# 序列的函数式抽象-Spark API设计

functional programming (1) (/categories.html#functional programming-ref)

scheme [1] (/tags.html#scheme-ref) spark [8] (/tags.html#spark-ref) lisp [1] (/tags.html#lisp-ref)

**30 August 2013**

## Background

最近钻研于SICP (*Structure and Interpretation of Computer Programs*)，深为lisp所抽象出的公共模式所吸引，联系一直以来所使用的Spark，想到两者对于公共模式提炼的相同之处，有感而发，记下自己的所想。

# 公共模式的提炼

思考程序设计中如下几种场景:

1. 将一个表里的所有元素按给定因子做一次缩放 2. 将一个表里的所有元素进行平方 3. 将一个表里的所有元素加上一个给定值

可以看到以上三种场景可以抽象为将表内的所有元素进行某种运算得到一个新的表，本质上都是表的映射，我们可以将这种公共模式表述为一个高阶过程**map**，它的参数一个运算过程参数和一个表参数，返回值是将这个运算过程作用于表中每一个元素后形成的新表。可以参看场景1的例子：

```
1. (define (scale-list items factor)
2.   (if (null? items)
3.     nil
4.     (cons (* (car items) factor)
5.           (scale-list (cdr items) factor))))
```

对于它的抽象如下：

```
1. (define (map proc items)
2.   (if (null? items))
3.     nil
4.     (cons (proc (car items))
5.           (map proc (cdr items)))))
```

因此`scale-list`可以简化为如下:

```
1. (define (scale-list items factor)
2.   (map (lambda (x) (* x factor))
3.        items))
```

**map**不仅定义了一种公共模式，同时也设置了抽象屏障，将表变换过程的实现，与如何提取表中元素及组合结果的细节隔离开来，提升了抽象层次。

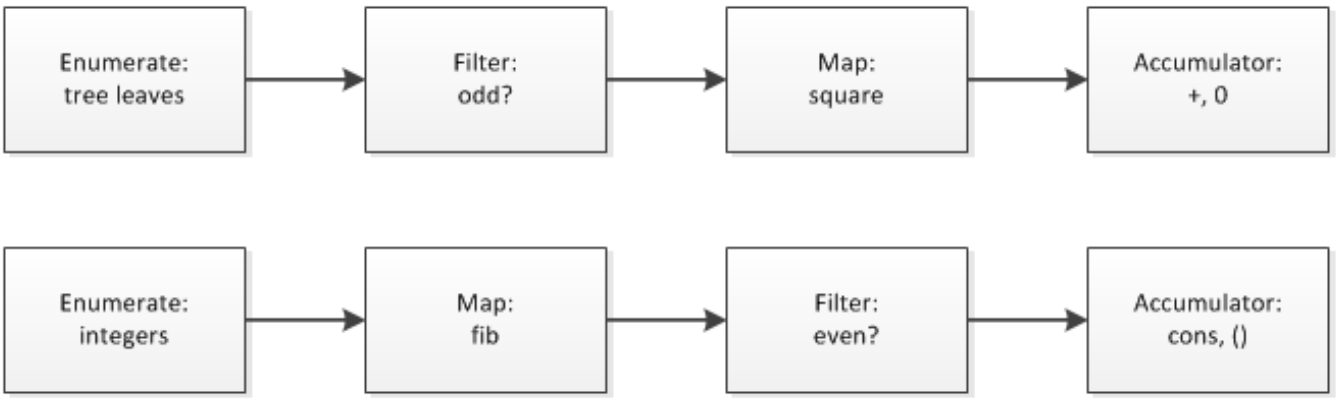
函数式语言提供了许多诸如**map**的公共模式，如**reduce**，**foreach**，**filter**，**accumulate**，利用这些公共模式，我们可以摆脱底层细节的纠缠，专注于算法的实现。

# 过程的公共模式拆解

同样思考以下两种场景:

1. 计算出树中所有叶子节点值为奇数的平方和 2. 构造出所有偶数的斐波那契数Fib(k)的一个表，其中k小于给定整数n

乍看之下上面描述的两个场景毫无共同点，无法进行抽象，实际对于复合过程的拆解会发现它们有极大的相似性，对于过程的分解可以参考下图：




类似于信号流图，通过一些级联的步骤对过程进行子过程分解和抽象，上述两个场景都可以用一些公共模式的串联来解决。我们来看一下场景1是如何利用公共模式构造和串联的。

**map**模式在之前已经构造出来了，接下来我们首先是要构造**filter**模式:

```
1. (define (filter predicate sequence)
2.   (cond ((null? sequence) nil)
3.         ((predicate (car sequence))
4.           (cons (car sequence)
5.                 (filter predicate (cdr sequence))))
6.         (else (filter predicate (cdr sequence)))))
```

同样我们还需要**accumulate**累加器对序列进行累加:

```
1. (define (accumulate op initial sequence)
2.   (if (null? sequence)
3.      initial
4.      (op (car sequence)
5.          (accumulate op initial (cdr sequence)))))
```

最后我们需要有一个输入序列:

```
1. (define (enumerate-tree tree)
2.   (cond ((null? tree) nil)
3.         ((not (pair? tree)) (list tree))
4.         (else (append (enumerate-tree (car tree))
5.                       (enumerate-tree (cdr tree))))))
```

将上面这些子过程串联起来我们就可以得出场景1的复合过程:

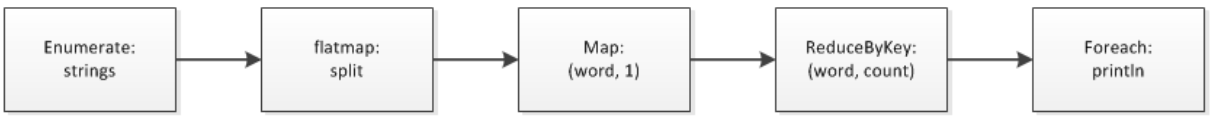
```
1. (define (sum-odd-square tree)
2.   (accumulate +
3.               0
4.               (map square
5.                    (filter odd?
6.                            (enumerate-tree tree)))))
```

由此我们可以看到，如果语言或者库为我们提供了抽象的公共模式，我们可以利用公共模式将我们的复杂算法分解为基本的公共模式并进行串联，这样可以使算法描述更为清晰、模块化。

# Spark API设计

我们知道Spark将数据集合抽象为RDD (*Resilient Distributed Dataset*)，所有的操作过程都是基于RDD的，而RDD本质上就是一个序列，因此Spark也提供了诸如**map**，**reduce**，**filter**等公共模式。

Spark提供了两大类公共模式: Transformations和Actions。Transformations是将一个RDD转换为另一个RDD，是一种映射。而Actions则提供了RDD计算结果的聚合和获得。我们以word count为例简单地表述一下如何利用公共模式的复合来表述算法。




可以看到复杂的算法被分解为信号流图的表述方式，并且将不同的公共模式应用于各阶段，清晰明白地描述出了算法的构成，而程序语言的构成如下所示：

```
1. rdd.flatMap(_.split(" ")).map(r => (r, 1)).reduceByKey(_ + _).foreach(println)
```

Spark屏蔽了RDD的转换和实现细节，设置了抽象屏障，使我们只需关注RDD如何转换并提供行为函数，并且Spark以统一且公认的函数式公共模式提供了API，使得熟悉函数式语言的用户可以轻松地了解这些公共模式的作用。同时由于提供了众多的公共模式，因此可以将算法清晰地分解为不同模式的组合，表述更为清晰和简洁。同样是基于map-reduce的算法框架，相比于Hadoop Spark提供了更精炼的API和更高的抽象模式，使得用户可以更清晰简单地描述其算法。

---

← Previous (/architecture/2013/05/22/SparkStreaming-Job-Troubleshooting-of-Dependency-Chain)
Archive (/archive.html) Next → (/arhitecture/2013/10/08/spark-storage-module-analysis)




**CATEGORIES**

lessons (1) (/categories.html#lessons-ref)

test (1) (/categories.html#test-ref)

architecture (6) (/categories.html#architecture-ref)

functional programming (1) (/categories.html#functional programming-ref)

arhitecture (1) (/categories.html#arhitecture-ref)

**LINKS**

阮一峰的网络日志 (http://www.ruanyifeng.com/blog/)

刘未鹏 (http://mindhacks.cn/)

酷壳 (http://coolshell.cn/)

BeiYuu.com (http://beiyuu.com/)

**MY FAVORITES**