Java

**用户标识符**

用户标识符是程序员对程序中的各个元素加以命名时使用的命名记号。

在Java语言中，标识符是以字母、下划线（”\_”）或美元符（”$”）开始，后面可以跟字母、下划线、美元符和数字的一个字符序列。

注意：标识符中的字符是区分大小写的。例如，Name和name被认为是两个不同的标识符。

保留字是特殊的标识符，具有专门的意义和用途，不能当作用户的标识符使用。Java语言中的保留字均用小写字母表示

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| abstract抽象 | break跳出 | byte字节 | boolean布尔型 | catch | case | class | char | continue |
| default | double | do | else | extends | false | final | float | For |
| finally | if | import | implements | int | interface | instanceof | long | 1ength |
| native | new | null | package | private | protected | public | return | switch |
| short | static | super | try | true | this | throw | throws | void |
| threadsafe | transient | while | synchronized |  | | | | |

数据类型

Java语言的数据类型可划分为基本数据类型和引用数据类型

基本数据类型

基本数据类型

引用数据类型

整数类型

字符类型

布尔类型浮点类型

单精度浮点型

双精度浮点型

字节型

短整型

整型

长整型

类

接口

数组

字符串

数据类型

数据类型 所占二进制位 所占字节 取值

byte 8 1 -27～27-1

short 16 2 -215～215-1

Int 32 4 -231～231-1

long 64 8 -263～263-1

char 16 2 任意字符，只能是一个半角字符，一位

boolean 8 1 true , false

float 32 4 -3.4E38(3.4×1038)～3.4E38(3.4×1038)

double 64 8 -1.7E308(1.7×10308)～1.7E308(1.7×10308)

**常量**

所谓常量就是在程序运行过程中保持不变的量即不能被程序改变的量，也把它称为最终量。

标识常量使用一个标识符来替代一个常数值，其定义的一般格式为：

**final 数据类型 常量名=value[,常量名=value …];**

注意：在程序中，为了区分常量标识符和变量标识符，常量标识符一般全部使用大写书写。

**变量**

变量是程序中的基本存储单元，在程序的运行过程中可以随时改变其存储单元的值。

变量的一般定义如下：

数据类型 变量名[=value] [, 变量名[=value] …];

**变量的作用域**是指变量自定义的地方起，可以使用的有效范围。在程序中不同的地方定义的变量具有不同的作用域。一般情况下，在本程序块（即以大括号“{}”括起的程序段）内定义的变量在本程序块内有效。

算术运算符

算术运算符用于数值量的算术运算，它们是：

+（加），-（减），\*（乘），/（除），%（求余数），++（自加1），--（自减1）

关系运算符和关系表达式

关系运算符用于两个量的比较运算，它们是：

>（大于），<（小于），>=（大于等于），<=（小于等于），==（等于），!=（不等于）

关系运算符组成的关系表达式（或称比较表达式）产生一个布尔值。若关系表达式成立产生一个true值，否则产生一个false值。

布尔逻辑运算符和布尔表达式

布尔逻辑运算符用于布尔量的运算，有3个布尔逻辑运算符：

1. !（逻辑非）

！运算的一般形式是： ! A

其中：A 是布尔逻辑或关系表达式。若A的值为true，则!A的值false，否则为true。

2. &&（逻辑与）

&&运算的一般形式是： A&&B

其中：A、B 是布尔逻辑或关系表达式。若A和B的值均为true，则A&&B的值为true，否则为false。

3. ||（逻辑或）

|| 运算的一般形式是： A || B

其中：A、B 是布尔逻辑或关系表达式。若A和B的值只要有一个为true，则A||B的值为true；若A和B的值均为false时，A||B的值为false。

赋值运算符（=）和赋值表达式

赋值运算符是最常用的运算符，用于把一个表达式的值赋给一个变量（或对象）。

与c、c++类似，Java也提供了复合的或称扩展的赋值运算符：

对算术运算有： +=，- =，\*=，/=，%=

例如：

x\*=x+y ; 相当于x=x\*(x+y) ;

x+=y; 相当于 x=x+y ;

条件运算符（ ? ：）及表达式

条件运算符是三元运算符，有条件运算符组成的条件表达式的一般使用格式是：

逻辑(关系)表达式 ？ 表达式1 ： 表达式2

其功能是：若逻辑（关系）表达式的值为true，取表达式1的值，否则取表达式2的值。

条件运算符及条件表达式常用于简单分支的取值处理。

例如，若已定义a,b为整型变量且以赋值，求a,b两个数中的最大者，并赋给另一个量max，可以用如下式子处理：

max=(a>b) ? a : b;

