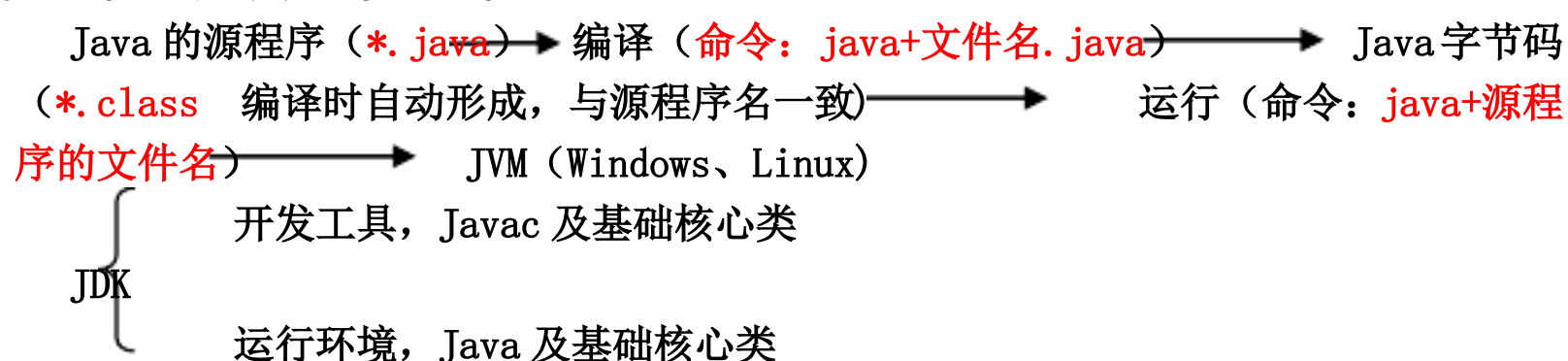


JAVA 知识点总结

1. JAVA 的特点：

- ① 简单易用、完全面向对象；
- ② 与平台无关性、可扩展性强；
- ③ 可移植性高、支持分布式编程；
- ④ 健壮、安全可靠并性能优异；
- ⑤ 支持多线程开发技术；
- ⑥ 支持动态开发。

2. JVM：Java 虚拟机（JVM 是 Java 实现跨平台的基础）。



3. 编写第一个 Java 程序：

Java 源文件扩展名为：“.java”

一个源文件中最好只有一个 java 类，但是可以包含多个类

public 修饰的类，文件名与类名必须一致(包括大小写)

被运行的类中需要有一个方法：

```
public static void main(String[ ] args) {}
```

一个源文件中最多有一个 public 修饰的类

例如：public class Test{

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

```
        System.out.println(“这个编写的第一个 java 程序!!!”);
```

```
    }
```

```
}
```

在运行程序之前先配置环境变量：

path 变量值为：JDK 安装目录下\bin;

classpath 变量值为：JDK 安装目录下\lib; 或 .;JDK 安装目录下\lib\tools.jar

在 dos 命令窗口中输入以下命令进行运行：

编译命令： javac Test.java

运行命令： java Test

生成文档命令： javadoc Test.java

4. Java 编程规范

A、命名规定

包：包名应该是小写的名词。

如：package shipping.objects

类：类名应该是名词，大小写混合，每个单词的首字母大写。

如：class AccountBook

接口：接口名的大小写应该与类名一样。

如：interface Account

方法：方法名应该动词，大小写混合，首字母小写。每个方法名中，以大写字母区分单词。限制使用下划线。

如：balanceAccount()

