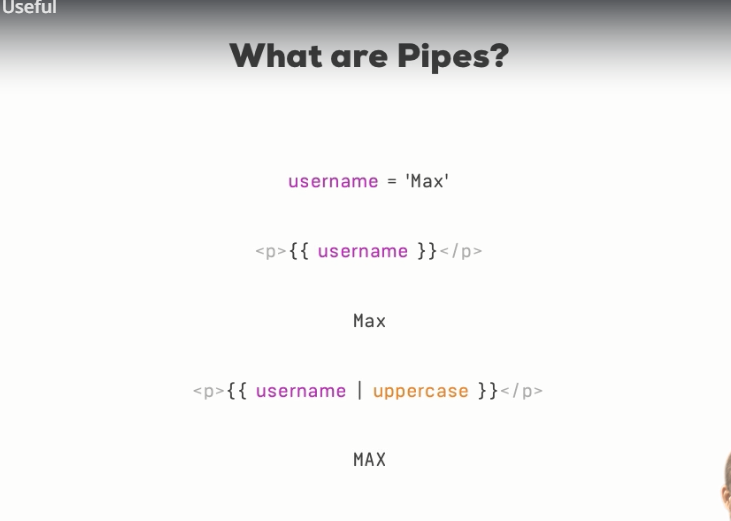
**Section 17: Using Pipes to Transform Output**

**Section 17: Lecture 225//Introduction and why Pipes Are Useful**

1. What are pipes?

Pipes is a feature in Angular 2, which allows you to transform output in your template. There are pipes for different types of output.

1. Suppose, you need to change value of some property when you render/output it on screen – here we can use pipe. Suppose, we need to convert the output into uppercase while displaying. Ex. <p>{{ username | uppercase }}</p>
2. 

**Section 17: Lecture 226//Using Pipes**

1. Here we will add few pipes in our code. Now, we know that the pipe will only transform the output, so we would need to apply/use it only at the html template.
2. app.component.html
3. <div class="container">
4. <div class="row">
5. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
6. <ul class="list-group">
7. <li
8. class="list-group-item"
9. \*ngFor="let server of servers"
10. [ngClass]="getStatusClasses(server)">
11. <span
12. class="badge">
13. {{ server.status }}
14. </span>
15. <strong>{{ server.name }}</strong> |
16. {{ server.instanceType | uppercase }} |
17. {{ server.started | date }}
18. </li>
19. </ul>
20. </div>
21. </div>
22. </div>

**Section 17: Lecture 227//Parameterizing Pipes**

1. So, here we want to display the date in the format we want, so we will parameterize this pipe. We can add the parameters by adding a colon behind the pipe. Before that a pipe needs to be able to handle the colon.
2. The date pipe expects to receive a string parameter, so, we will pass it as string.
3. app.component.html
4. <div class="container">
5. <div class="row">
6. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
7. <ul class="list-group">
8. <li
9. class="list-group-item"
10. \*ngFor="let server of servers"
11. [ngClass]="getStatusClasses(server)">
12. <span
13. class="badge">
14. {{ server.status }}
15. </span>
16. <strong>{{ server.name }}</strong> |
17. {{ server.instanceType | uppercase }} |
18. {{ server.started | date:'fullDate' }}
19. </li>
20. </ul>
21. </div>
22. </div>
23. </div>

**Section 17: Lecture 228//Where to learn more about pipes**

1. Official angular documentation 🡪 angular.io 🡪 DOCS 🡪API Reference 🡪 Filter – type pipes and search.

**Section 17: Lecture 229//Chaining Multiple Pipes**

1. Now, we will combine pipes in angular, we can simply chain pipes by adding another pipe symbol after a pipe.
2. app.component.html
3. <div class="container">
4. <div class="row">
5. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
6. <ul class="list-group">
7. <li
8. class="list-group-item"
9. \*ngFor="let server of servers"
10. [ngClass]="getStatusClasses(server)">
11. <span
12. class="badge">
13. {{ server.status }}
14. </span>
15. <strong>{{ server.name }}</strong> |
16. {{ server.instanceType | uppercase }} |
17. {{ server.started | date:'fullDate' | uppercase }}
18. </li>
19. </ul>
20. </div>
21. </div>
22. </div>

