Scala核心编程

-隐式转换和隐式参数

尚硅谷研究院

- 隐式转换

提出问题

先看一段代码，引出隐式转换的实际需要=>**指定**某些数据类型的相互转化

```
package com.atguigu.scala.conversion

object Scala01 {
  def main(args: Array[String]): Unit = {
    val num : Int = 3.5 //?错 高精度->低精度
    println(num)
  }
}
```

- 隐式转换

隐式函数基本介绍

隐式转换函数是以implicit关键字声明的带有单个参数的函数。这种函数将会**自动**应用，将值从一种类型转换为另一种类型

隐式函数快速入门

使用隐式函数可以优雅的解决数据类型转换，以前面的案例入门.

```
implicit def f1(d: Double): Int = {
  d.toInt
}
//Double 是输入类型, Int 是转换后的类型
```

- 隐式转换

隐式函数的底层工作原理

```
def main(args: Array[String]): Unit = {
      implicit def f1(d: Double): Int = {
        d.toInt
      }
      implicit def f2(l: Long): Int = {
        l.toInt
      }
      val num: Int = 3.5
      println(num)
      val num2: Int = 4.5
      println(num2)
      val num3: Int = 20l
 }
```

【案例演示+反编译+说明】

- 隐式转换

隐式转换的注意事项和细节

1) 隐式转换函数的函数名可以是任意的，隐式转换与函数名称无关，只与函数签名（函数参数类型和返回值类型）有关。

2) 隐式函数可以有多个(即：隐式函数列表)，但是需要保证在当前环境下，只有一个隐式函数能被识别

```
//在当前环境中，不能存在满足条件的多个隐式函数
implicit def a(d: Double) = d.toInt
implicit def b(d: Double) = d.toInt
val i1: Int = 3.5 //（X）在转换时，识别出有两个方法可以被使用，就不确定调用哪一个，所以出错
println(i1)
```

- 隐式转换丰富类库功能

基本介绍

如果需要为一个类增加一个方法，可以通过隐式转换来实现。（动态增加功能）比如想为MySQL类增加一个delete方法

分析解决方案

在当前程序中，如果想要给MySQL类增加功能是非常简单的，但是在实际项目中，如果想要增加新的功能就会需要改变源代码，这是很难接受的。而且违背了软件开发的OCP开发原则 (闭合原则 open close priceple)
在这种情况下，可以通过隐式转换函数给类动态添加功能。

- 隐式转换丰富类库功能

快速入门案例

使用隐式转换方式动态的给MySQL类增加delete方法。

```
class MySQL{
  def insert(): Unit = {
    println("insert")
  }
}
class DB {
  def delete(): Unit = {
    println("delete")
  }
}
```

```
implicit def addDelete(mysql:MySQL): DB = {
      new DB //
}
val mysql = new MySQL
mysql.insert()
mysql.delete() //?
//[案例演示+反编译]
```

- 隐式值

基本介绍

隐式值也叫**隐式变量**，将某个形参变量标记为implicit，所以编译器会在方法省略隐式参数的情况下去搜索作用域内的隐式值作为缺省参数

应用案例

```
implicit val str1: String = "jack"
def hello(implicit name: String): Unit = {
println(name + " hello")
}
hello //调用.不带()
//[案例演示+反编译]
```

- 隐式值

课堂测试题

下面的代码是正确，并解释scala底层实现机制,底层的函数名

```
object ImplicitVal {
  def main(args: Array[String]): Unit = {
    implicit val str1: String = "jack" //隐式值
    def hello(implicit name: String): Unit = { // hello$1
      println(name + " hello")
      def hello(): Unit = { // hello$2
        println("xxxx")
      }
    }
    hello // hello$1(str1)  使用隐式值不要带()
  }
  def hello(): Unit = { //函数名
    println("xx") }
} //案例演示+说明
```

