Cool RxJS Tricks

UtahJS Conf 2017

by Seth House @whiteinge

http://talks.eseth.com/#rxjs-tricks

Reactive Extensions

Who is this Talk For?

- This is not an introduction!

Who is this Talk For?

- This is not an introduction!
- We're going to go fast. http://talks.eseth.com/#rxjs-tricks

Who is this Talk For?

- This is not an introduction!
- We're going to go fast. http://talks.eseth.com/#rxjs-tricks
- Crucial to have self-motivation when diving into this topic.

Who is this Talk For?

- This is not an introduction!
- We're going to go fast. http://talks.eseth.com/#rxjs-tricks
- Crucial to have self-motivation when diving into this topic.

Presenter's $0.02:

- Mature, stable, six-year-old codebase.

Who is this Talk For?

- This is not an introduction!
- We're going to go fast. http://talks.eseth.com/#rxjs-tricks
- Crucial to have self-motivation when diving into this topic.

Presenter's $0.02:

- Mature, stable, six-year-old codebase.
- Ideas and techniques spanning many languages and implementations.

Who is this Talk For?

- This is not an introduction!
- We're going to go fast. http://talks.eseth.com/#rxjs-tricks
- Crucial to have self-motivation when diving into this topic.

Presenter's $0.02:

- Mature, stable, six-year-old codebase.
- Ideas and techniques spanning many languages and implementations.
- Rx knowledge will make you unafraid of async.

Who is this Talk For?

- This is not an introduction!
- We're going to go fast. http://talks.eseth.com/#rxjs-tricks
- Crucial to have self-motivation when diving into this topic.

Presenter's $0.02:

- Mature, stable, six-year-old codebase.
- Ideas and techniques spanning many languages and implementations.
- Rx knowledge will make you unafraid of async.
- Rx knowledge will show you a new way to compose behavior and programs.

A Note on RxJS Versions

RxJS 4

- Maintenance-only.
- Rock solid.
- Great performance.
- Maintained for the foreseeable future.

RxJS 5

- Better performance.
- Better stack traces.
- Test with marble diagrams!
- See the "Migrating From" doc.

Observables

Rx.DOM.ajax()

Like a Promise but *lazy* and *cancelable*.

```
var users = Rx.DOM.ajax('https://api.github.com/users');
var sub = users.subscribe(
    x => console.log('users', x),
    err => console.log('error', err),
    () => console.log('completed'));
```

A Shorthand: `logOb()`

```
// logOb :: String -> Observer
function logOb(name) {
    return {
        onNext: x => console.log(`${name} next:`, x),
        onError: err => console.log(`${name} error:`, err),
        onCompleted: () => console.log(`${name} completed`),
    };
}
```

Lazy

```
var users = Rx.DOM.ajax({
    url: 'https://api.github.com/users',
    responseType: 'json',
});
```

Lazy

```
var users = Rx.DOM.ajax({
    url: 'https://api.github.com/users',
    responseType: 'json',
});
```

```
// Observable({response: Array(30), status: 200, xhr: XMLHttpRequest, ...})
```

Lazy

```
var users = Rx.DOM.ajax({
    url: 'https://api.github.com/users',
    responseType: 'json',
});
```

```
// Observable({response: Array(30), status: 200, xhr: XMLHttpRequest, ...})
```

```
var userCount = users.pluck('response', 'length');
```

Lazy

```
var users = Rx.DOM.ajax({
    url: 'https://api.github.com/users',
    responseType: 'json',
});
```

```
// Observable({response: Array(30), status: 200, xhr: XMLHttpRequest, ...})
```

```
var userCount = users.pluck('response', 'length');
```

```
var sub = userCount.subscribe(logOb('count'));
```

Ajax

A Shorthand: `xhr()`

```
function xhr(url, ...params) {
    return Rx.DOM.ajax({
        url,
        responseType: 'json',
        ...params,
    })
    .pluck('response');
}
```

Combine Requests

```
var users = xhr('/users');
var repos = xhr('/repos');
```

Combine Requests

```
var users = xhr('/users');
var repos = xhr('/repos');
```

```
// Grab both immediately and emit each whenever.
users.merge(repos).subscribe(logOb('x'))
```