变量：所有的变量应该是大小写混合，首字母小写。由大写字母区分单词。限制下划线的使用。

如：currentCustomer

常数：基本类型常数应该是全部大写的由下划线区分的单词。

如：HEAD_COUNT、MAXIMUM_SIZE、MIN_VALUE

B、程序主体结构规定

控制结构：所有语句，即使是一条语句，如果是某控制结构的一部分，都要使用大括号（{ }）括起来。

空格：在任意行上只放置一条语句，并且使用 2 或 4 个空格缩进使代码更易读。

Java 代码的位置：所有代码都存在于一个类里

例如：修饰符 class 类名

```
{  
    程序代码  
}
```

注意：

Java 是严格区分大小写的；

功能执行语句以(;)结束，这个分号必须是英文输入法中的(;)；

连续的字符串不能换行，可使用(+)连接。

5、Java 代码的注释

作用：

1)、解释程序中某些部分的作用和功能，提高程序的可读性。

2)、可以使用注释暂时屏蔽某些语句，在程序调试时使用。

注释的形式：

1)、单行注释

int c = 10; // 定义一个整型

2)、多行注释

```
/*  
int c = 10; // 定义一个整型  
int x = 5;  
*/
```

注意：/*...*/中可以嵌套”//”注释，但不能嵌套”/*...*/”。

3)、文档注释

“/***/”。

6、Java 中的标识符

1)、变量，方法，类和对象的名称都是标识符，程序员需要标识和使用的东西都需要标识符。

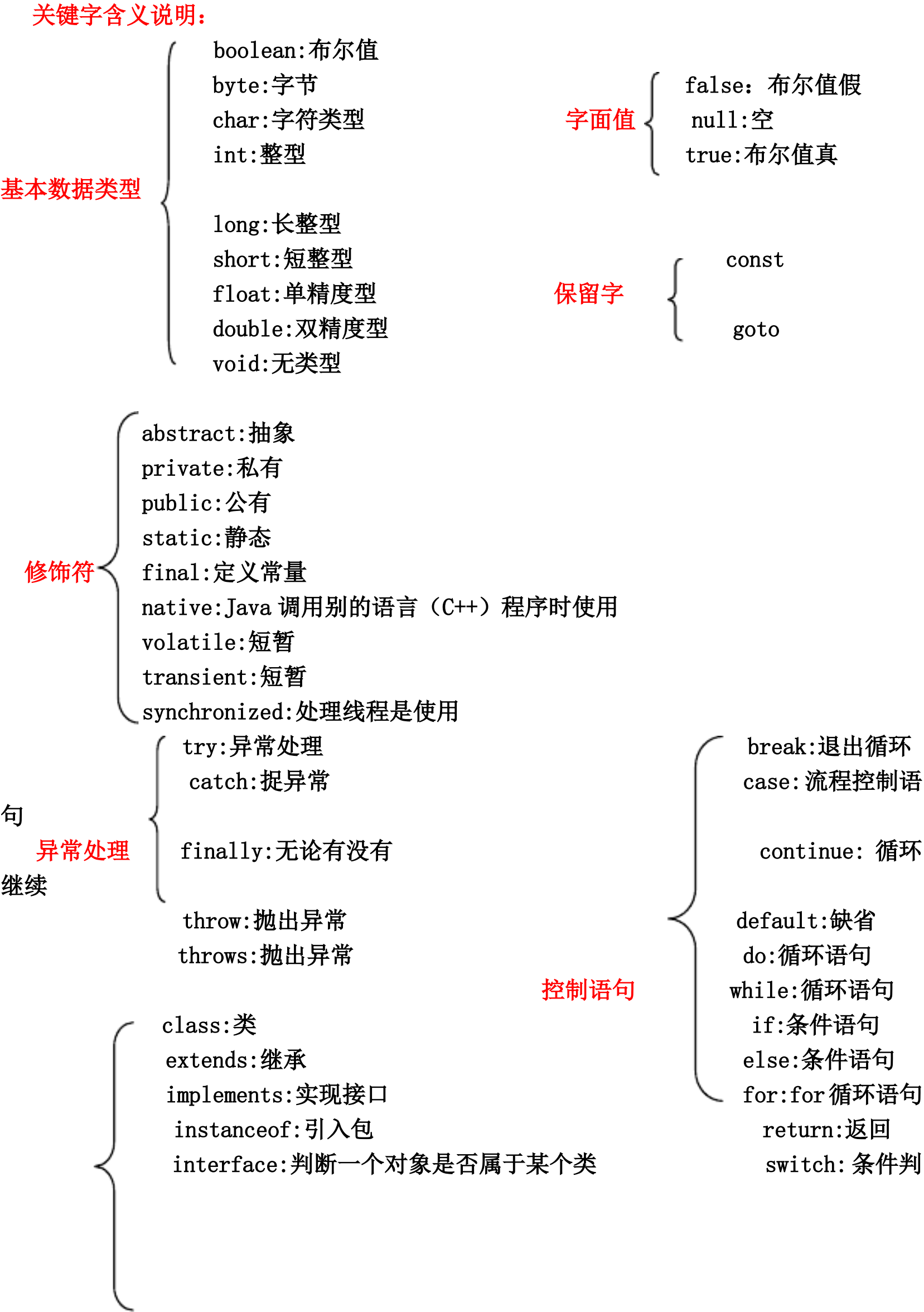
2)、在 Java 语言里标识符以字母或下划线、\$符号开头，后面字符可以是字母、数字、

