[

Java千百问_03基本语法（003）_public、privite、protected有什么区别

,

点击进入_更多_Java千百问

public、privite、protected有什么区别

首先，public、privite、protected都是Java的修饰符，我们先看看Java修饰符是什么

1、java修饰符是什么

Java修饰符是用来修饰Java中的标识符（包括变量名、方法名、类名、包名和参数名等等）的，用来改变它们的含义的关键词。 Java语言有各种各样修饰符，大体分为两类：访问控制修饰符、非访问控制修饰符。如图：

```
public class className {
 private boolean myFlag;
 static final double weeks = 9.5;
 protected static final int BOXWIDTH = 42;
 public static void main(String[] arguments) {

  }
}
```

2、public、privite、protected的区别是什么

public、private、protected都属于访问控制修饰符，用来设置类，变量，方法和构造函数的访问级别，如果不加任何访问控制修饰符，则为default。

public：

公共的。Java语言中访问限制最宽的修饰符，被其修饰的类、属性以及方法不仅可以跨类访问，而且允许跨包（package）访问。

private：

私有的。Java语言中对访问权限限制的最窄的修饰符，被其修饰的类、属性以及方法只能被该类的对象访问，其子类不能访问，更不能允许跨包访问。

protected：

被保护的。介于public和private之间的一种访问修饰符。被其修饰的类、属性以及方法只能被类本身的方法及子类访问，即使子类在不同的包中也可以访问。

default：

默认的。即不加任何访问修饰符。该模式下，只允许在同一个包中进行访问。

3、非访问控制修饰符有哪些

出了访问控制符之外的，都归类为非访问控制修饰符，来实现很多其他的功能。

static：

用来修饰静态类、方法和变量。

final：

用来修饰不会被修改的方法和变量。

abstract：

用来修饰抽象类和方法。

synchronized：

表明一个方法或一段代码需要同步执行。

volatile：

修饰被不同线程访问和修改的变量，都会直接读取原值，而不会因编译器的优化而读取备份。

点击进入ooppookid的博客

]

[

Java千百问_03基本语法（004）_java中的运算符都有哪些

,

点击进入_更多_Java千百问

java中的运算符都有哪些

Java提供了丰富的运算符来操纵变量。如果不知道什么是变量，看这里：局部变量、类变量、实例变量有什么区别

我们可以把所有的Java操作符为以下几组（除位运算和其它运算符之外，其他几种操作无几乎是java中使用频率最高的语法）：

算术运算符、关系运算符、逻辑运算符、赋值运算符、位运算符、其它运算符

下面来仔细说明。运算符的优先级看这里：java运算符的优先级是怎样的

1、算术运算符

算术运算符用于在数学表达式中，他们是在代数中使用的方法相同。假设整型变量a=20，b=10，则：

运算符	描述	实例
+	加法，两侧值相加	a+b=30
-	减法，左侧值减去右侧值	a-b=10
*	乘法，两侧值相乘	a*b=200
/	除法，左侧值除以右侧值	a/b=2
%	取模，左侧值除以右侧值所得的余数	a%b=0
++	自增，值加1	a++
--	自减，值减1	B--

2、关系运算符

关系运算符用来比较操作数，假设变量a=20，b=10，则：

运算符	描述	实例
==	检查两个操作数是相等的，如果是，则条件为true	a==b，false
!=	检查两个操作数是相等的，如果不是，则条件为true	a!=b，true
>	检查左侧操作数是否大于右侧操作数，如果是，则条件为true	a>b，true
<	检查左侧操作数是否小于右侧操作数，如果是，则条件为true	a<b，false
>=	检查左侧操作数是否大于或等于右侧操作数，如果是，则条件为true	a>=b，true
<=	检查左侧操作数是否小于或等于右侧操作数，如果是，则条件为true	a<=b，false

3、逻辑运算符

逻辑运算符用来描述与、或、非逻辑关系，假设变量a=true，b=false，则：

运算符	描述	实例
&&	逻辑与，如果两个布尔值都是true或非0，则为true，反之为false	a&&b，false
||	逻辑或，如果两个布尔值任意一个是true或非0，则为true，反之为false	a||b，true
!	逻辑非，如果布尔值是false，则为true，反之为false	!(a&&b)，true

4、位运算符

位运算符可以应用到整数类型，长型，整型，短整型，字符和字节。它作用于位，并执行逐位操作。二进制位操作具体看：二进制是怎样做位运算的

假设整型变量A=60（0011 1100）和变量B=13（0000 1101），则：

运算符	描述	实例
&	与AND，两侧值进行二进制与操作	a&b，12，0000 1100
|	或OR，两侧值进行二进制或操作	A|b，61，0011 1101
^	异或XOR，两侧值进行二进制异或操作	A^b，49，0011 0001
~	非，值进行二进制非操作	~a，-61，1100 0011
<<	左移，对左侧值进行二进制左移操作，右侧值为位移位数	a<<，240，1111 0000
>>	右移，对左侧值进行二进制右移操作，右侧值为位移位数	a>>，15，0000 1111
>>>	无符号右移，对左侧值进行二进制无符号右移操作，右侧值为位移位数	a>>>，15，0000 1111

