The Slow, Steady Industry Move Toward Tacit Programming

Michael Feathers
R7K Research & Conveyance
# return the number of contiguous spans of non-zeros in an array
#
# :: [Integer] -> Integer
def span_count ary
  ([0] + ary).lazy
    .each_cons(2)
      .count{|c,n| c == 0 && n != 0 }
end
For some reason we like to call this functional programming

```ruby
# return the number of contiguous spans of non-zeros in an array
#
def span_count ary
  ([0] + ary).lazy
    .each_cons(2)
    .count{|c,n| c == 0 && n != 0 }
end
```
But it's not.

# return the number of contiguous spans of non-zeros in an array
#
# :: [Integer] -> Integer
def span_count ary
  ([0] + ary).lazy
    .each_cons(2)
    .count{|c,n| c == 0 && n != 0 }
end
It's something more severe..
Collection pipelines are a programming pattern where you organize some computation as a sequence of operations which compose by taking a collection as output of one operation and feeding it into the next. (Common operations are filter, map, and reduce.) This pattern is common in functional programming, and also in object-oriented languages which have lambdas. This article describes the pattern with several examples of how to form pipelines, both to introduce the pattern to those unfamiliar with it, and to help people understand the core concepts so they can more easily take ideas from one language to another.

25 June 2015

Martin Fowler

Contents

First encounters
Defining Collection Pipeline
Exploring more pipelines and operations
  Getting total word counts (map and reduce)
  Getting the number of articles of each type (group-by)
Law of Demeter

Fluent Interfaces

Train-Wrecks
class ClassNode
  def initialize exp
    @expression = exp
  end

  def field_names
    @field_names ||= @expression.flatten
      .each_cons(2)
      .select { |marker, _| marker == :ivar }
      .map { |_,name| name }
      .uniq
  end

  def method_names
    method_nodes.keys
  end

  def names
    @names ||= field_names + method_names
  end

  def method_nodes
    @method_nodes ||= Hash[exp=select(@expression, [:def]).map { |exp| MethodNode.new(exp) }
                             .map { |mn| [mn.name, mn] }]
  end

  def dependencies
    @dependencies ||= method_nodes.map { |n,mn| [n, mn.dependencies(names)] }
  end
end
Why Now?
Why Now?

Lambdas
Why Now?

Lambdas

Revenge of C
Use the *Pipes and Filters* architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (Filters) that are connected by channels (Pipes).
Referential Transparency..

Use the Pipes and Filters architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (Filters) that are connected by channels (Pipes).
Singular Focus

Use the Pipes and Filters architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (Filters) that are connected by channels (Pipes).
```ruby
stems = ARGF.read
  .split
  .each_cons(2)
  .group_by { |word_pair| word_pair[0] }

def next_word ary
  ary[rand(ary.length).to_i][1]
end

e = Enumerator.new do |e|
  word = stems.first.first
  while word
    e << word
    word = next_word(stems[word] || stems.first)
  end
end
```
Generate and Select
# return a line that represents increases, decreases, and stable changes in a method's length
#
# :: [event] -> String -> String
def method_delta_line es, method_name
    es.select { |e| e.method_name == method_name }
        .map(&:method_length)
        .each_cons(2)
        .map { |c,n| "^","=","v"[(c <=> n) + 1] }
        .join
end
[1,2,3,4,5,6,7].take(5)
[1,2,3,4,5,6,7].take(5)

[1,2,3,4,5]
[1,2,3,4,5,6,7].drop(5)
[1, 2, 3, 4, 5, 6, 7].drop(5)

[6, 7]
[].drop(5)
[].drop(5)
take and drop are total functions
def pair_at(n, array)
    array.drop(n).take(2)
end
def pair_at(n, array)
    array.drop(n).take(2)
end

composable!!
[].select { |e| e.odd? }
[].reject { |e| e.odd? }
[].map { |e| e.odd? }
[].detect { |e| e.odd? }

nil
When you want a loop, each or recursion: 
select, map, or reduce
When you want to carry state: zip it up
[1,2,3].zip(["a","b","c"])
[1,2,3].zip(['a','b','c'])

[[1,'a'],[2,'b'],[3,'c']]

[[1,'a'],[2,'b'],[3,'c']]
def lookahead_pairs sequence
    sequence.zip(sequence.drop(1))
end
def lookahead_pairs sequence
    sequence.zip(sequence.drop(1))
end

[1,2,3] -> [[1,2],[2,3],[3,nil]]
# is an array sorted?

class Array
  def sorted?
    zip(self.drop(1)).map { |left, right| left <= right }.all?
  end
end
is an array sorted?

