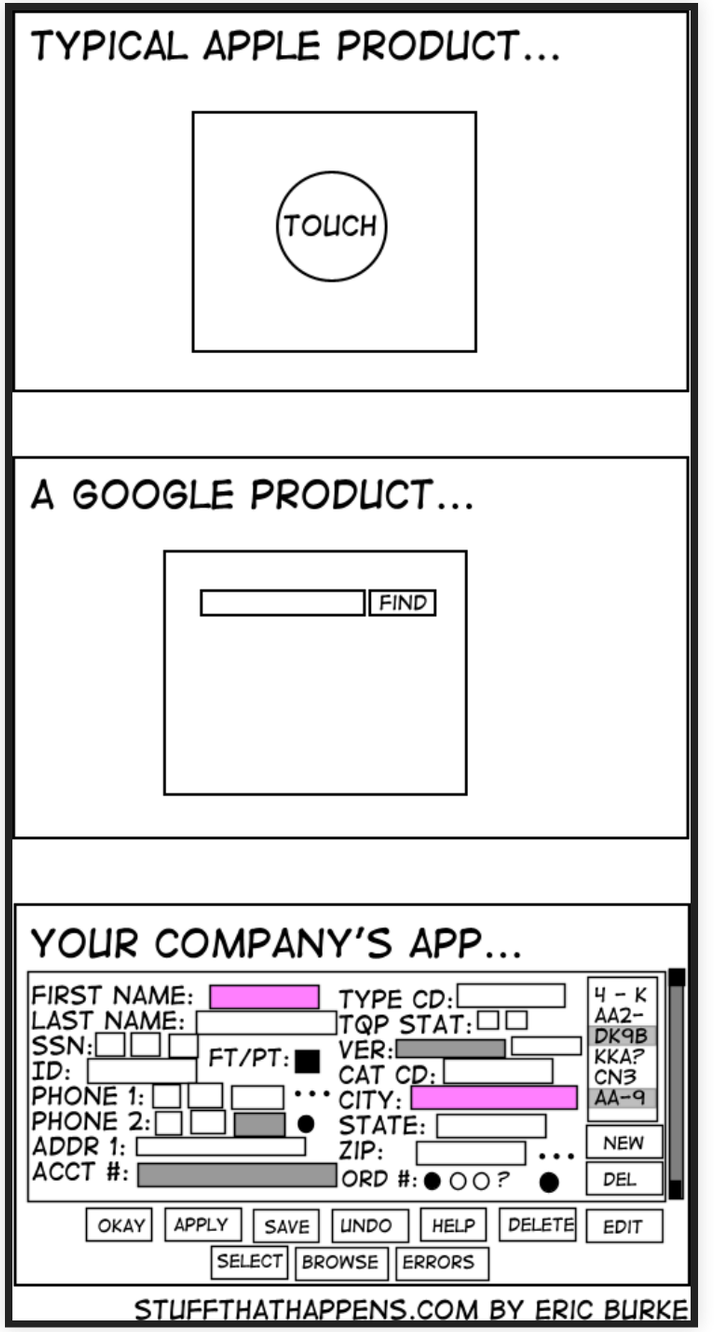
Observables and Monadic Streams

Building pipelines to tame complex data flows over time

Jeff Barczewski - 2019 Connect.tech

- @jeffbski
- jeff@codewinds.com
- https://codewinds.com/connect-rxjs-most


TYPICAL APPLE PRODUCT...
TOUCH
A GOOGLE PRODUCT...
FIND
YOUR COMPANY'S APP...
FIRST NAME:
LAST NAME:
SSN:
ID:
PHONE 1:
PHONE 2:
ADDR 1:
ACCT #:
FT/PT:
TYPE CD:
TQP STAT:
VER:
CAT CD:
CITY:
STATE:
ZIP:
ORD #:
4 - K
AA2-
DK9B
KKA?
CN3
AA-9
NEW
DEL
OKAY
APPLY
SAVE
UNDO
HELP
DELETE
EDIT
SELECT
BROWSE
ERRORS
STUFFTHATHAPPENS.COM BY ERIC BURKE

