Scala第九章节

章节目标

1. 理解包的相关内容
2. 掌握样例类, 样例对象的使用
3. 掌握计算器案例

---

1. 包

实际开发中, 我们肯定会遇到同名的类, 例如: 两个Person类. 那在不改变类名的情况下, 如何区分它们呢?

这就要使用到`包(package)`了.

1.1 简介

包就是文件夹, 用关键字`package`修饰, 它可以区分重名类, 且功能相似的代码可以放到同一个包中, 便于我们维护和管理代码.

> 注意:
>
> 1. 编写Scala源代码时, 包名和源码所在的目录结构可以不一致.
> 2. 编译后, 字节码文件和包名路径会保持一致(由编译器自动完成).
> 3. 包名由数字, 大小写英文字母, _(下划线), $(美元符)组成, 多级包之间用.隔开, 一般是公司域名反写.
>
> `例如: com.itheima.demo01,  cn.itcast.demo02`

1.2 格式

- **格式一: 文件顶部标记法, 合并版**

```
package 包名1.包名2.包名3      //根据实际需求, 可以写多级包名
//这里可以写类, 特质...
```

- **格式二: 文件顶部标记法, 分解版**

```
package 包名1.包名2
package 包名3                 //这种写法和
//这里可以写类, 特质...
```

- **格式三: 串联式包语句**

```
package 包名1.包名2 {
    //包名1的内容在这里不可见

    package 包名3 {
        //这里可以写类, 特质...
    }
}
```

> 注意: 上述这种写法, 建议包名嵌套不要超过3级.

参考代码

```
//案例: 演示包的3种写法
//格式一: 文件顶部标记法, 合并版.
/*package com.itheima.scala
class Person*/

//格式二: 文件顶部标记法, 分解版.
/*package com.itheima
package scala
class Person*/

//格式三: 串联式包语句
package cn {          //嵌套不建议超过3层.
  package itcast {
    package scala {
      class Person{}
    }
  }
}

//测试类
object ClassDemo01 {
  //main方法是程序的主入口, 所有的代码都是从这里开始执行的.
  def main(args: Array[String]): Unit = {
    val p = new cn.itcast.scala.Person()
    println(p)
  }
}
```

1.3 作用域

Scala中的包和其它作用域一样, 也是支持嵌套的, 具体的访问规则(作用域)如下:

1. 下层可以直接访问上层中的内容.

   `即: 在Scala中, 子包可以直接访问父包中的内容`

2. 上层访问下层内容时, 可以通过`导包(import)或者写全包名`的形式实现.

3. 如果上下层有相同的类, 使用时将采用就近原则来访问.

   `即: 上下层的类重名时,优先使用下层的类, 如果明确需要访问上层的类,可通过上层路径+类名的形式实现`

需求

1. 创建com.itheima包, 并在其中定义Person类, Teacher类, 及子包scala.
2. 在com.itheima.scala包中定义Person类, Student类.
3. 在测试类中测试.

参考代码

```
//案例: 演示包的作用域
package com.itheima{  //父包
  class Person{}      //这是com.itheima包中的Person类
  class Teacher{}     //这是com.itheima包中的Teacher类

  object ClassDemo01 { //父包中的测试类
```

```
    def main(args: Array[String]): Unit = {
      //方式一: 导包, 导包语句可以出现在Scala代码中的任意位置, 不一定是行首.
     /*import com.itheima.scala.Student
      val s = new Student()   //父包访问子包的类, 需要导包.
      println(s)*/

      //方式二: 全包名
      val s = new com.itheima.scala.Student()
      //因为当前包就是com.itheima, 所以上述代码可以简写成如下格式
      //val s = new scala.Student()
      println(s)
    }
  }

  package scala{    //子包
    class Person{}  //这是com.itheima.scala包中的Person类
    class Student{} //这是com.itheima.scala包中的Student类

    object ClassDemo02 {  //子包中的测试类.
      def main(args: Array[String]): Unit = {
        val t = new Teacher()      //子包可以直接访问父包中的内容.
        println(t)

        val p = new Person()       //子父包有同名类时, 采用就近原则来访问.
        println(p)

        val p2 = new com.itheima.Person() //子父包有同名类, 且想访问父包类时, 写全路径即可.
        println(p2)
      }
    }
  }
}
```

1.4 包对象

包中可以定义子包, 也可以定义类或者特质, 但是Scala中不允许直接在包中定义变量或者方法, 这是因为JVM的局限性导致的, 要想解决此问题, 就需要用到`包对象`了.