5、赋值运算符

赋值运算符是为变量赋值所使用，如下：

运算符	描述	实例
=	将右侧值赋值给左侧变量	c=a+b
+=	将左侧值与右侧值之和赋值给左侧变量	c+=a等价于c=c+a
-=	将左侧值与右侧值之差赋值给左侧变量	c-=a等价于c=c-a
*=	将左侧值与右侧值之积赋值给左侧变量	c*=a等价于c=c*a
/=	将左侧值与右侧值之商赋值给左侧变量	c/=a等价于c=c/a
%=	将左侧值与右侧值的余数赋值给左侧变量	c%=a等价于c=c%a
<<=	将左侧值左移赋值给左侧变量，位移位数位右侧值	c<<=a等价于c=c<<a
>>=	将左侧值右移赋值给左侧变量，位移位数位右侧值	c>>=a等价于c=c>>a
&=	将左侧值与右侧值做与操作后赋值给左侧变量	c&=a等价于c=c&a
|=	将左侧值与右侧值做或操作后赋值给左侧变量	c|=a等价于c=c|a
^=	将左侧值与右侧值做异或操作后赋值给左侧变量	c^=a等价于c=c^a

6、其它运算符_条件运算符

条件运算符也被称为三元运算符，可以作为赋值运算符种很特殊的一种，此运算符是确定哪些值应分配给变量。语法：

variable x = (expression) ? value if true : value if false

"?"号左侧为条件表达式ture或false，如果true则将":"左侧值赋值给"="左侧的变量；如果false则将":"右侧值赋值给"="左侧的变量。

实例：

```
public class Test {

   public static void main(String args[]){
      int a , b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );//结果：Value of b is : 30

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );//结果：Value of b is : 20
   }
}
```

6、其它运算符_instanceof运算符

instanceof运算符只用于对象引用变量，检查对象是否为特定类型（类或接口类型）。语法：

(Object reference variable) instanceof (class/interface type)

如果左侧值的类型与右侧的类/接口类型（包含父类）一致，则结果为 true。

实例1：

```
public class Test {

public static void main(String args[]){
String name = "James";
boolean result = name instanceof String;
System.out.println(result);//结果：true
}
}
```

实例2：

```
class Vehicle {}

public class Car extends Vehicle {
public static void main(String args[]){
Vehicle a = new Car();
boolean result = a instanceof Car;
System.out.println(result);//结果：true
}
}
```

点击进入ooppookid的博客

]

[

Java千百问_03基本语法（005）_二进制是怎样做位运算的

,

点击进入_更多_Java千百问

二进制是怎样做位运算的

程序中的所有数在计算机内存中都是以二进制的形式储存的。位运算说白了，就是直接对整数在内存中的二进制位进行操作。其他运算符看这里：java种的运算符都有哪些

大部分运算流程都是先将整数转换为二进制，然后进行相应二进制操作。常见的操作有如下几种：

运算	符号（java）	描述
按位与	&	与AND，两侧值进行二进制与操作
按位或	|	或OR，两侧值进行二进制或操作
按位异或	^	异或XOR，两侧值进行二进制异或操作
按位取反	~	非，值进行二进制非操作
左移	<<	左移，对左侧值进行二进制左移操作，右侧值为位移位数
带符号右移	>>	右移，对左侧值进行二进制右移操作，右侧值为位移位数
无符号右移	>>>	无符号右移，对左侧值进行二进制无符号右移操作，右侧值为位移位数

下面我们详细说明，运算符的优先级看这里：java运算符的优先级是怎样的

1、按位与 and

两个二进制数进行按位与操作：相同位的两个数字都为1，则为1；若有一个不为1，则为0。

例如：00101 & 11100 = 00100

通常用于二进制取位操作，例如一个数 & 1的结果就是取二进制的最末位。

这可以用来判断一个整数的奇偶，二进制的最末位为0表示该数为偶数，最末位为1表示该数为奇数。

2、按位或 or

两个二进制数进行按位或操作：相同位只要一个为1即为1。

例如：00101 | 11100 = 11101

通常用于二进制特定位上的无条件赋值，例如一个数 | 1的结果就是把二进制最末位强行变成1。

如果需要把二进制最末位变成0，对这个数 | 1之后再减一就可以了，其实际意义就是把这个数强行变成最接近的偶数。

3、按位异或 xor

两个二进制数进行按位异或操作：相同位不同则为1，相同则为0。

例如：00101 ^ 11100 = 11001

异或运算可以用于简单的加密，是因为xor运算的逆运算是它本身，也就是说两次异或同一个数最后结果不变，即（a xor b) xor b = a。

比如我想对某人说1314520，但怕别人知道，于是双方约定拿生日 19871112作为密钥。1314520 xor 19871112 = 20659024，我就把20659024告诉MM。MM再次计算20659024 xor 19871112的值，得到1314520。

4、按位取反 not

对一个二进制数进行按位取反操作：每一位的0变为1，1变为0。

例如：~0011 1100 = 1100 0011

使用not运算时要格外小心，你需要注意整数类型有没有符号。如果not的对象是无符号整数（不能表示负数），那么得到的值就是它与该类型上界的差，因为无符号类型的数是用00到$FFFF依次表示的。