**Section 15: Lecture 230//Creating a custom pipe**

1. Here we will create a new typescript file shorten.pipe.ts. In this File we will create class which will implement an interface called pipeTransform.
2. In this pipe class we need to add one special decorator i.e. the @Pipe decorator.
3. app.pipe.ts:
4. import { PipeTransform, Pipe } from "@angular/core";
5. @Pipe({
6. name:'shorten'
7. })
8. export class ShortenPipe implements PipeTransform{
9. transform(value: any){
10. if(value.length>10){
11. return value.substr(0, 10) + ' ...';
12. }
13. return value;
14. }
15. }
16. app.module.ts:
17. import { BrowserModule } from '@angular/platform-browser';
18. import { NgModule } from '@angular/core';
19. import { FormsModule } from '@angular/forms';
20. import { HttpModule } from '@angular/http';
21. import { AppComponent } from './app.component';
22. import { ShortenPipe } from './shorten.pipe';
23. @NgModule({
24. declarations: [
25. AppComponent,
26. ShortenPipe
27. ],
28. imports: [
29. BrowserModule,
30. FormsModule,
31. HttpModule
32. ],
33. providers: [],
34. bootstrap: [AppComponent]
35. })
36. export class AppModule { }

5. app.component.html:

<div class="container">

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<ul class="list-group">

<li

class="list-group-item"

\*ngFor="let server of servers"

[ngClass]="getStatusClasses(server)">

<span

class="badge">

{{ server.status }}

</span>

<strong>{{ server.name | shorten }}</strong> |

{{ server.instanceType | uppercase }} |

{{ server.started | date:'fullDate' | uppercase }}

</li>

</ul>

</div>

</div>

</div>

**Section 17: Lecture 231//Parametrizing a custom pipe**

1. Shorten.pipe.ts:
2. import { PipeTransform, Pipe } from "@angular/core";
3. @Pipe({
4. name:'shorten'
5. })
6. export class ShortenPipe implements PipeTransform{
7. transform(value: any, limit: number){
8. if(value.length>limit){
9. return value.substr(0, limit) + ' ...';
10. }
11. return value;
12. }
13. }
14. app.component.ts:
15. <div class="container">
16. <div class="row">
17. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
18. <ul class="list-group">
19. <li
20. class="list-group-item"
21. \*ngFor="let server of servers"
22. [ngClass]="getStatusClasses(server)">
23. <span
24. class="badge">
25. {{ server.status }}
26. </span>
27. <strong>{{ server.name | shorten:5 }}</strong> |
28. {{ server.instanceType | uppercase }} |
29. {{ server.started | date:'fullDate' | uppercase }}
30. </li>
31. </ul>
32. </div>
33. </div>
34. </div>

**Section 17: Lecture 232//Creating a Filter pipe**

1. Now to create a new pipe we will run the below command:

ng g p filter

1. Filter.pipe.ts:
2. import { Pipe, PipeTransform } from '@angular/core';
3. @Pipe({
4. name: 'filter'
5. })
6. export class FilterPipe implements PipeTransform {
7. transform(value: any, filterString: string, propName: string): any {
8. if(value.length === 0 || filterString ===''){
9. return value;
10. }
11. const resultArray=[];
12. for (const item of value){
13. if(item[propName]=== filterString){
14. resultArray.push()
15. }
16. }
17. return resultArray;
18. }
19. }

3. app.component.html:

<div class="container">

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<input type="text" [(ngModel)]="filteredStatus">

<hr>

<ul class="list-group">

<li

class="list-group-item"

\*ngFor="let server of servers | filter:filteredStatus:'status'"

[ngClass]="getStatusClasses(server)">

<span

class="badge">

{{ server.status }}

</span>

<strong>{{ server.name | shorten:5 }}</strong> |

{{ server.instanceType | uppercase }} |

{{ server.started | date:'fullDate' | uppercase }}

</li>

</ul>

</div>

</div>

</div>

**Section 17: Lecture 233//Pure and Impure Pipes (or: How to “fix” the Filter pipe)**

1. Filter.pipe.ts:
2. import { Pipe, PipeTransform } from '@angular/core';
3. @Pipe({
4. name: 'filter'
5. //, pure: false
6. })
7. export class FilterPipe implements PipeTransform {
8. transform(value: any, filterString: string, propName: string): any {
9. if(value.length === 0 || filterString ===''){
10. return value;
11. }
12. const resultArray=[];
13. for (const item of value){
14. if(item[propName]=== filterString){
15. resultArray.push()
16. }
17. }
18. return resultArray;
19. }
20. }

