RxJS Operators

Introduction

RxJS Operators © 2018, Code Whisperer Enterprises

Why This Course?

- Contains instructional videos for every RxJS operator – perfect place to become an RxJS expert
- RxJS operator skills are useful in Angular 2 apps and many other applications
- Various skills learned here will make you better at programming in general
- As compact as possible
- Easy and accessible pair-programming format
- 75% of videos are code-along demonstrations
- Money back guarantee!

What Will You Learn?

- How to use every RxJS operator – combination, filtering, creation, and more
- Maximally non-trivial use cases for the various operators
- How to implement common data structures like Redux using observables
- Useful Node.js development techniques

Tips for Coding Along at Home

- Demonstration will go quickly – pause whenever you need to catch up
- If you are stuck, try copying the code directly from the course files or the GitHub repository: `https://github.com/danielstern/rxjs-operators`
- Some Node packages are required for demo and version numbers are important
  - If stuck, clone the entire GitHub repository, and run npm install. This will update give you a local repository with correct, up-to-date plugins.
  - Global dependencies may require manual global installation, i.e, `npm install –g webpack-dev-server@2.11.1`

Stop! Before You Begin!

- We are assuming you have a basic knowledge of RxJS and have worked with observables at least a few times
- We will not be discussing basic concepts such as…
  - What is RxJS?
  - What are observables?
- If not, it is strongly recommended you watch RxJS – Mastering Observables (2017, Code Whisperer Enterprises) on Udemy
- 66% Off Discount Available With this Code: `https://www.udemy.com/rxjs-101/?couponCode=OPERATORS`

Basic Operators (Operators 101)

Setting Up The Project

- Get our IDE of choice open
  - Feel free to use any text editor or Integrated Development Environment you wish
  - WebStorm is recommended paid solution
    - I will be using it throughout this course
  - Atom is a good free option
- Install global dependencies and make sure we can run and ES6 Node Script
  - If we can, we're good to go!

range (start, length)

- Creates an observable that synchronously emits *length* integers starting at *start*
- For example, *range(1,3)* emits 1, 2, 3
- A useful starting point

of (… things)

- Takes any number of arguments and returns an observable that emits them one after the other

from (…)

- Converts an array, promise or iterator into an observable
    - **fromPromise can be used specifically for promises*

interval (…) & timer (…)

<u>interval (duration)</u>

- Emits a value each time the specified duration passes
- Emits the numbers 0, 1, 2, 3, (etc…)

<u>timer (duration, [interval])</u>

- Emits once after the specified duration has passed
- If a second argument is passed, it will then emit each time that interval passes, indefinitely

empty ()

- A static operator
- Creates an observable which completes immediately and returns no values
- Useful for testing, corner cases

map and mapTo

map

- Equivalent to JavaScript's array.prototype.map
- Converts each element to something new based on provided *mutator*

mapTo

- Converts each emitted value into a new value, without regard for the emitted value

filter

- Equivalent to array.prototype.filter
- Creates an observable that only emits the latest value from the source observable if it passes a predicate function

do

- Does a thing
- Discreetly executes a side-effect such as a *console.log()* statement
- More complex side-effects (API calls, etc.) should not be handled with *do()*
- Receives the last emitted value as an argument, but doesn’t return anything (last emitted value is passed to new operator automatically)
- Can’t change the emitted value

pluck

- Equivalent to Lodash’s *pluck*
- Used to map an observable of similar objects to a single property of those objects
- A string, not a function, is provided

first

- Roughly equivalent to array.prototype.find
- Creates an observable which completes as soon as the source observable emits an acceptable value
- Useful for extracting a value from an observable that will not complete, or that will take a long time to complete

startWith

- Creates a new observable that emits a provided value, then emits values from the source observable
- Useful for asynchronous observables that may not return a value for some time

create (…)

- Creates a new observable which emits, completes and errors under custom circumstances
- Powerful, but executing too much code inside *create* is an anti-pattern
- Remember, when you have a hammer…
  - Everything looks like a nail!

every

- Equivalent to array.prototype.every
- Emits true if each element emitted by the source array passed a provided predicate function
- Only emits after the source completes (singular)

distinctUntilChanged

- Creates an observable which only emits the latest value from the source observable if it is different than the one before it
- Useful for an observable that tends to emit the same value many times in a row

defaultIfEmpty

- Creates an observable that, if the source observable completes before emitting any values, emits the provided value
- Has no effect if the source observable emitted any values

Intermediate Operators (Operators 201)

delay & delayWhen

delay

- Emits values from the source array only after a specified duration has passed
- Duration is specified as a number

delayWhen

- Like delay
  - instead of a number, an observable which emits after the duration of the delay is provided

throw (...)