下划线、\$符号。（其他符号都不能用来命名标识符）

- 3)、标识符对字母大小写非常敏感，必须区分大小写，但是没有长度限制。
 - 4)、关键字不能用作标识符命名。
 - 5)、标识符中不能包含空格。
 - 6)、标识符是由 Unicode 字符组成的，因此可以使用汉字作为标识符(不推荐，尽量不用)；
- 7、关键字

Java 中一些赋以特定的含义、并用做专门用途的单词称为关键字，也可叫保留字。关键字不能作为普通的标识符使用。

所有 Java 关键字都是小写的，IF、THIS、NEW 等都不是 Java 关键字；
goto 和 const 虽然从未使用，但也作被为 Java 保留关键字；

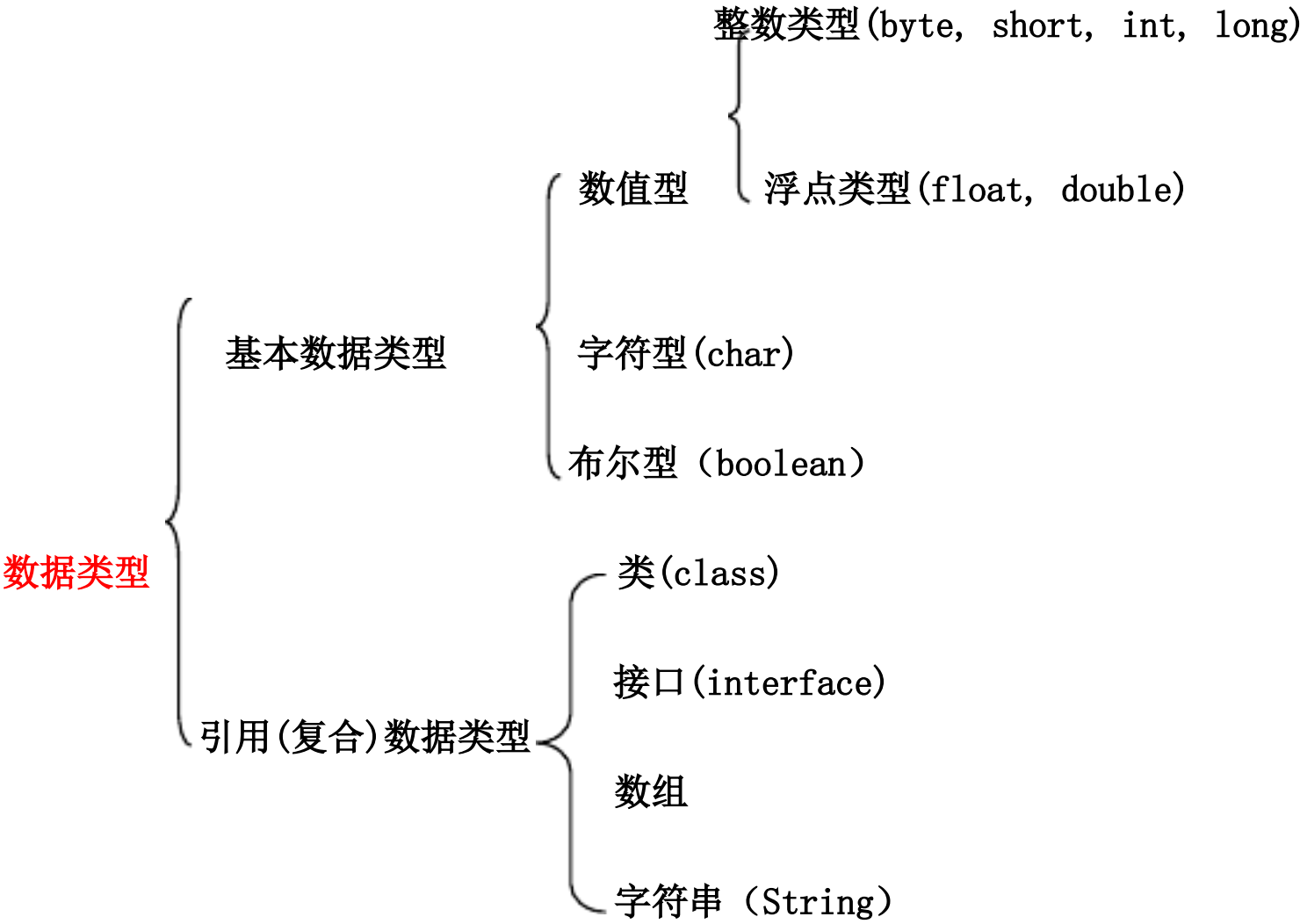


断

面向对象

- new: 创建新对象
- package: 包
- super: 超类
- this: 自己

8、java 基本数据类型



基本数据类型包括：整型、浮点型、字符型、逻辑型（布尔型）。

数据类型	名称	位长	默认值	取值范围
布尔型	boolean	1	false	true, false
字节型	byte	8	0	-128~127
字符型	char	16	‘\u0000’ 0’	‘\u0000’ ~ ‘\uffff’
短整型	short	16	0	-32768~32767
整型	int	32	0	-2147483648~2147483647
长整型	long	64	0	-9223372036854775808~9223372036854775807
浮点型	float	32	0.0	±1.4E-45~±3.4028235E+38
双精度型	double	64	0.0	±4.9E-324~±1.7976931348623157E+308

注意：

