Java 变量类型 Java 运算符

Java 修饰符

Java语言提供了很多修饰符，主要分为以下两类：

- 访问修饰符
- 非访问修饰符

修饰符用来定义类、方法或者变量，通常放在语句的最前端。我们通过下面的例子来说明：

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

访问控制修饰符

Java中，可以使用访问控制符来保护对类、变量、方法和构造方法的访问。Java 支持 4 种不同的访问权限。

- **default** (即缺省，什么也不写）: 在同一包内可见，不使用任何修饰符。使用对象：类、接口、变量、方法。
- **private** : 在同一类内可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）**
- **public** : 对所有类可见。使用对象：类、接口、变量、方法
- **protected** : 对同一包内的类和所有子类可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）。**

我们可以通过以下表来说明访问权限：

访问控制

修饰符	当前类	同一包内	子孙类(同一包)	子孙类(不同包)	其他包
public	Y	Y	Y	Y	Y
protected	Y	Y	Y	Y/N (说明)	N
default	Y	Y	Y	N	N
private	Y	N	N	N	N

默认访问修饰符-不使用任何关键字

使用默认访问修饰符声明的变量和方法，对同一个包内的类是可见的。接口里的变量都隐式声明为 `public static final`,而接口里的方法默认情况下访问权限为 `public`。

如下例所示，变量和方法的声明可以不使用任何修饰符。

实例

```
String version = "1.5.1";
boolean processOrder() {
return true;
}
```

私有访问修饰符-private

私有访问修饰符是最严格的访问级别，所以被声明为 **private** 的方法、变量和构造方法只能被所属类访问，并且类和接口不能声明为 **private**。

声明为私有访问类型的变量只能通过类中公共的 getter 方法被外部类访问。

Private 访问修饰符的使用主要用来隐藏类的实现细节和保护类的数据。

下面的类使用了私有访问修饰符：

```
public class Logger {
private String format;
public String getFormat() {
return this.format;
}
public void setFormat(String format) {
this.format = format;
}
}
```

实例中，Logger 类中的 format 变量为私有变量，所以其他类不能直接得到和设置该变量的值。为了使其他类能够操作该变量，定义了两个 public 方法：getFormat()（返回 format的值）和 setFormat(String)（设置 format 的值）

公有访问修饰符-public

被声明为 public 的类、方法、构造方法和接口能够被任何其他类访问。

如果几个相互访问的 public 类分布在不同的包中，则需要导入相应 public 类所在的包。由于类的继承性，类所有的公有方法和变量都能被其子类继承。

以下函数使用了公有访问控制：

```
public static void main(String[] arguments) {
// ...
}
```

Java 程序的 main() 方法必须设置成公有的，否则，Java 解释器将不能运行该类。

受保护的访问修饰符-protected

protected 需要从以下两个点来分析说明：

- **子类与基类在同一包中**：被声明为 protected 的变量、方法和构造器能被同一个包中的任何其他类访问；

- **子类与基类不在同一包中**：那么在子类中，子类实例可以访问其从基类继承而来的 protected 方法，而不能访问基类实例的protected方法。

protected 可以修饰数据成员，构造方法，方法成员，**不能修饰类（内部类除外）**。

接口及接口的成员变量和成员方法不能声明为 protected。 可以看看下图演示：

```
1  package com.runoob.test;
2
3  public interface Runoob {
4
5      default void fun() {}
6
7  }
```

子类能访问 protected 修饰符声明的方法和变量，这样就能保护不相关的类使用这些方法和变量。

下面的父类使用了 protected 访问修饰符，子类重写了父类的 openSpeaker() 方法。

```
class AudioPlayer {
protected boolean openSpeaker(Speaker sp) {
// 实现细节
}
}
class StreamingAudioPlayer extends AudioPlayer {
protected boolean openSpeaker(Speaker sp) {
// 实现细节
}
}
```

如果把 openSpeaker() 方法声明为 private，那么除了 AudioPlayer 之外的类将不能访问该方法。

如果把 openSpeaker() 声明为 public，那么所有的类都能够访问该方法。

如果我们只想让该方法对其所在类的子类可见，则将该方法声明为 protected。

> *protected 是最难理解的一种 Java 类成员访问权限修饰词，更多详细内容请查看 Java protected 关键字详解。*

访问控制和继承

请注意以下方法继承的规则：

- 父类中声明为 public 的方法在子类中也必须为 public。
- 父类中声明为 protected 的方法在子类中要么声明为 protected，要么声明为 public，不能声明为 private。
- 父类中声明为 private 的方法，不能够被继承。

