

全部课程 (/courses/) / Scala开发教程 (/courses/490) / 访问控制修饰符

在线实验，请到PC端体验

访问控制修饰符

一、实验介绍

1.1 实验内容

包的成员，类或对象可以使用访问控制修饰符。比如，用 `private` 和 `protected` 来修饰，通过这些修饰符可以控制其他部分对这些类，对象的访问。Scala 和访问控制大体上和 Java 类似，但也有些重要的不同，本节将介绍它们。

1.2 实验知识点

- 访问控制修饰符
- 为访问控制修饰符添加作用域

1.3 实验环境

- Scala 2.11.7
- Xfce 终端

1.4 适合人群

本课程难度为一般，属于初级级别课程，适合零基础或具有 Java 编程基础的用户。

二、开发准备

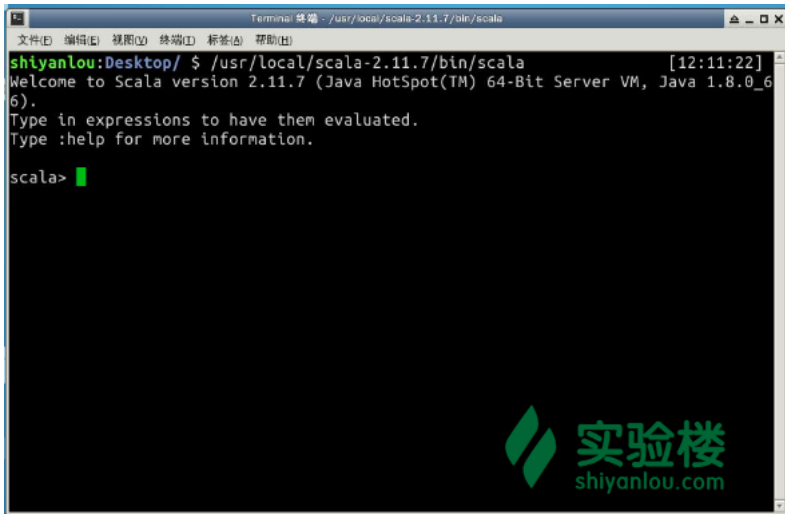
为了使用交互式 Scala 解释器，你可以在打开的终端中输入命令：

```
cd /usr/local/scala-2.11.7/bin/  
  
scala
```

当出现 `scala>` 开始的命令行提示符时，就说明你已经成功进入解释器了。如下图所示。

动手实践是学习 IT 技术最有效的方式！

开始实验



三、实验步骤

3.1 访问控制修饰符

3.1.1 私有成员

Scala 的私有成员和 Java 类似，一个使用 `private` 修饰过的类或对象成员，只能在该类或对象中访问。在 Scala 中，也可以在嵌套的类或对象中使用。比如：

```
class Outer{
  class Inner{
    private def f(){
      println("f")
    }

    class InnerMost{
      f() //OK
    }
  }

  (new Inner).f();// error: f is not accessible
}
```

在 Scala 中，`(new Inner).f()` 是不合法的，因为它是在 `Inner` 中定义的私有类型，而在 `InnerMost` 中访问 `f` 却是合法的。这是因为 `InnerMost` 是包含在 `Inner` 的定义中（子嵌套类型）。在 Java 语言中，两种访问都是可以的。Java 允许外部类型访问其包含的嵌套类型的私有成员。

3.1.2 保护成员

和私有成员类似，Scala 的访问控制比 Java 来说也是稍显严格些。在 Scala 中，由 `Protected` 定义的成员只能由定义该成员和其派生类型访问。而在 Java 中，由 `Protected` 定义的成员可以由同一个包中的其它类型访问。在 Scala 中，可以通过其它方式来实现这种功能。

下面为 `protected` 的一个例子：

```
class p{
  class Super{
    protected def f() {
      println("f")
    }
  }

  class Sub extends Super{
    f()
  }

  class Other{
    (new Super).f() //error: f is not accessible
  }
}
```

