
ANGULAR 6
Lesson 08


HTTP REQUESTS /
OBSERVABLES
Objectives

Http Requests / Observables

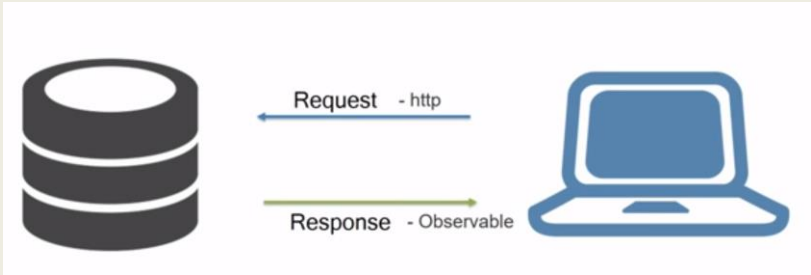
Capgemini

- HTTP Requests
- Sending GET Requests
- Sending a PUT Request
- Transform Responses with Observable Operators (map())
- Using the Returned Data
- Catching Http Errors
- Pipe(), map(), catchError()
- Interceptors
- Basics of Observables & Promises
- Analysing a Built-in Angular Observable
- Building & Using a Custom Observable
- Understanding Observable Operators
- Using Subjects to pass and listen to data


HTTP REQUESTS

HTTP Requests

- Angular applications often obtain data using http
- Application issues http get requests to a web server which returns http response-Observable to the application.
- Application then processes that data



Add instructor notes here.

Cap

Introducing RxJs

- RxJs stands for Reactive Extensions for Javascript, and its an implementation of Javascript.
- It is a ReactiveX library for JavaScript.
- It provides an API for asynchronous programming with observable streams.
- ReactiveX is a combination of the best ideas from the Observer pattern, the Iterat functional programming.
- Observable is a RxJS API. Observable is a representation of any set of values over time. All angular Http methods return instance of Observable. Find some of its operat
- map: It applies a function to each value emitted by source Observable and returns fin of Observable.
- catch: It is called when an error is occurred. catch also returns Observable.

The RxJS library is quite large.

It's up to us to add the operators we need

// Add map operator
https://cdnjs.cloudflare.com/ajax/libs/rxjs/4.1.0/rx.map

// Add all operators to Observable
https://cdnjs.cloudflare.com/ajax/libs/rxjs/4.1.0/rx.all.js

// Add map operator
import 'rxjs/add/operator/map'*;*

// Add all operators to Observable
import 'rxjs/Rx';

Transform Responses with Observable Operators

- Operators are functions that build on the observables foundation to enable sophisticated manipulation of collections.
- For example, RxJS defines operators such as map(), filter(), concat(), and flatMap().
- Operators take configuration options, and they return a function that takes a source observable.
- When executing this returned function, the operator observes the source observable's emitted values, transforms them, and returns a new observable of those transformed values.

Transform Responses with Observable Operators

- Map operator

```
import { map } from 'rxjs/operators';
const nums = of(1, 2, 3);
const squareValues = map((val: number) => val * val);
const squaredNums = squareValues(nums);
squaredNums.subscribe(x => console.log(x));
 // Logs
// 1
// 4
// 9
```

Catching Http Errors

- In addition to the error() handler, RxJS provides the catchError operator that lets you handle known errors in the observable recipe.
- For instance,
  - *suppose you have an observable that makes an API request and maps to the response from the server.*
  - *If the server returns an error or the value doesn't exist, an error is produced.*
  - *If you catch this error and supply a default value, your stream continues to process values rather than erroring out.*