其它运算符

1.数组下标运算符（[ ]）

主要用于数组

2.强制类型转换运算符（(类型)）

在高类型的数据向低类型的数据转换时，一般需要强制转换

3.( )运算符

( )运算符用在运算表达式中，它改变运算的优先次序

4.字符串连接符（+）

在表达式中，如果+号运算符前边的操作数是一个字符串，此时该+号运算符的作用是字符

类型转换

整型、实型、字符型数据可以混合运算。类型从低级到高级顺序示意如下：

低---------------------------------------------->高

byte—> short, char—> int —> long—> float —> double

注意:如果将高类型数据转换成低类型数据，则需要强制类型转换，这样做有可能会导致数据溢出或精度下降

**Java程序的注释方式：程序注释和程序文档注释**

**程序注释**

程序中的注释不是程序的语句部分，它可以放在程序的任何地方，系统在编译时忽略它们。

注释可以在一行上，也可在多行上。有如下两种方式的注释：

1. 以双斜杠（//）开始

例如，在下边的程序片段中使用注释：

//下面定义程序中所使用的量

int id ; //定义一整型变量id ，表示识别号码。

String name; //定义一字符串变量name，表示名字。

1. 以“/\*”开始，以“\*/”结束

当需要多行注释时，一般使用“/﹡……﹡/”格式作注释，中间为注释内容。

/\*

\* 本程序是一个示例程序，在程序中定义了如下两个方法：

\* setName(String ) ---设置名字方法

\* getName () --- 获取名字方法。

\*/

**程序文档注释**

程序文档注释是Java特有的注释方式，它规定了一些专门的标记，其目的是用于自动生成独立的程序文档。

程序文档注释通常用于注释类、接口、变量和方法。下面看一个注释类的例子：

/\*\*

\* 该类包含了一些操作数据库常用的基本方法，诸如：在库中建立新的数据表、

\* 在数据表中插入新记录、删除无用的记录、修改已存在的记录中的数据、查询

\* 相关的数据信息等功能。

\* @author unascribed (用于说明本程序代码的作者)

\* @version 1.50, 02/02/06 （用于说明程序代码的版本及推出时间）

\* @since JDK2.0 （用于说明开发程序代码的软件环境）

\*/

**分支控制语句**

**if条件分支语句：**

if (布尔表达式) 语句;

else if (布尔表达式2) 语句2;

……

else if (布尔表达式n-1) 语句n-1;

else 语句2;

功能：若布尔表达式（关系表达式或逻辑表达式）产生true (真)值，则执行语句，否则跳过该语句，继续判断else if内的条件，如果满足则执行else if后的语句，不满足就跳过，继续判断下一个，最后执行else后面的语句

**switch条件语句：**

如上所述，if~ else if ~ else是实现多分支的语句。但是当分支较多时，使用这种形式会显得比较麻烦，程序的可读性差且容易出错。Java提供了switch语句实现“多者择一”的功能。switch语句的一般格式如下：

switch(表达式)

{

case 常量1： 语句组1; [break;]

case 常量2： 语句组2;[break;]

……………………………

case 常量n-1： 语句组n-1;[break;]

case 常量n： 语句组n;[break;]

default： 语句组n+1;

}

注意：break关键字的作用是结束本switch结构语句的执行，跳到该结构外的下一个语句执行

**循环控制语句**

**for循环语句：**

for循环语句是最常见的循环语句之一。for循环语句的一般格式如下：

for ( 表达式1; 表达式2; 表达式3)

{

语句组； //循环体

}

1）表达式1一般用于设置循环控制变量的初始值，例如：int i=1；

2）表达式2一般是关系表达式或逻辑表达式，用于确定是否继续进行循环体语句的执行。例如：i<100;

3）表达式3一般用于循环控制变量的增减值操作。例如：i++; 或i--;

**While和do-While循环语句：**

一般情况下，for循环用于处理确定次数的循环；while和do-while循环用于处理不确定次数的循环。

1. while循环

while循环的一般格式是：

while(布尔表达式)

{

语句组； //循环体

}

2. do-While循环

do-while循环的一般格式是：

do

{

语句组； //循环体

}

while(布尔表达式)；

while先判断后循环（有可能一次也不执行），do~while先循环后判断（至少执行一次循环）

**break语句：**

可用于分支，循环语句的结构中。同样它也用来结束循环，使程序跳到循环结构后边的语句去执行。

**continue语句：**

只能用于循环结构中，它用来结束本轮次循环（即跳过循环体中下面尚未执行的语句），直接进入下一轮次的循环。

**返回语句return：**

return语句用于方法中，该语句的功能是结束该方法的执行，返回到该方法的调用者或将方法中的计算值返回给方法的调用者。

**类 和 对象**

**面向对象的特性**

1）继承

2）封装

3）多态：在Java中通过方法的重载和覆盖来实现多态性。

**类：（大驼峰）**

定义类 ：具有相同的属性和方法的对象的集合，定义了该集合中每个对象所共有的属性和方法

个类的定义包括如下两个方面：

定义属于该类对象共有的属性（属性的类型和名称）；

定义属于该类对象共有的行为（所能执行的操作即方法）。

类包含类的声明和类体两部分，其定义类的一般格式如下：

[访问限定符] [修饰符] class 类名 [extends 父类名] [implements 接口名列表>]//类声明

{ //类体开始标志 继承 实现

[类的成员变量（属性）说明] //属性说明

[类的构造方法（函数）定义]

[类的成员方法（函数）定义] //行为定义

} //类体结束标志

访问限定符：确定该定义类可以被哪些类使用。可用的访问限定符如下：

a)public 表明是公有的。可以在任何Java程序中的任何对象里使用公有的类。该限定符也用于限定成员变量和方法。如果定义类时使用public进行限定，则类所在的文件名必须与此类名相同（包括大小写）。其他类也可以调用

b)private表明是私有的。该限定符可用于定义内部类，也可用于限定成员变量和方法。只有本类能够调用

c)protected 表明是保护的。只能为其子类（只要是子类就行），和同一个包下所访问。

d)默认访问 若没有访问限定符，则系统默认是友元的 (friendly)。友元的类可以被本类包中的所有类访问。只有同一个包下可以访问。

**修饰符**：作用是确定该定义类如何被其他类使用。可用的类修饰符如下：

a)abstract 说明该类是抽象类。抽象类不能直接生成对象。

b)final 说明该类是最终类，最终类是不能被继承的。没有子类的类。

**成员变量：（小驼峰）**

成员变量用来表明类的特征（属性）。声明或定义成员变量的一般格式如下：

[访问限定符] [修饰符] 数据类型 成员变量名[=初始值];