5、左移

对一个二进制数进行左移操作：a << b就表示把a转为二进制后左移b位（在后面添b个0）。左移是不区分有无符号的。

例如: 1100100 << 2 = 110010000，即100 << 2 = 400。

可以看出，a << b的值实际上就是a乘以2的b次方，因为在二进制数后添一个0就相当于该数乘以2。

通常认为a << 1比a * 2更快，因为前者是更底层一些的操作。因此程序中乘以2或者2的倍数的操作请尽量用左移一位来代替。

定义一些常量可能会用到shl运算。你可以方便地用1 << 16 - 1来表示65535。

6. 带符号右移

对一个二进制数进行无符号右移操作：a >> b就表示把a转为二进制后右移b位，符号位保持不变。

例如: 1100100 >> 2 = 11001，即100 >> 2 = 25。

和<<相似，相当于a除以2的b次方（取整）。我们也经常用<< 1来代替除以 2，比如二分查找、堆的插入操作等等。

想办法用<<代替除法运算可以使程序效率大大提高。最大公约数的二进制算法用除以2操作来代替慢得出奇的mod运算，效率可以提高60%。

7.无符号右移

对一个二进制数进行带符号右移操作：a >> b就表示把a转为二进制后右移b位，符号位也要参与移动，移动后高位补0。

只是对32位和64位的值有意义。

[点击进入ooppookid的博客]

]

[

Java千百问_03基本语法（006）_java运算符的优先级是怎样的

,

点击进入_更多_Java千百问

java运算符的优先级是怎样的

运算符优先级决定运算的顺序，这会影响一个表达式如何计算。什么是运算符看这里：java中的运算符都有哪些

例如x= 7+3* 2；这里x被赋值13，而不是20，因为运算符*的优先级高于+，所以它首先被乘以3 * 2，然后加7。

这里整理了运算符的优先级，优先级高的在列表顶部，低的在底部。在运算表达式中，优先级较高的运算符将首先计算。

分类	运算符	同级执行顺序
分组	()、[]、.	从左至右
单值运算	++、--、!、~	从右至左
乘除	*、/、%	从左至右
加减	+、-	从左至右
位移	>>、<<、>>>	从左至右
非相等关系	>、>=、<、<=	从左至右
相等关系	== !=	从左至右
二进制与	&	从左至右
二进制异或	^	从左至右
二进制或	|	从左至右
逻辑与	&&	从左至右
逻辑或	||	从左至右
三元运算	?:	从右至左
赋值	=、+=、-=、*=、/=、%=、>>=、<<=、&=、^=、|=	从右至左
分割	,	从左至右

点击进入ooppookid的博客

]

[

Java千百问_03基本语法（007）_if else如何使用

,

点击进入_更多_Java千百问

if else如何使用

java中if else语句，是用来做逻辑判断的语法（另一种逻辑判断语句switch看这里：switch如何使用）。使用方式非常简单，可以用if做单独判断，可以用if...else if...else做多逻辑判断，还可以嵌套使用。可以说是java中使用最为广泛的语句。下面来说明这个语句具体如何使用。

1、if语句如何单独使用：

首先，if语句的语法为：

```
if(Boolean flag)
{
//如果flag为true时，进入这里，执行{}包起来的代码段；如果为false，则直接跳过，不执行该段。

}
```

这里要注意的是，如果{}包起来的代码只有一句，则可以省略{}，但是这种写法一般不建议写。
实例：

```
public class Test {

public static void main(String args[]){
int x = 10;

if(true){
System.out.print("==1==");
}

  if( x >= 20 )
System.out.print(" ==2==");

System.out.print(" ==3==");

  if( x < 20 )
System.out.print(" ==4==");
}
}
```

将会产生以下结果：
==1==
==3==
==4==

2、if...else if...else 语句：

if语句后面可以跟一个可选的else if...else语句。

else if...可以添加多个，用来判断不同的逻辑；else...一组if条件中只能有一个，没有被前面的条件匹配到，则会执行else里的代码。当然直接if....else....也是可以的。

它的语法是:

```
if(Boolean flag1){
//如果flag1为true时，进入这里，执行{}包起来的代码段；如果为false，则继续判断下面的else if条件。
}
else if(Boolean flag2){
 //如果flag2为true时，进入这里（当然前提是flag1为false），执行{}包起来的代码段；如果为false，则继续判断下面的else if条件。
}
else if(Boolean flag3){
 //同上一个else if，前提是flag1、flag2都为false。
}
...
else{
//如果flag1、flag2、flag3...都为false，则执行这里。

}
```

实例:

```
public class Test {

public static void main(String args[]){
int x = 30;

if( x == 10 ){
System.out.print("==1==");
}
}else if( x == 20 ){
System.out.print(" ==2==");
}else if( x == 30 ){
System.out.print(" ==3==");
 else{
System.out.print(" ==4==");
}
}
}
```

这将产生以下结果：

==3==

3、嵌套语句：

你可以在一个if或else if语句中使用另一个if或else if语句。

语法:

嵌套if...else的语法如下：

```
if(Boolean_expression 1){
//Executes when the Boolean expression 1 is true
if(Boolean_expression 2){
//Executes when the Boolean expression 2 is true
}
}
```

可以嵌套else if...else在类似的方式，因为我们有嵌套的if语句。

实例:

```
public class Test {

public static void main(String args[]){
int x = 30;
int y = 10;

if( x == 30 ){
if( y == 10 ){
System.out.print("X = 30 and Y = 10");
}
}
}
}
```

这将产生以下结果：

X = 30 and Y = 10

点击进入ooppookid的博客

]

[

Java千百问_03基本语法（008）_switch如何使用

,

点击进入_更多_Java千百问

switch如何使用

switch允比对一个变量的值，来执行不同情况的代码。

switch语句和if else语句类似，switch能够实现的功能if else完全可以实现，区别在于使用switch逻辑更为清晰。if else详情看这里：if else如何使用

