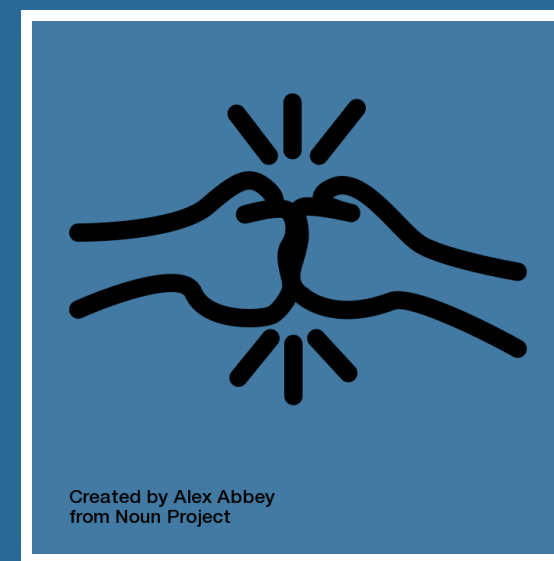


Angular 2 and Observables

The Zan and Jayna of Reactive Component Development



Just in case you don't know...

Wonder Twins

From Wikipedia, the free encyclopedia


The Wonder Twins, **Zan** and **Jayna**, are a fictional **extraterrestrial** twin brother and sister **superhero** duo who first appeared in Hanna-Barbera's American **animated** television series *Super Friends*. They subsequently appeared in comics based on the animated series, and were later introduced into the main **DC Comics Universe**. They have since appeared in other media, including animated series such as *Justice League Unlimited* and *Teen Titans Go!*, and the live-action TV series *Smallville*.

Contents	[hide]
1	Publication history
2	Fictional character biography
2.1	Super Friends
3	Powers and abilities
4	In comics
5	In other media
5.1	Television
5.2	Justice League/Justice League Unlimited
5.3	Adult Swim
5.4	Smallville
5.5	Arrowverse
5.6	Teen Titans Go!
6	References
7	External links

Publication history

[edit]

Wonder Twins



The Wonder Twins, from *Super Friends*.

Publication information	
First appearance	"Joy Ride" <i>The All-New Super Friends Hour</i> (September 10, 1977)
In-story information	
Alter ego	Zan and Jayna
Species	Exxorian
Place of origin	Exxor
Team affiliations	<div>Super Friends</div> <div>Justice League</div> <div>Ten Elements of the Universe</div>
Abilities	<div>Zan can transform into any form of water</div> <div>Jayna can transform into any</div>

The Wonder Twins of Angular 2 are
Components

RxJS Observables

They go together like Chocolate and Peanut Butter

He's Corny, Who is This Guy?

Ken Rimple - Director of Training, Chariot Solutions

- Long-time Java guy, Spring, now into JavaScript...
- Teaching and consulting at Chariot for 9+ years
- Author of various AngularJS courses at Chariot
- Co-author of Angular 2 and React courses at Chariot
- Father of 4 and inveterate bad punster

Where to find me

- On the tweeter at @krimple
- Chariot TechCast/DevNews podcast host
- Write articles at chariotsolutions.com/blog about JS frameworks and more

Angular 2 is

- A Component-based JavaScript Framework...
- girded by a *Functional Reactive API*, Microsoft's *RxJS 5.0*

What is a Component?

```
@Component({
  selector: 'message-area',
  template: `... html content here`,
  ...
})
export class MessageAreaComponent {
  constructor() { }
  message: string; // state variables
  doSomething() { ... } // user-defined methods
  ngOnInit() { ... } // lifecycle methods
}
```

You're never walking alone...

The application is bootstrapped with an outer component

A component in use by an *outer* component

```
// in the template of the outer component...  
  
<message-area  
  messageText="Don't walk and chew gum"  
  [messageOptions]="messageOpts"  
  (messageAccepted)="clearMessageText()">  
</message-area>
```

- We'll look at each of these assignments...