修饰符用来确定成员变量如何在其他类中使用。可用的修饰符如下：

a)static 表明声明的成员变量为静态的。静态成员变量的值可以由该类所有的对象共享，它属于类，而不属于该类的某个对象。即使不创建对象，使用“类名.静态成员变量”也可访问静态成员变量。

b）final 表明声明的成员变量是一个最终变量，即常量。最终不能被覆盖的方法。

**成员方法：**

方法用来描述对象的行为，成员方法用来实现类的行为。

方法也包含两部分，方法声明和方法体（操作代码）。

方法定义的一般格式如下：

[访问限定符] [修饰符] 返回值类型 方法名([形式参数表]) [throws 异常表]

try

catch

{

[ 变量声明 ] //方法内用的变量，局部变量

[ 程序代码 ] //方法的主体代码

[ return [ 表达式 ] ] //返回语句

}

修饰符用于表明方法的使用方式。可用于方法的修饰符如下：

a)abstract说明该方法是抽象方法，即没有方法体（只有“{}”引起的空体方法）。

b)final说明该方法是最终方法，即不能被重写。

c)static说明该方法是静态方法，可通过类名直接调用。

**构造方法：**

构造方法用来构造类的对象。如果在类中没有构造方法，在创建对象时,系统使用默认的构造方法。定义构造方法的一般格式如下：

[public] 类名（[形式参数列表]）

{

[方法体]

}

1）构造方法的名字就是类名。

2）访问限定只能使用public或缺省。一般声明为public，如果缺省，则只能在同一个包中创建该类的对象。

3）在方法体中不能使用return语句返回一个值。

4）构造方法可以重载，但不可以重写！

**对象**

**1. 声明对象**

声明对象的一般格式如下：

**类名 对象名;**

例如：

　　　Citizen p1,p2; //声明了两个公民对象

　　　Float f1,f2; //声明了两个浮点数对象

声明对象后，系统还没有为对象分配存储空间，只是建立了空的引用，通常称之为空对象（null）。因此对象还不能使用。

2**. 创建对象**

对象只有在创建后才能使用，创建对象的一般格式如下：

　　　对象名 = new 类构造方法名([实参表])

由于实参与形参按对应关系一一传递数据，因此在实参和形参的结合上必须保持“三一致”的原则，即：

1）实参与形参的个数一致；

2）实参与形参对应的数据类型一致；

3）实参与形参对应顺序一致。

**参数传递方式**

1. 按值传递方式：只将数值传递，无法改变参数值。（基本数据类型和string型）

Int a=1;

Int b=a;

1. 按引用传递方式 ：可以改变参数值。（引用数据类型：对象，数组）

Circle c1=new circle();

Circle c2=c1;//没有开辟新的内存地址，只是将c1的地址给了c2

**静态方法：**

用类名去调用，静态不能用对象去调用。

属于整个类，不属于某个对象

本类里：

静态 调用 非静态=创建对象并调用

静态　调用 静态=直接调用

非静态 调用 非静态=直接调用

非静态 调用 静态=直接调用

其他类里：

静态 调用 非静态=创建对象并调用

静态　调用 静态=类名.静态方法

非静态 调用 非静态=创建对象并调用

非静态 调用 静态=类名.静态方法

静态方法称之为类方法，非静态方法是成员方法

**main() 方法：**

所有项目的入口函数，必须经过main函数调用才能使用

main()方法就是一个静态的方法，main()方法也是一个特殊的方法，解释器在装入程序后首先运行的是main()方法。

main()方法和其他的成员方法在定义上没有区别，其格式如下：

public static void main(String [] args)（args是参数变量名）

{

//方法体定义

………

}

**最终（Final）方法**

在Java中，子类可以从从父类继承成员方法和成员变量，并且可以把继承来的某个方法重新改写并定义新功能。但如果父类的某些方法不希望再被子类重写，必须把它们说明为最终方法，final修饰即可。

**类的继承**

类继承的实现：

public class 新子类 extends 父类

覆盖（Override）和重载方法

覆盖特点：必须是子类的方法覆盖父类的方法，方法名相同，参数列表必须相同，返回值类型相同。

重载特点：子类父类皆可重载。方法名相同，参数列表不同为重载。

Super 和 this：

引用父类方法时，我们使用super关键字。

this代表当前对象对本类成员的引用；而super则代表当前对象对父类成员的引用。

在程序中对变量的引用时，什么情况下不需要加this、super？什么情况下需要加，加哪个？其规则如下：

1）当不涉及同名变量的定义时，对变量的引用不需要加this或super关键字。

2）当涉及同名变量的定义时，分两种情况：

a)方法变量和成员变量同名，在引用成员变量时，前边加this；

B）本类成员变量和父类成员变量同名，在引用父类成员变量时，前边加super。

**super 关键字是用来访问父类的成员（如方法、字段、构造方法）。但是，super 关键字在某些情况下是可以省略的，具体情况如下：**

**1. 调用父类的无参构造方法时**

如果子类的构造方法没有显式调用父类的构造方法（即没有使用 super()），Java 编译器会自动调用父类的无参构造方法（super()）。因此，在这种情况下，super() 是可以省略的。

class Animal {

Animal() {

System.out.println("Animal constructor");