语法:

```
switch(expression){
case value :
//Statements
break; //optional
case value :
//Statements
break; //optional
//You can have any number of case statements.
default : //Optional
//Statements
}
```

以下规则适用于switch语句：

1、在switch语句中使用的expression只能是一个字节，short，int和或char，enum本身为int，所以也可以使用（jdk1.7以后可以使用String）。

2、switch可以有任何数量的case语句。每个case后面是进行比较的值和":"。

3、case后面的值与expression类型必须相同，必须是一个常量。

4、当case后面的值与expression等于，则执行case对应的代码段，直到break语句为止。

5、直行到break语句，直接跳出switch，执行后续代码。

6、不是每一个case都要有break。如果没有break，则会继续执行下一个case对应的代码段，直到break为止。break关键字详解看这里：break与contine分别如何使用

7、switch语句可以有一个默认case，它必须出现在所有case之后。默认情况下，没有case是true时，执行default对应代码段。

例子：

```
public class Test {

public static void main(String args[]){
//char grade = args[0].charAt(0);
char grade = 'B';

switch(grade)
{
case 'A' :
System.out.println("Excellent!");
break;
case 'B' :
case 'C' :
```

```
System.out.println("Well done");
break;
case 'D' :
System.out.println("You passed");
case 'F' :
System.out.println("Better try again");
break;
default :
System.out.println("Invalid grade");
}
System.out.println("Your grade is " + grade);
}
}
```

编译并运行上面使用各种命令行参数的程序。这将产生以下结果：

Well done
Your grade is a B

]

[

Java千百问_03基本语法（009）_java中如何循环执行

,

点击进入_更多_Java千百问

java中如何循环执行

首先，我们看看循环是什么

1、循环是什么

当我们需要多次执行同样的代码段，通常被称为一个循环。伴随循环经常出现的关键字：break与contine分别如何使用

Java有非常灵活的三种循环机制：

while 循环

do...while 循环

for 循环

2、什么是while循环

while循环可以按照特定的次数重复执行任务。

语法:

```
while(Boolean flag)
{
//代码段
}
```

在执行时，如果flag的结果为true，则循环中的代码段将被执行。直到flag的结果为false，循环执行停止，继续执行循环代码的后续代码。

要注意，while循环的关键点是循环可能永远不会运行。当flag结果为 false，循环体将被跳过，在while循环之后的第一个语句将被执行。

例子：

```
public class Test {

public static void main(String args[]) {
int x = 10;

while( x < 15 ) {
System.out.println("value of x : " + x );
x++;
}
}

}
```

这将产生以下结果:

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

3、什么是do...while 循环

do ... while循环类似于while循环，不同的是一个do ... while循环是保证至少执行一次。

语法：

```
do
{
//Statements
}while(Boolean flag);
```

循环方式与while循环大体一致。不同的是，flag表达式出现在循环的结尾，在循环中的语句执行后才会判断flag是否为ture，所以代码段至少会执行一次。

例子：

```
public class Test {

public static void main(String args[]){
int x = 10;

do{
System.out.println("value of x : " + x );
x++;
}while( x < 8 );
}
}
```

这将产生以下结果:

value of x : 10

4、什么是for循环

for循环可以可以指定执行次数，控制任务执行次数是方便的一件事（当然while和do while也可以实现）。

语法：

```
for(initialization; Boolean flag; update)
{
//Statements
}
```

执行过程：

1、initialization首先被执行，并且仅执行一次。这个步骤可声明和初始化任何控制循环的变量。如不需要声明，则用写一个";"即可。

2、判断flag值。如果是true，则执行循环体。如果是false，则循环体不执行，跳出循环继续执行后续代码。

3、若flag为true，执行循环体后，会执行update语句，该语句允许变更任何循环变量。这个语句可以为空， 写一个";"即可。

4、在执行完update语句后，继续第二步操作，产生循环。直到flag为false，则循环终止。

例子：

```
public class Test {

public static void main(String args[]) {

for(int x = 10; x < 15; x = x+1) {
System.out.println("value of x : " + x );
```

```
}
}
}
```

这将产生以下结果:

value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14

5、什么是加强版for循环

Java5之后才有该语法，用来遍历集合体使用的循环。
语法：

```
for(declaration : expression)
{
//Statements
}
```

expression为一个可以遍历的集合体，declaration为每次遍历的集合中的值。集合全部遍历完成，则跳出循环。理论上循环次数与集合的size一致。
例子：

```
public class Test {

public static void main(String args[]){
int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ){
System.out.print( x );
System.out.print(",");
}
System.out.print("
");
String [] names ={"James", "Larry", "Tom", "Lacy"};
for( String name : names ) {
System.out.print( name );
System.out.print(",");
}
}
}
```

这将产生以下结果:

10,20,30,40,50,
James,Larry,Tom,Lacy,

]

[

Java千百问_03基本语法（010）_break与contine分别如何使用

,
点击进入_更多_Java千百问

break与contine分别如何使用

break，contine都是使用在循环体中的语句，都有终止执行的作用，具体不同看下面详解。循环语句看这里：

1、break关键字是如何使用的：

break是用来停止整个循环。必须在循环体中或switch语句中（switch语法看这里：switch如何使用）。它将停止本层循环的执行（多层循环只会停止break这一层，跳出后继续父级循环），并开始执行后续的代码。

