[] = new type[arraySize];
```

例如：

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

4.1数组的初始化

你可以在声明数组的同时进行初始化（静态初始化），也可以在声明以后进行初始化（动态初始化）。例如：

```
// 静态初始化
// 静态初始化的同时就为数组元素分配空间并赋值
int intArray[] = {1,2,3,4};
String stringArray[] = {"唯品会", "https://www.vip.com"};

// 动态初始化
float floatArray[] = new float[3];
floatArray[0] = 1.0f;
floatArray[1] = 132.63f;
floatArray[2] = 100F;
```

4.2数组引用

可以通过下标来引用数组：

```
 arrayName[index];
```

每个数组都有一个length属性来指明它的长度，例如 intArray.length 指明数组 intArray 的长度。

```
public static void main(String[] args) {
        int intArray[] = {1, 2, 3, 4, 5};
        long sum = 0;
        int len = intArray.length;

        // 计算数组元素的和
```

```
        for (int i = 0; i < len; i++) {
            sum += intArray[i];
        }

        System.out.println("所有数组元素的和为：" + sum);
    }
    /**
    所有数组元素的和为：15
    **/
```

4.3数组的遍历

实际开发中，经常需要遍历数组以获取数组中的每一个元素。最容易想到的方法是for循环。

不过，Java提供了"增强版"的for循环，专门用来遍历数组

```
    public static void main(String[] args) {
        int intArray[] = {1, 2, 3, 4, 5};
        long sum = 0;
        int len = intArray.length;

        // 计算数组元素的和
        for (int x : intArray) {
            sum += x;
        }

        System.out.println("所有数组元素的和为：" + sum);
    }
    /**
    所有数组元素的和为：15
    **/
```

这种增强版的for循环也被称为"foreach循环"，它是普通for循环语句的特殊简化版。所有的foreach循环都可以被改写成for循环。

但是，如果你希望使用数组的索引，那么增强版的 for 循环无法做到。

4.4二维数组

二维数组的声明、初始化和引用与一维数组相似：

```
int intArray[][] = { {1,2}, {2,3}, {4,5} };

int a[][] = new int[2][3];
a[0][0] = 12;
a[0][1] = 34;
// ......
a[1][2] = 93;
```

Java语言中，由于把二维数组看作是数组的数组，数组空间不是连续分配的，所以不要求二维数组每一维的大小相同。例如：

```
int intArray[ ][ ] = { {1,2}, {2,3}, {3,4,5} };

int a[ ][ ] = new int[2][ ];
a[0] = new int[3];
a[1] = new int[5];
```

通过二维数组计算两个矩阵的乘积。

```
    public static void main(String[] args) {
        // 第一个矩阵（动态初始化一个二维数组）
        int a[][] = new int[2][3];
        // 第二个矩阵（静态初始化一个二维数组）
        int b[][] = {{1, 5, 2, 8}, {5, 9, 10, -3}, {2, 7, -5, -18}};
        // 结果矩阵
        int c[][] = new int[2][4];

        // 初始化第一个矩阵
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 3; j++)
                a[i][j] = (i + 1) * (j + 2);

        // 计算矩阵乘积
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 4; j++) {
                c[i][j] = 0;
                for (int k = 0; k < 3; k++)
                    c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
```