Design Patterns Fonctionnels

Jean-Luc Falcone

4 juin 2013

Aggrégation

```
def sum( is: List[Int] ): Int =
  if( is.isEmpty ) 0
  else is.reduceLeft( _ + _ )

def forall( bs: List[Boolean] ): Boolean =
  if( bs.isEmpty ) false
  else bs.reduceLeft( _ && _ )

def concat[A]( lsts: List[List[A]] ): List[A] =
  if( lsts.isEmpty ) Nil
  else lsts.reduceLeft( _ ++ _ )

def pipeline[A]( lsts: List[A=>A] ): A=>A =
  if( lsts.isEmpty ) identity
  else lsts.reduceLeft( _ andThen _ )
```

Abstraction fonctionnelle

```
def aggreg[A]( lst: List[A], empty: A )
             ( reductor: (A,A)=>A      ): A =
  if( lst.isEmpty ) empty
  else lst.reduceLeft(reductor)

def sentence( words: List[String] ): String =
  aggreg( words, "" )( _ + " " + _ )

def forall( bs: List[Boolean] ): Boolean =
  aggreg( bs, false )( _ && _ )

def pipeline[A]( lsts: List[A=>A] ): A=>A =
  aggreg( lsts, identity )( _ andThen _ )
```

Abstraction objet

```
trait Aggregator[A] {
  def empty: A
  def append( a1: A, a2: A ): A
}

object Aggregator {
  def apply[A]( e: A )( f: (A,A)=>A ) = new Aggregator[A] {
    val empty = e
    def append( a1:A, a2:A ): A = f(a1,a2)
  }
  val intSum = apply(0)( _ + _ )
  val boolAnd = apply(0.0)( _ && _ )
  def listAggreg[A] = apply( List[A]() )( _ ::: _ )
}
```

Utilisation

```
def aggreg[A]( lst: List[A])( agg: Aggregator[A] ): A =
  if( lst.isEmpty ) agg.empty
  else lst.reduceLeft( agg.append )

def sum( is: List[Int] ): Int =
  aggreg( is )( Aggregator.intSum )

def forall( bs: List[Boolean] ): Boolean =
  aggreg( bs )( Aggregator.boolAnd )

def concat[A]( lsts: List[List[A]] ): List[A] =
  aggreg( lsts )( Aggregator.listAggreg[A] )
```

Injection de paramètres (implicite)

```
implicit val increment = 3

def inc( i: Int )( implicit incr: Int ) = i + incr

inc(2)(4) // => 6

inc(2)    // => 5
```

Injection de paramètres: Exemple (1)

```
def aggreg[A]( lst: List[A])( agg: Aggregator[A] ): A =
  if( lst.isEmpty ) agg.empty
  else lst.reduceLeft( agg.append )

object Aggregators {

  implicit val intSum = apply(0)( _ + _ )

  implicit val boolAnd = apply(0.0)( _ && _ )

  implicit def listAggreg[A] =
                  apply( List[A]() )( _ ::: _ )
}
```

Injection de paramètres: Exemple (2)

```
import Aggregators._

val b = aggreg( List(true,false,true) )
val i = aggreg( List(1,2,3,4) )
val l = aggreg( List( List( 1, 2 ), List( 3, 4 ) )
```

Algèbre: Monoïdes

Définition (wikipedia)

Formellement, $(E, *, e)$ est un *monoïde* lorsque :

- $\forall (x, y) \in E^2, x * y \in E$ (stabilité)
- $\forall (x, y, z) \in E^3, x * (y * z) = (x * y) * z$ (associativité)
- $\exists\ e \in E, \forall x \in E, x * e = e * x = x$ (existence d'un élément neutre)

Histogrammes (scala)

```
implicit def histoAggreg[A]: Aggregator[Map[A,Int]] =
  apply( Map[A,Int]() ){ (map1,map2) =>
    val keys = map1.keySet ++ map2.keySet
    keys.map{ =>
    ( k, (map1.getOrElse(k,0) + map2.getOrElse(k,0)) )
    }.toMap
  }

def histogram[A]( lst: A ): Map[A,Int] = {
  val hs = lst.map( a => Map(a->1) )
  aggreg( hs )
}
```

Histogrammes (java)

```
Map<A,Int> histogram[A]( List<A> lst ) {
  Map<A,Int> h = new HashMap<A,Int>();
  for( A a: lst ) {
    int count = 0;
    if( h.contains(a) ) {
      count += h.get(a);
    }
    h.put( a, count );
  }
  return h;
}
```