Angular 2 Pipes

Sang Shin
JPassion.com
"Code with Passion!"

Topics

- What is a Pipe?
- Built-in Pipes
- Custom Pipes
- Pipes and change detection
- Pure and Impure pipes
- Quick tutorial on Promise and Observable
- Async pipe

What is Pipe?

What is a Pipe?

- A pipe takes in data as input and transforms it to a desired output
  - The hero's birthday is {{ birthday | date }}
- Pipe can take parameters
  - The amount is {{ amount | currency:'EUR' }}
- Pipe can take multiple parameters
  - The sliced string is {{ name | slice:1:5 }}
- Pipes can be chained
  - The birthday is {{ birthday | date | uppercase}}
  - The birthday is {{ birthday | date:'yyyy-MM-dd' | uppercase}}

Built-in Pipes

Built-in Pipes

- date
- uppercase
- lowercase
- slice
- currency
- percent
- json
- decimal
- async

Lab: Use built-in Pipes

```
{{ myValue | lowercase }}
{{ myValue | uppercase }}
{{ myValue | slice:2 }}
{{ myValue | uppercase | slice:2:5 }}

currency with filter: {{ 10000000 | currency:"USD"}}
currency with filter: {{ 10000000 | currency:"EUR"}}

date without filter: {{ myDate }}</p>
date using filter format2: {{ myDate | date:"fullDate"}}
date using filter format1: {{ myDate | date:"dd/MM/yy"}}
date using filter format2: {{ myDate | date:"MM/dd/yy"}}
```

Lab: Use built-in types

- Suppose you have the following in the component class

  object: Object = {name: 'sang', email: 'sang@jpassion.com'};

- The following results

  {{object}} -> [object Object]

  {{object|json}} -> { "name": "sang", "email": "sang@jpassion.com" }

Custom Pipes

Creating Custom Pipe

- Create a class with @Pipe annotation
  - Give a name with “name” property
- Implement PipeTransform interface
  - Implement transform(value: any, args?: any) method

```
@Pipe({
  name: 'mySquare'
})
export class MySquarePipe implements PipeTransform {

  transform(value: any, args?: any): any {
    return value * value;
  }

}
```

Lab: Create a custom pipe

- Create a pipe that returns square value of a number
  - {{5|mySquare}} should display 25
- Generate a pipe
  - ng g pipe my-square
- Register the pipe to the hosting component
  - pipes: [MySquarePipe]
- Implement transform method
- Try argument – {{5|mySquare:true}}

```
transform(value: any, args?: any): any {
  if (args == undefined ||  args == true) { return value * value; }
  else { return value }
}
```

Filter Pipe

- Filter Pipe receives an array and returns an array after filtering operation

Lab: Create a filter pipe

- Create a filter pipe that receives an array of string and argument string and returns only the strings that contains the argument string
- Generate a pipe
  - ng g pipe my-filter
- Implement transform method
  - Use item.match('^.*' + args + '.*$') for match operation

```
for (let item of value) {
  if (item.match('^.*' + args + '.*$')) {
    myArray.push(item);
  }
}
```

Pipes and Change Detection

Angular Change Detection Process

- Angular looks for changes to data-bound values through a change detection process that runs after every JavaScript event:
  - every keystroke, mouse move, timer tick, and server response.
- This could be expensive
  - Angular strives to lower the cost whenever possible and appropriate
- Angular picks a simpler, faster change detection algorithm when we use a pipe
  - As a default, it does not get applied for every change detection cycle

Pure vs Impure Pipes

- There are two categories of pipes: pure and impure
  - Pipes are pure by default (for faster processing)
- We make a pipe impure by setting its pure flag to false
  - Angular executes an impure pipe during every component change detection cycle

```
@Pipe({
  name: 'flyingHeroesImpure',
  pure: false
})
```

Quick Tutorial on Promise, Observable

Promise

- The Promise object is used for asynchronous computations
- A Promise represents a value which may be available now, or in the future, or never

```
let myPromise = new Promise(
                    (resolve, reject) => {
                        setTimeout(() => resolve("JavaOne 2016"), 3000);
                    });

myPromise.then(value => console.log(value))
            .catch(error => console.log(error));
```

Observable

- Rx is a library for composing asynchronous and event-driven programs using observable sequences
- The API supports Observable type

```
let myObservable = Observable.create((subscriber) => {
  subscriber.next("Hello");
  subscriber.next("World!");
  subscriber.next("2016!");
  subscriber.complete();
});

myObservable.subscribe((data) => console.log(data),
                       (error) => console.log(error),
                       () => console.log("completed"));
```

Async Pipe

Async pipe

- The async pipe returns the latest value it has received
  - It calls then() method for Promise internally
  - It calls subscribe() method for Observable internally

```
<h4>Asynch pipes</h4>
<p>not filtered with async filter: {{myAsynchValue}}</p>
<p>filtered with async filter: {{myAsynchValue|async}}</p>
```

Lab: Async Pipe

- Create Promise object

```
myAsynchValue = new Promise(
        (resolve, reject) => {  setTimeout(() => resolve("Hello!"), 3000) }
)
```

- Use Asych pipe

```
<h4>Asynch pipes</h4>
<p>not filtered with async filter: {{myAsynchValue}}</p>
<p>filtered with async filter: {{myAsynchValue|async}}</p>
```

Code with Passion!
JPassion.com