1.4.1 概述

在Scala中, 每个包都有一个`包对象`, 包对象的名字和包名必须一致, 且它们之间是平级关系, 不能嵌套定义.

> 注意:
>
> 1. 包对象也要定义到父包中, 这样才能实现`包对象和包`的平级关系.
> 2. 包对象一般用于对`包的功能进行补充, 增强等`

1.4.2 格式

```
package 包名1 {        //父包
    package 包名2 {    //子包

    }

    package object 包名2 { //包名2的包对象

    }
}
```

1.4.3 示例

需求

1. 定义父包com.itheima, 并在其中定义子包scala.
2. 定义scala包的包对象, 并在其中定义变量和方法.
3. 在scala包中定义测试类, 并测试.

参考代码

```
package com.itheima { //父包

  package scala { //子包
    //测试类
    object ClassDemo03{
      def main(args: Array[String]): Unit = {
        //访问当前包对象中的内容.
        println(scala.name)
        scala.sayHello()
      }
    }
  }

  package object scala {   //scala包的包对象, 和scala包之间是平级关系.
    //尝试在包中直接定义变量和方法, 发现报错, 我们要用包对象解决.
    val name = "张三"
    def sayHello() = println("Hello, Scala!")
  }
}
```

1.5 包的可见性

在scala中, 我们也是可以通过访问权限修饰符(private, protected, 默认), 来限定包中一些成员的访问权限的.

格式

```
访问权限修饰符[包名]          //例如: private[com] var name = ""
```

需求

1. 定义父包com.itheima, 并在其中添加Employee类和子包scala.
2. 在Employee类中定义两个变量(name, age), 及sayHello()方法.
3. 在子包com.itheima.scala中定义测试类, 创建Employee类对象, 并访问其成员.

参考代码

```
//案例: 演示包的可见性
package com.itheima { //父包
  class Employee { //父包com.itheima中的Employee类
    //解释: private表示只能在本类中使用,  [itheima]表示只要是在itheima包下的类中, 都可以访问.
    private[itheima] val name = "张三"
    //什么都不写, 就是公共权限, 类似于Java中的public
    val age = 23

    //解释: private表示只能在本类中使用,  [com]表示只要是在com包下的类中, 都可以访问.
    private[com] def sayHello() = println("Hello, Scala!")
  }

  package scala { //子包
    object ClassDemo04 {  //子包com.itheima.scala中的测试类
      def main(args: Array[String]): Unit = {
        //创建Employee类的对象, 访问其中的成员.
        val e = new Employee()
        println(e)
        println(e.name)
        println(e.age)
        e.sayHello()
      }
    }
  }
}
```

1.6 包的引入

1.6.1 概述

在Scala中, 导入包也是通过关键字`import`来实现的, 但是Scala中的import功能更加强大, 更加灵活, 它不再局限于编写到scala文件的顶部, 而是可以编写到scala文件中任何你需要用的地方. 且Scala默认引入了java.lang包, scala包及Predef包.

1.6.2 注意事项

1. Scala中并不是完全引入了scala包和Predef包中的所有内容, 它们中的部分内容在使用时依旧需要先导包.

   `例如: import scala.io.StdIn`

2. import语句可以写到scala文件中任何需要用到的地方, 好处是: 缩小import包的作用范围, 从而提高效率.

3. 在Scala中, 如果要导入某个包中所有的类和特质, 要通过`_(下划线)`来实现.

   `例如:import scala._ 的意思是, 导入scala包下所有的内容`

4. 如果仅仅是需要某个包中的某几个类或者特质, 则可以通过`选取器(就是一对大括号)`来实现.

   `例如:import scala.collection.mutable.{HashSet, TreeSet}表示只引入HashSet和TreeSet两个类.`

5. 如果引入的多个包中含有相同的类, 则可以通过`重命名或者隐藏`的方式解决.