整数类型数据用于描述一个一定大小范围内的整数。

浮点类型数据用于描述一个范围很大的实数；

浮点类型数据有一定的精度限制。

字符类型为 char，它用于表示一个字符,使用单引号’ 在 Java 中 char 类型为 16 字节，采用 Unicode 表示。

逻辑类型为 boolean，它用于表示真和假； boolean 类型只有两个值真（true），假

(false);

boolean 类型有自己的运算，不能参与其他数据类型之间的运算。

9、常量

常量就是程序里持续不变的值，是不能改变的数据。

声明常量的格式如下：

final 类型 常量名[, 常量名]=值;

Java 中的常量包括整型常量、浮点型常量、布尔常量、字符常量等。

整型常量：

十进制：不能以 0 开头，多个 0~9 之间的数字

十六进制：以 0x 或 0X 开头 0x8a 0X56d

八进制：必须以 0 开头 034 0376

长整型：必须以 L 结尾 87L 345L

浮点数常量：

float 型：2e3f 0.6f

double 型：4.1d 1.23d

布尔常量： true 和 false

字符常量： 'a' '5'

字符串常量： "hello" "8698" "\nmain" 转义字符\n 表示换行

null 常量： null，表示对象的引用为空。

10、变量

在 java 语言中存储一个数据信息，必须将它保存到一个变量中。变量在使用前必须有定义，即有确定的类型和名称。

声明变量的语法：

类型 变量名[, 变量名][=初值];

例如：int i; char c; float a, b, c;

变量的声明有三种形式：

1、声明变量。

例如：int i;

2、变量赋值。在变量赋值之前要先声明变量。

例如：int i;

i=5;

