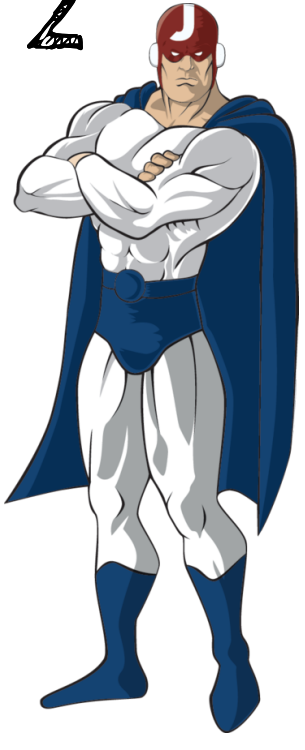
Asynchronous Data Streams in Angular 2

by Gerard Sans (@gerardsans)



A little about me

- Angular GDE (Google Developer Expert)
- JavaScript and Angular fanboy ^_^
- Blog | @gerardsans | github.com/gsans
- Community activist
  - Looking for Angular Meetup organisers
- Meetup Group. Follow us!
  - AngularJS Labs (@angularjs_labs)

ng-nl 2016 - February 18th

@ngnlconf

ng-conf - May 4-6th

@ngconf

2 Days: Angular 2 Workshop - April 14-15th

Jfokus - Stockholm

TypeScript, Dependency Injection, Template Syntax, Components, RxJS, HTTP, Component Router, Forms, Unit Testing, Migration to Angular 2

AngularConnect - Sept 27-28th

@AngularConnect

Asynchronous Data Streams

Streams

- ***Asynchronous***, register a *callback* to be notified when results are available
- ***Data***, raw information: Number, String, Objects (Arrays, Sets, Maps)
- ***Streams***, sequences of data made available over time

Examples

Stream

Array

Streams timeline

- Also known as
  - pipes (Unix 3, 1973)
  - streams (Node.js, 2009)
  - async pipes (Angular 2, 2015)
  - Observable sequences, Observables

Functional Programming

- Array methods
  - forEach, map, filter and reduce
- Composition

Array Methods

```
var team = [
  { name: "Igor Minar", commits: 159 },
  { name: "Jeff Cross", commits: 84 },
  { name: "Brian Ford", commits: 113 }
];
```

forEach

```
for(var i=0, ii=team.length; i<ii; i+=1){
  console.log(team[i].name);
}
```

```
team.forEach(function(member){
  console.log(member.name);
});
```

map

```
var newTeam = [];
for(var i=0, ii=team.length; i<ii; i+=1){
  newTeam.push(team[i]);
}
```

```
var onlyNames = team.map(function(member){
  return { name: member.name };
});
```

filter

```
var onlyOver100Commits = [];
for(var i=0, ii=team.length; i<ii; i+=1){
  if (team[i].commits>100) {
    onlyOver100Commits.push(team[i]);
  }
}
```

```
var onlyOver100Commits = team
  .filter(function(member){
    return (member.commits>100);
});
```

reduce

```
var total = 0; // initial value
for(var i=0, ii=team.length; i<ii; i+=1){
  total = total + team[i].commits;
}
```

```
var total = team.
  reduce(function(total, member){
    return total + member.commits;
}, 0); // initial value
```

Composition

```
function over100Commits(member){
  return (member.commits>100);
}
function projectName(member){
  return member.name;
}
function toUpperCase(input){
  return input.toUpperCase();
}
```

```
team
  .filter(over100Commits)
  .map(projectName)
  .map(toUpperCase)
  .forEach(log);
```

Key Concepts

- Observable
- Operators
  - merge, concat, map and more!
- Observer
  - onNext, onError, onCompleted
- Subscribe/Unsubscribe

Observable API

```
//Observable constructor
let observable = new Observable(observer => {
  try {
    //pushing values
    observer.next(1);
    observer.next(2);
    observer.next(3);

    //finish stream
    observer.complete();
  }
  catch(e) {
    //error handling
    observer.error(e);
  }
});
```