- Creates an observable which immediately enters an error state while emitting no values
- Useful for testing error handling

take

take

- Emits only the first few values of the source observable
- Number of emitted values is specified by provided number

takeWhile

- Like take, but emits values from the source only until a provided predicate returns false
- Passing values subsequent to the first failing value will not be emitted
  - This is unlike filter

takeUntil

- Like take, but emits values from the source observable only until provided observable emits
  - Common example: timer

skip

skip

- Ignores the first few elements of a source observable
- Number provided as argument determines how many are skipped

skipWhile

- Ignores elements from a source observable until a provided predicate function returns *false*

skipUntil

Ignores elements from a source observable until a provided observable emits a value

last

- Returns the last element of a source observable to pass a predicate, after that observable completes
- Unlike *first*, source must complete

concat

- Loosely equivalent to array.prototype.concat
- Creates an observable which emits all values from a source observable, then emits all values from a provided observable

concatAll

- When a source observable emits other observables, subscribe to each one and emit its values
- Does not subscribe to one observable until the previous one completes

concatMap & concatMapTo

concatMap

- Like map, but the value returned from the mutator must be an observable
- The observable returned from the mutator is subscribed to
  - Results are passed to the next observer

concatMapTo

- Like concatMap, but maps to a constant observable with no regard for the incoming values

single

- Emits just one value which passes a predicate function, after the source observable completes
- If more than one value passes the predicate, an error will be thrown
  - This is unlike *first*

ignoreElements

- Doesn’t emit and values from source observable, but does emit an error or complete state from the source
- Usage is obscure

sample

- Emits the latest element from the source observable at a specified interval
- Useful if the frequency at which new elements are added, and the frequency as which you need to access elements, vary greatly
- *Not* equivalent to *_.sample*
    - _.sample returns a random element from an array

reduce & scan

reduce

- Equivalent to array.prototype.reduce
- Aggregate all the elements of an observable after it completes

scan

- Every time the source observable emits, aggregate all the values so far and emit the aggregated value
  - Like reduce, but emits multiple times

groupBy

- After the source observable completes…
  - Separate all the emitted values into groups based on an accessor
  - Emit each of those groups as an observable

timeout

- Creates an observable that throws an error if the source observable waits longer than the specified duration to emit two consecutive values
- Once source completes, timeout no longer applies

fromEvent(…)

- Creates an observable which emits values as they come in from a generic event source
- Event source can be many common JavaScript form controls…
  - Button
  - Text input
  - Other DOM events

merge

merge

- Creates new observable which combines the source and provided observable
- Works like concat, but all observables are subscribed to at once
    - Does not wait for previous observable to complete to start next one
- Hard to determine post-merge what the source was

mergeAll(…)

- Merges all provided observables

mergeMap

- If the source observable emits observables, continuously subscribe to those and emit any value that comes from any of them
- Subscriber doesn’t know when a new observable has been merged in

buffer

buffer

- Collects values from source observable until provided observable emits
- Provided observable can emit anything
- Collected values are emitted as an array
- Starts buffering again immediately

bufferCount

- Like buffer, but waits until a specified number of values are emitted from source before emitting buffered values

bufferTime

- Like buffer, but waits a specified amount of time before emitting buffered values

buffer [cont'd]

bufferToggle

- Like buffer, but takes two arguments – an opening and closing observable
  - Closing observable is provided a factory function
- Buffer starts a buffer when opening observable emits
- Emits values when closing observable emits
- Can have multiple buffers going simultaneously

bufferWhen

- Like bufferToggle, but requires no opening observable
- Like buffer, but factory function is provided instead of observable

partition

- Separates stream into two groups – one that passes the predicate, and one does not
- Like combining the results of *filter* with everything that was filtered out

throttle

- Does not emit any observables until a duration of time, specified by the provided observable, has passed between source emissions
- Only emits the latest value
- Metaphor: Someone who is looking for deals on an online store. They find the best deal and then wait 5 minutes to see if a better one appears.
  - If a better deal comes, they forget the last one and wait 5 more minutes before doing anything
  - If one doesn’t, they make the purchase (emission)
  - The purchasing activity is throttled by the frequency of new deals appearing

throttleTime

- Like throttle, except duration is determined by a specified number and not an observable

Advanced Operators (Operators 301)

zip (… provided)

- Bundles the latest emissions of a number of observables into a single observable
- Indexes of bundled emissions must match
- Zipped observable will emit at the pace of the slowest ancestor

combineLatest (… provided)

- Once each of the provided observables has emitted at least once, emit a bundle containing all the latest values
- After that, emit an updated bundle whenever any provided observable emits
- Works like zip, except indexes do not have to match
  - Moves faster than the pace of the slowest provided observable

forkJoin (…)