}

}

class Dog extends Animal {

Dog() {

// super(); // 这行是可选的，Java 会自动添加这一行

System.out.println("Dog constructor");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog(); // 输出：Animal constructor Dog constructor

}

}

在 Java 中，每当你创建一个子类的对象时，父类的构造函数 总是会被调用。这个调用顺序是：首先调用父类的构造函数，然后调用子类的构造函数。

super()

/在子类构造函数中，如果没有显式调用 super()，Java 编译器会隐式地添加 super()，也就是说，父类的无参构造函数会被自动调用。

/如果父类没有无参构造函数（没有默认构造函数），并且子类没有显式调用父类的构造函数，编译器会报错。

执行顺序：

在 main 方法中，创建 Dog 类的对象时：

Animal 类的构造函数 首先被调用，输出 "Animal constructor"

然后，Dog 类的构造函数 被调用，输出 "Dog constructor"

当父类中存在一个有参的构造函数时，需要使用super（）

class Animal {

private String name;

// 父类的有参构造方法

public Animal(String name) {

this.name = name;

System.out.println("Animal's name: " + name);

}

}

class Dog extends Animal {

private int age;

// 子类的构造方法，通过 super 调用父类的有参构造方法

public Dog(String name, int age) {

super(name); // 调用父类的有参构造方法

this.age = age;

System.out.println("Dog's age: " + age);

}

}

public class Main {

public static void main(String[] args) {

// 创建 Dog 类的实例，构造函数会调用父类的有参构造方法

Dog dog = new Dog("Buddy", 5);

}

}

//输出：

Animal's name: Buddy

Dog's age: 5

2. 调用父类的方法时

如果子类需要直接使用父类的方法，新建子类对象直接调用父类的方法时可以不使用 super。

class Animal {

void makeSound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

// 直接调用父类的方法

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.makeSound(); // 输出：Animal sound

}

}

如果要使用父类方法并对父类方法进行编辑，则需要用super（）

class Animal {

public void speak() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

@Override

public void speak() {

// 调用父类的speak方法

super.speak();

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.speak();

//输出：

Animal makes a sound

Dog barks

}

}

**抽象类**

声明抽象方法的一般格式如下：

[访问限定符] abstract 数据类型 方法名([参数表]);

注意：抽象方法只有声明，没有方法体，所以必须以“；”号结尾。

1）所谓抽象方法，是指在类中仅仅声明了类的行为，并没有真正实现行为的代码。也就是说抽象方法仅仅是为所有的派生子类定义一个统一的接口，方法具体实现的程序代码交给了各个派生子类来完成，不同的子类可以根据自身的情况以不同的程序代码实现。没有方法体。

2）抽象方法只能存在于抽象类中，正像刚才所言，一个类中只要有一个方法是抽象的，则这个类就是抽象的。

3）构造方法、静态（static）方法、最终（final）方法和私有（private）方法不能被声明为抽象的方法！

3）一个抽象类中可以有一个或多个抽象方法，也可以没有抽象方法。如果没有任何抽象方法，这就意味着要避免由这个类直接创建对象。

4）抽象类只能被继承（派生子类）而不能创建具体对象即不能被实例化。

**内部类、匿名类及最终类**

**内部类**

所谓内部类（Inner Class），是指被嵌套定义在另外一个类内甚至是一个方法内的类，因此也把它称之为类中类。嵌套内部类的类称为外部类（Outer Class），内部类通常被看成是外部类的一个成员

**包**

**Java中常用的标准类包**

SUN公司在JDK中提供了各种实用类，通常被称之为标准的API（Application Programming Interface）

1）java.lang

包中存放了Java最基础的核心类，诸如System、Math、String、Integer、Float类等等。在程序中，这些类不需要使用import语句导入即可直接使用。例如前边程序中使用的输出语句System.out.println()、类常数Math.PI、数学开方方法Math.sqrt()、类型转换语句Float.parseFloat()等等。

2）java.awt

包中存放了构建图形化用户界面(GUI)的类。如Frame、Button、TextField等，使用它们可以构建出用户所希望的图形操作界面来。

3）javax.swing

包中提供了更加丰富的、精美的、功能强大的GUI组件，是java.awt功能的扩展，对应提供了如JFrame、JButton、JTextField等等。在前边的例子中我们就使用过JoptionPane类的静态方法进行对话框的操作。它比java.awt相关的组件更灵活、更容易使用。

4）java.applet

包中提供了支持编写、运行applet(小程序)所需要的一些类。

5）java.util

包中提供了一些实用工具类，如定义系统特性、使用与日期日历相关的方法以及分析字符串等等。

6）java.io

包中提供了数据流输入/输出操作的类。如建立磁盘文件、读写磁盘文件等等。

7）java.sql

包中提供了支持使用标准SQL方式访问数据库功能的类。

1. java.net

包中提供与网络通讯相关的类。用于编写网络实用程序

**包(package)的创建及包中类的引用**

**创建包package语句的一般格式如下：**

package 包名;