Basic Stream

```
//ASCII Marble Diagram

----0----1----2----3----> rx.interval(1000)
----0----1----2----|      rx.interval(1000).take(3)

---> is the timeline
0, 1, 2, 3 are emitted values
X is an error
| is the 'completed' signal
```

RxMarbles

Basic Stream

```
----0----1----2----3----> rx.interval(1000)
----0----1----2----|      rx.interval(1000).take(3)


rx.Observable
 .interval(1000)
 .take(3)
 .subscribe(item => console.log(item)); // shows 0, 1, 2
```

Observer

```
observable.subscribe(
  function onNext(value){
    console.log(value);
  },
  function onError(error){
    console.log(error);
  },
  function onCompleted(){
    console.log('Completed');
  });
```

```
var observer = new Observer(
  function onNext(result){
    console.log(result);
  },
  function onError(err){
    console.log(err);
  },
  function onCompleted(){
    console.log('Completed');
  }
);
observable.subscribe(observer);
```

Unsubscribe

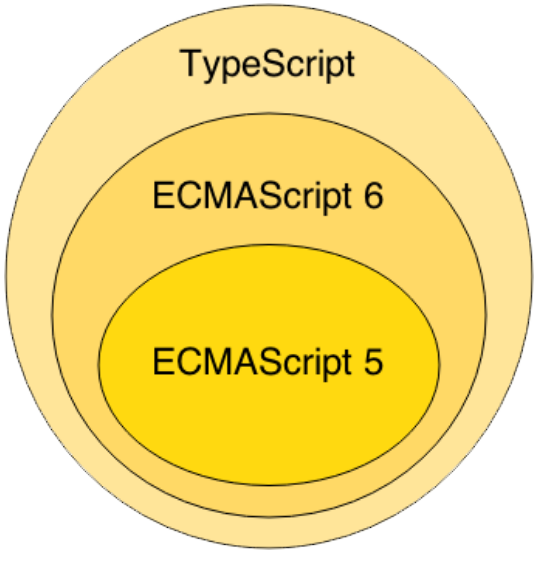
```
var subscriber = observable.subscribe(
  value => console.log(value),
  error => console.log(error),
  () => console.log('Completed')
);

//cleanup handlers
subscriber.unsubscribe();
```

Arrays vs Streams

- Arrays (sync)
  - map and filter create new arrays
  - source must be in memory
- Streams (sync/async)
  - only deal with one item at a time
  - consume less memory
  - good for large datasets

video

ES6 / TypeScript

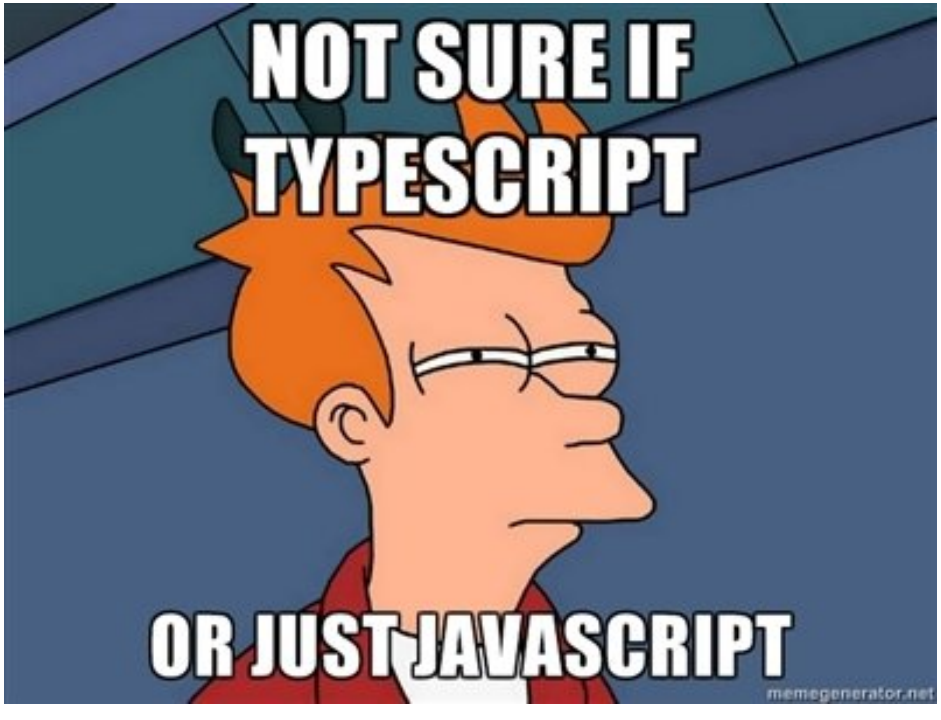


ES6 (ES2015)

