
学校的理想装备
电子图书·学校专集
校园网上的最佳资源
Java语言规格说明
BOOK
内部资料 非卖品

Java 语言规格说明

(译自 Java language specification)

1.程序结构 Java 语言的源程序代码由一个或多个编译单元(compilation unit)组成，每个编译单元只能包含下列内容(空格和注释除外)：

* 一个程序包语句(package statement)
* 引入语句(import statements)
* 类的声明(class declarations)
* 界面声明(interface declarations)每个 Java 的编译单元可包含多个类或界面，但是每个编译单元却至多有一个类或者界面是公共的。

Java 的源程序代码被编译之后，便产生了 Java 字节代码(bytecode)。

Java 的字节代码由一些不依赖于机器的指令组成，这些指令能被 Java 的运行系统(runtime system)有效地解释。Java 的运行系统工作起来如同一台虚拟机。

在当前的 Java 实现中，每个编译单元就是一个以.java 为后缀的文件。

每个编译单元有若干个类，编译后，每个类生成一个.class 文件。.class 文件是 Java 虚机器码?2. 词法问题在编译的过程中，Java 源程序代码中的字符被划分为一系列的标记(token)。Java 编译器可以识别五种标记: 标识符、关键字、字面量、运算符以及特殊分隔符。注释以及诸如空格、制表符、换行符等字符 ，都不属于标识之列，但他们却常被用来分隔标记。

Java 程序的编写采用泛代码 Unicode 字符集，若采用其它的字符集，则需在编译前转换成 Unicode。

2.1 注释

Java 语言提供了 3 种形式的注释：

//text 从//到本行结束的所有字符均作为注释而被编译器忽略。

/* text */从/*到*/ 间的所有字符会被编译器忽略。

/** text */当这类注释出现在任何声明之前时将会作特殊处理，它们不能再用在代码的任何地方。这类注释意味着被括起来的正文部分，应该作为声明项目的描述，而被包含在自动产生的文档中。

2.2 标识符标识符的首字符必须是一个字母，下划线("_")或美元符号("$")。 后面的字符也可是数字 0-9。 Java 使用泛代码字符集，为了便于识别好一合法标识符，下面列出它的“字母”:

* 大写字母“A”～“Z”
* 小写字母“a”～“z”
* 泛代码（Unicode）中所有字符编码在十六进制数 00C0 之前的字符。标识符中，首字母后的字符可以是任意的。当然，Unicode 区段中那些被保留作特殊字符的除外。

由此，“garton”及“MjΦlner”都是合法标识符，但是，包括诸如“π”的字符串却不是合法的。

为了取得更多的有关泛代码标准的信息，请参阅“The UnicodeStandard”，“World Wide Character Encoding version 1.0，volumes 1 &2”，Unicode 公司的 FTP 地址是 unicode.org。

2.3 关键字

下面的标识符被保留用作关键字，他们不能作任何其它的用途。

abstract default goto*null synchronizedboolean do if package thisbreakdouble implementsprivate threadsafebyteelseimportprotectedthrowbyvalve *extendsinstanceofpublictransientcasefalseint returntruecatchfinalinterface shorttrycharfinally long staticvoidclassfloat nativesuperwhileconst * fornew switchcontinue

其中，加*标记后是被保留但当前却未使用的。

2.4 字面量字面量(literal)是某些类型值的基本表述，这些类型包括整型，浮点型，布尔量，字符及字符串。

2.4.1 整型字面量整数可有三种表示形式： 十进制，八进制和十六进制。一个十进制整型字面量由一系列的数字组成，但它的第一个数字不能是 0(有时十进制数字也可象下面讲的那样加后缀)。整数也可表达成八进制或十六进制形式。以 0 开头的整型字面量，意味着它是一个十六进制的。十六进制整数可以包括数字 0-9 以及字母 a-f 及 A-F。八进制整数中则只能是出现数字 0-7。在超过 32 位所能表示的范围之前，整型字面量的类型即为 int，否则为 long 型。一个整型字面量可通过加后缀 L 或 l 而强迫成 long 型。

