Scala ve Veri Akışları

Çağdaş Şenol

February 17, 2013

Outline

Listeler ve Methodlar

```
val numberList = 1::2::3::4::5::Nil

val stringList = "1"::"2"::"3"::"4"::Nil

def add1(x: Int) = x + 1
def mul2(x: Int) = x * 2
def toInt(x: String) = x.toInt
```

Basit Liste İşlemleri - Değiştirme

```
def add1ToAll(lon: List[Int]): List[Int] =
  if(lon.isEmpty)
    Nil
  else
    add1(lon.head) :: add1ToAll(lon.tail)
```

Desen Eşleme ve Tekrar Eden Desenler

- Pattern Matching İle daha şık yazmak mümkün

```
def mul2WithAll(lon: List[Int]): List[Int] =
  lon match {
    case Nil => Nil //Pattern 1
    case x::xs => mul2(x) :: mul2WithAll(xs) //Pattern
  }

def toIntAll(lon: List[String]): List[Int] =
  lon match {
    case Nil => Nil //Pattern 1
    case x::xs => toInt(x) :: toIntAll(xs) //Pattern 2
  }
```

Soyutlama 1

- Yapılan İşlem Aynı.
- Bir Liste al ve bir fonksiyon uygulanmış halini döndür

```
def mapper[A,B](f: A => B)(loa:List[A]): List[B] =
  loa match {
    case Nil => Nil //Pattern 1
    case x::xs => f(x) :: mapper(f)(xs) //Pattern 2
  }
```

- Ödev: Filter yazın

Basit Liste İşlemleri - Tüketme

```
val list = 1::2::3::4::5::Nil

def sum (a:Int, b:Int) = a + b
def mul (a:Int, b:Int) =  a * b
def concat (a:Int, b:String) = a.toString + b
```

Basit Liste İşlemleri - Tüketme

```
def listSum(lon:List[Int]): Int = {
  lon match {
    case Nil => 0 // Pattern 1
    case x::xs => sum(x , listSum(xs)) //Pattern 2
  }
}

def listMul(lon:List[Int]): Int =
  lon match {
    case Nil => 1 // Pattern 1
    case x::xs => mul(x , listMul(xs)) //Pattern 2
  }
```

Yine Benzer Desen

```
def listConcat(lon:List[Int]): String =
  lon match {
    case Nil => "" // Pattern 1
    case x::xs => concat(x, listConcat(xs)) //Pattern 2
  }
```

Soyutlama

- Sadece Int tipi ile oldu

```
def fold1(z:Int)(f: (Int,Int) => Int)(lon:List[Int]) :
  lon match {
    case Nil => z
    case x::xs => f(x , fold1(z)(f)(xs))
  }
```

Soyutlama

- Sadece Ayni tipler üzerinde çalışır halde oldu.

```
def fold2[A](z:A)(f: (A,A) => A)(lon:List[A]) : A =
  lon match {
    case Nil => z // Pattern 1
    case x::xs => f(x , fold2(z)(f)(xs)) //Pattern 2
  }
```

Soyutlama

-En genel Halde Liste tüketme işi

```
def folder[A,B](z:B)(f: (A,B) => B)(loa:List[A]) : B =
 loa match {
   case Nil => z // Pattern 1
   case x::xs => f(x , folder(z)(f)(xs)) //Pattern 2
 }
```

- Ödev: Foreach yazın

Kullanalım Bunları

```
val listNumbers = 1::2::3::4::5::Nil
val listStrings = List("1", "2", "3", "4", "5")

def add1(x: Int) = x + 1
def mul2(x: Int) = x * 2
def toInt(x: String) = x.toInt
def sum(a:Int, b:Int) = a + b
```

Kullanım 1

```
val result1 = listStrings.map(toInt).fold(0)(sum)
println(result1 == 15)
```

Kullanım 2

- Anonim fonksiyonlar

```
val result2 = listStrings
                .map((x: String) =>x.toInt)
                .fold(0)((x:Int, y:Int) => x + y)
println(result2 == 15)
```

Kullanım 3

- Underscore

```
val result3 = listStrings.map(_.toInt).fold(0)( _ + _)
println(result3 == 15)
```

Gerçek Hayatta Ne İşimize Yarayacak

```
//dosyadaki sayilari toplama
val file = scala.io.Source.fromFile("/tmp/hede")
val fileResult = file.getLines()
                     .map(_.toInt)
                     .fold(0)(_ + _)
file.close()
```

Enumerator

- Ya dosya çok büyükse ya da I/O bizi çok bekletiyorsa
- Simdilik Enumerator u liste ile ayni varsayalim

```
val l = Enumerator(1,2,3,4,5)
```

Enumeratee / Mapper

- Listeler icin map yazar gibi.
- Tek farkli liste.map diye cagirmak yerine
- Map Fonksiyounu Liste alacak şekilde önceden hazırlıyoruz.
- Gibi düşünelim

```
val add1 = Enumeratee.map( (x:Int) => x + 1)
val mul2 =  Enumeratee.map( (x:Int) => x * 2)
```

Iteratee

- Folder imiz gibi Listeyi tükettebilirz de
- Bunun için Iteratee ler var

```
val sum = Iteratee.fold(0)( (x:Int, y:Int) =>  x + y)
val printer = Iteratee.foreach(println _)
```

- Tüketmek üzere liste bekliyor

Enumeratorler ve Composition

- Enumeratorler composoble.
- yani l.map(_ + 1).map(_ * 3) yazmak gibi

```
val addMul = add1.compose(mul2)
val addMul2 = add1 >=> mul2
```

- Meşhur fish operator

Değiştir ve Tüket

```
val consume = addMul2 &>> sum
val f = 1 |>>> consume
val f2 =  1 |>>> add1 ><> mul2 &>> sum
```

- f ve f2 nin tipleri Future.
- |»> Asenkron bir çağrı
- ExecutionContex ile non-blocking

Gerçek Hayatta Ne İşimize Yarayacak

```
def ene(path:String) = {
  val r = new java.io.BufferedReader(
            new java.io.FileReader(path))
  Enumerator.fromCallback1( b => {
    val line = r.readLine
    val chunk = if(line == null) None else Some(line)
    scala.concurrent.Future.successful(chunk)
  }, r.close                                )
}
```

Gerçek Hayatta Ne İşimize Yarayacak

```
val res = e |>>> toInt &>> sum
res.onFailure{ case t => println(t.getCause)
res.onSuccess{ case t => println(t)}
```

Toparlarken

- Map, Filter, Fold ve Foreach ile neredeyse her türlü
- Liste ve Stream İşlemini yapabilirsiniz.
- Play Iteratee Kütüphanesi standalone halde play bağımsız
- Listeler Nasıl Head ve Tail den oluşuyorsa
- Streamler de Cont, Done ve Empty Tipinden oluşuyor

Son

- Teşekkürler.
- Sorular.
- Scala-Türkiye Google groups
- scala-turkiye github