1）此语句必须放在整个源程序第一条语句的位置（注解行和空行除外）。

2）包名应符合标识符的命名规则，习惯上，包名使用小写字母书写。

**引用类包中的类：**

1. 使用import语句导入类，其应用的一般格式如下：

import 包名.\*; //可以使用包中所有的类

或： import 包名.类名; //只装入包中类名指定的类

在程序中import语句应放在package语句之后，如果没有package语句，则import语句应放在程序开始，一个程序中可以含有多个import语句，即在一个类中，可以根据需要引用多个类包中的类。

2）在程序中直接引用类包中所需要的类。其引用的一般格式是：

包名.类名

**接口**

在Java中可以把接口看作是一种特殊的抽象类，它只包含常量和和抽象方法的定义，而没有变量和方法的实现

**接口的定义**

与类的结构相似，接口也分为接口声明和接口体两部分。定义接口的一般格式如下：

[public] interface 接口名 [extends 父接口名列表] //接口声明

{ //接口体开始

//常量数据成员的声明及定义

**数据类型 常量名=常数值;**

……………

//声明抽象方法

**返回值类型 方法名([参数列表]) [throw 异常列表] ;**

…………………

} //接口体结束

1）接口的访问限定只有public和缺省的。

2）interface是声明接口的关键字，与class类似。

3）接口的命名必须符合标识符的规定，并且接口名必须与文件名相同。

4）允许接口的多重继承，通过“extends 父接口名列表”可以继承多个接口。

5）**对接口体中定义的常量**，系统默认为是“static final”修饰的，不需要指定。

6）**对接口体中声明的方法**，系统默认为是“abstract”的，也不需要指定；对于一些特殊用途的接口，在处理过程中会遇到某些异常，可以在声明方法时加上“throw 异常列表”

**接口的实现**

所谓接口的实现，即是在实现接口的类中重写接口中给出的所有方法，书写方法体代码，完成方法所规定的功能。定义实现接口类的一般格式如下：

[访问限定符] [修饰符] class 类名 [extends 父类名] implements 接口名列表

{ //类体开始标志

[类的成员变量说明] //属性说明

[类的构造方法定义]

[类的成员方法定义] //行为定义

/\*重写接口方法\*/

**接口方法定义**  //实现接口方法

} //类体结束标志

**接口的实例化**

在 Java 中，接口是一个包含抽象方法（没有实现方法）和常量的类型，不能直接实例化。也就是说，你不能直接通过 new 关键字创建接口的实例。

但是，接口可以通过以下方式来实现实例化：定义一个类来实现接口，并通过类的实例来创建对象。子类必须提供接口中所有抽象方法的实现。

interface Animal {

void sound();

}

class Dog implements Animal {

@Override

public void sound() {

System.out.println("Woof!");

}

}

public class Main {

public static void main(String[] args) {

// 通过 Dog 类实例化接口 Animal

**Animal myDog = new Dog()**;

myDog.sound(); // 输出 "Woof!"

}

}

**系统常用类**

**Objcet类:**

在Java中Object类是所有类的父类（直接的或间接的），也就是说Java中所有其他的类都是从Object类派生而来的

主要方法：

1）boolean equals(Object obj) 用来比较两个对象是否相同，相同时返回true，否则返回false。

2）Class getClass() 获取当前对象所属类的信息，返回的是Class对象。

3）String toString() 返回对象本身的相关信息，返回值是字符串。

4）Object clone() 创建且返回一个本对象的复制对象（克隆）。

5）void wait() 该线程等待，直到另一个线程叫醒它。

6）int hashCode() 返回对象的哈希码值。

7）void notify() 叫醒该对象监听器上正在等待的线程。

**System类**

System类是最基础的类，它提供了标准的输入/输出、运行时系统（Runtime）信息。

**一，属性**

System类提供了如下三个属性：

(1) final static PrintStream out 用于标准输出（屏幕）；

(2) final static InputStream in 用于标准输入（键盘）；

(3) final static PrintStream err 用于标准错误输出（屏幕）。

**二，几个常用方法**

(1) static long currentTimeMillis() 用来获取1970年1月1日0时到当前时间的微秒数。

(2) static void exit(int status) 退出当前java程序。status为0时表示正常退出，非0时表示因出现某种形式的错误而退出。

(3) static void gc() 回收无用的内存空间进行重新利用。

(4) static void arraycopy(Object src,int srcPos, Object dest,int destPos, int length) 将数组 src中srcpos位置开始的length个元素复制到dest数组中以destPos位置开始的单元中。

(5) static String setProperty(String key, String value) 设置由key指定的属性值为value。

(6) static String getProperties(String properties) 返回properties属性的值。

**基本数据类型类：Integer类**

**一，Integer类的常用属性**

(1) static int MAX\_VALUE 最大整型常量2147483647。

(2) static int MIN\_VALUE 最小整型常量-2147483648。

(3) static int SIZE 能表示的二进制位数32。

**二，构造器**

(1) Integer(int value) 以整数值构造对象。

(2) Integer(String s) 以数字字符串构造对象。

**三，常用方法**

(1) byte byteValue() 返回整数的字节表示形式。

(2) short shortValue() 返回整数的short表示形式。

(3) int intValue() 返回整数的int表示形式。/写法/i.intValue()

(4) long longValue() 返回整数long的表示形式。

(5) float floatValue() 返回整数float的表示形式。

(6) double doubleValue() 返回整数double的表示形式。

(7) **int compareTo(Integer anotherInteger) 与另一个整数对象相比较，若相等返回0；若大于比较对象，返回1；否则返回-1。**