- Classes, modules, arrow functions

TypeScript

- Types, annotations
- Better editor support
- Angular Team uses it ^_^


NOT SURE IF
TYPESCRIPT
OR JUST JAVASCRIPT
memegenerator.net

ES6: Overview

- Arrow functions and let keyword; Block scopes
- Classes and inheritance; Default parameters
- Destructured assignment
- Generators; Iterators; Maps
- Promises; Rest parameters; Sets
- Spread operator; Template Literals

Resources

- jsbin (ES6/Babel)
- Just another introduction to ES6

ES6 modules

ES6

```
// ES6 deconstructing
import { member } from "module-name";
import { reallyLongMemberName as alias } from "module-name";
import { member1 , member2, [...] } from "module-name";
import defaultMember from "module-name";
```

```
//------ lib1.js ------
export function add(x, y) {
  return x + y;
}
export function square(x) {
  return x * x;
}
```

```
//------ lib2.js ------
export default function square(x) {
  return x * x;
}
```

```
//------ main.js ------
import { add as sum, square } from 'lib1';
import mySquare from 'lib2';

sum(1,2); // 3
mySquare(2,2); // 4
```

ES6: Classes

ES6

```
class Person {
  //class constructor
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  //class method
  sayName() {
    console.log("I'm "+this.name);
  }
}

let p = new Person("John", 25);
p.sayName(); // I'm John
```

```
function Person(name, age) {
  //class constructor
  this.name = name;
  this.age = age;
}
Person.prototype = {
  //class method
  sayName: function () {
    console.log("I'm "+this.name);
  }
};

var p = new Person("John", 25);
p.sayName(); // I'm John
```

ES6: Destructuring

ES6

```
let [one, two] = [1, 2];
let {three, four} = {three: 3, four: 4};

console.log(one, two, three, four); // 1 2 3 4
```

```
var a = [1, 2];
var one = a[0];
var two = a[1];

var b = {three: 3, four:  4};
var three = b.three;
var four = b.four;

console.log(one, two, three, four); // 1 2 3 4
```

ES6: Template literals

ES6

```
//multi-line
let text = `string text line 1
string text line 2`;

//interpolation
let p = {name: "John", credits: 20};
let tpl = `Name: ${p.name}. Credits: ${p.credits + 10}`;
// Name: John. Credits: 30
```

```
//multi-line
var text = "string text line 1\n" +
  "string text line 2";

//interpolation
var p = {name: "John", credits: 20};
var tpl = "Name: " + p.name + ". Credits: " + (p.credits + 10);
// Name: John. Credits: 30
```

ES6: Arrow functions

ES6

```
let square = x => x * x;
let add = (a, b) => a + b;
let pi = () => 3.1415;
```

```
var square = function(x) { return x * x; };
var add = function(a, b) { return a + b; };
var pi = function() { return 3.1415; };
```

ES6: Default parameters

ES6

```
function sayMsg(msg='This is a default message.') {
  console.log(msg);
}
sayMsg();
sayMsg('This is a different message!');
```

```
function sayMsg(msg){
  var msg = msg || 'This is a default message.';
  console.log(msg);
}
sayMsg();
sayMsg('This is a different message!');
```

TypeScript +1.6

- Basic types: boolean, string, number, any
- Type annotations and compile-time type checking
- Type inference, Interfaces, Enumerated type, Mixins, Generics, Tuple