Combine Requests

```
var users = xhr('/users');
var repos = xhr('/repos');
```

```
// Grab both immediately and emit each whenever.
users.merge(repos).subscribe(logOb('x'))
```

```
// Grab both immediately and emit each in order.
users.concat(repos).subscribe(logOb('x'))
```

Combine Requests

```
var users = xhr('/users');
var repos = xhr('/repos');
```

```
// Grab both immediately and emit each whenever.
users.merge(repos).subscribe(logOb('x'))
```

```
// Grab both immediately and emit each in order.
users.concat(repos).subscribe(logOb('x'))
```

```
// Grab both immediately and emit once together.
// (Unbounded; careful of uneven producers.)
users.zip(repos).subscribe(logOb('x'))
```

Combine Requests

```
var users = xhr('/users');
var repos = xhr('/repos');
```

```
// Grab both immediately and emit each whenever.
users.merge(repos).subscribe(logOb('x'))
```

```
// Grab both immediately and emit each in order.
users.concat(repos).subscribe(logOb('x'))
```

```
// Grab both immediately and emit once together.
// (Unbounded; careful of uneven producers.)
users.zip(repos).subscribe(logOb('x'))
```

```
// Grab both immediately and emit once together.
// (Cachest last emit from each producer.)
users.combineLatest(repos).subscribe(logOb('x'))
```

Sequential Requests

```
// Get all users:
var allUsers = xhr('/users');
```

Sequential Requests

```
// Get all users:
var allUsers = xhr('/users');
```

```
// Grab the first user:
var firstUser = allUsers
    .flatMap(userList => xhr(userList[0].url));
```

Sequential Requests

```
// Get all users:
var allUsers = xhr('/users');
```

```
// Grab the first user:
var firstUser = allUsers
    .flatMap(userList => xhr(userList[0].url));
```

```
// Grab that user's repos:
var userRepos = firstUser
    .flatMap(userDetails => xhr(userDetails.repos_url));
```

Sequential Requests

```
// Get all users:
var allUsers = xhr('/users');
```

```
// Grab the first user:
var firstUser = allUsers
    .flatMap(userList => xhr(userList[0].url));
```

```
// Grab that user's repos:
var userRepos = firstUser
    .flatMap(userDetails => xhr(userDetails.repos_url));
```

```
// Grab the first repo for that user:
var firstUserRepo = userRepos
    .flatMap(userReposList => xhr(userReposList[0].url));
```

Sequential Requests

```
// Get all users:
var allUsers = xhr('/users');
```

```
// Grab the first user:
var firstUser = allUsers
    .flatMap(userList => xhr(userList[0].url));
```

```
// Grab that user's repos:
var userRepos = firstUser
    .flatMap(userDetails => xhr(userDetails.repos_url));
```

```
// Grab the first repo for that user:
var firstUserRepo = userRepos
    .flatMap(userReposList => xhr(userReposList[0].url));
```

```
// Grab the details for that repo:
var sub = firstUserRepo
    .subscribe(logOb('User's first repo details'));
```

Sequential Requests (together)

Get details for the first user's first repo:

```
xhr('/users')
    .flatMap(userList => xhr(userList[0].url)
        .flatMap(userDetails => xhr(userDetails.repos_url)
            .flatMap(userReposList => xhr(userReposList[0].url))))
    .subscribe(logOb('First user repo details'));
```

Coordinate Many Requests

But what if we wanted all repos from all users?

Coordinate Many Requests

But what if we wanted all repos from all users?

```
xhr('/users')
    .flatMap(userList => Rx.Observable.from(userList))
    .flatMap(user => xhr(user.url)
        .flatMap(userDetails => xhr(userDetails.repos_url)
            .flatMap(userReposList => Rx.Observable.from(userReposList))
            .flatMap(repo => xhr(repo.url))));
```