Jeff Barczewski

- Married, Father, Catholic
- 30 yrs (somewhat seasoned :-)
- JS (since 95, primary last 8 years)
- Work: USAF, RGA, MasterCard ApplePay, Elsevier, Monsanto, Sketch
- Founded CodeWinds, live/online training, mentoring, consulting
- Languages: Fortran, C, C++, Java, Ruby, JavaScript, Go, Rust

Modern Developer's Challenges

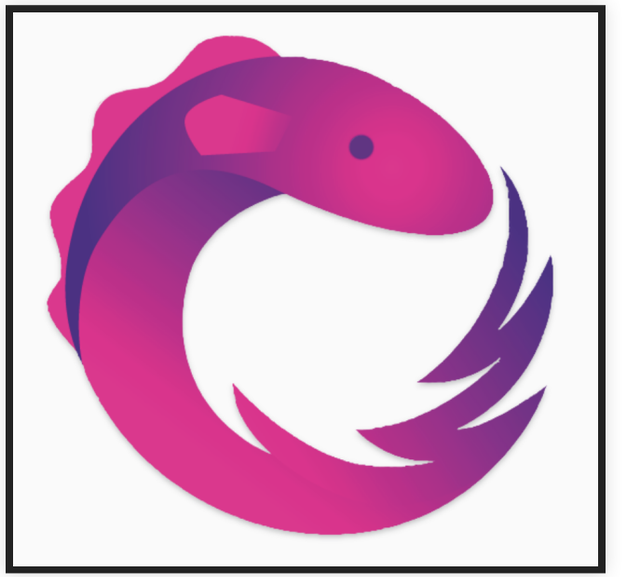
- events - inputs, environmental, gps, gryo, IoT, actions/commands
- async I/O
- state
- concurrency
- failures / reslilience
- cloud services - latency, concurrent limits, network failures
- streams of external data
- time, grouping
- cancellation
- dependencies

What is an Observable?

Observables are lazy Push collections of multiple values.

It can emit 0 to N values and can be cancelled.

- RxJS Main Docs: https://rxjs-dev.firebaseapp.com/
- RxJS API: https://rxjs-dev.firebaseapp.com/api
- Tutorial Content: https://www.learnrxjs.io/ (by Brian Troncone)
- RxJS Marbles Diagrams: https://rxmarbles.com/



Sneak Peak - RxJS Demo

- Auto search using RxJS Ajax with debouncing and cancellation of requests
  - https://stackblitz.com/edit/react-pdjjaj
- Drag and Drop using RxJS
  - https://stackblitz.com/edit/react-gaceaj

Observable.create

```
import {Observable} from 'rxjs';
const ob$ = Observable.create(obs => {
  // your code here
  obs.next('foo');
  obs.next('bar');
  obs.next('baz');
  // obs.error(Error('my error'))
  obs.complete();
  return () => { /* optional teardown logic */ }
});

ob$.subscribe({
  next: x => console.log(x), // foo, bar, baz
  error: err => console.log('error', err),
  complete: () => console.log('complete') // complete
});
// can also subscribe with 3 fns instead of an object
ob$.subscribe(
  x => console.log('ob$ next', x), /* next */
  err => console.log('ob$ error', err), /* error */
  () => console.log('ob$ complete') /* complete */
);
```

Other ways to create observables

```
import {of, throwError, from, pairs, interval, timer, fromEvent,
  EMPTY, NEVER} from 'rxjs';
import { ajax } from 'rxjs/ajax';
import { fromFetch } from 'rxjs/fetch';
import { webSocket } from 'rxjs/webSocket';

const ob$ = of(123, [3,4], {a:1}); // three events 123, [3,4], {a:1}
const ob$ = throwError(Error('my error'));

const ob$ = from(['a', 'b']); // two events a, b
const ob$ = from(promise); // 1 event when promise resolves
const ob$ = pairs({a: 123, b: 234}); // 2 events [a, 123], [b, 234]

const ob$ = interval(1000); // index increasing every 1s
const ob$ = timer(2000, 1000); // after 2s delay, idx every 1s
const ob$ = fromEvent(element, 'click'); // on every element click
const ob$ = EMPTY; // empty observable, ends with 0 events
const ob$ = NEVER; // empty observable that never ends

const ob$ = ajax.getJSON('https://myserver.com');
const ob$ = fromFetch('https://myserver.com');
const ob$ = webSocket({ url: 'ws://localhost:8010' });
```