Static assignment

```
messageText="Don't walk and chew gum"  
enabled="true"
```

- Uses the **propertyName="value"** syntax, and expects literal values
- Used for simple options

Property binding

```
[propertyName]="variableOrExpression"
```

- Looks in the current component for a member variable or evaluates an expression
- The value is assigned *by reference* to the property **propertyName** in the inner component

Event Binding

```
(eventName) = "methodCall( )"
```

- When the **message-area** component emits the named event, execute the method **methodCall()** in the outer component

Differences from AngularJs 1.x?

(Tons)

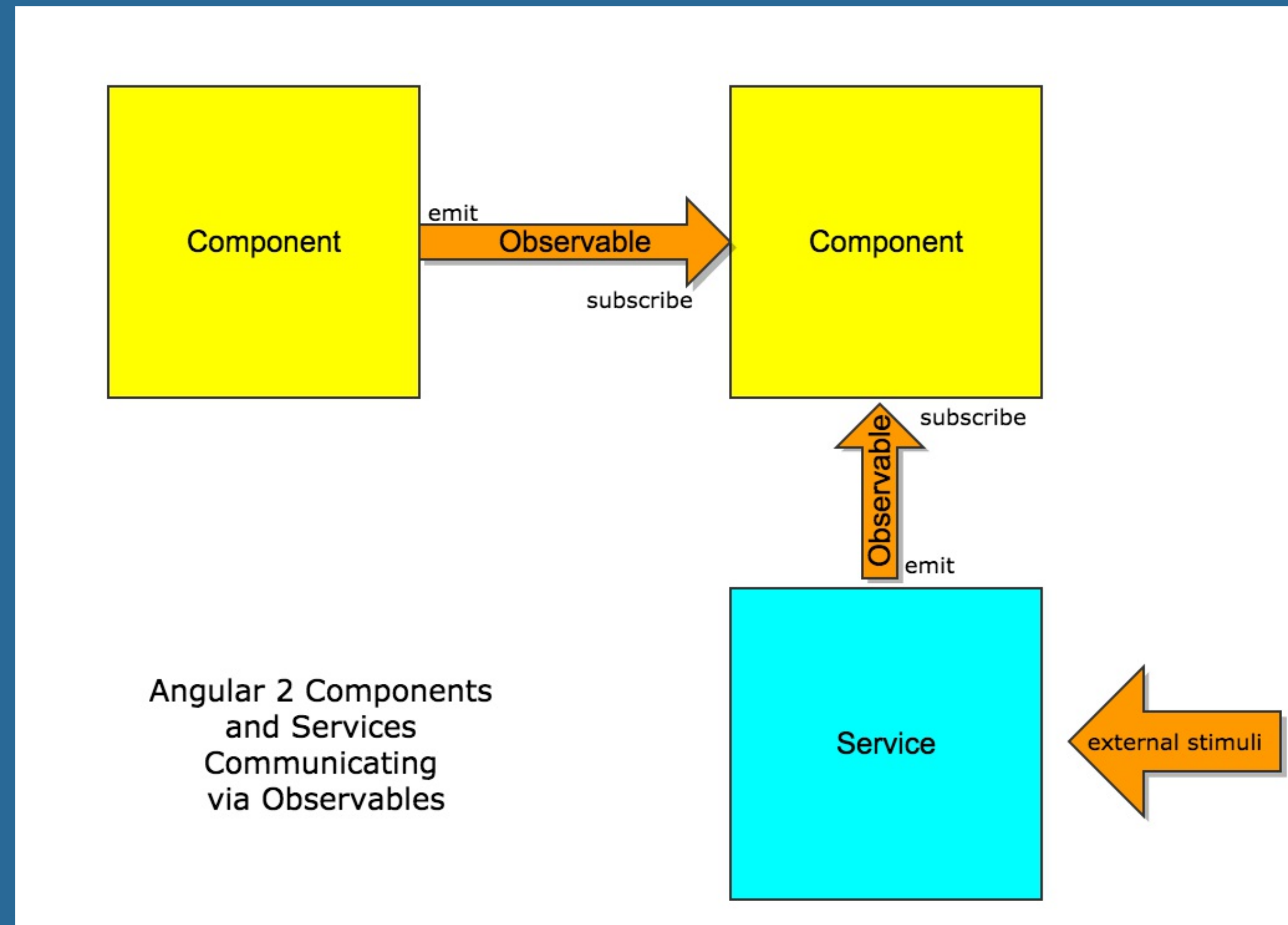
- No more **\$scope, controller**
- Still have **Services**, though created differently
- Still have "filters" but they're called **Pipes** now
- Dependencies injected via constructor injection
- TypeScript classes instead of plain ES5
- The old message bus is replaced by RxJS
- Promises now use ES2015 Promise syntax or are replaced by RxJS observables