(8) static Integer decode(String nm) 把字符串nm译码为一个整数。

(9) **static int parseInt(String s) 返回字符串的整数表示形式。**

(10) static int parseInt(String s, int radix) 以radix为基数返回字符串s的整数表示形式。

(11) static String toBinaryString(int i) 返回整数i的二进制字符串表示形式。

(12) static String toHexString(int i) 返回整数i的十六进制字符串表示形式。

(13) static String toOctalString(int i) 返回整数i的八进制字符串表示形式。

(14) static String toString(int i) 返回整数i的字符串表示形式。

(15) static String toString(int i, int radix) 以radix为基数返回i的字符串表示形式。

(16) static Integer valueOf(String s) 返回字符串s的整数对象表示形式。

(17) static Integer valueOf(String s, int radix) 以radix为基数返回字符串s的整数对象表示形式。

(18) static int bitCount(int i) 返回i 的二进制表示中”1”位的个数。

**Math类**

Math类提供了用于数学运算的标准方法及常数

**一，属性**

(1) static final double **E=2.718281828459045**；

(2) static final double **PI=3.141592653589793**；

**二，常用方法**

**(1) static 数据类型 abs(数据类型 a) 求a的绝对值**。其中数据类型可以是int、long、float和double。这是重载方法。

**(2) static 数据类型 max(数据类型 a, 数据类型 b) 求a，b中的最大值**。数据类型如上所述。

**(3) static 数据类型 min(数据类型 a, 数据类型 b) 求a，b中的最小值**。数据类型如上所述。

(4) static double acos(double a) 返回Arccosa的值。

(5) static double asin(double a) 返回Arcsina的值。

(6) static double atan(double a) 返回Arctga的值。

(7) static double cos(double a) 返回cosa的值。

(8) static double exp(double a) 返回ea的值。

(9) static double log(double a) 返回lna的值。

(10) static double pow(double a, double b) 求ab的值。

**(11) static double random() 产生0~1之间的随机值，包括0而不包括1。**

(12) static double rint(double a) 返回靠近a的且等于整数的值，相当于四舍五入去掉小数部分。

(13) static long round(double a) 返回a 靠近long的值.。

(14) static int round(float a) 返回a靠近int的值。

(15) static double sin(double a) 返回sina的值。

(16) static double sqrt(double a) 返回a的平方根。

(17) static double tan(double a) 返回tga的值。

(18) static double toDegrees(double angrad) 将angrad表示的弧度转换为度数。

(19) static double toRadians(double angdeg) 将angdeg表示的度数转换为弧度。

**数组**

**一维数组：相同类型的对象的集合**

和其他变量一样，数组必须先声明定义，而后赋值，最后被引用。

一维数组声明的一般格式如下：

**数据类型 [] 数组名；**

其中：

1）数据类型说明数组元素的类型，可以是Java中任意的数据类型。

2）数组名是一个标识符，应遵照标识符的命名规则。

**一维数组大小的定义及初始化**

一般情况下，使用new运算符定义数组大小，例如下边的程序语句：

int []intArray; //声明一个整型数组

intArray = new int[5]; //定义数组可以存放5个整数元素

//为数组中每个元素赋值

intArray[0]=1; //数组下标从0开始

intArray[1]=2;

intArray[2]=3;

intArray[3]=4;

intArray[4]=5;

通常我们也采用如下方式为数组元素赋初值并由初值的个数确定数组的大小：

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

String []stringArray={"abc", "How", "you"};

**一维数组元素的引用**

数组元素的引用方式为：

**数组名[下标]**

1）下标可以为整型常数或表达式，下标值从0开始。

2）数组是作为对象处理的，它具有长度（length）属性，用于指明数组中包含元素的个数。因此数组的下标从0开始到length-1结束。如果在引用数组元素时，下标超出了此范围，系统将产生数组下标越界的异常（ArrayIndexOutOfBoundsException）。

**二维及多维数组**

在Java语言中，多维数组是建立在一维数组基础之上的，以二维数组为例，可以把二维数组的每一行看作是一个一维数组

二维数组的声明一般格式如下：

**数据类型 [ ][ ] 数组名;**

**二维数组大小的定义及初始化**

**例如有一个二维数组：{{1,2,3},{4,5,6},{7,8,9}}**

1）**先声明而后定义最后再赋值**

int [][]matrix; //声明二维整型数组matrix

matrix = new int[3][3] //定义matrix包含3×3九个元素

matrix[0][0]=1; //为第一个元素赋值

matrix[0][1]=2; //为第二个元素赋值

………………

matrix[2][2]=9; //为第九个元素赋值

2) **直接定义大小而后赋值**

int matrix=new int[3][3];//定义二维整型数组matrix包含3×3九个元素

matrix[0][0]=1; //为第一个元素赋值

………………

matrix[2][2]=9; //为第九个元素赋值

3）**由初始化值的个数确定数组的大小**

int [][]matrix={{1,2,3},{4,5,6},{7,8,9}};//由元素个数确定3行3列

**遍历一个数组的方法：**

1. **普通for**

**一维数组：**

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

// 使用传统的 for 循环

for (int i = 0; i < arr.length; i++) {

System.out.println(arr[i]);

}

**二维数组：**

int[][] arr = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

// 使用传统的 for 循环

for (int i = 0; i < arr.length; i++) { // 外层循环遍历行

for (int j = 0; j < arr[i].length; j++) { // 内层循环遍历列

System.out.print(arr[i][j] + " ");

