RxJS and Ajax

Downloads: http://bit.ly/centric-ng4

Agenda

Datum	Onderwerp
1-2-2017	TypeScript Introduction
15-2-2017	Advanced TypeScript
1-3-2017	Angular Introduction
15-3-2017	Angular Building Blocks
29-3-2017	Components
12-4-2017	RxJS and Ajax
26-4-2017	Data Entry
10-5-2017	Single Page Applications

Downloads: http://bit.ly/centric-ng4

What are we going to cover?

RxJS

Angular HTTP requests

The async pipe

HTTP request headers

RxJS

RxJS is an API for asynchronous programming with observable streams.

RxJS and Angular

The reactive extensions are used throughout Angular.

- An AJAX request returns an observable stream
- An EventEmitter is also an observable stream

Why RxJS?

Most actions are not standalone occurrences.

- Example: A mouse click triggers an Ajax request which triggers a UI update

RxJS is a great library to compose these streams in a functional style.

The RxJS Observable

An Observable is the object that emits a stream of event.

- The observer is the code that subscribes to the event stream

RxJS operators

Operators are used to operate on the event stream between the source and the subscriber.

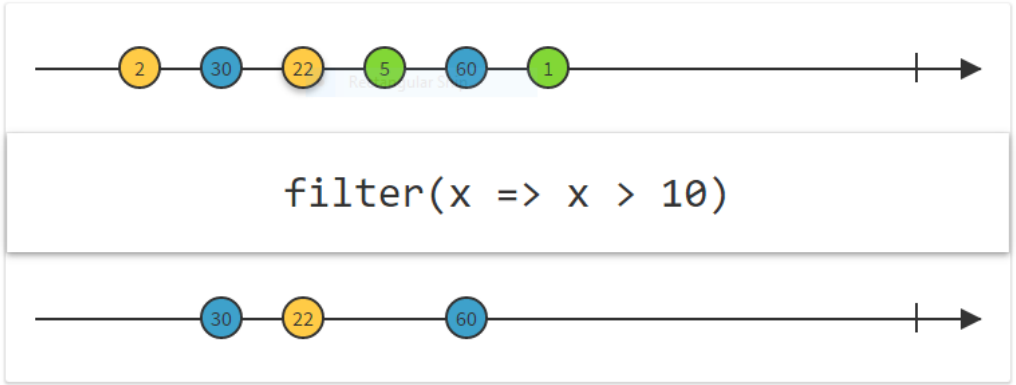
There are many operators for all sorts of purposes:

- Creating observables
- Transforming
- Filtering
- Combining
- Error handling
- Aggregate
- …

Example: The filter operator

An observable filter only allows items through if they pass the filter.

- These are normally documented using a before and after observable timeline



Basic RxJS example

```
Rx.Observable.range(1, 10)
  // 1 to 10
  .filter(x => x % 2 === 0)
  // Only even numbers
  .map(x => x * 10)
  // Multiplied by 10
  .subscribe(x => console.log(x));
```

Combining RxJS streams

```
// Start with 10 numbers
Rx.Observable.range(1, 10)
  // Switch to promises
  .concatMap(page => fetch(`/api/movies?page=${page}`))
  // Get the result per page using a promise
  .flatMap(rsp => rsp.json())
  // Switch to the resulting movies array per page
  .map(json => json.results)
  // Convert stream of arrays to stream of movies
  .flatMap(e => e)
  // Print each movie object
  .subscribe(movie => console.log(movie));
```

Angular HTTP requests

In Angular, AJAX requests are done using the HTTP service request() function.

- Each request is returned as an RxJS observable

The HTTP service has convenience functions for the most common request:

- **get**, **post**, **put**, **patch**, **delete**, **options**, and **head**.

Working with the response

An HTTP request returns an observable HTTP response object.

- Use the observable map operator to convert to the JSON payload

Use **import 'rxjs/add/operator/map';** to use the map operator.

The movies HTTP service example

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import { Movie } from './movie';

@Injectable()
export class MoviesService {
  constructor(private http: Http) { }

  getMovies(): Observable<Movie[]> {
    return this.http
      .get('/movies.json')
      .map(resp => resp.json());
  }
}
```

The movies component example

```
export class MoviesComponent implements OnInit {
  movies: Movie[] = [];

  constructor(private moviesService: MoviesService) { }

  ngOnInit() {
    this.moviesService.getMovies()
      .subscribe(movies => this.movies = movies);
  }
}
```

The template

```
<ul>
  <li *ngFor="let movie of movies">
    {{movie.title}}
  </li>
</ul>
```

The async pipe

The async pipe unwraps a promise or observable and returns the last value emitted.

- Makes databinding to the result of an HTTP request very simple

Use the **share** operator to prevent multiple subscriptions from multiple bindings.

The async pipe component example

```
export class MoviesComponent implements OnInit {
  movies: Observable<Movie>;

  constructor(private moviesService: MoviesService) { }

  ngOnInit() {
    this.movies = this.moviesService.getMovies()
  }
}
```

The async pipe template example

```
<ul>
  <li *ngFor="let movie of movies | async">
    {{movie.title}}
  </li>
</ul>
```

HTTP request headers

Angular doesn’t set many HTTP request headers.

- Settings headers for security or content negotiation is often required
- Request headers can be set per request or a default using DI

Default HTTP request headers example

```
@Injectable()
export class DefaultRequestOptions extends BaseRequestOptions {
  headers = new Headers({
    'Accept': 'application/json',
    'X-Requested-By': 'Angular',
  });
}

@NgModule({
  // Other settings
  providers: [MoviesService, {
    provide: RequestOptions,
    useClass: DefaultRequestOptions }],
})
export class AppModule { }
```

Conclusion

RxJS Observables are used in many places by Angular.

- They are much more powerful than promises

The HTTP service is used to do Ajax requests in Angular

- It is based around Observables