Coordinate Many Requests

But what if we wanted all repos from all users?

```
xhr('/users')
    .flatMap(userList => Rx.Observable.from(userList))
    .flatMap(user => xhr(user.url)
        .flatMap(userDetails => xhr(userDetails.repos_url)
            .flatMap(userReposList => Rx.Observable.from(userReposList))
            .flatMap(repo => xhr(repo.url))));
```

```
 xhr('/users')
-    .flatMap(userList => xhr(userList[0].url)
+    .flatMap(userList => Rx.Observable.from(userList))
+    .flatMap(user => xhr(user.url)
         .flatMap(userDetails => xhr(userDetails.repos_url)
-            .flatMap(userReposList => xhr(userReposList[0].url))))
+            .flatMap(userReposList => Rx.Observable.from(userReposList))
+            .flatMap(repo => xhr(repo.url))));
```

Limit Concurrent Requests

```
 xhr('/users')
     .flatMap(userList => Rx.Observable.from(userList))
-    .flatMap(user => xhr(user.url)
+    .flatMapWithMaxConcurrent(5, user => xhr(user.url)
         .flatMap(userDetails => xhr(userDetails.repos_url)
             .flatMap(userReposList => Rx.Observable.from(userReposList))
-            .flatMap(repo => xhr(repo.url))));
+            .flatMapWithMaxConcurrent(5, repo => xhr(repo.url))));
```

Rate-limit Outgoing Requests

```
 xhr('/users')
     .flatMap(userList => Rx.Observable.from(userList))
+     .map(x => Rx.Observable.of(x).delay(1000))
+     .concatAll()
     .flatMapWithMaxConcurrent(5, user => xhr(user.url)
         .flatMap(userDetails => xhr(userDetails.repos_url)
             .flatMap(userReposList => Rx.Observable.from(userReposList))
+             .map(x => Rx.Observable.of(x).delay(1000))
+             .concatAll()
             .flatMapWithMaxConcurrent(5, repo => xhr(repo.url))));
```

A Shorthand: "lettable" Functions

```
function listToRateLimitedStream(delayBy = 1000) {
  return o => o
    .flatMap(xs => Rx.Observable.from(xs))
    .map(x => Rx.Observable.of(x).delay(delayBy))
    .concatAll();
}
```

A Shorthand: "lettable" Functions

```
function listToRateLimitedStream(delayBy = 1000) {
    return o => o
        .flatMap(xs => Rx.Observable.from(xs))
        .map(x => Rx.Observable.of(x).delay(delayBy))
        .concatAll();
}
```

```diff
 xhr('/users')
-    .flatMap(userList => Rx.Observable.from(userList))
-    .map(x => Rx.Observable.of(x).delay(1000))
-    .concatAll()
+    .let(listToRateLimitedStream(1000))
     .flatMapWithMaxConcurrent(5, user => xhr(user.url)
         .flatMap(userDetails => xhr(userDetails.repos_url)
-            .flatMap(userReposList => Rx.Observable.from(userReposList))
-            .map(x => Rx.Observable.of(x).delay(1000))
-            .concatAll()
+            .let(listToRateLimitedStream(1000))
             .flatMapWithMaxConcurrent(5, repo => xhr(repo.url))));
```

Send Different Data Through Different Streams

```
var obs = xhr('/users')
    // ...snip...
    .share();

var [superStars, regularStars] = obs
    .partition(repoDetails => repoDetails.stars > 1500);

obs.count()
    .subscribe(logOb('Total repos'));
regularStars.count()
    .subscribe(logOb('Regular repos'));
superStars.reduce(countTimesSeenUser, {})
    .subscribe(logOb('Super stars'));


function countTimesSeenUser(acc, repo) {
    acc[repo.userId] = (acc[repo.userId] || 0) + 1;
    return acc;
}
```

Ajax Failures

Catch & Replace Errors

```diff
 xhr('/users')
     .let(listToRateLimitedStream(1000))
     .flatMapWithMaxConcurrent(5, user => xhr(user.url)
+        .catch(() => Rx.Observable.empty())
         .flatMap(userDetails => xhr(userDetails.repos_url)
+            .catch(() => Rx.Observable.empty())
             .let(listToRateLimitedStream(1000))
-            .flatMapWithMaxConcurrent(5, repo => xhr(repo.url))));
+            .flatMapWithMaxConcurrent(5, repo => xhr(repo.url)
+                .catch(() => Rx.Observable.empty()))));
```