语法
break语法是任何循环中一个单独的语句：

```
break;
```

例子：

```
public class Test {

public static void main(String args[]) {
int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) {
if( x == 30 ) {
break;
}
System.out.println(x);
}
}
}
```

这将产生以下结果:
10
20

2、continue关键字是如何使用的：

continue是用来停止循环中的本次代码，但循环仍然继续。它使循环立即跳转到下一次迭代，继续循环。

在for循环中，continue会立即执行更新语句。在一个while循环或do/while循环中，会立即判断flag值。java循环语法看这里：java中如何循环执行
语法
continue 语法是任何循环中一个单独的语句：

```
continue;
```

例子：

```
public class Test {

public static void main(String args[]) {
int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) {
if( x == 30 ) {
continue;
}
System.out.println(x);
}
}
}
```

这将产生以下结果:

```
10
20
40
50
```

]

[

Java千百问_03基本语法（011）_final,finally,finalize有什么区别

,

点击进入_更多_Java千百问

1、final,finally,finalize有什么区别

final,finally是java的关键字，finalize是jdk的一个方法名，它们虽说字面意思类似，但是他们的使用却完全不同。具体如下：

final

final是一个修饰符关键字。
如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承。
如果一个变量或方法声明为final，可以保证它们在使用中不被改变。被声明为final的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为final的方法也同样只能使用，不能重载。
因此一个类不能既被声明为abstract的，又被声明为final的。

finally

finally是一个异常相关的关键字，用于try后面，finally块中的代码总是执行，不论是否发生异常（即无论是否执行catch块中代码）。一般用于清理工作、关闭链接等类型的语句。

了解finally关键字如何使用看这里：finally关键字如何使用
finalize

finalize是一个jdk方法名，它是Object类的一个protected方法，在垃圾收集器将对象从内存中清除出去之前会被调用，默认没有任何操作，源代码如下：

```
protected void finalize() throws Throwable { }
```

当垃圾收集器在确定这个对象没有被引用时，会调用finalize()，任何一个Object的子类都可以去覆盖这个方法。

了解protected关键字看这里：public、privite、protected有什么区别
了解方法覆盖看这里：java中覆盖是什么

]

[

Java千百问_03基础语法（012）_transient关键字有什么用

,

点击进入_更多_Java千百问

1、transient关键字有什么用

transient是java语言的关键字，是变量修饰符。

如果用transient声明一个实例变量，当对象存储时，它的值不需要维持。

Java的serialization（序列化）提供了一种持久化对象实例的机制，当持久化对象时，可能有一个特殊的对象数据成员，我们不想用serialization机制来保存它。为了在一个特定对象的一个域上关闭serialization，可以在这个域前加上关键字transient。

了解序列化看这里：[Serializable接口有什么用][2]
[2]:

当一个对象被序列化的时候，transient型变量的值不包括在序列化的表示中，然而非transient型的变量是被包括进去的。

]

[

Java千百问_03基础语法（013）_>、>>、>>>有什么区别

,

点击进入_更多_Java千百问

1、>、>>、>>>有什么区别

了解java运算符看这里：java种的运算符都有哪些
了解java运算符优先级看这里：java运算符的优先级是怎样的
“>”属于关系运算符，而“>>”、”>>>”则按位运算符，看一下它们各自的含义：

1. 运算符>
   关系运算符，表示大于。
   如：if(a>b)…结果是boolean类型。

2. 运算符>>
   按位运算符，表示右移。<<表示左移。
   j<< i。相当于num除以2的n次幂，j/(int) (Math.pow(2, i))。
   如：int i=15; i>>2的结果是3，移出的部分将被抛弃。
   转为二进制的形式可能更好理解，0 0000 1111(15)右移2位的结果是0 0000 0011(3)。
   了解二进制运算看这里：二进制是怎样做位运算的

3. 运算符>>>
   按位运算符，表示无符号右移。
   与>>类似，但移动时忽略符号位，空位都以0补齐。
   无符号右移运算符>>>只是对32位和64位的值有意义（以及负值），其余情况与>>一致。

实例：

```
System.out.println("1、以下测试>:");
int a = 15, b = 2, c = -15;
System.out.println(a > b);
System.out.println("\n2、以下测试>>:");
System.out.println(a + " / (int) (Math.pow(2, " + b + "))) = "
              + (a / (int) (Math.pow(2, b))));
System.out.println(a + " >> " + b + " = " + (a >> b));
System.out.println(c + " >> " + b + " = " + (c >> b));
System.out.println("\n3、以下测试>>>:");
System.out.println(a + " >>> " + b + " = " + (a >>> b));
System.out.println(c + " >>> " + b + " = " + (c >>> b));
```

执行结果如下：

```
1、以下测试>:
true
2、以下测试>>:
15 / (int) (Math.pow(2, 2))) = 3
15 >> 2 = 3
```

-15 >> 2 = -4
3、以下测试>>>:
15 >>> 2 = 3
-15 >>> 2 = 1073741820

]

[

Java千百问_03基础语法（014）_volatile关键字有什么用

,

点击进入_更多_Java千百问

1、volatile关键字有什么用

volatile是java语言的关键字，是变量修饰符。它是被设计用来修饰被不同线程访问和修改的变量。

volatile的作用是： 作为指令关键字，确保本条指令不会被编译器优化，且应用的所有线程读取这个变量的值是一致的。

简单的说，就是禁止编译器对代码进行优化，且强迫所有线程从共享内存读取该变量（而不是读取寄存器中的备份），变量发生改变时强行存入共享内存。

java内存模型需要具有以下规则：原子性（Atomicity）、可见性（Visibility）、可排序性（Ordering）。
用volatile修饰的变量，就会具有可见性，且不允许线程内部缓存和重排序，但是它不能使变量具有原子性。

