Scala Iterator（迭代器） Scala Trait(特征)

Scala 类和对象

类是对象的抽象，而对象是类的具体实例。类是抽象的，不占用内存，而对象是具体的，占用存储空间。类是用于创建对象的蓝图，它是一个定义包括在特定类型的对象中的方法和变量的软件模板。

我们可以使用 new 关键字来创建类的对象，实例如下：

```
class Point(xc: Int, yc: Int) {
   var x: Int = xc
   var y: Int = yc

   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("x 的坐标点: " + x);
      println ("y 的坐标点: " + y);
   }
}
```

Scala中的类不声明为public，一个Scala源文件中可以有多个类。

以上实例的类定义了两个变量 **x** 和 **y** ，一个方法：**move**，方法没有返回值。

Scala 的类定义可以有参数，称为类参数，如上面的 xc, yc，类参数在整个类中都可以访问。

接着我们可以使用 new 来实例化类，并访问类中的方法和变量：

```
import java.io._

class Point(xc: Int, yc: Int) {
   var x: Int = xc
   var y: Int = yc

   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("x 的坐标点: " + x);
      println ("y 的坐标点: " + y);
   }
}

object Test {
   def main(args: Array[String]) {
      val pt = new Point(10, 20);

      // 移到一个新的位置
      pt.move(10, 10);
```

```
    }
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
x 的坐标点: 20
y 的坐标点: 30
```

Scala 继承

Scala继承一个基类跟Java很相似, 但我们需要注意以下几点：

- 1、重写一个非抽象方法必须使用override修饰符。
- 2、只有主构造函数才可以往基类的构造函数里写参数。
- 3、在子类中重写超类的抽象方法时，你不需要使用override关键字。

接下来让我们来看个实例：

```
class Point(xc: Int, yc: Int) {
   var x: Int = xc
   var y: Int = yc

   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("x 的坐标点: " + x);
      println ("y 的坐标点: " + y);
   }
}

class Location(override val xc: Int, override val yc: Int,
   val zc :Int) extends Point(xc, yc){
   var z: Int = zc

   def move(dx: Int, dy: Int, dz: Int) {
      x = x + dx
      y = y + dy
      z = z + dz
      println ("x 的坐标点 : " + x);
      println ("y 的坐标点 : " + y);
      println ("z 的坐标点 : " + z);
   }
}
```

Scala 使用 extends 关键字来继承一个类。实例中 Location 类继承了 Point 类。Point 称为父类(基类)，Location 称为子类。

override val xc 为重写了父类的字段。

继承会继承父类的所有属性和方法，Scala 只允许继承一个父类。

实例如下：

```
import java.io._

class Point(val xc: Int, val yc: Int) {
   var x: Int = xc
   var y: Int = yc
   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
      println ("x 的坐标点 : " + x);
      println ("y 的坐标点 : " + y);
   }
}

class Location(override val xc: Int, override val yc: Int,
   val zc :Int) extends Point(xc, yc){
   var z: Int = zc

   def move(dx: Int, dy: Int, dz: Int) {
      x = x + dx
      y = y + dy
      z = z + dz
      println ("x 的坐标点 : " + x);
      println ("y 的坐标点 : " + y);
      println ("z 的坐标点 : " + z);
   }
}

object Test {
   def main(args: Array[String]) {
      val loc = new Location(10, 20, 15);

      // 移到一个新的位置
      loc.move(10, 10, 5);
   }
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
x 的坐标点 : 20
```

```
y 的坐标点 : 30
z 的坐标点 : 20
```

Scala重写一个非抽象方法，必须用override修饰符。

```
class Person {
  var name = ""
  override def toString = getClass.getName + "[name=" + name + "]"
}

class Employee extends Person {
  var salary = 0.0
  override def toString = super.toString + "[salary=" + salary + "]"
}

object Test extends App {
  val fred = new Employee
  fred.name = "Fred"
  fred.salary = 50000
  println(fred)
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
Employee[name=Fred][salary=50000.0]
```

Scala 单例对象

在 Scala 中，是没有 static 这个东西的，但是它也为我们提供了单例模式的实现方法，那就是使用关键字 object。

Scala 中使用单例模式时，除了定义的类之外，还要定义一个同名的 object 对象，它和类的区别是，object对象不能带参数。

当单例对象与某个类共享同一个名称时，他被称作是这个类的伴生对象：companion object。你必须在同一个源文件里定义类和它的伴生对象。类被称为是这个单例对象的伴生类：companion class。类和它的伴生对象可以互相访问其私有成员。

单例对象实例

```
import java.io._

class Point(val xc: Int, val yc: Int) {
   var x: Int = xc
   var y: Int = yc
   def move(dx: Int, dy: Int) {
      x = x + dx
      y = y + dy
```

```
   }
}

object Test {
   def main(args: Array[String]) {
      val point = new Point(10, 20)
      printPoint

      def printPoint{
         println ("x 的坐标点 : " + point.x);
         println ("y 的坐标点 : " + point.y);
      }
   }
}
```

执行以上代码，输出结果为：

```
$ scalac Test.scala
$ scala Test
x 的坐标点 : 10
y 的坐标点 : 20
```

伴生对象实例

```
/* 文件名: Marker.scala
 * author:菜鸟教程
 * url:www.runoob.com
 */

// 私有构造方法
class Marker private(val color:String) {

  println("创建" + this)

  override def toString(): String = "颜色标记："+ color

}

// 伴生对象，与类名字相同，可以访问类的私有属性和方法
object Marker{

    private val markers: Map[String, Marker] = Map(
      "red" -> new Marker("red"),
      "blue" -> new Marker("blue"),
      "green" -> new Marker("green")
    )
```

```
    def apply(color:String) = {
      if(markers.contains(color)) markers(color) else null
    }


    def getMarker(color:String) = {
      if(markers.contains(color)) markers(color) else null
    }
    def main(args: Array[String]) {
        println(Marker("red"))
        // 单例函数调用，省略了.(点)符号
        println(Marker getMarker "blue")
    }
}
```

执行以上代码，输出结果为：

```
$ scalac Marker.scala
$ scala Marker
创建颜色标记：red
创建颜色标记：blue
创建颜色标记：green
颜色标记：red
颜色标记：blue
```

← Scala Iterator（迭代器）

Scala Trait(特征) →

点我分享笔记