使用Package（包）

一、实验介绍

1.1 实验内容

软件开发过程中，减小程序之间的“耦合性”至关重要。降低耦合性的一个方法是模块化。Scala 提供和 Java 类似的分包机制，但又稍有不同。因此，即使你了解 Java 语言，还是建议继续学习本节内容。

1.2 实验知识点

- 包的概念
- 引用包中的代码
- 使用 import
- 隐含的 import
- 包对象

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。

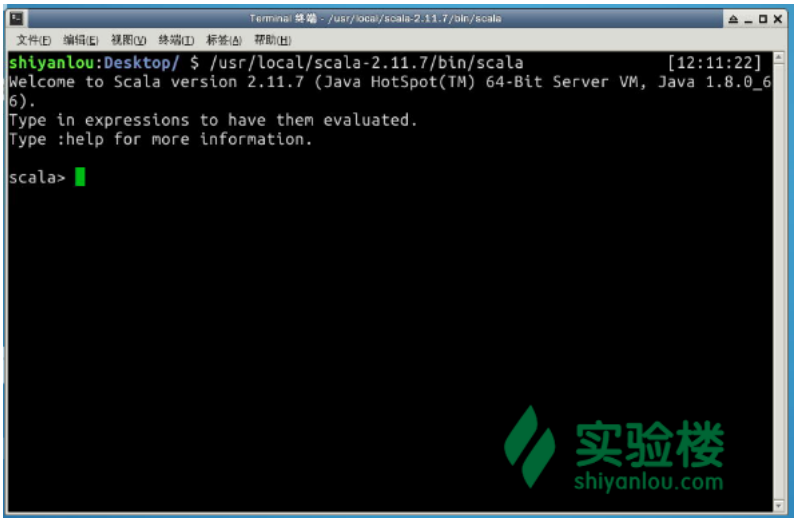
二、开发准备

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。



三、实验步骤

3.1 将代码放入包中

我们之前的例子中，没有明确使用 `package` ，因此它们存在于“未命名”的包中，或是默认包中。

在 Scala 将代码定义到某个包中有两种方式：

第一种方法和 Java 一样，在文件的头定义包名，这种方法就后续所有代码都放在该包中。

比如：

```
package bobsrockets.navigation
class Navigator
```

第二种方法有些类似 C# ，如：

```
package bobsrockets.navigation {
  class Navigator

}
```

第二种方法，可以在一个文件中定义多个包。

3.2 引用包中的代码

当我们把代码以层次关系放到包中时，它不仅仅可以帮助人们浏览代码，同时也说明了同一包中的代码具有某些相关性。 Scala 可以利用这些相关性来简化代码引用，你可以使用短名称，而无需使用包的全路径来访问类定义。

下面我们给出三个简单的例子：

```
package bobsrockets{
  package navigation{
    class Navigator{
      var map =new StarMap
    }

    class StarMap
  }

  class Ship {
    val nav= new navigation.Navigator
  }


  class fleets{
    class Fleet{
      def addShip() {new Ship}
    }
  }
}
```

在第一个例子中，正如你可以预见的一样：访问同一包中定义的类型，无需使用前缀。直接使用类型的名称即可访问，也就是本例可以直接使用 `new StarMap`。类 `StarMap` 和 `Navigator` 定义在同一个包中。

第二个例子，嵌套的 `package` 也可以在其父包中被同级别的其它类型直接访问，而无需使用全称。因此，第二个例子可以使用 `navigation` ，直接访问 `navigation` 包，而无需添加 `bobsrockets` 。

第三个例子，但使用包定义的 `{}` 语法结构时，内层的类型可以直接访问其外层定义的类型。因此，在类 `Fleet` 中，可以直接访问外层定义的类型 `Ship` 。

要注意的是，这种用法只适用于你明确嵌套包定义。如果你采用 Java 语言风格——一个文件定义一个包。那么你只能访问该包中定义的类型。

访问包定义的类型，还有一个技巧值得说明一下：比如你定义的一些类型之间可能存在相互隐藏的关系，也就是内层定义的同名类型可能会隐藏外层定义的同名类型。那么你怎么来访问外层定义的类型呢？请看下例：

```
package launch{
  class Booster3
}

package bobsrockets{
  package navigtion{
    package launch{
      class Booster1

    }

    class MissionControl{
      val booster1 =new launch.Booster1
      val booster2=new bobsrockets.launch.Booster2
      val booster3=new _root_.launch.Booster3

     }
    }


    package launch{
      class Booster2
    }
}
```

如何来访问 `Booster1`、 `Booster2` 和 `Booster3` 呢？访问 `Booster1` 比较容易， `Booster2` 可以通过全称来访问。那么如何访问最外层的 `Booster3` 呢？内层的包 `launch` 隐藏了这个外部的同名包。为解决这种情况， Scala 提供了 `_root_` ，也就是所有最外层的类型都可以当成定义在 `_root_` 包中。因此， `_root_.launch.Booster3` 可以到访问最外层定义的类型。

3.2.1 使用import

和 Java 一样， Scala 也是通过 `import` 语句引用其它包中定义的类型。类型引入后，可以使用短名称来引用该类型而无需使用全路径。要注意的 Scala 使用“ _ ”而非“ * ”作为通配符。

```
//easy access to Fruit
import bobsdelights.Fruit

//easy access to all members of bobdelights
import bobsdelights._

//easy access to all member of Fruits
import bobsdelights.Fruits._
```