了解java内存模型看这里：java内存模型是什么样的
在目前多线程频繁使用的年代，并不建议使用这种可靠性低、且对开发者或者场景要求高的方式来完成多线程的操作，由于volatile很容易被误用于进行原子性操作，如果使用不当则会错误频出。

]

[

Java千百问_03基础语法（015）_System.exit(0)有什么用

,

点击进入_更多_Java千百问

1、System.exit(0)有什么用

查看java.lang.System的源代码，我们可以找到System.exit(status)这个方法，源代码如下：

```
public final class System {
/**
     * Terminates the currently running Java Virtual Machine. The
     * argument serves as a status code; by convention, a nonzero status
     * code indicates abnormal termination.
     * <p>
     * This method calls the <code>exit</code> method in class
     * <code>Runtime</code>. This method never returns normally.
     * <p>
     * The call <code>System.exit(n)</code> is effectively equivalent to
     * the call:
     * <blockquote><pre>
     * Runtime.getRuntime().exit(n)
     * </pre></blockquote>
     *
     * @param      status  exit status.
     * @throws  SecurityException
     *        if a security manager exists and its <code>checkExit</code>
     *        method doesn't allow exit with the specified status.
     * @see        java.lang.Runtime#exit(int)
     */
    public static void exit(int status) {
        Runtime.getRuntime().exit(status);
    }
}
```

注释中说的很清楚，这个方法是用来结束当前正在运行中的java虚拟机。参数status作为状态码，按照惯例，一个非零状态码表示异常，所以说status如果是非零的，那么表示是非正常退出。调用System.exit(n)实际上相当于调用:

```
Runtime.getRuntime().exit(n);
```

对于System.exit(0)来说，有以下几点需要注意：

1. System.exit(0)是将整个虚拟机都停掉了，也就是说连内存也都被释放了。而dispose()只是关闭这个窗口，但是并没有停止整个应用。
2. System.exit(0)是正常退出程序，而System.exit(1)或者说status非0表示非正常退出程序。
3. 不管status为何值都会退出程序。

]

[

Java千百问_03基础语法（016）_main方法是什么

,

点击进入_更多_Java千百问

1、main方法是什么

某个类中有main()方法，说明这是一个java应用程序，可以直接启动运行的程序（操作系统中安装了jdk或者jre）。任何一个非抽象类/接口都可以添加main()方法。

了解抽象类、接口看这里：接口和抽象类有什么区别
在java的规范中，main()方法的声明为：

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

}
```

当通过java运行工具运行某个类时：java 类名，jre会运行类中的main()方法。在运行这个Java应用程序的时候，首先会调用main方法，由于main方法是public static，所以调用时不实例化这个类的对象，而是通过类名直接调用。

了解public、private看这里：public、privite、protected有什么区别
对于main有以下几点需要注意：

1. 对于java中的main方法，由于规范中main的返回值类型为void，所以main方法不能有返回值。
2. main方法的输入参数，类型为String[]，规范中main()方法中必须有一个入参String[]，当然参数的名字是可以自己设定的。根据习惯，这个字符串数组的名字一般和规范范例中mian参数名保持一致，取名为args。
3. main(String[] args)方法的参数args可以在运行时指定，例如：java TestMain 1 2 3，则args为[1,2,3]。
4. main方法中可以通过throws Exception声明抛出异常，如果发生异常，则会直接抛在运行工具中（运行工具一般会将错误日志打印到console）。
5. 带有main方法的类同普通类一样，执行main之前也会先执行类的静态代码块static{}。

结合以上几点的一个例子：

```
public class TestMain {

    static {
        System.out.println("Hello Wordld");
    }

    public static void main(String[] args) throws Exception {
        if (args.length > 0) {
            for (String arg : args) {
                System.out.println("args:" + arg);
            }
        }
        if (args.length <= 0) {
            throw new Exception("Exception");
        }
    }
}
```

执行“java TestMain”结果如下：

Hello Wordld
Exception in thread “main” java.lang.Exception: Exception

at com.test.TestMain.main(TestMain.java:16)

执行“java TestMain 1 2 3”结果如下：

Hello Wordld
args:1
args:2
args:3

]

[

Java千百问_03基础语法（017）_static有什么用

,

点击进入_更多_Java千百问

1、static有什么用

static是java的关键字，用static声明达到静态的目的。所谓静态，就是在程序编译后就能被使用，不需要创建任何实例。
static能够修饰类、方法、变量以及类代码块，具体如下：

static方法

被static声明的方法叫做静态方法，不需要实例化对象通过类名直接调用的方法，最常见的是main()。

了解main方法看这里：main方法是什么
例如：

```
public class TestStatic {

    public static void main(String[] args) {
        TestStatic.testStatic();
    }

    public static void testStatic() {
        System.out.println("my testStatic");
    }
}
```

运行结果如下：

my testStatic

static属性

被static声明的属性叫做静态属性（类变量），不需要实例化对象通过类名直接使用。

了解实例变量、类变量看这里：局部变量、类变量、实例变量有什么区别
例如：