下面的均为合法的整型字面量。

22L07772.4.2 浮点字面量一个浮点字面量可包括以下部分：一个十进制整数，一个小数点“ . ”，小数部分(另外一个十进制整数)，指数部分，一个类型后缀。指数部分是一个 e 或 E 后跟一个整数。浮点字面量至少包含有一个数字，外加或者一个小数点或者一个 e(或 E)，下面举一些浮点字面量的例子：

3.14153.1E12.1e122E12 就象在后面描述的那样，Java 语言有两种浮点类型： float 及 double，用户可按以下写法区分：

2.0d 或 2.0Ddouble 型

2.0f 或 2.0F 或 2.0float 型 2.4.3 布尔字面量

布尔(boolean)字面量有两个值： true 及 false。

2.4.4 字符字面量字符字面量是一个由单引号括起的字符(或者是由一组字符来表述一个字符)。字符属于 char 类型，并且均从泛代码字符集中得来。而下面列出的转义序列则用来描述一些非图形字符，它们以反斜杠“ \ ”开始以作转义用。

续行符头<newline> \换行 NL(LF)\n 垂直制表符 HT\t 退格 BS\b 回车 CR\r 走纸换页 FF\f 反斜杠 \ \\单引号 ' \'双引号 " \"八进制数 0ddd \ddd 十六进制数 0xdd \xdd 泛代码字符 0xdddd\udddd2.4.5 串字面量串字面量是双引号引起的零个或多个字符的序列。每个串字面量被看作是一个串对象，而并非是一个字符的数组，例如“ abc ”创建了一个新的串类的实例。下面的都是合法的串字面量：

```
" "\\空串
"\" " \\只包含一个双引号的串
"This is a string"
"This is a \
 two-line string"2.5  运算符及特殊分隔符
```

下面这些字符在 Java 源程序中作运算符或分隔符用：

+ — ! %^&*|～/ ><

() { } [ ] ;? : ,· =

另外，下面这些复合字符被用作运算符：

++--== <=>= != << >>

>>> +=-=*= /=&= /=

^= %= <<= >>= >>>= ‖ &&

后面还要在运算符一节中作详细介绍。

3．类型任何一个变量或表达式都有一个类型，类型决定变量可能的取值范围，决定对这些值允许的操作，以及这些操作的意义是什么。Java 语言中提供了内置定义类型，程序员也可以利用类及界面(interface)机制构造新类型。

Java 语言有两种类型： 简单类型和复合类型。简单类型指那些不能再分割的原子类型。如：整型、浮点型、布尔型、字符型均为简单类型。

复合类型建立在简单类型的基础上。Java 语言有三种复合类型：数组、类及界面。在本节中，我们主要讨论简单类型及数组。

3.1 数值类型 3.1.1 整数类型整数与 C 及 C++中相似，但有两点区别： 其一，所有的整数类型均是独立于机器的；其二，对某些传统的定义作出改变，以反映自 C 问世以来所带来的变化，四种整数类型分别具有 8 位、16 位、32 位及 64 位的宽度，并且均是有符号的(signed)。如下所示：宽度类型名 8byte16short32int64long 一个变量的类型不会直接影响它的存储的分配，类型仅仅决定变量的算术性质以及合法的取值范围。如果把一个超出合法范围的值赋给一变量，那么这个值将是对合法值域取模后的值。

3.1.2 浮点类型关键字 float 表示单精度(32 位)，而 double 则表示双精度(64 位)，两个 float 型数运算的结果仍是 float 型，若有其中之一为 double 型，则结果为 double 型。浮点运算及数据格式按 IEEE754 中的定义，细节问题请参阅 “ 附录 A： 浮点 ” 中有关浮点实现的细节。

3.1.3 字符类型 Java 全部使用泛代码字符集，因此 char 类型数据被定义成一个 16 位的无符号整数。

3.2 布尔类型当一个变量的取值或为 ture 或为 false，或者是当一个方法的返回值为 ture 或 false 时，它?嵌际遣级 嘈偷摹A硗猓 叵翟怂愕慕峁 嗍遣?尔型的。

