Transducers Internals

Transducers in Practice Workshop - CUFP 2017

Universality of fold

- `reduce` as the prototypical recursive iterative process
- Redefinition of sequential processing in terms of `reduce`
- Extract transformations and I/O details away
- Let’s refactor `map` and `filter` to find out

Step 1: express like reduce

```
(defn map [f result coll]
  (if (not= '() coll)
    (map f (f result (first coll)) (rest
coll))
    result))

(map (fn [result el] (conj result (inc
el))) [] (range 10))
```

```
(defn filter [f result coll]
  (if (not= '() coll)
    (filter f (f result (first coll))
(rest coll))
    result))

(filter (fn [result el] (if (odd? el)
(conj result el) result)) [] (range 10))
```

Step 2: then call it reduce

```
(defn reduce [f result coll]
  (if (not= '() coll)
    (reduce f (f result (first coll))
(rest coll))
    result))

(reduce (fn [result el] (conj result (inc
el))) [] (range 10))
```

```
(defn reduce [f result coll]
  (if (not= '() coll)
    (filter f (f reduce (first coll))
(rest coll))
    result))

(reduce (fn [result el] (if (odd? el)
(conj result el) result)) [] (range 10))
```

Step 3: which is already in STD lib

```
(reduce (fn [result el] (conj result (inc el))) [] (range 10))
```

```
(reduce (fn [result el] (if (odd? el) (conj result el) result)) [] (range 10))
```

Step 4: extract “essence” into fn

```clojure
(defn mapping [result el]
  (conj result (inc el)))


(reduce mapping [] (range 10))
```

```clojure
(defn filtering [result el]
  (if (odd? el)
    (conj result el)
    result))

(reduce filtering [] (range 10))
```

Step 5: introduce param “rf”

```clojure
(defn mapping [rf]
  (fn [result el]
    (rf result (inc el))))


(reduce (mapping conj) [] (range 10))
```

```clojure
(defn filtering [rf]
  (fn [result el]
    (if (odd? el)
      (rf result el)
      result)))

(reduce (filtering conj) [] (range 10))
```

Step 6: introduce param f / pred?

```clojure
(defn mapping [f]
  (fn [rf]
    (fn [result el]
      (rf result (f el)))))
```

```clojure
(reduce ((mapping inc) conj) [] (range 10))
```

```clojure
(defn filtering [pred]
  (fn [rf]
    (fn [result el]
      (if (pred? el)
        (rf result el)
        result))))
```

```clojure
(reduce ((filtering odd?) conj) [] (range 10))
```

Step 7: extract transduce fn

```clojure
(defn mapping [f]
  (fn [rf]
    (fn [result el]
      (rf result (f el)))))


(defn transduce [xf rf coll]
  (reduce (xf rf) (rf) coll))

(transduce (mapping inc) conj (range 10))
```

```clojure
(defn filtering [pred]
  (fn [rf]
    (fn [result el]
      (if (pred? el)
        (rf result el)
        result))))

(defn transduce [xf rf coll]
  (reduce (xf rf) (rf) coll))

(transduce (filtering odd?) conj (range 10))
```

Create your own transducers

- Our `mapping` and `filtering` are how transducers are in the STD lib!
- Well almost. A good “transducer” also knows how to behave.

Designing a transducer

- Terminating or non terminating computations (1-arg arity).
- Providing an initial value (0-arg arity).
- Stateful or stateless.
- How to terminate early (when required).
- Be respectful of surrounding transducers (mandatory calls).

Resources

- A tutorial on the universality and expressiveness of fold

Lab 02
Custom Transducers

Goal of Lab2

- Task1: create a logging transducer to print intermediate results
- Task 2: create a (stateful) moving average transducer

Open `transducers-workshop.lab02` to start.