3、变量的初始化。

例如：int i = 8;

11、基本数据类型之间的兼容性

基本数据类型之间的转换：“小”的数据类型可以直接赋给“大”的数据类型。“大”的不能赋值给“小”的数据类型（会出现编译错误）。

数据类型大小关系如下：

整数类： long > int > short > byte

浮点型： double > float

整型数据类型可以赋给浮点数据类型比如：

float ← short float ← int float ← long double ← long

注意：

char 可以赋给 long 和 int ，但是不能赋给 short 和 byte（编译错误）。

char 可以赋给 float 和 double。

当整数型常量被声明为 long 类型时，只能赋值给 long 型变量。
当整数型常量在 0~65535 之间时，可以被赋值给 char 型变量。
char 型常量可以被赋值给整数类变量，只要整数变量的类型可以容纳 char 型文字常量所表示的数值。
浮点型常量默认为 double 型，而 double 型常量不能赋值给 float 型变量。
boolean 与其他数据类型没有兼容性。

12、数据类型转换

1、自动类型转换（隐式类型转换）

- 需要同时满足两个条件：
- 1)、两种类型彼此兼容
 - 2)、目标类型的取值范围要大于源类型

2、强制类型转换（显示类型转换）

当两种类型不兼容，或目标取值类型范围小于源类型时，自动类型转换无法进行，需要进行强制类型转换。
数据类型强制转换的格式为：
（数据类型）数据表达式；
例如：int i = 5; byte b = (byte)i;

13、变量的作用域

变量的作用域指一个变量起作用的范围，它决定了一个变量何时可以访问、何时不可以访问。Java 中任何变量的作用域都从该变量声明之后开始，并且只在该声明的语句块中使用，也就是该变量只能在声明它的那个花括号 {} 中使用。
变量有分为成员变量和局部变量。
成员变量：在类中声明的变量称为成员变量，又叫全局变量。
使用范围：通常在类开始处声明，可在整个类中使用。
局部变量：在方法或块（块由两个花括号）中声明的变量称为局部变量。
使用范围：从声明处开始到它所在方法或块的结束处。

例：

```
{
int x = 4;
//这之间只有 x 可以访问
int y = 1;
//x 和 y 可以访问
{
int z = 2;
//x、y、z 都可以访问
z = 5;
}
x = 4;
//只有 x 和 y 可以访问，不可以访问 z
}
```

14、Java 中的运算符

算术运算符、关系运算符、赋值运算符、逻辑运算符、位运算符、条件运算符

1)、算术运算符

运算符	运算	范例	结果
-----	----	----	----

+	正号	+3	3
-	负号	b=4;-b;	-4
+	加	5+5	10
-	减	6-4	2
*	乘	3*4	12
/	除	5/5	1
%	取模（求余）	5%5	0
++	自增（前）	a=2;b=++a;	a=3;b=3;
++	自增（后）	a=2;b=a++;	a=3;b=2;
--	自减（前）	a=2;b=--a;	a=1;b=1;
--	自减（后）	a=2;b=a--;	a=1;b=2;
+	字符串相加	“he” + “llo”	“hello”

两个整数之间的相除 (/) 运算结果还是整数，其结果是除的结果的整数部分。

例如：5/2 结果为 2

要获得实数结果，运算中至少一个浮点数。

例如：5/2.0 结果为 2.5

2)、关系运算符

运算符	运算	范例	结果
==	相等于	4==3	false
!=	不等于	4!=3	true
<	小于	4<3	false
>	大于	4>3	true
<=	小于等于	4<=3	false
>=	大于等于	4>=3	true

3)、赋值运算符

运算符	运算	范例	结果
=	赋值	a=3;b=2;	a=3;b=2;
+=	加等于	a=3;b=2;a+=b;	a=5;b=2;
-=	减等于	a=3;b=2;a-=b;	a=1;b=2;
=	乘等于	a=3;b=2;a=b;	a=6;b=2;
/=	除等于	a=3;b=2;a/=b;	a=1;b=2;
%=	模等于	a=3;b=2;a%=b;	a=1;b=2;