布尔值不是数值型，因此不能用强制类型转换把它们转化成数值。

3.3 数组数组在 Java 语言中属第一类对象。由它们代替了指针运算，所有的对象(包括数组)都可通过标识来引用。即使被当作数运算，标识的值也不应被破坏。通过 new 运算符可创建一个数组。

char s[]=new char[30];数组第一元素的下标为 0，在声明中指定维数是不允许的。每次都必须显式地用 new 分配数组：

int i [] =new int [3];

Java 语言不支持多维数组，但是，程序员却可以创建数组的数组。

int i [ ] [ ]=new int [3][4];

至少有一维要明确给定，而其它维则可在以后再确定。例如：

int i[] []=new int [3] [ ]

是一个合法的声明。

除了在变量名及方法名后跟方括号这种 C 风格的声明之外，Java 语言允许方括号跟在数组类型之后，下面两行是等价的：

int iarray[ ];
int [ ] iarray;
同样地，方法声明也一样：
byte f(int n)[ ];
byte [ ] f(int n);
运行时检查下标保证它们是合法的：
int a[ ]=new int [10];
a[5]=1;
a[1]=a[0]+a[2];a[-1]=4; // 运行时引发一个ArrayIndexOutOfBoundsException(数组下标越界)异常 a[10]=2; //运行时引发一个 ArrayIndexOutOfBoundsException(数组下标越界)异常
数组的大小必须使用整数表达式：
int n;
float arr[ ]=new float[n+1]
数组的长度可通过.length 查找：
int a[] []=new int [10][3];
println (a.length) //打印出 10
println (a[0].length) //打印出 3。
3.3.1 数组细节我们所定义的数组都是 Object 类的一个子类的实例，在类的层次结构中有一个被称为 Array 的子类，它有一个实例变量“ length ”。对每一个基本类型，都有一个相应的 Array 的子类。同理，每一个类也都有一个相应的 Array 子类存在。例如：new Thread[n] 创建一个 Thread[ ]的实例。如果类 A 是类 B 的超类，那么，A[]是 B[]的超类，见下图：
Object
Array Aint[] float[] A[]B
B[]
因此，可以把数组赋给一个 Object 变量。
Object o;
int a[]=new int [10];
o=a;
并且可通过强制类型转换把 object 变量赋给一数组变量。
a=(int [])o;
Array 类本身不能显式地产生子类。
4. 类类(class)是传统面向对象编程模型的象征。它们支持数据抽象以及实现对数据的约束，在 Java 中，每一个新的类创建一个新类型。
要想得到一个新的类，程序员必须首先找到一个已有的类，新类即在这个已有类的基础上构造，我们称之为派生(derived)。派生出的类亦称为原来类的子类，而这个类我们称为超类(super class)。
类的派生具有传递性： 如果 B 是 A 的子类，C 是 B 的子类，则 C 是 A 的子类。
一个类的直接超类以及指示这个类如何实现的界面(interface)，在类的声明中，由关键字 extends 及 implements 标出。如下示(黑体表示关键字)：:
[doc_ comment] [modifer] class ClassName
extends Superclassname

```
implements  interface {, interface } ] {
  class body
}
```

举例：

```
/** 2 dimension point */
public class Points {
float x, y;
 ......
}
/** printable point */
class PinttablePoint extends Points implements Printable {
......
public void Print ( ) {
}
```

}所有的类均从一个根类 Object 中派生出来。除 Object 之外的任何类都有一个直接超类。如果一个类在声明时未指明其直接超类，那么缺省即为 Object。如下述：

```
class Point {
float x, y
}
```

与下面写法等价

```
class Point extends Object {
 float  x,   y;
```

}Java 语言仅支持单继承，通过一个被称作“界面”的机制，来支持某些在其它语言中用多继承实现的机制(详见“界面”一节)。Java 之所以没有采用 C++ 的多继承机制，是为了避免多继承带来的诸多不便，例如：可能产生的二义性，编译器更加复杂，程序难以优化等问题。