Timeout Long Requests

```
 xhr('/users')
     .let(listToRateLimitedStream(1000))
     .flatMapWithMaxConcurrent(5, user => xhr(user.url)
+        .timeout(15000)
         .catch(() => Rx.Observable.empty())
         .flatMap(userDetails => xhr(userDetails.repos_url)
+            .timeout(15000)
             .catch(() => Rx.Observable.empty())
             .let(listToRateLimitedStream(1000))
             .flatMapWithMaxConcurrent(5, repo => xhr(repo.url)
+                .timeout(15000)
                 .catch(() => Rx.Observable.empty()))));
```

Retrying Failures

```
 xhr('/users')
     .let(listToRateLimitedStream(1000))
     .flatMapWithMaxConcurrent(5, user => xhr(user.url)
         .timeout(15000)
+        .retryWhen(no => no.flatMap(() => Rx.Observable.timer(1000)))
         .catch(() => Rx.Observable.empty())
         .flatMap(userDetails => xhr(userDetails.repos_url)
             .timeout(15000)
+            .retryWhen(no => no.flatMap(() => Rx.Observable.timer(1000)))
             .catch(() => Rx.Observable.empty())
             .let(listToRateLimitedStream(1000))
             .flatMapWithMaxConcurrent(5, repo => xhr(repo.url)
                 .timeout(15000)
+                .retryWhen(no => no.flatMap(() => Rx.Observable.timer(1000)))
                 .catch(() => Rx.Observable.empty()))));
```

A Shorthand: Retry

```
function retry({
        timeout = 15000,
        retry = 10000,
    }) {
    return o => o
        .timeout(timeout)
        .retryWhen(no => no.flatMap(() => Rx.Observable.timer(retry)))
        .catch(() => Rx.Observable.empty());
}
```

Retry -- Forever...

```
 xhr('/users')
     .let(listToRateLimitedStream(1000))
     .flatMapWithMaxConcurrent(5, user => xhr(user.url)
-        .timeout(15000)
-        .retryWhen(no => no.flatMap(() => Rx.Observable.timer(1000)))
-        .catch(() => Rx.Observable.empty())
+        .let(retry({retry: 1000}))
         .flatMap(userDetails => xhr(userDetails.repos_url)
-            .timeout(15000)
-            .retryWhen(no => no.flatMap(() => Rx.Observable.timer(1000)))
-            .catch(() => Rx.Observable.empty())
+            .let(retry({retry: 1000}))
             .let(listToRateLimitedStream(1000))
             .flatMapWithMaxConcurrent(5, repo => xhr(repo.url)
-                .timeout(15000)
-                .retryWhen(no => no.flatMap(() => Rx.Observable.timer(1000)))
-                .catch(() => Rx.Observable.empty())))));
+                .let(retry({retry: 1000})))));
```

Retry with Backoff

```
function retry({
        timeout = 15000,
        retry = 10000,
+        backoffMax = 5,
    }) {
    return o => o
        .timeout(timeout)
-        .retryWhen(no => no.flatMap(() => Rx.Observable.timer(retry)))
+        .retryWhen(no => no
+            .scan(count => count + 1, 0)
+            .flatMap(function(i) {
+                var backoff = i < backoffMax
+                    ? i * retry
+                    : backoffMax * retry;
+                return i >= backoffMax
+                    ? Rx.Observable.throw('Giving up')
+                    : Rx.Observable.timer(backoff);
+            }))
        .catch(() => Rx.Observable.empty());
}
```

Ajax Progress Events

Track all Ajax Requests

```
+var XhrProgress = new Rx.Subject();
+
 function xhr(url, ...params) {
+   var rqid = Symbol(url);
+   var sendProg = type => ev => XhrProgress.onNext({
+       url,
+       params,
+       xhr: ev ? ev.target : null,
+       ev,
+       rqid,
+       type,
+   });
+
    return Rx.DOM.ajax({
        url,
        responseType: 'json',
+       progressObserver: {
+           onNext: sendProg('next'),
+           onError: sendProg('error'),
+           onCompleted: sendProg('completed'),
+       },
        ...params,
    })
```