class Array
  def sorted?
    zip(self.drop(1)).map {lleft,right| left <= right }.all?
  end
end

ruby-1.9.2-p180 :007 > [1,2,3].sorted?
ArgumentError: comparison of Fixnum with nil failed
  from (irb):4:in `<='
  from (irb):4:in `block in sorted?'
  from (irb):4:in `map'
  from (irb):4:in `sorted?'
  from (irb):7
  from /Users/michaelfeathers/.rvm/rubies/ruby-1.9.2-p180/bin/irb:16:in `<main>'
is an array sorted?

class Array
  def sorted?
    zip(self.drop(1)).map { |left, right| left <= right }.all?
  end
end

Problem:

[1,2,3].zip([1,2,3].drop(1))
=> [[1, 2], [2, 3], [3, nil]]
is an array sorted?

class Array
  def sorted?
    zip(self.drop(1)).map { |left, right| left <= right }.all?
  end
end

Problem:

[1,2,3].zip([1,2,3].drop(1))
=> [[1, 2], [2, 3], [3, nil]]

zip adds nuls when the sequences are not of equal size
# enter each_cons

class Array
  def sorted?
    each_cons(2).all? { |left, right| left <= right }
  end
end
# enter each_cons

class Array
  def sorted?
    each_cons(2).all? { |left, right| left <= right }
  end
end

This works!
day_data = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,  
16,17,18,19,20,21,22,23,24,25,26,27,28,29]

Take this ordered data and turn it into an array of arrays, each containing successive runs of seven elements. The last array may have less than seven.
day_data = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,  
16,17,18,19,20,21,22,23,24,25,26,27,28,29]

(0...day_data.count).zip(day_data)  
  .group_by { |day,_| day / 7 }  
  .map { |__,vl| vl.map { |__,v| v } }
day_data = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
            16,17,18,19,20,21,22,23,24,25,26,27,28,29]

(0...day_data.count).zip(day_data)

=> [[0, 0], [1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6], [7, 7], [8, 8], [9, 9],
   [10, 10], [11, 11], [12, 12], [13, 13], [14, 14], [15, 15], [16, 16], [17, 17],
   [18, 18], [19, 19], [20, 20], [21, 21], [22, 22], [23, 23], [24, 24], [25, 25],
   [26, 26], [27, 27], [28, 28], [29, 29]]
```ruby
day_data = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,  
            16,17,18,19,20,21,22,23,24,25,26,27,28,29]

(0...day_data.count).zip(day_data)  
  .group_by { |day,_| day / 7 }

=> {0=>[[0, 0], [1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6]], 1=>[[7, 7], [8, 8], [9, 9], [10, 10], [11, 11], [12, 12], [13, 13]], 2=>[[14, 14], [15, 15], [16, 16], [17, 17], [18, 18], [19, 19], [20, 20]], 3=>[[21, 21], [22, 22], [23, 23], [24, 24], [25, 25], [26, 26], [27, 27]], 4=>[[28, 28], [29, 29]]}
```
day_data = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,  
16,17,18,19,20,21,22,23,24,25,26,27,28,29]

(0...day_data.count).zip(day_data)
 .group_by { |day, _| day / 7 }  
 .map { |_, v| v }

=> [[0, 0], [1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6]], [[7, 7], [8, 8], [9, 9],  
[10, 10], [11, 11], [12, 12], [13, 13]], [[14, 14], [15, 15], [16, 16], [17, 17],  
[18, 18], [19, 19], [20, 20]], [[21, 21], [22, 22], [23, 23], [24, 24], [25, 25],  
[26, 26], [27, 27]], [[28, 28], [29, 29]]]
day_data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]

(0...day_data.count).zip(day_data)
  .group_by { |day, _| day / 7 }
  .map { |_, v| v.map { |_, vl v |} }

=> [[0, 1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12, 13], [14, 15, 16, 17, 18, 19, 20], [21, 22, 23, 24, 25, 26, 27], [28, 29]]
day_data = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29]

day_data.each_slice(7).to_a

=> [[0, 1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12, 13], [14, 15, 16, 17, 18, 19, 20], [21, 22, 23, 24, 25, 26, 27], [28, 29]]
gccmock.rb

# gccmock - generate exploding link stubs from linker error messages
#
# Usage: gcc [files] 2>&1 ruby gccmock.rb > [filename].c

LINK_SYMBOL = /.*\(_([_a-z0-9]*)\)/,

puts "#include <assert.h>\n"
puts "#define EXPLODE(name) void name() { assert(!"unexpected call"); }"
puts ARGF.select { |line| LINK_SYMBOL =~ line } \  
   .map { |line| LINK_SYMBOL.match(line)[1] } \  
   .uniq \  
   .sort \  
   .map { |name| "EXPLODE(#{name});" } \  
   .join($/)
# gccmock - generate exploding link stubs from linker error messages
#
# Usage: gcc [files] 2>&1 ruby gccmock.rb > [filename].c