4.1 类类型之间的强制转换 Java 语言支持在两个类型之间的强制转换，因为每个类即是一个新的类型。Java 支持类类型之间的强制转换，如果 B 是 A 的子类，那么 B 的一个实例亦可作为 A 的实例来使用，虽然不需要显式的转换，但显式转换亦是合法的，这被称作拓宽(widening)。如果 A 的一个实例，想当作 B 的实例使用，程序员就应写一个类型转换叫作削窄（narrowing）的强制。从一个类到其子类的强制转换在运行时要作例行的检查以确保这个对象就是其子类的一个实例(或其子类之一)。兄弟类之间的强制类型转换是一个编译错误，类的强制转换的语法如下??

```
(classname) ref
```

其中，(classname)是要转换的目的类，而 ref 是被转换的对象。

强制转换仅仅影响到对象的引用，而不会影响对象本身。然而，对实例变量的访问却受到对象引用的类型的影响。一个对象从一个类型到另一类型的强制转换后，可以使同一变量名对不同的实例变量访问。

```
classClassA{
String  name = "ClassA"
}
class ClassB  extends  ClassA {//ClassB 是 ClassA 的子类
```

```
 String name="ClassB";
}
class AccessTest {
void test( ) {
ClassB  b=new ClassB( );
println (b.name);//打印： ClassB
ClassA  a
a=(ClassA)b;
println (a.name);//打印： ClassA
}
```

}4.2 方法方法(method)是可施于对象或类上的操作，它们既可在类中，也可在界面中声明。但是他们却只能在类中实现(Java 中所有用户定义的操作均用方法来实现)。

类中的方法声明按以下方式：

[Doc_ comment] [Access Specifiers] ReturnType methodName(parameterList){

method body(本地的 native 及抽象的方法没有体部分)

}除构造函数可以无返回类型外，其余的方法都有一个返回类型。如果一个不是构造函数的方法不返回任何值，那么它必须有一个 void 的返回类型。

参数表由逗号分隔的成对的类型及参数名组成，如果方法无参数，则参数表为空。方法内部定义的变量(局部变量)不能隐藏同一方法的其它变量或参数。例如： 如果一个方法带以名为 i 的参数实现，且方法内又定义一个名为 i 的局部变量，则会产生编译错误，例如：

```
class Rectangle {
 void vertex (int i, int j)  {
 for (int i=0; i<=100; i++)  { //出错
 …
 }
 }
}
```

方法体内循环语句中声明的 i 是一个编译错误。

Java 语言允许多态方法命名，即用一个名字声明方法，这个名字已在这个类或其超类中使用过，从而实现方法的覆盖(overriding)及重载(overloadding)。所谓覆盖是对继承来的方法提供另一种不同的实现。而重载是指声明一个方法，它与另外一个方法有相同的名字，但参数表不同。

注： 返回类型不能用来区别方法，即在一个类的范围内，具有相同的名字，相同的参数表(包括个数、位置及类型)的方法，必须返回相同的类型。若这样的两个方法有不同的返回类型，将会产生一个编译错误。

4.2.1 实例变量实例变量(instance variables)是指那些在类内声明，但在方法的作用域之外尚未被 static 标记的变(参照 “ 静态方法，变量及初始化 ” 段)。

而在一个方法的作用域之内声明的变量是局部变量。实例变量可以有修饰符(见修饰符)。

实例变量可以是任何的类型，并且可以有初始值。如果一个实例变量无初

始值，它将被初始化成 0。布尔型变量被初始化成 flase，对象被初始化成 null。下面是一个实例变量 j 具有初始化值的例子：

```
class A{
  int j =23;
……
```

}4.2.2 this 和 super 变量在一个非静态方法的作用域内，this 这个名字代表了当前对象。例如： 一个对象可能需要把自己作为参数传给另一个对象的方法：

```
class MyClass {
 void Method (OtherClass obj) {
 …
obj.Method (this)
  …
 }
}
```

不论何时，一个方法引用它自己的实例变量及方法时，在每个引用的前面都隐含着“ this ”。

如：

```
class Foo {
int a, b, c;
……
void myPrint ( ) {
print (a+ "\n");//  a=="this.a"
}
……
 }
```

super 变量类似于 this 变量。this 变量实际上是对当前对象的引用，它的类型就是包含当前正在处理的方法的类。而 super 变量则是对其超类类型对象的引用。