2. app.component.html:

<div class="container">

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<input type="text" [(ngModel)]="filteredStatus">

<br>

<button class="btn btn-rimary" (click)="onAddServer()">Add Server</button>

<hr>

<ul class="list-group">

<li

class="list-group-item"

\*ngFor="let server of servers | filter:filteredStatus:'status'"

[ngClass]="getStatusClasses(server)">

<span

class="badge">

{{ server.status }}

</span>

<strong>{{ server.name | shorten:5 }}</strong> |

{{ server.instanceType | uppercase }} |

{{ server.started | date:'fullDate' | uppercase }}

</li>

</ul>

</div>

</div>

</div>

3. app.component.ts:

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

servers = [

{

instanceType: 'medium',

name: 'Production Server',

status: 'stable',

started: new Date(15, 1, 2017)

},

{

instanceType: 'large',

name: 'User Database',

status: 'stable',

started: new Date(15, 1, 2017)

},

{

instanceType: 'small',

name: 'Development Server',

status: 'offline',

started: new Date(15, 1, 2017)

},

{

instanceType: 'small',

name: 'Testing Environment Server',

status: 'stable',

started: new Date(15, 1, 2017)

}

];

filteredStatus = '';

getStatusClasses(server: {instanceType: string, name: string, status: string, started: Date}) {

return {

'list-group-item-success': server.status === 'stable',

'list-group-item-warning': server.status === 'offline',

'list-group-item-danger': server.status === 'critical'

};

}

onAddServer(){

this.servers.push({

instanceType: 'small',

name: 'New Server',

status: 'stable',

started: new Date(26,5,2018)

});

}

}

**Section 17: Lecture 234//Understanding the “async” pipe**

1. There is another built in pipe which us different from all the other pipes - it helps us in handling the Asynchronous data.
2. app.component.ts:
3. import { Component } from '@angular/core';
4. @Component({
5. selector: 'app-root',
6. templateUrl: './app.component.html',
7. styleUrls: ['./app.component.css']
8. })
9. export class AppComponent {
10. appStatus = new Promise(
11. (resolve, reject)=>{
12. setTimeout(()=>{
13. resolve('stable');
14. }, 2000);
15. }
16. );
17. servers = [
18. {
19. instanceType: 'medium',
20. name: 'Production Server',
21. status: 'stable',
22. started: new Date(15, 1, 2017)
23. },
24. {
25. instanceType: 'large',
26. name: 'User Database',
27. status: 'stable',
28. started: new Date(15, 1, 2017)
29. },
30. {
31. instanceType: 'small',
32. name: 'Development Server',
33. status: 'offline',
34. started: new Date(15, 1, 2017)
35. },
36. {
37. instanceType: 'small',
38. name: 'Testing Environment Server',
39. status: 'stable',
40. started: new Date(15, 1, 2017)
41. }
42. ];
43. filteredStatus = '';
44. getStatusClasses(server: {instanceType: string, name: string, status: string, started: Date}) {
45. return {
46. 'list-group-item-success': server.status === 'stable',
47. 'list-group-item-warning': server.status === 'offline',
48. 'list-group-item-danger': server.status === 'critical'
49. };
50. }
51. onAddServer(){
52. this.servers.push({
53. instanceType: 'small',
54. name: 'New Server',
55. status: 'stable',
56. started: new Date(26,5,2018)
57. });
58. }
59. }

3. app.component.html:

<div class="container">

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<input type="text" [(ngModel)]="filteredStatus">

<br>

<button class="btn btn-rimary" (click)="onAddServer()">Add Server</button>

<br><br>

<h2>App Status: {{ appStatus | async }}</h2>

<hr>

<ul class="list-group">

<li

class="list-group-item"

\*ngFor="let server of servers | filter:filteredStatus:'status'"

[ngClass]="getStatusClasses(server)">

<span

class="badge">

{{ server.status }}

</span>

<strong>{{ server.name | shorten:5 }}</strong> |

{{ server.instanceType | uppercase }} |

{{ server.started | date:'fullDate' | uppercase }}

</li>

</ul>

</div>

</div>

</div>

================================END OF SECTION=================================