Component Demos

Components and Observables

What is an Observable?

- An abstraction in *RxJS* that can act as a *data stream*
 - emits typed information
 - notifies subscribers when the information changes



This is a functional,
reactive framework,
enabled by Components
and RxJS...

Observables can be operated on functionally

- Compose streams of observable events
- Chain one operator to another
- Transform the stream to what you want
- Allow for more than one subscriber

Key Observable Type - Event Emitter

- Used in UI to report events from one component to another
- Bound using the **@Output** annotation

```
@Component({  
  ...  
})  
export class ReportingComponent {  
  @Output() onStatusChanged: EventEmitter<string> =  
    new EventEmitter<string>();  
}
```

Emitting an event

```
@Component({
  ...
  template: `
    <button (click)='emitEvent()'>go!</button>
  `
})
...
export class ReportingComponent {
  ...
  emitEvent() {
    this.onStatusChanged.emit('OUCH!');
  }
}
```

Accepting an emitted event

Inject **@Input** parameters using event syntax and subscribe in your code

```
in template:
<reporting
  (onStatusChanged)=
    "addStatusToHistory( $event ) ">
</reporting>
```

Demo - emitting an
event from an input
field

Key Observable Example - Angular API

```
// fetching data
http.get('/api/customers')
  .then(
    (data) => {
      this.data = data;
    },
    (error) => {
      console.error('error occurred in fetch');
      console.dir(error);
    }
  );
```

Observable updates for Http

```
// updates
http.post('/api/customers', {})
  .then(
    () => { /* leave location */ },
    (error) => { /* show error */ });
```


Demo - REST

Key Observable Type - Subject

- Acts as an **Observable** and an **Observer**
- Meaning you can use it to emit events in a *stream*
- Not tied to an UI component

```
// creating - stream of arrays of strings
let dataStream = new Subject<string[]>();

// emitting - send an array chunk
dataStream.next(['b', 'c', 'asdfkalsjdf']);

// subscribing
dataStream.subscribe(
  (payload) => { console.log(payload); },
  (error) => { alert('error with stream'); },
  () => { console.log('stream closed'); });
```