Subject

```
import {Subject} from 'rxjs';

const subject$ = new Subject();

// subject$ is read and write
subject$
  .subscribe({
    next: x => console.log(x),
    error: err => console.error(err),
    complete: () => console.log('complete')
  });

// ob$ is only read
const ob$ = subject$.asObservable();
ob$.subscribe(console.log);

// emitting events, error, complete
subject.next(123);
subject.next('abc');
// subject.error(Error('my error'))
subject.complete();
```

Transforming events

```
import {of, map, filter, reduce, scan, tap, catchError} from 'rxjs/operators';
of(1, 2, 3).pipe(
  map(x => x * 10), // 10, 20, 30
  filter(x => x > 15), // 20, 30
  reduce((acc, x) => acc + x, 0) // 50
).subscribe(console.log); // 50
```

```
of(1, 2, 3).pipe(
  map(x => x * 10), // 10, 20, 30
  scan((acc, x) => acc + x, 0)
).subscribe(console.log); // 10, 30, 60
```

```
of(1, 2, 3).pipe(
  tap(x => console.log(x)), // 1, 2, 3
  map(x => x * 10), // 10, 20, 30
  tap({
    next: x => console.log(x), // 10, 20, 30
    error: err => console.error(err),
    complete: () => console.log('complete') // complete
  }),
  catchError(err => of({error: err.toString()}))
).subscribe();
```

Converting to and from observables

```
import {from, bindCallback, bindNodeCallback, fromEvent, of} from 'rxjs';
import {map, toArray} from 'rxjs/operators';

const ob$ = from(promise); // from promise to an observable
const prom = ob$.toPromise(); // from observable to a promise

const fn = x => x * 2;
const boundFn = bindCallback(fn);
const ob$ = boundFn(4); // 8

const readFileAsObservable = bindNodeCallback(fs.readFile);
const ob$ = readFileAsObservable('./file.txt', 'utf8');

const ob$ = fromEvent(element, 'click');

// collecting into an array and returning as promise
const arrProm = of(1, 2, 3).pipe(
  map(x => x * 10)
  toArray() // collects all into an array and emits
).toPromise();
```

Filtering Operators

```
import {of, timer} from 'rxjs';
import {filter, find, findIndex, first, last, take, skip, takeLast, takeWhile,
  skipWhile, distinct, distinctUntilChanged} from 'rxjs/operators';

// still need to call .subscribe on these
of(10, 20, 30).pipe(filter(x => x > 15)) // 20, 30
of(10, 20, 30).pipe(find(x => x > 15))  // 20
of(10, 20, 30).pipe(findIndex(x => x > 15))  // 1
of(10, 20, 30).pipe(first()) // 10
of(10, 20, 30).pipe(last()) // 30
of(10, 20, 30).pipe(take(2)) // 10, 20
of(10, 20, 30).pipe(skip(1)) // 20, 30of(10, 20, 30).pipe(takeLast(2)) // 20, 30
of(10, 20, 30).pipe(takeWhile(x => x < 25)) // 10, 20
of(10, 20, 30).pipe(skipWhile(x => x < 25)) // 30
of(10, 20, 10, 30).pipe(distinct()) // 10, 20, 30
of(10, 10, 20, 10, 30, 30, 40).pipe(distinctUntilChanged()) // 10, 20, 10, 30, 40
of(10, 10, 20, 10, 30, 30, 40).pipe(
  disctintUntilChanged((a, b) => a === b)) // 10, 20, 10, 30, 40

const timer$ = timer(1000);
ob$.pipe(takeUntil(timer$)) // take events until 1s timer fires
```