}

1. **加强for循环**

**一维数组：**

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

// 使用增强 for 循环

for (int element : arr) {

System.out.println(element);

}

**二维数组：**

int[][] arr = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

// 使用增强 for 循环

for (int[] row : arr) { // 遍历每一行

for (int element : row) { // 遍历每一行中的元素

System.out.print(element + " ");

}

**数组（Arrays）类**

1. **static void sort(数据类型 [] d)**

用于对数组d进行排序（升序），数据类型是除boolean之外的任何数据类型。

1. **static void sort(数据类型 [] a, int start,int end)**

对数组a中指定范围从start到end位置之间的数据元素进行排序。

1. **static void fill(数据类型 [] a,数据类型 value)**

设置a数组各个元素的值为value。

1. **static void fill(数据类型 [] a,int start,int end,数据类型 value)**

设置a数组中从start到end位置的元素的值为value。

1. **static int binarySeach(数据类型[] a,数据类型 key)**

利用二进制搜索数组(排过序)内元素值为key的所在位置。

1. **static boolean equals(数据类型[] d1,数据类型[] d2)**

判断d1和d2两数组是否相等。

类方法的一般引用格式如下： **类名.方法名（参量表）**

例如：

Arrays.sort(数组名); //给数组排序

**String类**

1. **构造字符串对象**

**String类生成的对象是不变的**

在前边我们使用字符串时，是直接把字符串常量赋给了字符串对象：

String a = “abcf ergg”;

其实String类提供了如下一些常用的构造函数用来构造字符串对象：

1）String() 构造一个空的字符串对象。

2）String( char chars[ ] ) 以字符数组chars的内容构造一个字符串对象。

3）String( char chars[ ], int startIndex, int numChars )

以字符数组chars中从startIndex位置开始的numChars个字符构造一个字符串对象。

4）String(byte [] bytes) 以字节数组bytes的内容构造一个字符串对象。

5）String(byte[] bytes, int offset, int length)

以字节数组bytes中从offset位置开始的length个字节构造一个字符串对象。

二，String类对象的常用方法

1）**public int length()** 此方法返回字符串的字符个数。

2）**public char charAt(int index)**

此方法返回字符串中index位置上的字符，其中index 值的 范围是0~length-1。

例如：

String str1=new String(“This is a string.”); //定义字符串对象str1

int n=str1.length(); //获取字符串str1的长度n=17

char ch1=str1.charAt(n-2); //获取字符串str1倒数第二个字符,ch1=’g’

3）**public int indexOf(char ch)** 返回字符ch在字符串中第一次出现的位置。

4）**public lastIndexOf(char ch)** 返回字符ch在字符串中最后一次出现的位置。

5）**public int indexOf(String str)** 返回子串str在字符串中第一次出现的位置。

6）**public int lastIndexOf(String str)** 返回子串str在字符串中最后一次出现的位置。

7）**public int indexOf(int ch,int fromIndex)**

返回字符ch在字符串中fromIndex位置以后第一次出现的位置。

8）**public lastIndexOf(in ch ,int fromIndex)**

返回字符ch在字符串中fromIndex位置以后最后一次出现的位置

1. **public int indexOf(String str,int fromIndex)**

返回子串str在字符串中fromIndex位置后第一次出现的位置。

1. **public int lastIndexOf(String str,int fromIndex)**

返回子串str在字符串中fromIndex位置后最后一次出现的位置。

例如：

String str2=new String(“too wonderful for words;most intriguing.”) ;

int n=str2.indexOf(‘o’); // n=1

n=str2.lastIndexOf(‘o’); // n=25

n=str2.indexOf(“wo”); // n=4

n=str2.lastIndexOf(“wo”); // n=18

n=str2.indexOf(‘o’,16); // n=19

n=str2.indexOf(‘r’,21); // n=32

1. **public String substring(int beginIndex)**

返回字符串中从beginIndex位置开始的字符子串。

1. **public String substring(int  beginIndex, int  endIndex)**

返回字符串中从beginIndex位置开始到endIndex位置(不包括该位置)结束的字符子串。例如：

String str3=new String(“it takes time to know a person”);

String str4=str3.substring(16); //str4=” know a person”

String str5=str3.substring(3,8); //str5=”takes”

13）**public String contact(String str)** 用来将当前字符串与给定字符串str连接起来。

14）**public String replace(char oldChar,char newChar)**

用来把串中所有由oldChar指定的字符替换成由newChar指定的字符以生成新串。

15）**public String toLowerCase()** 把串中所有的字符变成小写且返回新串。

16）**public String toUpperCase()** 把串中所有的字符变成大写且返回新串。

17）**public String trim()**  去掉串中前导空格和拖尾空格且返回新串。

18）public String[] split(String regex) 以regex为分隔符来拆分此字符串。

**StringBuffer类**

在字符串处理中，String类生成的对象是不变的，即String中对字符串的运算操作不是在源字符串对象本身上进行的，而是使用源字符串对象的拷贝去生成一个新的字符串对象，其操作的结果不影响源串。

StringBuffer中对字符串的运算操作是在源字符串本身上进行的，运算操作之后源字符串的值发生了变化。StringBuffer类采用缓冲区存放字符串的方式提供了对字符串内容进行动态修改的功能，即可以在字符串中添加、插入和替换字符。StringBuffer类被放置在java.lang类包中。