Potential Subject Strategies

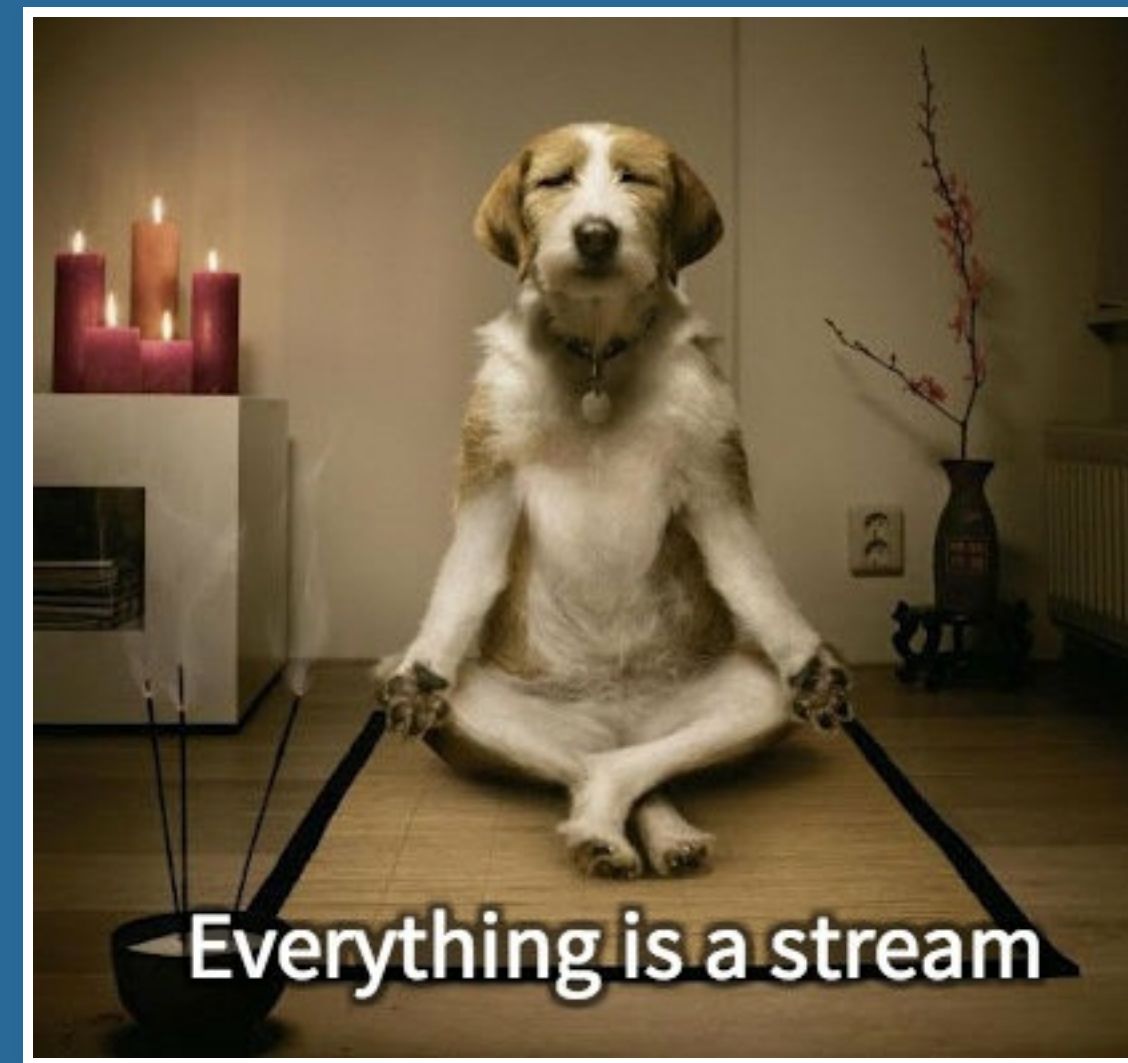
- Use as an adapter to translate data from another API
 - Midi Input
 - User-defined events
- Great to decouple sources and sinks of data
- Data is sent asynchronously
- Angular uses this internally
- Example in RxJS - `Observable.webSocket`

Demo - Synthesizer in Angular and RxJS

Advanced Features of Observables - Operators

- Observable streams can be composed and operated on with
 - Higher-order functions like **map**, **debounce**, **filter**, etc..
 - Dozens built-in to RxJS 5.0 - see [Observable API](#) for examples

Excellent reference



Summary

Angular 2 is Powerful

- But you need to learn Observables to make it reactive
- You need to master Components and **@Input** and **@Output** to communicate
- Look at each Angular API and learn how the Observables are used

Resources and Q&A

- Look on Twitter at [@krimple](#) for the slide deck and code samples later tonight
- Chariot's Angular and React training courses, as well as Scala chariotsolutions.com/training
- Ken's blog articles about Angular, as well as those from other Charioteers on Scala, React, Mobile and more - chariotsolutions.com/blog
- We have a PhillyETE video watching party coming up - chariotsolutions.com/events