   - `重命名`的格式

```
import 包名1.包名2.{原始类名=>新类名, 原始类名=>新类名}

//例如: import java.util.{HashSet=>JavaHashSet}
```

- 隐藏的格式

```
import 包名1.包名2.{原始类名=>_, _}

//例如:import java.util.{HashSet=>_, _} 表示引入java.util包下除了HashSet类之外所有的类
```

1.6.3 示例

需求

1. 创建测试类, 并在main方法中测试上述的5点注意事项.
2. 需求1: 导入java.util.HashSet类.
3. 需求2: 导入java.util包下所有的内容.
4. 需求3: 只导入java.util包下的ArrayList类和HashSet类
5. 需求4: 通过重命名的方式, 解决多个包中类名重复的问题
6. 需求5: 导入时, 隐藏某些不需要用到的类, 即: 导入java.util包下除了HasSet和TreeSet之外所有的类.

参考代码

```
//案例: 演示包的引入
object ClassDemo05 {
  //测试方法1
  def test01() = {
    //1.导入java.util.HashSet
    import java.util.HashSet    //好处: 缩小import包的作用范围, 从而提高效率.
    val hs = new HashSet()
    println(hs.getClass)

    //2.导入java.util包下所有的内容.
    import java.util._
    val hm = new HashMap()
    println(hm.getClass)
    val list = new ArrayList()
    println(list.getClass)
  }

  //测试方法2
  def test02() = {
    //val hs = new HashSet()      //这样写会报错, 因为没有导包.

    //3.只导入java.util包下的ArrayList类和HashSet类
    /*import java.util.{ArrayList, HashSet}
    val list = new ArrayList()
    val hs = new HashSet()
    val hm = new HashMap()        //这样写会报错, 因为没有导包.*/

    //4.通过重命名的方式, 解决多个包中类名重复的问题
    /*import java.util.{HashSet => JavaHashSet}
    import scala.collection.mutable.HashSet
    val hs = new HashSet()
```

```
    val jhs = new JavaHashSet()
    println(hs.getClass)
    println(jhs.getClass)*/

    //5. 导入时, 隐藏某些不需要用到的类.
    //导入java.util包下除了HasSet和TreeSet之外所有的类
    import java.util.{HashSet=>_, TreeSet=>_, _}
    //val hs = new HashSet()      //这样写会报错
    val hm = new HashMap()
    println(hm.getClass)
  }

  //main方法, 程序的入口.
  def main(args: Array[String]): Unit = {
    //调用test01()和test02()这两个测试方法.
    test01()
    println("-" * 15)
    test02()
  }
}
```

2. 样例类

在Scala中, 样例类是一种特殊类, 一般是用于**保存数据**的(类似于Java POJO类), 在并发编程以及Spark、Flink这些框架中都会经常使用它。

1.1 格式

```
case class 样例类名([var/val] 成员变量名1:类型1, 成员变量名2:类型2, 成员变量名3:类型3){}
```

- 如果不写, 则变量的默认修饰符是val, 即: val是可以省略不写的.
- 如果要实现某个成员变量值可以被修改, 则需手动添加var来修饰此变量.

1.2 示例

需求

- 定义样例类Person, 包含姓名和年龄这两个成员变量.

  `其中: 姓名用val修饰, 年龄用var修饰`
- 在测试类中创建Person类的对象, 并打印它的属性值.
- 尝试修改姓名, 年龄这两个成员变量的值, 并观察结果.

参考代码

```
//案例: 样例类入门
object ClassDemo01 {
  //1. 创建一个Person样例类, 属性为: 姓名, 年龄.
  case class Person(name:String = "张三",var age:Int = 23) {}

  def main(args: Array[String]): Unit = {
    //2. 创建Person类型的对象, 然后打印属性值.
    val p = new Person()
    println(p)
```

```
    //3. 尝试修改对象p的属性值
    //p.name = "李四"     //这样写会报错, 因为样例类的成员变量默认修饰符是: val
    p.age = 24
    println(p)
  }
}
```

1.3 样例类中的默认方法

1.3.1 简介

当我们定义一个样例类后，编译器会自动帮助我们生成一些方法, 常用的如下:

- apply()方法
- toString()方法
- equals()方法
- hashCode()方法
- copy()方法
- unapply()方法

1.3.2 功能详解

- **apply()方法**
  - 可以让我们快速地使用类名来创建对象, 省去了new这个关键字
  - 例如: `val p = Person()`
- **toString()方法**
  - 可以让我们通过输出语句打印对象时, 直接打印该对象的各个属性值.
  - 例如: `println(p) 打印的是对象p的各个属性值, 而不是它的地址值`
- **equals()方法**
  - 可以让我们直接使用`==`来比较两个样例类对象的所有成员变量值是否相等.
  - 例如: `p1 == p2 比较的是两个对象的各个属性值是否相等, 而不是比较地址值`
- **hashCode()方法**
  - 用来获取对象的哈希值的. 即: 同一对象哈希值肯定相同, 不同对象哈希值一般不同.
  - 例如:

```
val p1 = new Person("张三", 23)
val p2 = new Person("张三", 23)
println(p1.hashCode() == p2.hashCode())     //结果为: true
```