所定义的类型中包 bobsdelights 中：

```
package bobsdelights

abstract class Fruit(
  val name: String,
  val color:String
)

object Fruits{
  object Apple extends Fruit ("apple","red")
  object Orange extends Fruit("orange","orange")
  object Pear extends Fruit("pear","yellowish")
  val menu=List(Apple,Orange,Pear)
}
```

第一个为引用单个类型，第二个为按需引用。和 Java 不同的是，是使用“_”代替“*”，第三个类似于 Java 中的静态引用，可以直接使用 Fruits 中定义的对象。

此外，Scala 中的 import 语句的使用比较灵感，可以用在代码的任意部分，而不一定需要在文件开头定义。比如下面的 import 定义在函数内部：

```
import bobsdelights.Fruit

def showFruit(fruit:Fruit){
  import fruit._

  println(name+"s are" + color)
}
```

方法 showFruit 引入 fruit 对象（非类型）的所有成员， fruit 的类型为 Fruit 。因此，可以在函数直接使用 fruit 的成员变量，而无需使用 fruit 限定符。这个方法和下面代码是等价的：

```
import bobsdelights.Fruit

def showFruit(fruit:Fruit){
   println(fruit.name+"s are" + fruit.color)
}
```

和 Java 相比，Scala 的 import 的使用更加灵活：

- 可以出现在文件中任何地方
- 可以 import 对象（singleton 或者普通对象）和 package 本身
- 支持对引入的对象重命名或者隐藏某些类型

下面的例子直接引入包名称，而非包中成员。引入包后，可以使用相对名称来指代某个类型（有些类型文件系统的路径）。

```
import java.util.regex

class AStarB {
  val pat= regex.Pattern.compile("a*b")
}
```

import 也可以用来重命名或者隐藏某些类型，比如：

```
import Fruits.{Apple,Orange}
```

仅仅引用 Fruits 中的 Apple 和 Orangle 类型。

下面的例子使用 => 重命名类型：

```
import Fruits.{Apple=>MaIntosh,Orange}
```

同样重命名也可以重新定义包名称，比如：

```
import java.{sql => S}
```

将引入的包 java.sql 改名为 java.S ，因此可以使用 S.Date 来代替 sql.Date 。

如果需要隐藏某个类型，可以使用 Type => _ 将某个类型改名为 _ ，就可以达到隐藏某个类型的效果。比如：

```
import Fruits.{Apple=>_,_}
```

这个引用中，引入了 Fruits 中除 Apple 之外的其它类型。

3.2.2 隐含的 import

Scala 默认为每个文件添加如下几个 package 。这几个包无需明确指明。

```
import java.lang._   //everything in the java.lang package
import scala._       //everything in the scala package
import Predef._      //everything in the Predef object
```

因此在写 Scala 应用之前，先了解下这些缺省包定义了那些类和功能。

此外这三个包的顺序也需要了解一下，比如： StringBuilder 类定义在包 scala 和 java.lang 包中，后定义的 import 会覆盖前面的定义。因此，如果不明确指明， StringBuilder 为 scala.StringBuilder 而非 java.lang.StringBuilder 。

注意这里的 scala._ 指所有 scala 下的包，包括子包。你可以在Scala的官方中查看到它们的全部信息 (http://www.scala-lang.org/files/archive/api/2.10.3/#package)，如下图所示。

Predef 为一对象（非报名），因此可以直接使用 Predef 对象定义的方法（静态引用）。因此在写代码之前了解下 scala 包和 Predef 定义的功能尤其重要。

3.2.3 包对象

到目前为止，我们看到的添加到包的都是类型、 Trait 和单例对象(Object)。这些都是指包的定级层次定义的类型。

Scala 的定级层次除了可以定义类、 `Trait` 、 `Object` 之外，其它的像可以在类、 `Trait` 、 `Object` 内部定义的类型，也都可以直接定义在包中。比如一些通用的函数、变量，你都可以直接定义在包中。

在 Scala 中，可以把这些函数或方法放在一个称为“包对象”中。每个包只有一个包对象，任何放在包对象的类型都可以认为是包自身的成员。例如：

```
//in file bobsdelights/package.scala

package object bobsdelights{
  def showFruit(fruit: Fruit){
    import fruit._
     println(name + "s are " + color)
  }
}

//in file PrintMenu.scala

package printmenu
import bobsdelights.Fruits
import bobsdelights.showFruit

object PrintMenu{
  def main(args:Array[String]){
    for(fruit <- Fruits.menu){
      showFruit(fruit)
    }
  }
}
```

本例中，对象 `PrintMenu` 可以引入包对象中定义的函数 `showFruit` ，方法和引入一个类定义一样，也是通过 `import` 语句。

包对象通常被编译为 `package.class` ，其包名为定义的包。所有按照惯例一般包对象定义放在 `package.scala` 中。比如，上面的包对象可以放在 `bobsdelights` 目录下的 `package.scala` 中。

四、实验总结

包是用来分类管理类文件的，包相当于文件夹，而类则相当于文件。学习完本实验，希望你能够在今后的开发工作中能够利用包的特性，更好地组织项目文件结构。

上一节 (/courses/490/labs/1694/document)

下一节 (/courses/490/labs/1696/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)