Catching Http Errors

- Here's an example of using the catchError operator

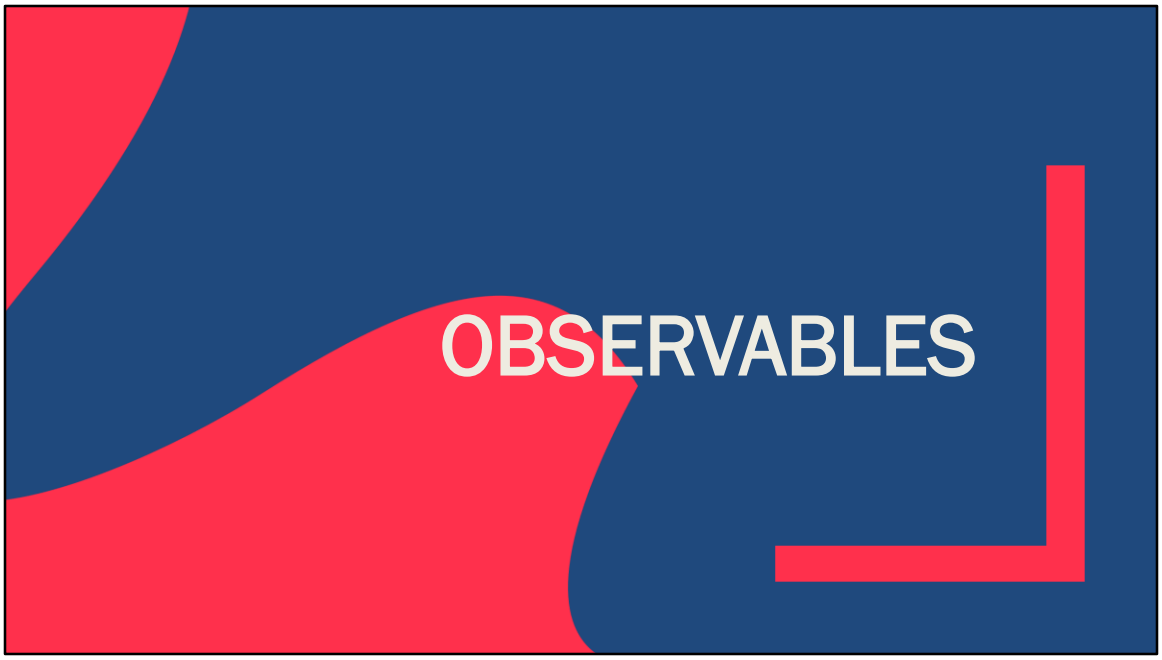
```
import { ajax } from 'rxjs/ajax';
import { map, catchError } from 'rxjs/operators';
// Return "response" from the API. If an error happens,// return an empty array.
const apiData = ajax('/api/data').pipe(
        map(res => {
                if (!res.response) {
                        throw new Error('Value expected!');
                }
                return res.response;
        }),
        catchError(err => of([]))
);
apiData.subscribe({
        next(x) { console.log('data: ', x); },
        error(err) { console.log('errors already caught... will not run');
}
});
```

Add instructor notes here.

Demo

- Demo HttpGet
- Demo HttpDelete
- Demo HttpGetErrorHanding

Add the notes here.


OBSERVABLES

Add instructor notes here.

Basics of Observables & Promises

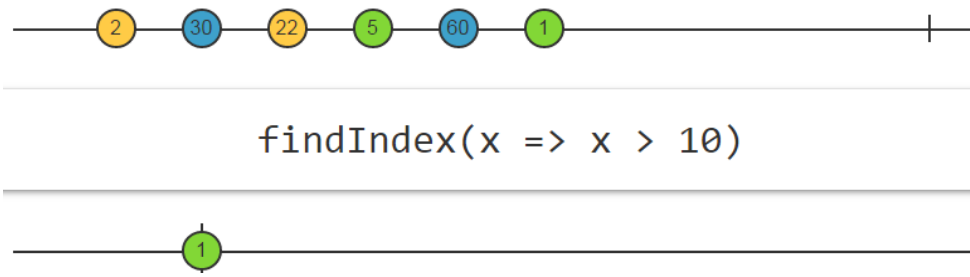
- Observables provide support for passing messages between publishers and subscrib application.
- Observables offer significant benefits over other techniques for event handling, asyn programming, and handling multiple values.
- Observables are declarative—that is, you define a function for publishing values, but executed until a consumer subscribes to it. The subscribed consumer then receives the function completes, or until they unsubscribe.

Data sequences can take many forms such as a stream of data from backend web service or a set of system notifications or a series of even such as user input.

Reactive extensions represent a data sequence as an observable sequenc commonly just called an observable.

A method can be subscribed to an observable to receive asynchronou notifications as new data arrives. The method can then react with th arrived data. The method is notified when there is no more data or one a error occurs. Since an observable works like an array we can use the ma operator.

We can visualizing observable sequences with interactive diagrams fro



Marble diagram

Add instructor notes here.

