Pipes

BUILDING WEB APPLICATIONS USING ANGULAR

Contents

- What are pipes and why do we need them?
- How to create our own pipes
- Using Pure and Impure pipe

Introduction

- Applications work with data and this data often needs to be displayed to the user, usually we want to transform this data into information. For example we often store financial data as integer values, in pence. Rarely does a user want to see their prices in pence though.
- Converting our applications data into presentable information is often thought of as a stylistic task, and is well suited to be done in the view. After all, that's where we should be concerned about presentation.
- This is the job of pipes

Pipes

- A pipe takes input data and outputs it in the desired format

```
<p>My name is {{fullname | titlecase}}</p>
```

- Angular has a number of built in pipes

AsyncPipe	CurrencyPipe	DatePipe
DecimalPipe	I18nPluralPipe	I18nSelectPipe
JsonPipe	LowerCasePipe	PercentPipe
SlicePipe	TitleCasePipe	UpperCasePipe

Pipes

- In order to configure the pipe’s output we can use parameters through use of the colon ‘:’ operator

```
<p>My name is {{DOB | date: “dd MMM”}}</p> //03 May
```

- Angular has a number of built in pipes

AsyncPipe	CurrencyPipe	DatePipe
DecimalPipe	I18nPluralPipe	I18nSelectPipe
JsonPipe	LowerCasePipe	PercentPipe
SlicePipe	TitleCasePipe	UpperCasePipe

- Pipes can be chained together

```
<p>My name is {{DOB | date: “dd MMM” | uppercase }}</p> //03 MAY
```

Custom Pipes

- Custom pipes can be built by creating a pipe class that implements the PipeTransform interface and is decorated by the @Pipe() decorator with a ‘name’ property
- The class should take 1 + n arguments – the initial value of the data, followed by n arguments that the pipe can receive. It then returns the transformed data

```
@Pipe({
   name: 'custom'
})
export class CustomPipe implements PipeTransform {

   transform(value: string, replace: string, replacement?: string): string {
      if (!replacement) {
         replacement = "bananas"
      }

      return value.replace(replace, replacement);
   }
}
```

Pure and Impure Pipes

- Pipes are pure by default and can be made impure by setting their pure flag to false
- Pure pipes execute only when a pure change is executed – that is: a primitive value changes, or an object reference changes
- If you need a pipe to execute on mutation of objects, then you need an impure pipe!

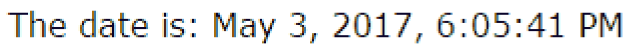
```
@Pipe({
   name: ‘custom’,
   pure: false
})
export class CustomPipe implements PipeTransform {…}
```

Angular's AsyncPipe

- Angular's AsyncPipe is a noteworthy pipe due to it's utility and is a good example of a stateful, impure pipe
- The async pipe takes a Promise or Observable and unwraps emitted values as they arrive
- We can use this to minimise the amount of boiler plate code going in to our components

```
{{ timer | async | date: medium }}
```

```
timer: Observable<Date> = new Observable<Date>(observer => {
   setInterval(()=>{
      observer.next(new Date());
   }, 1000);
})
```



```
The date is: May 3, 2017, 6:05:41 PM
```

Exercise

USING PIPES TO PRESENT OUR DATA