```
public class TestStatic {

    public static void main(String[] args) {
        TestStatic.staticString = "staticString";
        System.out.println(TestStatic.staticString);
    }

    public static String staticString;
}
```

运行结果如下：

staticString

static代码块

被static修饰的代码块在类加载时就会执行，例如：

```
public class TestStatic {

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

```
    static {
        System.out.println("my static code");
    }
}
```

运行结果如下：

my static code

staitc类

通常情况下，是不可以用static修饰类的。如果一定要用static修饰类的话，只能修饰内部类。如果想在内部类中使用statitc属性或方法，需要使用static内部类（普通内部类只能使用static final），例如：

```
public class TestStatic {

    static class StaticA {

        private static String str = "my str";

        public static String getStr() {
            return str;
        }
    }

    public static void main(String[] args) {
        System.out.println(TestStatic.StaticA.getStr());
    }
}
```

运行结果如下：

my str

]

[

Java千百问_03基础语法（018）_注释是什么

,

点击进入_更多_Java千百问

1、注释是什么

java中的注释和其他编程语言中的注释一样，注释的内容不会被编译运行，只是源代码中对代码的解释说明。
通过添加代码注释可以提高源代码的可读性，使得Java程序条理清晰，易于区分代码行与注释行。另外，通常还会在类、方法或者代码段开头加入作者、添加/修改时间、程序版本以及代码功能等内容注释，方便后来的维护以及程序员的交流。（当然，对于好的代码，不写注释也能够有很高的可读性）

2、注释如何使用

对于Java注释，我们可以添加在代码中任何地方，主要语法有如下三种：

1. //
   注释一行，一般用来解释说明某个变量或者某行代码的含义。

2. /* ..... */
   注释若干行，注释多行写法如下：

```
/* ......
 * ......
 * ......
 */
```

一般用来描述某段代码的编写思路、执行过程或者注意事项。

1. /* * ..... */
   文档注释，这种写法同多行注释，不同的是，它可以通过jdk提供的javadoc工具生成代码文档（html等格式），方便形成开发文档。它不但可以多行注释，还可以添加一些标签，供生成文档使用，如下：

```
/**
* .........
* .........
* @author sunjie at 2016年6月18日
* @version 1.0.0
* @param name
*  名称
* @param sex
*  性别
* @return boolean
*  是否成功
*/
```

这里要注意的是javadoc只提取/* * ..... */这种类型的注释。
了解javadoc看这里：[javadoc是什么][2]

]

[

Java千百问_03基础语法（019）_注解是什么

,

点击进入_更多_Java千百问

1、注解是什么

java中的注解（Annotation），也叫元数据。是Java 5以后版本引入的一个特性。

注解与类、接口、枚举是在同一个层次，可以用来标注包、类、字段、方法、局部变量、方法参数等元素，达到对这些元素的描述和说明。

注解是可以允许jvm在运行中读取它，这一点与注释完全不同。并且包含多种加载策略，可以灵活配置。

了解注解、注释区别看这里：注解、注释有什么区别
如何自定义注解看这里：如何使用注解

2、注解有哪些加载策略

注解包含3中可配置的加载策略（RetentionPolicy），根据不同的需要进行不同的配置，具体如下：

```
public enum RetentionPolicy {
    // 此类型会被编译器丢弃
    SOURCE,
    // 此类型注解会保留在class文件中，但JVM会忽略它，默认策略
    CLASS,
    // 此类型注解会保留在class文件中，JVM会读取它
    RUNTIME
}
```

3、注解有什么作用

注解主要功能有以下几点：

1. 编写文档
   通过代码里标识的元数据生成文档，这一点与注释类似。

2. 代码分析
   通过代码里标识的元数据对代码进行分析，一般使用反射获取注解信息。

3. 编译检查
   通过代码里标识的元数据让编译器能够实现基本的编译检查，例如方法覆盖@Override。

4、JDK有那些内置注解

jdk提供了若干内置注解，常见的如下：

@Override

它用来对覆盖父类方法、实现接口方法进行标记，如果被标记的方法并没有实际覆盖父类方法，则编译器会发出错误警告。
例子：

```
public class SuperTest {
    public String tostring() {
        return "父类";
    }
}
```

```
public class Test extends SuperTest {
    @Override
    public String tostring() {
        return "子类注解";
    }
}
```

@Deprecated

它用来标记过期方法、不推荐使用方法。对于某些已经过期、不推荐使用的方法，但又不能直接删除（有其他地方仍使用），我们会使用@Deprecated进行标记，当使用这些方法时，会在编译时进行提示。

例子：

```
public class Test {
    public static void main(String[]args) {
        // 使用DeprecatedClass里声明被过时的方法
        DeprecatedClass.DeprecatedMethod();
    }
}

class DeprecatedClass {

    @Deprecated
    public static void DeprecatedMethod() {
    }
}
```

@SuppressWarnings

它用来标记不想被提示的警告，警告类型可以通过参数控制，具体如下：

deprecation，使用了过时的类或方法时的警告
unchecked，执行了未检查的转换时的警告
fallthrough，当Switch程序块直接通往下一种情况而没有Break时的警告
path，在类路径、源文件路径等中有不存在的路径时的警告
serial，当在可序列化的类上缺少serialVersionUID定义时的警告
finally，任何finally子句不能正常完成时的警告
all，关于以上所有情况的警告

例子：

```
public class Test {

    public static Listlist = newArrayList();

    @SuppressWarnings("unchecked")
    public void add(String data) {
        list.add(data);
    }
}
```

]