Cap

Basics of Observables & Promises

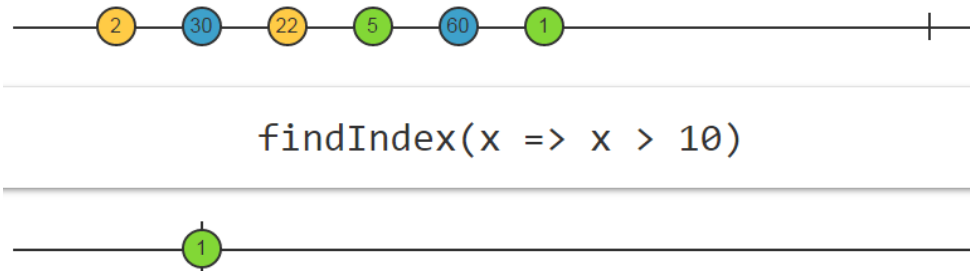
- An observable can deliver multiple values of any type—literals, messages, or events, the context. The API for receiving values is the same whether the values are delivere or asynchronously. Because setup and teardown logic are both handled by the obser application code only needs to worry about subscribing to consume values, and whe unsubscribing. Whether the stream was keystrokes, an HTTP response, or an interva interface for listening to values and stopping listening is the same.
- Because of these advantages, observables are used extensively within Angular, and recommended for app development as well.

Data sequences can take many forms such as a stream of data from backend web service or a set of system notifications or a series of even such as user input.

Reactive extensions represent a data sequence as an observable sequenc commonly just called an observable.

A method can be subscribed to an observable to receive asynchronou notifications as new data arrives. The method can then react with th arrived data. The method is notified when there is no more data or one a error occurs. Since an observable works like an array we can use the ma operator.

We can visualizing observable sequences with interactive diagrams fro



Marble diagram

Add instructor notes here.

Basics of Observables & Promises

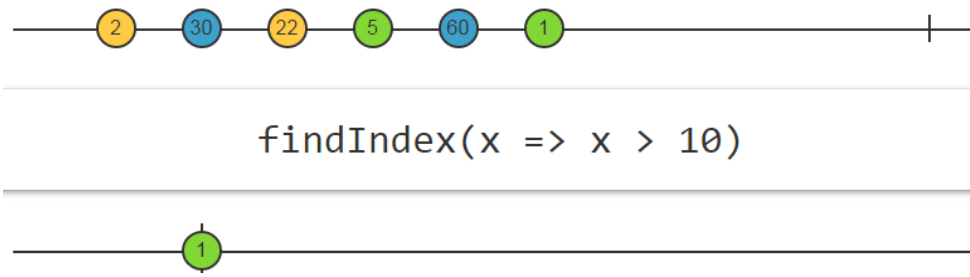
- Observables is a part of ReactiveX library also known as rxjs
  - *import { Observable } from 'rxjs/Observable';*
- Observables is like an array whose items arrived asynchronously. The role of Rea asynchronously programming
- Observable help to manage asynchronous data, such as data coming from a back data we are going to subscribe
- Observable work with multiple value
- Observable are cancellable
- Observable use javaScript function such as map filter & reduce

Data sequences can take many forms such as a stream of data from backend web service or a set of system notifications or a series of even such as user input.

Reactive extensions represent a data sequence as an observable sequenc commonly just called an observable.

A method can be subscribed to an observable to receive asynchronou notifications as new data arrives. The method can then react with th arrived data. The method is notified when there is no more data or one a error occurs. Since an observable works like an array we can use the ma operator.

We can visualizing observable sequences with interactive diagrams fro



Marble diagram

Capgemini

Building & Using a Custom Observable

- Use the Observable constructor to create an observable stream of any type.
- The constructor takes as its argument the subscriber function to run when the observable's subscribe() method executes.
- A subscriber function receives an Observer object, and can publish values to the observer's next() method.

Building & Using a Custom Observable

- Create observable with constructor

```
// This function runs when subscribe() is called
function sequenceSubscriber(observer) {
// synchronously deliver 1, 2, and 3, then complete
observer.next(1);
observer.next(2);
observer.next(3);
observer.complete();
// unsubscribe function doesn't need to do anything in this
// because values are delivered synchronously
return {unsubscribe() {}};}
 // Create a new Observable that will deliver the above sequence
const sequence = new Observable(sequenceSubscriber);
 // execute the Observable and print the result of each notification
sequence.subscribe({
 next(num) { console.log(num); },
complete() { console.log('Finished sequence'); }});
// Logs:// 1// 2// 3// Finished sequence
```