Resources

- Handbook

TS: Basic types

TS

```ts
var isDone: boolean = false;              // boolean
var height: number = 6;                   // number
var name: string = "bob";                 // string

var list: number[] = [1, 2, 3];           // arrays
var list: Array<number> = [1, 2, 3];      // generics

enum Color {Red, Green, Blue};            // enums
var c: Color = Color.Green;

var notSure: any = 4;                     // if not defined
var list: any[] = [1, true, "free"];

function warnUser(): void {               // return type
  alert('WAT?');
}
```

TS: IDEs

Angular 2 Basic App

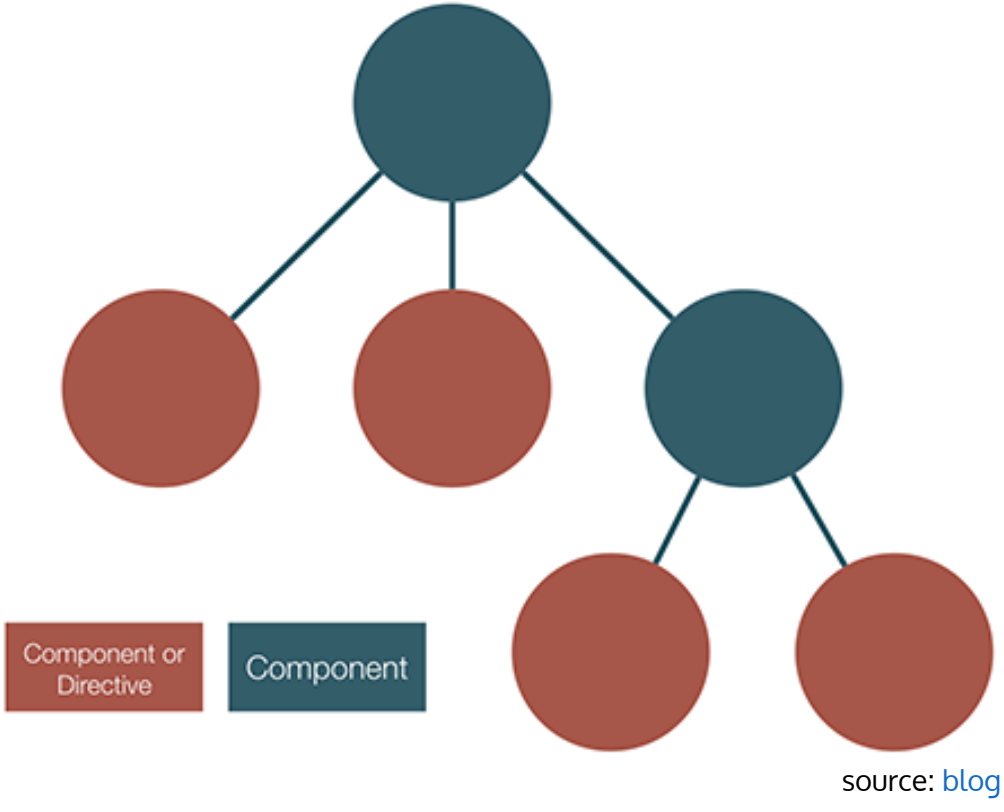
We are going to explore a basic Application covering:

- Bootstrapping
- Components, Services
- Template syntax
- Basic Http Service

Bootstrapping

- Angular Application instantiation
- Root component
- Global Dependencies
  - Classes. Eg: Router, Forms, Http
  - Global Values
  - Vendor dependencies

Component Tree



source: blog

Component

```
import {Component} from 'angular2/core';
import {FORM_DIRECTIVES} from 'angular2/common';
import {UsersService} from '../services/usersService';

@Component({
  selector:     'home', // <home></home>
  styles:       [`h1 { color: red }`],
  template:     `<h1>Home</h1>`,
  directives:   [FORM_DIRECTIVES],
  providers:    [usersService]
})
export class Home { ... }
```

Template Syntax

Syntax	**Binding type**
{{title}} <input **[value]**="firstName"> <li **[class.active]**="isActive"> <div **[style.width.px]**="mySize">	Interpolation Property Class Style
<button **(click)**="onClick($event)">	Event
[(ngModel)]="data.value"	Two-way

Additional Resources

- Angular 2 website (angular.io)
- Plunker (ES5/ES6/TS)


Thanks!