Scala的Option monad和C#的null-conditional操作符比较

这篇文章中将介绍C# 6.0的一个新特性，这将加深我们对Scala monad的理解。

Null-conditional操作符

假如我们有一个嵌套的数据类型，然后我们需要访问这个嵌套类型里面的某个属性。比如Article可以没有作者（Author）信息；Author可以没有Address信息；Address可以没有City信息，如下：

```
//////////////////////////////////////////////////////////////////////
 User: 过往记忆
 Date: 2016年2月24日
 Time: 22:25:23
 bolg:
 本文地址：/archives/1583
 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 过往记忆博客微信公共帐号：iteblog_hadoop
//////////////////////////////////////////////////////////////////////

public class Address {
 public string Street { get; set; }
 public string City { get; set; } // can be null
}

public class Author {
 public string Name { get; set; }
 public string Email { get; set; }
 public Address Address { get; set; } // can be null
}

public class Article {
 public string Title { get; set; }
 public string Content { get; set; }
 public Author Author { get; set;} // can be null
}

Console.WriteLine(article.Author.Address.City.ToUpper());
```

上面的代码会有产生NullReferenceException的风险，所以在使用这些对象之前，我们必须先做n

ull校验来避免NullReferenceException：

```
if (article != null) {
 if (article.Author != null) {
  if (article.Author.Address != null) {
   if (article.Author.Address.City != null) {
    Console.WriteLine(article.Author.Address.City.ToUpper());
   }
  }
 }
}
```

呸，使用如此复杂的代码来做一件非常简单的事！而且这段代码的可读行真的很差而且很混乱！幸运的是，C# 6.0 引入了一个null-conditional操作符。新的操作符使用?.表示，而且可以在操作符左边对象为null的如何情况下使用。

比如，下面的代码可以解释成：如果bob不为空，那么调用它的ToUpper方法；否则将bobUpper设置为null：

```
var bob = "Bob";
var bobUpper = bob?.ToUpper();
```

回到之前的例子，我们可以将它简写成：

```
Console.WriteLine(article?.Author?.Address?.City?.ToUpper());
```

Option类型

在Scala中，我们一般都会避免使用null。然而，我们依然想将某些数据设置成可选的。Option[T]这类可以显示地将某个值设置成可选状态。比如，下面的someBob对象的类型是Option[String]，这代表它既可以持有String类型的值，或者什么都没有：

```
val someBob: Option[String] = Some("Bob")
val noBob: Option[String] = None
```

因此，上面的例子在Scala中我们可以使用下面的代码实现：

```
case class Address(street: String, city: Option[String])
case class Author(name: String, email: String, address: Option[Address])
case class Article(title: String, content: String, author: Option[Author])
```

注意，和C#对比，在Scala中，我们必须显示地声明哪些字段是可选的！

然后我们打印出某篇文章作者的城市的小写形式：

```
if (article.author.isDefined) {
  if (article.author.get.address.isDefined) {
    if (article.author.get.address.get.city.isDefined) {
      println(article.author.get.address.get.city.get)
    }
  }
}
```

这种原生的方法和C#比较并没有什么提升，然而在Scala中，我们可以将上面代码写成下面：

```
for {
  author <- article.author
  address <- author.address
  city <- address.city
} yield println(city.toLowerCase())
```

虽然这个版本的代码也没有C#的 null-conditional操作符版本短，但是，重要的是，我们摆脱了使用嵌套的if语句，而且这个版本的代码可读性更高。这就是Scala中for语法和Option类型的monadic面的结合。

Option的monad特性

在我解释前面代码之前，首先让我来多介绍一点关于Option类型的方法。你是否记得List类型的map方法？它接收一个函数，然后将这个函数使用在list里面的所有元素中。有趣的是，Option类型也有map函数，我们可以将Option类型想象成一个List，它里面可以有一个（Some）或者零个（None）元素。所以，Option.map也可以接收一个函数，如果其中含有一个值，那么它将

把那个函数应用到那个值上；如果其中不含有值，那么Option.map直接返回None，如下：

```
//////////////////////////////////////////////////////////////////////
 User: 过往记忆
 Date: 2016年2月24日
 Time: 22:25:23
 bolg:
 本文地址：/archives/1583
 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 过往记忆博客微信公共帐号：iteblog_hadoop
//////////////////////////////////////////////////////////////////////

scala> val address = Address("street", Some("New York"))
address: HelloScala.Address = Address(street,Some(New York))

scala> address.city.map(city => city.toLowerCase())
res1: Option[String] = Some(new york)

scala> val address = Address("street", None)
address: HelloScala.Address = Address(street,None)

scala> address.city.map(city => city.toLowerCase())
res2: Option[String] = None
```

现在，我们是否可以将她使用到之前的例子中呢？

```
val cityLowerCase = article.author.map { author =>
  author.address.map { address =>
    address.city.map(city => city.toLowerCase)
  }
}
```

我想这个代码片段比直接使用if语句要好的多。使用这个代码的问题是，cityLowerCase的返回类型是Option[Option[Option[String]]]。最终的结果类型嵌套了这么多层！我们想要的类型是Option[String]。如果我们使用flatMap，我们将可以得到我们期望的类型：

```
val cityLowerCase: Option[String] = article.author.flatMap { author =>
  author.address.flatMap { address =>
    address.city.map(city => city.toLowerCase)
  }
```

```
}
```

Option.flatMap接收一个函数，并将option中的元素转换成另一个option，最后返回这个转换的值。这个方法和List的List.flatMap方法功能类似，其接收一个函数，并将list里面的元素转换成另外一个list，最后它将合并list里面的所有元素：

```
scala> List(1, 2, 3, 4).flatMap(el => List(el, el + 1))
res3: List[Int] = List(1, 2, 2, 3, 3, 4, 4, 5)
```

Option[T]和List[T]拥有flatMap方法意味着它们可以很容易地进行组合。在Scala中，所有拥有flatMap方法的类型都是monad！换句话说，monad就是说任何带有类型的泛型并且可以和其他同类型的泛型实例进行组合（使用flatMap方法）。现在，让我们回到之前的for语句，之所以没有在里面使用嵌套语句，是因为flatMap和map的语法糖导致的，我们可以把

```
val city = for {
  author <- article.author
  address <- author.address
  city <- address.city
} yield println(city.toLowerCase())
```

翻译成

```
val cityLowerCase: Option[String] = article.author.flatMap { author =>
  author.address.flatMap { address =>
    address.city.map(city => city.toLowerCase)
  }
}
```

为了加深对monad的理解，让我们来看看一个关于list的例子：

```
scala> for {
  el <- List(1, 2, 3, 4)
  list <- 1 to el
} yield list
```

```
res4: List[Int] = List(1, 1, 2, 1, 2, 3, 1, 2, 3, 4)
```

对于第一个list里面的元素，我们生成一个从1到那个元素的范围，最后我们将所有返回的list进行组合。

总结

这篇文章中，我的观点是来说明C#和Scala引入的一些特性来避免嵌套代码的编写。C#中引入了null-conditional操作符，来处理if语句中的嵌套null值检测；Scala则拥有一个更加通用的for语句和flatMap函数机制来避免嵌套代码的编写。

本文翻译自Scala's Option monad versus null-conditional operator in C#：http://www.codewithstyle.info/2016/02/scalas-option-monad-versus-null.html

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接: 【】（）