4)、逻辑运算符

运算符	运算	范例	结果
&	AND(与)	false & true	false
	OR(或)	false true	true
^	XOR(异或)	false ^ true	true
!	NOT(非)	!true	false
&&	AND(短路与)	false && true	false
	OR(短路或)	false true	true

在使用短路与 (&&) 时，如果第一个操作数（或表达式）为“假”，则不再计算第二个操作数（或表达式），直接返回“假”。

在使用短路或 (||) 时，如果第一个操作数（或表达式）为“真”，则不再计算第二个

操作数（或表达式），直接返回“真”。

逻辑运算符只能用于布尔（boolean）类型之间；其结果值为布尔（boolean）类型。

5)、位运算符

运算符	运算	计算规则
&	按位与	只有参加运算的两位都为 1，‘&’ 运算的结果才为 1，否则为 0。
	按位或	只有参加运算的两位都为 0，‘ ’ 运算的结果才为 0，否则为 1。
^	按位异或	只有参加运算的两位不同，‘^’ 运算的结果才为 1，否则为 0。
<<	左移位	左移指定位数，右边补 0。
>>	右移位	右移高位是 0，左边补 0；高位是 1，左边补 1。
>>>	无符号右移位	左边补 0。
~	按位取反	1 取反是 0, 0 取反是 1。

位运算符只可用于整数类型、char 类型，不可应用于浮点类型。

6)、条件运算符

条件运算符是一个三目运算符，也是唯一的一个三元运算符，符号为“?:”，在程序中能实现简单的判断功能。

语法格式：

表达式 1?表达式 2:表达式 3

其中表示 1 是一个布尔表达式，如果表达式 1 结果为 true，则执行表达式 2，否则执行表达式 3。

举例说明：求 a，b 的最大值。

```
int a=4,b=6,max;  
max=a>b?a:b;//将 a 和 b 中的较大值赋给 max
```

15、Java 中的控制语句

- (1)、if(表达式).....else..... 条件语句;
- (2)、for（表达式）..... 循环语句;
- (3)、while（表达式）..... 循环语句;
- (4)、do.....while(表达式)..... 循环语句;
- (5)、switch 多分支选择结构;
- (6)、continue 结束本次循环语句;
- (7)、break 终止执行 switch 或循环语句;
- (8)、return 从方法返回语句。

1、条件语句

分为四种：

单分支条件语句

```
语法格式为：if(条件表达式){  
    语句或语句块;  
}
```

二分支条件语句

```
语法格式为：if(条件表达式){  
    语句或语句块 1;  
}else{
```



```
        语句或语句块 2;  
    }
```

嵌套条件语句

语法格式为：if(条件表达式) {
 if(条件表达式) {
 语句或语句块;
 }
 }else{
 语句或语句块 2;
 }

多分支条件语句

语法格式为：if(条件表达式 1) {
 语句或语句块 1;
 }else if(条件表达式 2) {
 语句或语句块 2;
 }

 } else if(条件表达式 n) {
 语句或语句块 n;
 }else{
 语句 0;
 }

2、for 循环语句

语法格式为：for(变量初始化表达式; 循环条件表达式; 迭代部分)
 {
 语句或语句块; //循环体
 }

for 循环有一个特殊的循环，叫**死循环**。表现形式为：

```
boolean isOk = true;        或    for(;;) {}  
for(;isOk;) {}
```

for 循环可以再嵌套 for 循环。

注意：在 for 循环的初始化或迭代部分，可以有多个表达式，表达式之间用逗号隔开。例如：

```
int count =0;  
for(int a= 1,b=10;a<b;a++,b--){  
    count++;  
} 共循环多少次？
```

3、while 循环语句

语法格式为：while(循环条件 表达式) {
 语句或语句块;
 }

4、do-----while 循环语句






语法格式为：do{
 语句或语句块;
 } while(表达式); (注意分号绝对不能省略)