[

Java千百问_03基础语法（020）_注解、注释有什么区别

,

java注释注解×注释注解区别×注释注解混淆×java注释注解区别×注解与注释不同点×
点击进入_更多_Java千百问

1、注解、注释有什么区别

了解注释看这里：注释是什么
了解注解看这里：注解是什么

注解和注释很多人会混淆，它们之间的应用场景和具体使用完全不同，具体如下：

1. 用途不同
   注解通过标注包、类、字段、方法、局部变量、方法参数等元素，告诉JVM这些元素的附加信息（元信息）。
   注释是用来告诉开发人员这段代码的逻辑、说明、特点等，可以无限制的自由发挥。

2. 具体使用不同
   注解通过@来标注响应的元素，对于位置、语法、内容都有严格的限制，如果有任何错误，编译过程中就会异常。
   注释非常随意，在注释中可以填写任何内容，完全没有任何限制，甚至支持编写html。

3. 编译过程不同
   注解可以通过配置，在运行中让JVM去读取它，并完成对应的操作，一般会通过反射来获取我们为某个元素标注的注解。
   注释会被编译器完全忽略，完全是提供给开发人员作为参考使用。

4. 重要性不同
   注解目前来讲越来越重要，由于他可以对几乎所有元素进行描述和说明，在大多数框架中都非常喜欢使用注解来辅助配置。
   注释在代码规范越来越完善今天，作用越来越不明显，一段好的代码即便没有任何注释也能让开发者一目了然，而代码零注释的声音也越来越响，它的唯一作用可能就是用来生成javadoc了。

]

[

Java千百问_03基础语法（021）_如何自定义注解

,

点击进入_更多_Java千百问color="#ff0000"

1、如何自定义注解

要学习使用注解，我们需要了解注解，并且了解java提供的常见注解。更重要的是学会自定义注解。

了解注释看这里：注释是什么
我们自定义注解需要使用@interface关键字，具体语法如下：

```
public @interface 注解名 {定义体} default {默认值}
```

使用@interface自定义注解时，自动继承了java.lang.annotation.Annotation接口，由编译程序自动完成其他细节。在定义注解时，不能继承其他的注解或接口。

@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型。通过default来声明参数的默认值，如果不指定default，则说明该参数为必填。

注解参数（注解方法的返回值）的只能用public或默认（default）这两个访问权修饰，并且只支持以下数据类型：

1. 所有基本数据类型（int,float,boolean,byte,double,char,long,short)
2. String类型
3. Class类型
4. enum类型
5. Annotation类型
6. 以上所有类型的数组

我们一般是用反射来获取注解，一个简单的自定义注解和使用注解实例：

注解类：

```
@Retention(RetentionPolicy.RUNTIME)
public @interface TestAnnotation {

    String name() default "";
}
```

被注解的类：

```
public class TestInfo {

    @TestAnnotation(name = "公司名称")
    private String company;

    @TestAnnotation(name = "职位")
    private String position;
}
```

使用注解：

```
public class TestMain {

    public static void main(String[] args) {
        Field[] fields = TestInfo.class.getDeclaredFields();
        for (Field field : fields) {
            if (field.isAnnotationPresent(TestAnnotation.class)) {
                TestAnnotation testAnnotation = (TestAnnotation) field.getAnnotation(TestAnnotation.class);
                System.out.println("field:" + field.getName() + ", annotation:" + testAnnotation.name());
            }
        }
```

```
    }
}
```

我们运行后结果如下：

```
field:company, annotation:公司名称
field:position, annotation:职位
```

其中注解类中的注解@Retention(RetentionPolicy.RUNTIME)，是用来修饰注解的注解，即元注解，用来描述和限定自定义注解的使用场景和约束。具体如下：

了解元注解看这里：[元注解是什么][3]
[3]:

]

[

Java千百问_03基础语法（022）_元注解是什么

,

点击进入_更多_Java千百问

1、元注解是什么

了解注释看这里：注释是什么
了解自定义注解看这里：如何自定义注解
元注解就是负责注解其他注解。Java 5定义了4个标准的元注解（meta-annotation）：@Target、@Retention、@Documented、@Inherited，这些元注解的类在java.lang.annotation包中，具体如下：

@Target
@Target说明了注解所修饰的对象范围，即所定义的注解可以用在什么地方。具体的取值有：

1. CONSTRUCTOR
   用于描述构造器
2. FIELD
   用于描述类中的属性（域）
3. LOCAL_VARIABLE
   用于描述局部变量
4. METHOD
   用于描述方法
5. PACKAGE
   用于描述包
6. PARAMETER
   用于描述方法参数
7. TYPE
   用于描述类、接口(包括注解类型) 或enum声明

@Retention
@Retention定义了注解如何被保留，即被描述的注解在什么范围内有效。具体的取值有：

1. SOURCE
   在源文件中有效（即源文件保留）
2. CLASS
   在class文件中有效（即class保留）
3. RUNTIME
   在运行时有效（即运行时保留）

我们在《如何自定义注解》文章中的实例中使用了@Retention(RetentionPolicy.RUNTIME)，也就是说该自定义注解在运行时能够被读取和使用。

@Documented
@Documented定义了注解可以作为公共API，可以被例如javadoc此类的工具文档化。Documented是一个标记注解，并没

有任何成员。

@Inherited
@Inherited允许子类继承父类的注解。即一个使用了@Inherited修饰的注解被用于某个类，则该类的子类也等同于被该注解修饰。

]