非访问修饰符

为了实现一些其他的功能，Java 也提供了许多非访问修饰符。

static 修饰符，用来修饰类方法和类变量。

final 修饰符，用来修饰类、方法和变量，final 修饰的类不能够被继承，修饰的方法不能被继承类重新定义，修饰的变量为常量，是不可修改的。

abstract 修饰符，用来创建抽象类和抽象方法。

synchronized 和 volatile 修饰符，主要用于线程的编程。

static 修饰符

- **静态变量：**

  static 关键字用来声明独立于对象的静态变量，无论一个类实例化多少对象，它的静态变量只有一份拷贝。 静态变量也被称为类变量。局部变量不能被声明为 static 变量。

- **静态方法：**

  static 关键字用来声明独立于对象的静态方法。静态方法不能使用类的非静态变量。静态方法从参数列表得到数据，然后计算这些数据。

对类变量和方法的访问可以直接使用 **classname.variablename** 和 **classname.methodname** 的方式访问。

如下例所示，static修饰符用来创建类方法和类变量。

```
public class InstanceCounter {
private static int numInstances = 0;
protected static int getCount() {
return numInstances;
}
private static void addInstance() {
numInstances++;
}
InstanceCounter() {
InstanceCounter.addInstance();
}
public static void main(String[] arguments) {
System.out.println("Starting with " +
InstanceCounter.getCount() + " instances");
for (int i = 0; i < 500; ++i){
new InstanceCounter();
}
System.out.println("Created " +
InstanceCounter.getCount() + " instances");
}
}
```

以上实例运行编辑结果如下:

```
Starting with 0 instances
Created 500 instances
```

final 修饰符

final 变量：

final 表示"最后的、最终的"含义，变量一旦赋值后，不能被重新赋值。被 final 修饰的实例变量必须显式指定初始值。

final 修饰符通常和 static 修饰符一起使用来创建类常量。

实例

```
public class Test{
final int value = 10;
// 下面是声明常量的实例
public static final int BOXWIDTH = 6;
static final String TITLE = "Manager";
public void changeValue(){
value = 12; //将输出一个错误
}
}
```

final 方法

类中的 final 方法可以被子类继承，但是不能被子类修改。

声明 final 方法的主要目的是防止该方法的内容被修改。

如下所示，使用 final 修饰符声明方法。

```
public class Test{
public final void changeName(){
// 方法体
}
}
```

final 类

final 类不能被继承，没有类能够继承 final 类的任何特性。

实例

```
public final class Test {
// 类体
}
```

abstract 修饰符

抽象类：

抽象类不能用来实例化对象，声明抽象类的唯一目的是为了将来对该类进行扩充。

一个类不能同时被 abstract 和 final 修饰。如果一个类包含抽象方法，那么该类一定要声明为抽象类，否则将出现编译错误。

抽象类可以包含抽象方法和非抽象方法。

实例

```
abstract class Caravan{
private double price;
private String model;
private String year;
```

```
public abstract void goFast(); //抽象方法
public abstract void changeColor();
}
```

抽象方法

抽象方法是一种没有任何实现的方法，该方法的的具体实现由子类提供。

抽象方法不能被声明成 final 和 static。

任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类。

如果一个类包含若干个抽象方法，那么该类必须声明为抽象类。抽象类可以不包含抽象方法。

抽象方法的声明以分号结尾，例如：**public abstract sample();**。

实例

```
public abstract class SuperClass{
abstract void m(); //抽象方法
}
class SubClass extends SuperClass{
//实现抽象方法
void m(){
.........
}
}
```

synchronized 修饰符

synchronized 关键字声明的方法同一时间只能被一个线程访问。synchronized 修饰符可以应用于四个访问修饰符。

实例

```
public synchronized void showDetails(){
.......
}
```

transient 修饰符

序列化的对象包含被 transient 修饰的实例变量时，java 虚拟机(JVM)跳过该特定的变量。

该修饰符包含在定义变量的语句中，用来预处理类和变量的数据类型。

实例

```
public transient int limit = 55; // 不会持久化
public int b; // 持久化
```

volatile 修饰符

volatile 修饰的成员变量在每次被线程访问时，都强制从共享内存中重新读取该成员变量的值。而且，当成员变量发生变化时，会强制线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量的同一个值。

一个 volatile 对象引用可能是 null。

实例

```
public class MyRunnable implements Runnable
{
```

```
private volatile boolean active;
public void run()
{
active = true;
while (active) // 第一行
{
// 代码
}
}
public void stop()
{
active = false; // 第二行
}
}
```

通常情况下，在一个线程调用 run() 方法（在 Runnable 开启的线程），在另一个线程调用 stop() 方法。 如果 ***第一行*** 中缓冲区的 active 值被使用，那么在 ***第二行*** 的 active 值为 false 时循环不会停止。

但是以上代码中我们使用了 volatile 修饰 active，所以该循环会停止。

← Java 变量类型 Java 运算符 →