Dispatching Types with Flux

UtahJS / PolyJS

by Seth House

@whiteinge
seth@eseth.com

reduce

Background

```
// Flux! (Seriously.)
var Dispatcher = new Rx.Subject();
var send = (tag, data) => Dispatcher.onNext({tag, data});
```

Background

```
// Flux! (Seriously.)
var Dispatcher = new Rx.Subject();
var send = (tag, data) => Dispatcher.onNext({tag, data});
```

```
var ThingStore = Dispatcher
    .filter(({tag}) => tag === 'THING_ADDED' || tag === 'THING_REMOVED')
    .scan(function(acc, {tag, data}) {
        switch (tag) {
            case 'THING_ADDED':
                acc[data.id] = data;
                return acc;
            case 'THING_REMOVED':
                delete acc[data.id];
                return acc;
            default:
                return acc;
        }
    }, {})
    .shareReplay(1);
```

Example

```
var sub1 = ThingStore.subscribe(x => console.log('Current value of ThingStore', x));

send('THING_ADDED', {id: 'foo'})
```

Generalize?

```
[...]
.filter(({tag}) => tag === 'THING_ADDED' || tag === 'THING_REMOVED')
.scan(function(acc, {tag, data}) {
  switch (tag) {
    case 'THING_ADDED':
      acc[data.id] = data;
      return acc;
    case 'THING_REMOVED':
      delete acc[data.id];
      return acc;
    default:
      return acc;
  }
}, {})
[...]
```

Types

Semigroup

- Types with a `concat` method.

Semigroup

- Types with a `concat` method.
- All about *combining* values.

Semigroup

- Types with a `concat` method.
- All about *combining* values.
- Prepend, append, add, multiply, merging, etc.

Semigroup

- Types with a `concat` method.
- All about *combining* values.
- Prepend, append, add, multiply, merging, etc.
- [Formal specification or "laws" not pictured here.]

You Already Know Semigroups

```
'foo'.concat('bar')
```

You Already Know Semigroups

```
'foo'.concat('bar')
```

```
['foo'].concat(['bar'])
```

Monoid

- Is a semigroup.

Monoid

- Is a semigroup.
- All about *combining* values.

Monoid

- Is a semigroup.
- All about *combining* values.
- Types with an `empty` method.

Monoid

- Is a semigroup.
- All about *combining* values.
- Types with an `empty` method.
- Allows you to start combining things from nothing. I.e., a "seed" value.

Augment the Builtin Types (o_O)

```
String.empty = () => '';
Array.empty = () => [];
```

Generalize

```
var fold = M => xs => xs.reduce(
    (acc, x) => acc.concat(x),
    M.empty());
```

Combine via Reduction

```
fold(String)(['foo', 'bar'])
```

Combine via Reduction

```
fold(String)(['foo', 'bar'])
```

```
fold(Array)(['foo', 'bar'])
```

Make your own Types

```
var Collection = daggy.taggedSum('Collection', {
    Add: ['val'],
    Del: ['val'],
});
```

Make your Type a Monoid

```
Collection.prototype.concat = function(newVal) {
    function addRemove(oldVal) {
        if (Collection.Add.is(newVal)) {
            return Collection.Add(Object.assign({}, oldVal, newVal.val));
        }
        if (Collection.Del.is(newVal)) {
            Object.keys(newVal.val).forEach(function(key) {
                var val = newVal.val[key];
                if (oldVal[key]) {
                    delete oldVal[key];
                }
            });
            return Collection.Add(oldVal);
        }
        return oldVal;
    };

    return this.cata({
        Add: addRemove,
        Del: addRemove,
    });
};
```

Generalize in Rx too

```
Rx.Observable.prototype.foldp = foldp;
function foldp(M, seed = M.empty()) {
    return this.scan((acc, x) => acc.concat(x), seed)
        .startWith(seed);
}
```

Tie it all Together

```
var ThingStore = Dispatcher
    .filter(({tag}) => tag === 'THING_ADDED' || tag === 'THING_REMOVED')
    .pluck('data')
-   .scan(function(acc, {tag, data}) {
-       switch (tag) {
-           case 'THING_ADDED':
-               acc[data.id] = data;
-               return acc;
-           case 'THING_REMOVED':
-               delete acc[data.id];
-               return acc;
-           default:
-               return acc;
-       }
-   }, {})
+    .foldp(Collection)
    .shareReplay(1);
```

Blindly Dispatch those Types

The correct reduction logic is now implicit in the type.

```
send('THING_ADDED', Collection.Add({foo: 'Foo'}));
send('THING_ADDED', Collection.Add({bar: 'Bar'}));
send('THING_REMOVED', Collection.Del({bar: 'Bar'}));
```

Additional Reading

JavaScript Resources

Tom Harding's Fantas, Eel, and Specification

http://www.tomharding.me/

Professor Frisby's Mostly Adequate Guide to Functional Programming

https://drboolean.gitbooks.io/mostly-adequate-guide/content/