Thinking Immutably About Pipelines

David Pollak

Silesia JUG June 10, 2016

About @dpp

- Wrote a bunch of Spreadsheets
- Founded Lift & Wrote *Beginning Scala*
- VP Engineering Kiva.org (a PHP shop)

What We're Covering

- The Unix way
- Scala & Clojure
- Hash Array Mapped Tries
- Java Streams and Lambdas

Unix Piping

```
tr 'A-Z' 'a-z' <fnord.txt | tr -cs 'a-z' '\n' | \
   sort | uniq | comm -23 - /usr/share/dict/words
```

- Simple Spell Checker
- Each Program (function) in the chain does something specific
- Composable and Incremental to Learn/Understand

Why did Unix Pipes Win?

- Each Program Excellent on its Own
- Simple Exploration
- Composing Pipes forces Thinking & Isolation
- Did pipes win? *Windows PowerShell*

How do Humans Learn?

- Incrementally
- Via Exploration
- Applying Isolated Pieces to Form a Whole

How Many Things Can You Hold in Your Head?

- Dunno... but it's limited
- Each time we add "something", something else has to drop out
- So... less is more

What About Inside a Program?

- Can we "chain" or compose operations?
- Can we isolate logic and perhaps re-use the logic?
- Can we build logic Incrementally?

Yes!!

Some People call it "Functional Programming"

Detour: Cyclomatic Complexity

example

```
int age = 29;

if (age < 13)
{
  System.out.println("You are but a wee child!");
}
else if (age < 19)
{
  System.out.println("You are no longer a child, but a budding teenager.");
}
else
{
  if (age < 65)
  {
    System.out.println("You are an adult!");
  }
  else
  {
    System.out.println("You are now a senior, enjoy the good life friends!");
  }
  System.out.println("Also, since you are over the age of 19, you deserve a drink!");
}
```

This gets outputted because we know we’re inside that outer else block of code. Remember to follow through those curly braces {} to know exactly where you are.

Detour: Cyclomatic Complexity

- More Code Paths == Complexity
- Complexity == Bad
- Complexity means harder to understand and harder to keep track of the impact of changes
- More stuff to remember 🡒 more requirements that your brain executes code

Take While, Java Edition

```
final String x = "Elwood Eats Mice";
final StringBuilder ret = new StringBuilder();
for (char c : x.toCharArray()) {
    if (c != ' ') ret.append(c);
    else break;
}

return ret.toString();
```

Take While, Functional Editions

Scala:

```
scala> val x = "Elwood Eats Mice"
x: String = Elwood Eats Mice

scala> x.takeWhile(_ != ' ')
res0: String = Elwood
```

Clojure:

```
(def x "Elwood Eats Mice")

(->> x
     (take-while #(not= % \space ))
     clojure.string/join)
;; "Elwood"
```

Yeah, So?

- Fewer Lines of Code
- More Readable: Eyes drawn to logic
- Logic isolated and you can understand each bit of it
- The "What" not the "How"

Age Thing, Less Complex

```
(def rules [[#(< % 13) "You are but a wee child!"]
            [#(and (>= % 13) (<= % 19)) "You are no longer a child, but a budding teenager."]
            [#(and (> % 19) (< % 65)) "You are an adult!"]
            [#(>= % 65) "You are now a senior, enjoy the good life friends!"]
            [#(>= % 18) "Since you are 18 or over, you deserve a drink!"]])

(defn messages [age] (->>
                      rules
                      (filter #((first %) age))
                      (map second)))
```

```
user> (messages 13)
;; => ("You are no longer a child, but a budding teenager.")
user> (messages 32)
;; => ("You are an adult!" "Since you are 18 or over, you deserve a drink!")
user> (messages 3)
;; => ("You are but a wee child!")
user> (messages 67)
;; => ("You are now a senior, enjoy the good life friends!" "Since you are 18 or over, you deserve a drink!")
```

Chaining

First Name of Valid Persons, Sorted by Age

```
def validByAge(in: List[Person]) =
    in.filter(_.valid).
    sort(_.age < _.age).
    map(_.first)

(->> in
     (filter :valid)
     (sort-by :age)
     (map :first))
```

Chaining Is Readable

Chaining Allows Function Re-Use

Immutable Data Structures

- Immutable Means never saying "synchronized"
- Great for Multi-Threaded Systems
- Great for Distributed Systems

Performance of O(Log N)?

Not Any More

- HAMT (Hash Array Mapped Tries)
- Effectively O(1) for most operations
- Developed at EPFL
- Built into Clojure & Scala… not so much for Java (Guava?)

But, What About Java?

Where can we get some of the is goodness?

Java 8 Streams and Lambdas

- Streams: Lazy Collections (like Unix Pipes, Clojure `Seq`, and Scala `Stream`)
- Lambdas: Functions that close over `final` local scope

Java Example

```
public static Stream<String> validByAge(Stream<Person> in) {
    return in.filter(Person::isValid).
             sorted((a, b) -> a.getAge() - b.getAge()).
             map(Person::getFirst);
}
```

That's more like it!

But What if We Want a `List`?

`Collect` it...

```
public static List<String> validByAge2(Stream<Person> in) {
    return in.filter(Person::isValid).
             sorted((a, b) -> a.getAge() - b.getAge()).
             map(Person::getFirst).
             collect(Collectors.toList());
}
```

`Collect` "reduces" lazy stream to something concrete.

What About This Lazy Thing?

- Lazy allows dealing with unbounded streams
- Data consumed "as needed"
- Much lower memory footprint

```
public static IntStream primes(int until) {
    return IntStream.rangeClosed(1, until).filter(Main::isPrime);
}
```

Computed all the primes up to 1B.

Conclusion

- The "Unix Way" is the Functional Way
- Scala and Clojure led FP charge on JVM
- Java8 has good features to allow chaining on logic
- Lambdas and Streams reduce cyclomatic complexity: more maintainable code

Questions & Thanks!

Reference

- HAMT https://idea.popcount.org/2012-07-25-introduction-to-hamt/
- Wikipedia https://en.wikipedia.org/wiki/Hash*array*mapped_trie