Watch for Server Errors

```
var serverErrors = XhrProgress
    .pluck('xhr', 'status')
    .filter(status => status === 500);

serverErrors.subscribe(logOb('Seen error'));
```

Track All In-Flight Requests

```
var inFlight = XhrProgress.scan(function(active, x) {
    if (x.type === 'next') active.set(x.rqid, x);
    else active.delete(x.rqid);
    return active;
}, new Map());
```

Poll Until Condition

```
xhr('/some/path')
    .repeatWhen(no => no.delay(10000))
    .takeUntil(XhrProgress
        .filter(x => x.path === '/some/path')
        .pluck('xhr', 'response')
        .filter(resp => resp != null && resp !== ''));
```

Cache Responses

```
xhr('/some/path').shareReplay(1);
```

Render a Page

Shorthand: Wrap in a Function

...or don't. Embrace laziness & use function params for parse-time configuration (not run-time!).

```
+function getAllUserRepos(rateLimit = 1000, retryAfter = 1000) {
-       .let(listToRateLimitedStream(1000))
+       .let(listToRateLimitedStream(rateLimit))
-             .let(retry({retry: 1000}))
+             .let(retry({retry: retryAfter}))
        .flatMap(userDetails => xhr(userDetails.repos_url)
-                .let(listToRateLimitedStream(1000))
+                .let(listToRateLimitedStream(rateLimit))
           .flatMapWithMaxConcurrent(5, repo => xhr(repo.url)
+                   .let(retry({retry: 1000}))
+                   .let(retry({retry: retryAfter}))));
+
+           return getRepos.map(repo => ({repo, user}));
+       });
+}
```

Data Flow

```
getAllUserRepos()
    .scan(sumStarsForUser, {})
    .map(users => Object.values(users).sort((a, b) => a.id - b.id))
    .map(visualizeStarsForUser)
    .subscribe(render('body'));
```

Aggregate Results

```
function sumStarsForUser(acc, {user, repo}) {
    acc[user.id] = acc[user.id] || user;
    acc[user.id].stars = (acc[user.id].stars || 0) + repo.stars;
    return acc;
}
```

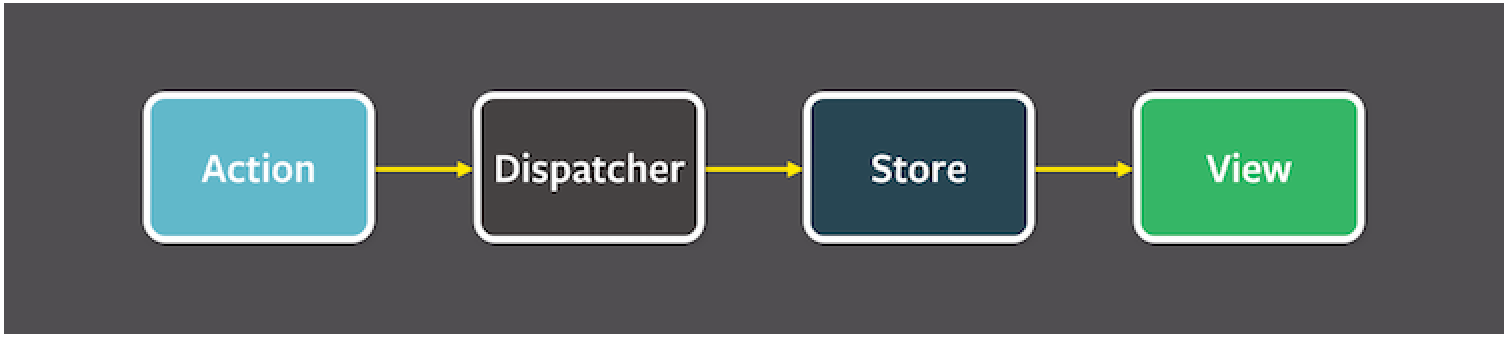
Format Results

```
function visualizeStarsForUser(userList) {
    var repoBoxes = userList.map(function(user) {
        return `
            <div style="float: left; width: 64px; text-align: center; padding: 2px;">
                <img src="data:image/png;base64,${user.avatar}" alt="${user.id}">
                <span>${user.stars} ★</span>
            </div>
        `;
    });
    return repoBoxes.join('');
}
```