Time / Sequence

```
import {timeoutWith, debounceTime, throttleTime, auditTime, sampleTime,
  delay, bufferTime, bufferCount} from 'rxjs/operators';

ob$.pipe(
  timeoutWith(2000, of({ a: 123 })), // use {a:123} if no events in 2s
  debounceTime(100), // wait for pause, then emit most recent event
  throttleTime(300), // drop events that are too frequent, use first in series
  auditTime(300), // drop events that are too frequent, use last in series
  sampleTime(300), // send last new event every 300ms
  delay(1000), // delay events by 1s
  bufferTime(100), // collect values within time and send as array
  bufferCount(10) // collect values into arrays with at most 10 elements
).subscribe();
```

Combination

```
import {concat, merge, forkJoin, zip, combineLatest, race} from 'rxjs';

concat(of(10, 20), of('a', 'b')).subscribe() // 10, 20, a, b

merge(
  interval(600).pipe(map(() => 'A'))
  interval(1000).pipe(map(() => 'B'))
).subscribe() // A, B, A, A, B

forkJoin([ of(10, 20), of('a', 'b') ]).subscribe() // [20, b]

zip(of(10, 20), of('a', 'b')).subscribe() // [10, a], [20, b]

race(ob1$, ob2$).subscribe() // winner of which responds first

combineLatest(
  interval(600).pipe(map(x => `A${x}`))
  interval(1000).pipe(map(x => `B${x}`))
).subscribe() // [A0, B0], [A1, B0], [A2, B0], [A2, B1]
```

mergeMap, concatMap, switchMap

```
import {of} from 'rxjs';
import {mergeMap, concatMap, switchMap} from 'rxjs/operators';
import {ajax} from 'rxjs/ajax';

// order of results is based on response that returns first
of('react', 'redux').pipe(
  mergeMap(x => ajax.getJSON(`https://myserver.com/${x}`))
).subscribe();

// order is preserved, requests are started serially
of('react', 'redux').pipe(
  concatMap(x => ajax.getJSON(`https://myserver.com/${x}`))
).subscribe();

// only last request is delivered, previous is cancelled
of('react', 'redux').pipe(
  switchMap(x => ajax.getJSON(`https://myserver.com/${x}`))
).subscribe();
```

Live RxJS Demos

- Auto search using RxJS Ajax with debouncing and cancellation of requests https://stackblitz.com/edit/react-pdjjaj
- Run Concurrently - limit concurrency: https://stackblitz.com/edit/react-q7w4dc
- Chaining operations with grouping: https://stackblitz.com/edit/react-tsn2ru
- Drag and Drop: https://stackblitz.com/edit/react-gaceaj
- Auto search using Axios and debouncing https://stackblitz.com/edit/react-2o6qbw
- Auto search using Fetch and debouncing https://stackblitz.com/edit/react-cqwf8t

Resilience - retry

```
import { interval, of, throwError } from 'rxjs';
import { mergeMap, retry } from 'rxjs/operators';

interval(1000).pipe(
  mergeMap(val => {
    if(val > 3){
      return throwError('My Error');
    }
    return of(val);
  }),
  //retry 2 times on error
  retry(2)
).subscribe(); // 0, 1, 2, 3
               // 0, 1, 2, 3
               // 0, 1, 2, 3, My Error
```

Resilience - retryWhen

```
import {from, throwError, timer} from 'rxjs';
import {mergeMap, retryWhen} from 'rxjs/operators';

ob$.pipe(
  mergeMap(x => {
    return from(fetchData(x))
      .pipe(
        retryWhen(attempts => attempts.pipe(
          mergeMap((err, idx) => {
            // we can interrogate errors and handle differently
            if (err.statusCode === 500) return throwError(err);

            // can allow N number of retries before failing
            if (idx > 3) return throwError(Error('max retries exceeded'));

            // can delay the retry linearly or based on index calculation
            return timer(idx * 1000); // delay backoff
          })
        ))
      );
  })
).subscribe();
```

Most.js - Monadic Event Streams

High-performance reactive event stream programming that powers Most.js.

@most/core features Most's battle-tested, high-performance architecture in a leaner, functions-only, curried API in a tree-shakeable package. It helps you write highly interactive apps by composing event streams, and without many of the hazards of side effects and mutable shared state.