1. 创建StringBuffer类对象

使用StringBuffer类创建StringBuffer对象，StringBuffer类常用的构造方法如下：

1）StringBuffer() 用于创建一个空的StringBuffer对象；

2）StringBuffer(int length) 以length指定的长度创建StringBuffer对象；

3）StringBuffer(String str) 用指定的字符串初始化创建StringBuffer对象。

注意： 与String类不同，必须使用StringBuffer类的构造函数创建对象，不能直接定义StringBuffer类型的变量。

如：StringBuffer sb = ”This is string object!”; 是不允许的。

必须使用：StringBuffer sb= new StringBuffer(”This is string object!”);

由于StringBuffer对象是可以修改的字符串，所以在创建StringBuffer对象时，并不一定都进行初始化工作。

2. 常用方法

**1) 插入字符串方法insert()**

insert()方法是一个重载方法，用于在字符串缓冲区中指定的位置插入给定的字符串。它有如下形式：

（1）insert(int index, 类型 参量) 可以在字符串缓冲区中index指定的位置处插入各种数据类型的数据（int、double、boolean、char、float、long、String、Object等）。

（2）insert（int index, char [] str, int offset, int len） 可以在字符串缓冲区中index指定的位置处插入字符数组中从下标offset处开始的len个字符。如：

StringBuffer Name=new StringBuffer(“李青青”);

Name.insert(1,”杨”);

System.out.println(Name.toString());//输出：李杨青青

**2）删除字符串方法**

StringBuffer类提供了如下常用的删除方法：

（1）**delete(int start,int end)** 用于删除字符串缓冲区中位置在start~end之间的字符。

（2）**deleteCharAt(int index)** 用于删除字符串缓冲区中index位置处的字符。

如：

StringBuffer Name=new StringBuffer(“李杨青青”);

Name.delete(1,3);

System.out.println(Name.toString());//输出：李青

**3) 字符串添加方法 append()**

append()方法是一个重载方法，用于将一个字符串添加到一个字串缓冲区的后面，如果添加字符串的长度超过字符串缓冲区的容量，则字符串缓冲区将自动扩充。它有如下形式：

（1）append (数据类型 参量名) 可以向字符串缓冲区添加各种数据类型的数据（int、double、boolean、char、float、long、String、Object等）。

（2）append(char[] str,int offset,int len) 将字符数组str中从offset指定的下标位置开始的len个字符添加到字符串缓冲区中。如：

StringBuffer Name=new StringBuffer(“李”);

Name.append(”杨青青”);

System.out.println(Name.toString());//输出：李杨青青

**4) 字符串的替换操作方法 replace()**

replace()方法用于将一个新的字符串去替换字串缓冲区中指定的字符。它的形式如下：

replace(int start,int end,String str) 用字符串str替换字符串缓冲区中从位置start到end之间的字符。如：

StringBuffer Name=new StringBuffer(“李杨青青”);

Name.replace(1,3,“ “);

System.out.println(Name.toString());//输出：李 青

**5）获取字符方法**

StringBuffer提供了如下从字串缓冲区中获取字符的方法：

(1) charAt(int index) 取字符串缓冲区中由index指定位置处的字符；

(2) getChars(int start, int end, char[] dst, int dstStart) 取字符串缓冲区中start~end之间的字符并放到字符数组dst中以dstStart下标开始的数组元素中。

如：

StringBuffer str=new StringBuffer(”三年级一班学生是李军”)

char[] ch =new char[10];

str.getChars(0, 7, ch, 3);

str.getChars(8, 10, ch, 0);

chr[2]=str.charAt(7);

System.out.println(ch); //输出：李军是三年级一班学生

6) 其他几个常用方法

(1) toString() 将字符串缓冲区中的字符转换为字符串。

(2) length() 返回字符串缓冲区中字符的个数。

(3) capacity() 返回字符串缓冲区总的容量。

(4) ensureCapacity(int  minimumCapacity) 设置追加的容量大小。

**(5) reverse() 将字符串缓冲区中的字符串翻转。**如：

StringBuffer str = new StringBuffer("1东2西3南4北5");

str.reverse();

System.out.println(str.toString()); //输出：5北4南3西2东1

(6) lastIndexOf(String  str) 返回指定的字符串str在字符串缓冲区中最右边（最后）出现的位置。

(7) lastIndexOf(String str,int fromIndex) 返回指定的字符串str在字符串缓冲区中由fromIndex指定的位置前最后出现的位置。

(8) substring(int start) 取字串。返回字符串缓冲区中从start位置开始的所有字符。

(9) substring(int start, int end) 取字串。返回字符串缓冲区中从位置start开始到end之前的所有字符。

**Date类**

Date类用来操作系统的日期和时间。

1. 常用的构造器

1）Date() 用系统当前的日期和时间构建对象。

2）Date(long date) 以长整型数date构建对象。date 是从1970年1月1日零时算起所经过的毫秒数。

2. 常用的方法

1）boolean after(Date  when) 测试日期对象是否在when之后。

2）boolean before(Date when) 测试日期对象是否在when之前。

3）int compareTo(Date anotherDate) 日期对象与anotherDate比较，如果相等返回0值；如果日期对象在anotherDate之后返回1，否则在anotherDate之前返回-1。

4）long getTime() 返回自1970.1.1 00:00:00以来经过的时间（毫秒数）。

5）void setTime(long time) 以time(毫秒数)设置时间。