- 隐式值

课堂练习

看下面几个案例，判断是否正确

```
object ImplicitVal02 {
  def main(args: Array[String]): Unit = {
    // 隐式变量（值）

    implicit val name: String = "Scala"
    implicit val name1: String = "World"

    def hello(implicit content: String = "jack"): Unit = {
      println("Hello " + content)
    } //调用hello

    hello
  }
}
//【案例演示+小结】
```

```
object Scala03 {
  def main(args: Array[String]): Unit = {

    // 隐式变量（值）
    implicit val name : String = "Scala"
    def hello( implicit content : String = "okook" ): Unit = {
      println("Hello " + content)
    }
    //调用hello
    hello
  }
}
```

输出什么内容？

题2

```
object Scala03 {
  def main(args: Array[String]): Unit = {

    // 隐式变量（值）
    implicit val name : Int = 10
    def hello( implicit content : String = "okook" ): Unit = {
      println("Hello " + content)
    }
    //调用hello
    hello
  }
}
```

输出什么内容？

题:3

```
object Scala03 {
  def main(args: Array[String]): Unit = {

    // 隐式变量（值）
    implicit val name : Int = 10
    def hello( implicit content : String ): Unit = {
      println("Hello " + content)
    }
    //调用hello
    hello
  }
}
```

输出什么？

题4

- 隐式类

基本介绍

在scala2.10后提供了**隐式类**，可以使用implicit声明类，隐式类的非常强大，同样可以扩展类的功能，比前面使用**隐式转换丰富类库功能**更加的方便，在集合中隐式类会发挥重要的作用。

隐式类使用有如下**几个特点**：

1) 其所带的构造参数有且只能有一个
2) 隐式类必须被定义在“类”或“伴生对象”或“包对象”里，即**隐式类不能是 顶级的(top-level objects)**。
3) 隐式类不能是case class（case class在后续介绍 **样例类**）
4) 作用域内不能有与之相同名称的标识符

- 隐式类

应用案例

看一个关于隐式类的案例，进一步认识隐式类。

```
class MySQL1 {
  def sayOk(): Unit = {
    println("sayOk")
  }
}
```

```
def main(args: Array[String]): Unit = {
    //DB1会对应生成隐式类
    implicit class DB1(val m: MySQL1) {
      def addSuffix(): String = {
        m + " scala"
      }
    }
    val mysql1 = new MySQL1
    mysql1.sayOk()
    //mysql1.addSuffix() ==> DB1$1(mysql1).addSuffix()
    //DB1$1(mysql1)返回的类型是 ImplicitClass$DB1$2
    println(mysql1.addSuffix())
  }//案例演示+说明
```

- **隐式的转换时机**

1) 当方法中的参数的类型与目标类型不一致时
2) 当对象调用所在类中不存在的方法或成员时，编译器会自动将对象进行隐式转换（根据类型）

- **隐式解析机制**

即编译器是如何查找到缺失信息的，解析具有以下两种规则：

1) 首先会在当前代码作用域下查找隐式实体（隐式方法、隐式类、隐式对象）。(一般是这种情况)
2) 如果第一条规则查找隐式实体失败，会继续在隐式参数的类型的作用域里查找。类型的作用域是指与该类型相关联的全部伴生模块，一个隐式实体的类型T它的查找范围如下(第二种情况范围广且复杂在使用时，应当尽量避免出现)：

a) 如果T被定义为T with A with B with C,那么A,B,C都是T的部分，在T的隐式解析过程中，它们的伴生对象都会被搜索。

b) 如果T是参数化类型，那么类型参数和与类型参数相关联的部分都算作T的部分，比如List[String]的隐式搜索会搜索List的伴生对象和String的伴生对象。

c) 如果T是一个单例类型p.T，即T是属于某个p对象内，那么这个p对象也会被搜索。

d) 如果T是个类型注入S#T，那么S和T都会被搜索。

- **隐式转换的前提**

在进行隐式转换时，需要遵守两个基本的前提：

1) 不能存在二义性
2) 隐式操作不能嵌套使用 // [举例：]如:隐式转换函数

```
//使用隐式函数来解决
//说明
//1. 在底层会生成一个函数 f1$1...
implicit def f1(d:Double): Int = {
  d.toInt

  val num3:Int = 5.6// 错误.==> f1(5.6)
}
```


图片1.z

谢谢！ 欢迎收看！