- @most/core use for new projects: https://github.com/mostjs/core
- @most/core API: https://mostcore.readthedocs.io/en/latest/
- Community code: https://github.com/mostjs-community

Differences from Observables

- High performance - 2-100x faster depending on operations. RxJS@6 is fast, but Most.js is even faster. Benchmark: https://github.com/mostjs/core/tree/master/packages/core/test/perf
- Smaller feature set
- Scheduler clock for app, time is passed with event.
- Most.js 1.x had Fantasy Land support, @most/core is planning on adding Static Land support
- Less popularity and contributors. Also less examples and articles available.
- runEffects and run vs subscribe
- npm install @most/core @most/scheduler

@most/core creating streams

```
import {now, empty, never, at, periodic, throwError} from '@most/core';
import { newDefaultScheduler, currentTime } from "@most/scheduler";
import { create as createSubject } from "most-subject";

now('a'); // create stream with single event starting now
empty(); // create empty stream which ends immediatly
never(); // create empty stream which never ends
at(123, 'a'); // create stream with event 'a' at time 123ms
periodic(100); // create stream that emits every 100ms
throwError(Error('my error')); // create stream with an error
fromPromise(promise); // create stream from resolution of promise

const scheduler = newDefaultScheduler(); // create one for app

const [subject, stream] = createSubject();
subject.event(currentTime(scheduler), "aa");
subject.event(currentTime(scheduler), "bb");
// subject.error(currentTime(scheduler), Error('my error'));
subject.end(currentTime(scheduler));
```

@most/core stream usage

```
import {now, empty, never, at, periodic, throwError} from '@most/core';
import { newDefaultScheduler, currentTime } from "@most/scheduler";
import { create as createSubject } from "most-subject";
import { pipe } from 'lodash/fp';

const scheduler = newDefaultScheduler(); // create one for app

const stream = pipe(
  tap(x => console.log(x)),
  debounce(100),
  skipRepeats,
  filter(x => x < 1000),
  map(x => x * 10),
  chain(x => fromPromise(fetchData(x))), // like mergeMap
  scan((acc, x) => acc + x, 0),
  tap(sum => console.log('sum', sum))
)(now(123));

runEffects(stream, scheduler);
```

Converting from promise and an ES Observable

```
import {fromPromise} from '@most/core';
import fromObservable from 'most-observable';

const stream = fromPromise(promise);
const stream = fromObservable(esObservable)
```

Converting stream back to a promise

```
import { run } from '@most/core';
import { curry2 } from '@most/prelude';
import { newDefaultScheduler } from "@most/scheduler";

const scheduler = newDefaultScheduler(); // create one for app

const toPromise = curry2((scheduler, mostStream) => {
  return new Promise((resolve, reject) => {
    let lastValue;
    const sink = {
      event(t, x) { lastValue = x; },
      end(t) { resolve(lastValue); },
      error(t, err) { reject(err); }
    };
    run(sink, scheduler, mostStream);
  });
});

const promise = toPromise(scheduler, stream);
```

Converting stream back to RxJS Observable

```
import { Observable } from "rxjs";
import { curry2 } from "@most/prelude";
import { run } from "@most/core";

const scheduler = newDefaultScheduler(); // create one for app

const toObservable = curry2((scheduler, mostStream) => {
  const observable = Observable.create(observer => { // on subscription
    const sink = {
      event(t, x) { observer.next(x); },
      end(t) { observer.complete(); },
      error(t, err) { observer.error(err); }
    };
    const disposable = run(sink, scheduler, mostStream);
    return () => { // on unsubscribe
      disposable.dispose();
    };
  });
  return observable;
});
const ob$ = toObservable(scheduler, stream);
```

Live Most.js Demos

- toPromise and toObservable: https://stackblitz.com/edit/js-vx457x
- auto search axios: https://stackblitz.com/edit/react-zvrrgp

Thanks

- https://codewinds.com/connect-rxjs-most (slides, resources)
- https://codewinds.com/ (blog, tips/training, consulting)
- jeff@codewinds.com
- @jeffbski