- Runs a number of observables, waits until they all finish, then bundle the results and emit
  - Forking – the process of running all the observables at once
  - Joining – the process of combining the results
- If any error, forkJoin will error
- Useful for when you need the results of all of a number of non-sequential API calls, or none at all
- Resolves a very common web development use case

publish

- Returns an observable with a special method, *connect*
  - Works similar to a Subject
- Unlike normal observable, published observable does not start executing code as soon as it is subscribed to
  - Multiple subscribers can subscribe and all get identical data!
- To start the functioning of the observable, like a normal observable responding to subscribe, call *connect*

share

- Like publish, but *connect* is omitted
- Observable starts executing code as soon as it is subscribed to, but does not start a new thread upon the 2nd subscription, 3rd subscription, and so on
- Useful for a long-lived process that gradually returns values
    - I.e., A notifications service with many widgets subscribed to it

multicast

- Like publish, but returns a Subject instead of an observable with a special property
- BehaviorSubject, ReplaySubject and others can all be used

race

- Waits until one observable from a group of provided observables emits, discard everything else
- Subsequent emissions from the “winner” will be emitted while the “losers” of the race will be ignored

retry & retryWhen

retry

- If the source observable throws an error, suppress the error and try again a specified number of times
- Number of repetitions is specified by a provided number
- Has no effect if the source never errors

retryWhen

- Like retry, but retries the source when provided observable emits

exhaustMap

- Subscribe to any observables emitted by source observable
  - The observable that is subscribed to is called the *subscribed observable*
- Emit any values that are emitted by the subscribed observable
- When the source observable emits again, discard the current subscribed observable and subscribe to the new one
- Like concatMap, but discards values
  - Impatient personality type!

withLatestFrom

- Creates new observable that combines emissions from the source observable with the latest value from a provided observable
- Subscribers are notified only when the source observable emits, but get both values
  - No notification is received when the provided observable emits

window

window

- All emissions within a specified window of time are bundled into an array and emitted together
- Like buffer, but emissions are bundled into an array
- A provided observable indicates when to “close” the window
  - New window is opened immediately
  - Observable emits an array at this time
  - First window opens automatically

windowCount

- Like window, but shifts to the next window when the current window has accumulated a specified number of values

windowTime

- Like window, but shifts to the next window after the current window has been opened for a specified amount of time

window [cont'd]

windowToggle

- Like window, but takes two observables – one which opens a window and another which closes it
- Closing observable is a factory function, like bufferToggle

windowWhen

- Like window, but opening observable is a factory function

let

- An operator which returns an observable that replaces the current one
  - Receives the current observable as an argument
- Usage promotes excessive trickiness

debounce & debounceTime

debounce

- Discard any values that are emitted within a specified period of time after the previous emission
- Like throttle but with an initial value
- Duration is specified by a provided observable

debounceTime

- Like debounce, but duration is specified by a number

Culminating Activity – RxJS Redux

Culminating Activity – RxJS Redux

- Implement a Redux store using observables
  - Features subscribe and dispatch
  - Omit additional features (middleware, etc.)
- Powered by the *scan* observable
- A very useful class for day-to-day programming!

Conclusion

Key Takeaways

- There are operators for almost every conceivable use case
- Using the correct operator results in succinct, readable, durable and versatile code (wow!)
  - Using the incorrect operator is rarely better than an equivalent hack
- Highly general operators (such as *create* and *let*) should be avoided under most circumstances
  - Before you act, remember TAOFT! *There's An Operator For That!*
- Only practice can result in mastery of operators

Challenge Task – RxJS Stackoverflow Client

- Build a web application which displays the latest questions from StackOverflow
  - Use APIs, i.e, https://api.stackexchange.com/2.0/questions?site=stackoverflow
  - Requests are limited to 300 per day - register for free to get up to 10,000
- Questions must be filterable by tag
- The body of the question, which is not included by default, and the answers, should be fetched separately as required
- All state management and processing of incoming data streams is to be done with RxJS
- The most elegant and clear chains of operators are to be used at all times
- The following operators should be avoided: *do*, *create*, *let*

Continue Learning with Great Deals!

- TodoMVC Application in Vue, React and Angular (50% Off)
  - https://www.udemy.com/todo-mvc/?couponCode=OPERATORS
- Ultimate JavaScript Objects (50% Off)
  - https://www.udemy.com/js-objects/?couponCode=OPERATORS
- Comprehensive TypeScript (93% Off!!!)
  - https://www.udemy.com/typescript101/?couponCode=OPERATORS

Thank You!