5、switch——case 多分支选择语句

语法格式为：switch(条件表达式) {
 case 常量 1:
 语句 1;
 break;
 case 常量 2:
 语句 2;
 break;

 case 常量 N:
 语句 N;
 break;
 [default:语句;break;]
 }

注意：

-  条件表达式的返回值类型必须是以下类型之一：int、byte、char、short。
-  case 子句中的值常量 N 必须是常量，而且所有 case 子句中的值应是不同的。
-  default 子句是可选的。
-  break 语句用来在执行完一个 case 分支后，是程序跳出 switch 语句，即终止 switch 语句的执行。
-  在一些特殊情况下，多个不同的 case 值要执行一组相同的操作，这时可以不用 break。

6、continue 结束本次循环语句

使用 continue 语句可以立刻重新开始下一轮的循环，而不再执行循环后面的语句。

7、break 终止执行 switch 和循环语句

使用 break 语句可以立刻终止循环，开始执行循环后面的语句。

8、return 从方法返回语句

return 语句主要作用是退出当前方法，将程序控制转移到方法的调用者。

一般格式为：return [值或表达式];

16、方法调用语句

方法调用语句是执行对象的某个方法。

一个完整的方法调用语句由某个方法调用加上一个分号构成。

调用语法格式为：类对象名称.方法名（参数）；

例如：

```
System.out.println(“This is a statement call a method!”);
```

调用对象 System.out 的 println 方法。

17、表达式语句

表达式语句就是由一个表达式加一个分号构成的语句。

例如常见的赋值语句：i=i+1;而 i=i+1 只是一个表达式。

18、空语句

空语句就是只有一个分号构成的语句。例如：；

19、复合语句

复合语句又叫块，由 {} 将一些语句括起来就构成一个复合语句。

```
例如：{
    a=b+c;
    t=a*100;
}
```

20、数组

(1) 定义：是用来存储一组或多组相同类型数据的数据类型。

(2) 数据类型：可以是基本数据类型（例如：数字型、字符型、布尔型），也可以是复合数据类型（例如：数组、类、字符串和接口）。

※数组本身就是一种复合数据类型，因此，数组的元素也可以是数组，这样就构成了二维数组和多维数组。

(3) 数组作为复合数据类型，与基本数据类型最大的区别：
数组是通过引用来控制的，而基本数据类型是通过值来控制的。

1、一维数组的声明：

格式： 类型 数组名[]; 或 类型[] 数组名；

举例：int a[]; String[] b;

数组的初始化有两种方法：一种是直接初始化，另一种是动态初始化。

直接初始化

格式： 类型 数组名[] = { 值 1, 值 2, ..., 值 n};

举例：int a[] = {1, 2, 3};
int b[]; b = {1, 2, 3};

动态初始化：

格式：

(1) **声明时初始化：**

类型 数组名[] = new 类型[数组长度];

(2) **声明后初始化：**

类型 数组名[];
数组名 = new 类型[数组长度];

2、访问数组元素

数组元素是通过数组名和下标来访问。未被初始化的数组，不能进行访问。

格式： 数组名[下标]

Java 中，数组的下标从 0 开始，直到 <数组长度-1>结束。

获得数组的长度，通过 **length** 属性来获得。

3、数组的复制

方法：

System.arraycopy(源数组, 源数组起始位置, 目标数组, 目标数组起始位置, 长度);

★注：不管是要复制的数组，还是被复制的数组，都必须先初始化。

举例：

```
int a[ ] = {1, 2, 3}, b[ ] = new int[3];
System.arraycopy(a, 0, b, 0, 3); //将数组 a 复制到 b
```

4、对象数组：

举例：

```
//定义一个对象数组，StringBuffer 是一个字符串缓存类
StringBuffer a[ ] = new StringBuffer[2];
//给对象赋值
```

```
a[0] = new StringBuffer( "Array[0]" );
a[1] = new StringBuffer( "Array[1]" );
System.out.println( "a[0]=" + a[0] + " a[1]=" + a[1] );
a[0].append( " is Modified" ); //追加字符串
System.out.println( "a[0]=" + a[0]);
```