Manage Side Effects

```
function render(container) {
    var el = document.querySelector(container)
    return function(content) {
        el.innerHTML = content;
    }
}
```

Flux



Flux implemented in Rx

```
var Actions = new Rx.Subject();
var send = tag => data => Action.onNext({tag, data});

var Dispatcher = Actions
    .asObservable();
    .share();

var userStarsStore = Dispatcher
    .filter(x => x.tag === 'USER_STARS')
    .flatMap(getAllUserRepos())
    .scan(sumStarsForUser, {})
    .map(users => Object.values(users).sort((a, b) => a.id - b.id));

var userStarsView = userStarsStore
    .map(visualizeStarsForUser);

var mainSubscription = theApp
    .subscribe(render('body'));
```

Misc

Komami Code

```
// up, up, down, down, left, right, left, right, b, a
var konami = Rx.Observable.from([38, 38, 40, 40, 37, 39, 37, 39, 66, 65]);

var result = $('#result');

Rx.Observable.fromEvent(window, 'keyup')
    .map(ev => ev.keyCode)
    .windowWithCount(10, 1) // get the last 10 keys
    .selectMany(x => x.sequenceEqual(konami)) // compare konami sequence
    .filter(x => x) // where we match
    .subscribe(logOb('KONAMI!'));
```

Drag and Drop

```
var mouseup = Rx.Observable.fromEvent(dragTarget, 'mouseup'),
    mousemove = Rx.Observable.fromEvent(document, 'mousemove'),
    mousedown = Rx.Observable.fromEvent(dragTarget, 'mousedown');

var mousedrag = mousedown.flatMap(function(md) {
    // calculate offsets when mouse down
    var startX = md.offsetX, startY = md.offsetY;

    // Calculate delta with mousemove until mouseup
    return mousemove.map(function(mm) {
        mm.preventDefault();
        return {
            left: mm.clientX - startX,
            top: mm.clientY - startY
        };
    })
    .takeUntil(mouseup);
});

var subscription = mousedrag.subscribe(function(pos) {
    dragTarget.style.top = pos.top + 'px';
    dragTarget.style.left = pos.left + 'px';
});
```

What is Reactive Extensions?

More than "Lodash for async".

A Common Specification

A lingua franca across twenty languages/environments.

http://reactivex.io/languages.html

C#, C# (Unity), C++, Clojure, Dart, Elixir, Go, Groovy, JRuby, Java, JavaScript, Kotlin, Lua, PHP, Python, Ruby, RxAndroid, RxCocoa, RxNetty, Scala, Swift

Rich Heritage of Ideas

ReactiveX, LINQ, Haskell, Category Theory.

What is Rx?

- Unified API for async operations.

What is Rx?

- Unified API for async operations.
- Both the consumer and the producer are in control.

What is Rx?

- Unified API for async operations.
- Both the consumer and the producer are in control.
- Resource allocation and automatic cleanup.

What is Rx?

- Unified API for async operations.
- Both the consumer and the producer are in control.
- Resource allocation and automatic cleanup.
- Subscription tracking.

What is Rx?

- Unified API for async operations.
- Both the consumer and the producer are in control.
- Resource allocation and automatic cleanup.
- Subscription tracking.
- Useful for: composition, data flow, messaging, state management, async, streams, processes, threads, coroutines, workers.

Resources

Learning Rx Advice

http://talks.eseth.com/#rxjs-tricks

- Pull up a browser console and *try* it!
- Use Subjects for slow, deliberate, step-by-step experiments.
- Keep experiments small and focused.

Learning Rx Resources

First Steps

- Introduction to Reactive Programming
- Subjects and multicasting
- Learning Observable By Building Observable

Find the Right Operator

- Which RxJS creation operator?
- Which RxJS instance operator?
- Broad Rx Decision Tree
- The Big List of Operators (TM)

Fun

- Snake
- State Management and Animations