LINK_SYMBOL = /.*"_(\_a-z0-9\*)"/,

puts "#include <assert.h>\n"
puts "#define EXPLODE(name) void name() { assert(!"unexpected call"); }"
puts ARGF.grep(LINK_SYMBOL) { |string| "EXPLODE(#{$1})" }  \  
  .uniq  
  .sort  
  .join($/)
STRING_COUNT = 6

def tab_column string, fret
    ["---"] * (string - 1) +
    [fret.ljust(3,'-')] +
    ["---"] * (STRING_COUNT - string)
end

puts ARGF.each_line
    .map(&:split)
    .map { |string,fret| tab_column(string.to_i, fret) }
    .transpose
    .map(&:join)
    .join($/)
def class_months es, class_name
  es.select { |e| e.class_name == class_name }
    .map { |e| e.date.month_start }
    .uniq
    .sort
end

def percent_active es, class_name, upto_date = Time.now.month_start
  range = class_months(es, class_name)
  range.count.to_f / month_range(range.first, upto_date).count * 100.0
end
render (EditBuffer topLine (x,y) contents) = do
  writeAt home $ window screenLines
  where window = unlines . take yExtent . drop topLine
  screenLines = (lines contents) ++ (repeat "\~")
[6] \(L \leftarrow (L' \vdash:) \uparrow L \vdash L\) \hspace{1cm} \text{n. drop To:}
[7] \(L \leftarrow \text{LJUST } VTOM',', L\) \hspace{1cm} \text{n. mat with one entry per row}
[8] \(S \leftarrow 1++/\backslash L \not\vdash 'c'\) \hspace{1cm} \text{n. length of address}
[9] \(X \leftarrow 0\Gamma/S\)
[10] \(L \leftarrow S\Phi(-\rho L+0, X) \uparrow L\) \hspace{1cm} \text{n. align the (names)}
[11] \(A \leftarrow (1\uparrow \rho L), X \uparrow L\) \hspace{1cm} \text{n. address}
[12] \(N \leftarrow 0 \downarrow \text{DLTB}(0, X) \downarrow L\) \hspace{1cm} \text{n. names}
[13] \(N \leftarrow ', 'a', N\)
[14] \(N[C(N='')/\rho N] \leftarrow '\) \hspace{1cm} \text{n. change _ to blank}
[15] \(N \leftarrow 0 \downarrow \text{RJUST } VTOM N\) \hspace{1cm} \text{n. names}
[16] \(S \leftarrow /\backslash ' \not\Phi N\) \hspace{1cm} \text{n. length of last word in name}
The following is an implementation of quicksort demonstrating *tacit programming*. Tacit programming involves composing functions together and not referring explicitly to any variables. J's support for *forks* and *hooks* dictates rules on how arguments applied to this function will be applied to its component functions.

```j
quicksort=: (($:@(<#[]), (=#[]), $:@(>#[])) (¬ ?@#)) ^: (1<#)
```
! 3 4 5
22 $ 1
2 2 $ 1

1 1
1 1
23 $ 1

1 1 1
1 1 1
222 $1

11
11
11
11
11
i. 4

0 1 2 3
23 $ i. 2

0 1 0
1 0 1
def game_of_life(name, size, generations, initial_life=nil)
  board = new_board size
  seed_board, size, initial_life
  print_board board, 0, name
  reason = generations.times do |gen|
    new = evolve board, size
    print_board new, gen+1, name
    break :all_dead if barren? new, size
    break :static if board == new
    board = new
  end
  if reason == :all_dead then puts "no more life."
  elsif reason == :static then puts "no movement"
  else puts "specified lifetime ended"
  end
end

def new_board(n)
  Array.new(n) {Array.new(n, 0)}
end

def seed(board, n, points=nil)
  if points.nil?
    # randomly seed board
    srand
    indices = []
    n.times { |x| n.times { |y| indices << [x, y] }}
    indices.shuffle[0,10].each { |x,y| board[y][x] = 1 }
  else
    points.each { |x, y| board[y][x] = 1 }
  end
end

def evolve(board, n)
  new = new_board n
  n.times { |i| n.times { |j| new[i][j] = fate board, i, j, n} }
end

def fate(board, i, j, n)
  i1 = [0, i-1].max; i2 = [i+1, n-1].min
  j1 = [0, j-1].max; j2 = [j+1, n-1].min
  sum = 0
  for ii in (i1..i2)
    for jj in (j1..j2)
      sum += board[ii][jj] if not (ii == i and jj == j)
    end
  end
  return sum
end
(_3 _3 (+/ e. 3+0.4&{}@,;._3 ])@ (0,0,~0.,0.,~)]