5、二维数组和多维数组


前面提到过，数组的元素也可以是数组，如果一个数组的每一个元素都是一个一维数组，这样就构成一个二维数组。

定义格式：类型 数组名[][]; 或 类型[][] 数组名;

举例： int a[][]; double[][] b;

这几种定义不合法：int a[2][]; int b[][2]; int c[2][2];

二维数组的初始化：二维数组的初始化也有直接和动态初始化两种方式。

 **直接初始化格式：**类型 数组名[][] = {{ 值 1, 值 2, ..., 值 n }, { 值 1, 值 2, ..., 值 n }....};

举例：int a[][] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

 **动态初始化格式**

① 类型 数组名=new 类型[长度 1][长度 2]; 长度 1 表示行，长度 2 表示列。

举例：int a[][]=new[3][5];

② 类型 数组名=new 类型[长度 1][];

数组名[0]=new 类型[长度 20];

数组名[1]=new 类型[长度 21];

.....

数组名[长度 1-1]=new 类型[长度 2n];

举例：int a[][];

a=new int[3][];

a[0]=new int[5];

a[1]=new int[8];

a[2]=new int[6];

6、二维数组的应用

举例：两个矩阵相乘的例子。

//声明并初始化数组

```
int a[ ][ ] = {{8, 13}, {4, 7}, {5, 2}};
```

```
int b[ ][ ] = {{3, 4, 11}, {6, 1, 10}};
```

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

```
int i, j, k;
```

//通过嵌套循环实现矩阵相乘的运算

```
for(i=0;i<a.length;i++){
```

```
    for(j=0;j<b[0].length;j++){
```

```
        result[i][j] = 0;
```

```
        for(k=0;k<b.length;k++){
```

```
            result[i][j] += a[i][k] * b[k][j];
```

```
        }
```

```

    }
}
//打印结果
System.out.println( "The result of a * b is : " );
for(i=0;i<result.length;i++){
    for(j=0;j<result [i].length;j++){
        System.out.print(result[i][j] + "    ");
    }
    System.out.println( ); //换行 }

```

7、一维数组的应用

例 1：将整数序列 {3, 15, 28, 11, 34, 78, 95, 27, 18} 首末颠倒过来。

```

int a[ ] = {3, 15, 28, 11, 34, 78, 95, 27, 18};
int i, length, temp;
length = a.length;
for(i=0;i<(length/2);i++){
    //以下将数组元素 a[i]和 a[length -1 - i]的值互换
    temp = a[i];
    a[i] = a[length -1 - i];
    a[length -1 - i] = temp;
}
for(i=0;i<length;i++){
    System.out.print(a[i] + "    ");
}

```

例 2：从数据：32 25 78 69 13 97 86 38 62 9 中找到数据 97 所在的位置。

【用顺序查找（线性查找）方法编写程序】。

```

//声明并初始化数组
int a[ ] = {32, 25, 78, 69, 13, 97, 86, 38, 62, 9};
int index = -1, i;
//逐个元素与 97 相比较，找到则退出循环，否则继续
for(i=0;i<a.length;i++){
    if(a[i] == 97){
        index = i;
        break;
    }
}
if(index == -1){ //表示该数不存在
    System.out.println( " 97 这个数不存在!" );
}else{
    System.out.println( "97 这个数的下标是： " + index);
}

```

例 3：将数据：37 28 51 13 64 为例，用【冒泡法】进行升序排列。

下面编写程序代码：（冒泡法）

```

int a[ ] = {37, 28, 51, 13, 64};
int i, j, n, temp;

```



```
n = a.length;
for(j=1;j < n ;j++ ){ //共执行 n-1 轮
    for(i=0;i< n - j; i++){ //第 j 轮
        if (a[i] > a[i+1]){
            //交换 a[i]与 a[i+1]的值
            temp = a[i];
            a[i] = a[i+1];
            a[i+1] = temp;
        }
    }
}
//打印排序后的结果
for(i=0;i < n;i++){
    System.out.print(a[i] + " ");
}
```