3.1.3 公开成员

`public` 访问控制为 Scala 定义的默认方式，所有没有使用 `private` 和 `protected` 修饰的成员都是“公开的”，可以被自由访问。Scala 不需要使用 `public` 来指定“公开访问”修饰符。

3.2 为访问控制修饰符添加作用域

Scala 的访问修饰符可以添加作用域参数。作用域的语法如下：

```
private[x]
```

或者是：

```
protected[x]
```

其中 `x` 代表某个包、类或者对象，表示可以访问这个 `private` 或的 `protected` 的范围直到 `x`。

通过为访问修饰符添加作用域参数，可以非常精确的控制所定义的类型能够被其它类型访问的范围。尤其是可以支持 Java 语言支持的 `package private`、`package protected` 等效果。

下面的例子为这种用法的一个示例：

```
package bobsrockets

package navigation{
  private[bobsrockets] class Navigator{
    protected[navigation] def useStarChart(){
      class LegOfJourney{
        private[Navigator] val distance=100
      }

      private[this] var speed = 200
    }
  }
}

package launch{
  import navigation._
  object Vehicle{
    private[launch] val guide=new Navigator
  }
}
```

在这个例子中，类 `Navigator` 使用 `private[bobsrockets]` 来修饰。这表示这个类可以被 `bobsrockets` 中所有类型访问。比如，通常情况下 `Vehicle` 无法访问私有类型 `Navigator`，但使用包作用域之后，`Vehicle` 中可以访问 `Navigator`。

这种技巧，对于分散在多个 Package 的大型项目而言，非常有用。它允许你进行定义，使其在多个子包中可以访问，但对使用这些 API 的外部客户代码隐藏，而这种效果在 Java 中是无法实现的。

动手实践是学习 IT 技术最有效的方式！

开始实验

此外, Scala 还支持一种比 `private` 还要严格的访问控制, 例如本例中的 `private[this]`。它只允许在定义该成员的类型中访问, 它表示该成员不仅仅只能在定义该成员的类型中访问, 而且只能是由该类型本身访问。比如:

本例中的 `speed`, 使用的 `protected[this]`、`speed` 和 `this.speed` 只在定义该成员的实例中可以访问。下面的用法也是不合法的, 即使它们也在 `Navigator` 里面。由于是新创建的另外的实例, 编译会出错 (此错误通常在 `ScalaIDE` 中能够给出, 此处作为了解即可):

```
package navigation{
  private[bobsrockets] class Navigator{
    protected[navigation] def useStarChart(){}
    class LegOfJourney{
      private[Navigator] val distance=100
    }

    private[this] var speed = 200

    def demo {
      val other=new Navigator
      val sp=other.speed
    }
  }
}

package launch{
  import navigation._
  object Vehicle{
    private[launch] val guide=new Navigator
  }
}
```

Symbol speed is inaccessible from this place

四、实验总结

通过访问控制修饰符, 能够较好地管理各个组件的“权限”。你也能够通过它们为部分代码“上锁”, 仅暴露出需要的部分。

至此, Scala 的基础知识部分就已经全部学习完毕了。我们将在稍候提供对应的项目课。

[← 上一节 \(/courses/490/labs/1695/document\)](#)

课程教师



引路蜂

共发布过6门课程

CSDN 专家博主, 擅长Java ME, Blackberry ,LWUIT , iPhone, Android, Windows Mobile, Mono , Windows Phone 7等平台开发, 主页 <http://www.imobilebbs.com/>

[查看老师的所有课程 > \(/teacher/164063\)](#)

进阶课程

[Scala 专题教程 - Case Class和模式匹配 \(/courses/514\)](#)

[Scala 专题教程 - 隐式变换和隐式参数 \(/courses/515\)](#)

[Scala 专题教程 - 抽象成员 \(/courses/516\)](#)

[Scala 专题教程 - Extractor \(/courses/526\)](#)



动手做实验, 轻松学IT



动手实践是学习 IT 技术最有效的方式!

开始实验