- **copy()方法**
  - 可以用来快速创建一个属性值相同的实例对象，还可以使用带名参数的形式给指定的成员变量赋值.
  - 例如:

```
val p1 = new Person("张三", 23)
val p2 = p1.copy(age = 24)
println(p1)     //结果为: 张三, 23
println(p2)     //结果为: 张三, 24
```

- **unapply()方法**
  - 一般用作`提取器`, 但是它需要一些铺垫知识, 我们暂时还没学到, 在后续章节详细解释.

1.3.3 示例

需求

1. 创建Person样例类, 指定姓名, 年龄.
2. 在测试类中创建Person类的对象, 并测试上述的5个方法.

参考代码

```
//案例: 演示样例类的默认方法.
object ClassDemo02 {
  //1. 定义一个样例类Person.
  case class Person(var name:String, var age:Int) {}

  def main(args: Array[String]): Unit = {
    //2. 测试样例类中的一些默认方法.
    val p1 = Person("张三", 23)        //免new, 说明支持apply方法
    println(p1)                        //直接打印对象输出属性值, 说明支持toString方法

    val p2 = Person("张三", 23)
    println(p1 == p2)                  //比较的是属性值, 说明重写了equals方法
    println("-" * 15)

    //记忆: 同一对象哈希值肯定相同, 不同对象哈希值一般不同.
    println(p1.hashCode())             //可以获取哈希值, 说明重写了hashCode方法
    println(p2.hashCode())

    //演示copy方法, 就是用来复制对象的.
    //需求:  想创建一个张三, 50这样的对象.
    val p3 = p2.copy(age = 50)
    println(p3)                       //结果为: 张三, 50
  }
}
```

3. 样例对象

在Scala中, **用case修饰的单例对象就叫: 样例对象, 而且它没有主构造器** , 它主要用在两个地方:

1. 当做枚举值使用.

   > 枚举: 就是一些固定值, 用来统一项目规范的.

2. 作为没有任何参数的消息传递

   > 注意: 这点目前先了解即可, 后续讲解Akka并发编程时会详细讲解.

3.1 格式

```
case object 样例对象名
```

3.2 示例

需求

- 定义特质Sex, 表示性别, 且它只有两个实例(Male: 表示男，Female: 表示女)
- 定义Person类，它有两个成员变量（姓名、性别）

- 在测试类中创建Person类的对象, 并测试.

参考代码

```
object ClassDemo03 {
  //1. 定义一个特质Sex, 表示性别.
  trait Sex
  //2. 定义枚举Male, 表示男.
  case object Male extends Sex
  //3. 定义枚举Female, 表示女.
  case object Female extends Sex

  //4. 定义Person样例类, 属性: 姓名, 性别.
  case class Person(name:String, sex:Sex) {}

  def main(args: Array[String]): Unit = {
    //5. 创建Person类型的对象.
    val p = Person("张三", Male)
    //6. 打印属性值.
    println(p)
  }
}
```

4. 案例: 计算器

8.1 需求

- 定义样例类Calculate, 并在其中添加4个方法, 分别用来计算两个整数的`加减乘除`操作.
- 在main方法中进行测试.

8.2 目的

- 考察`样例类`的相关内容.

8.3 参考代码

```
//案例: 计算器.
object ClassDemo04 {
  //1. 定义样例类Calculate, 添加加减乘除这四个方法.
  case class Calculate(a:Int, b:Int){
    //加法
    def add() =  a + b
    //减法
    def subtract() =  a - b
    //乘法
    def multiply() =  a * b
    //除法
    def divide() =  a / b
  }

  //main方法, 程序的入口
  def main(args: Array[String]): Unit = {
    //2. 测试Calculate类中的加减乘除这四个方法.
    val cal = Calculate(10, 3)
```

```
    //测试加法
    println("加法: " + cal.add())
    //测试减法
    println("减法: " + cal.subtract())
    //测试乘法
    println("乘法: " + cal.multiply())
    //测试除法
    println("除法: " + cal.divide())
  }
}
```