Лекция 23. Angular: Директивы, Сервисы, DI, Pipe

[**Директивы**](#_m03zj4frdf2u) **3**

[ngClass](#_57oryd4ldmo3) 3

[ngStyle](#_dcgwii4xrk8k) 4

[Создание директив](#_oqg5vnz9f2ne) 5

[Структурные директивы ngIf, ngFor, ngSwitch](#_71gz2p9bf1x) 6

[ngIf](#_fsx6caqy3jmm) 6

[ngFor](#_a8l18oc5kdy6) 7

[ngSwitch](#_1gzf2urdeg6) 7

[**Сервисы**](#_y4v81gdqqtfd) **8**

[**Pipe**](#_vxv4xw39454d) **9**

[Встроенные pipes](#_mpjg9xvkj9eg) 9

[Параметры в pipes](#_lbfzkhuw62f0) 9

[Создание своих pipes](#_qmq8nl6ywznx) 10

# Директивы

Директивы определяют набор инструкций, которые применяются при рендеринге html-кода. Директива представляет класс с директивными метаданными. В TypeScript для прикрепления метаданных к классу применяется декоратор @Directive.

**Компоненты**: компонент по сути также является директивой, а декоратор @Component расширяет возможности декоратора @Directive с помощью добавления функционала по работе с шаблонами.

**Атрибутивные**: они изменяют поведение уже существующего элемента, к которому они применяются. Например, ngModel, ngStyle, ngClass

**Структурные**: они изменяют структуру DOM с помощью добавления, изменения или удаления элементов hmtl. Например, это директивы ngFor и ngIf

## ngClass

Директива ngClass позволяет определить набор классов, которые будут применяться к элементу. Например, определим следующий компонент:

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

@Component({

selector: 'my-app',

template: `<div [ngClass]="{verdanaFont:true}">

<h1>Hello Angular 6</h1>

<p [ngClass]="{segoePrintFont:true}">

Angular 6 представляет модульную архитектуру приложения

</p>

</div>`,

styles: [

`.verdanaFont{font-size:13px; font-family:Verdana;}

.segoePrintFont{font-size:14px; font-family:"Segoe Print";}`

]

})

export class AppComponent { }

В шаблоне для привязки класса к элементу применяется директива [ngClass]="{verdanaFont:true}". Эта директива принимает js-объект, в котором ключи - это названия классов. Этим названиям присваиваются булевые значения true (если класс применяется) и false (если класс не применяется). То есть в данном случае класс verdanaFont будет применяться ко всему блоку div.

В качестве альтернативы мы можем использовать следующие выражения привязки:

<div [class.verdanaFont]="true">

<h1>Hello Angular 6</h1>

<p [class.verdanaFont]="false" [class.segoePrintFont]="true">

Angular 5 представляет модульную архитектуру приложения

</p>

</div>

## ngStyle

Директива ngStyle позволяет задать набор стилей, которые применяются к элементу. В качестве значения директива принимает js-объект, в котором ключи - названия свойств CSS:

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

@Component({

selector: 'my-app',

template: `<div [ngStyle]="{'font-size':'13px', 'font-family':'Verdana'}">

<h1>Hello Angular 5</h1>

<p [ngStyle]="{'font-size':'14px', 'font-family':'Segoe Print'}">

Angular 5 представляет модульную архитектуру приложения

</p>

</div>`,

styles: [

`.verdanaFont{font-size:13px; font-family:Verdana;}

.segoePrintFont{font-size:14px; font-family:"Segoe Print";}`

]

})

export class AppComponent { }

Аналогично для установки стилей можно применять свойства объекта style:

<div [style.fontSize]="'13px'" [style.fontFamily]="'Verdana'">

<h1>Hello Angular 5</h1>

<p [style.fontSize]="'14px'" [style.fontFamily]="'Segoe Print'">

Angular 5 представляет модульную архитектуру приложения

</p>

</div>

Директивы ngClass и ngStyle позволяют устанавливать привязку к выражениям, благодаря чему мы можем динамически менять стили или классы. Например:

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

@Component({

selector: 'my-app',

template: `<div [ngClass]="{invisible: visibility}">

<h1>Hello Angular 5</h1>

<p>

Angular 5 представляет модульную архитектуру приложения

</p>

</div>

<button (click)="toggle()">Toggle</button>`,

styles: [ `.invisible{display:none;}`]

})

export class AppComponent {

visibility: boolean = true;

// переключаем переменную

toggle(){

this.visibility=!this.visibility;

}

}

## Создание директив

Атрибутивные директивы меняют поведение элемента, к которому они применяются. Например, директива ngClass позволяет установить для элемента класс CSS. При этом сама директива применяется к элементу в виде атрибута.

И при необходимости мы можем сами создавать какие-то свои директивы атрибутов для каких-то определенных целей. Итак, создадим свою директиву. Добавим в папку src/app новый файл, который назовем bold.directive.ts:

import {Directive, ElementRef} from '@angular/core';

@Directive({

selector: '[bold]'

})

export class BoldDirective{

constructor(private elementRef: ElementRef){

this.elementRef.nativeElement.style.fontWeight = "bold";

}

}

Директива как и компонент может получать некоторые входные параметры извне. Для этого также используется декоратор **Input**.

import {Directive, Input, OnInit} from '@angular/core';

@Directive({

selector: '[bold]'

})

export class BoldDirective implements OnInit{

@Input() selectedSize = "18px";

@Input() defaultSize = "16px";

private fontSize : string;

private fontWeight = "normal";

ngOnInit(){

this.fontSize = this.defaultSize;

}

constructor(){}

}

<p bold [selectedSize]=”’11px’” ></p>

## Структурные директивы ngIf, ngFor, ngSwitch

### ngIf

Директива ngIf позволяет удалить или, наоборот, отобразить элемент при определенном условии. Например, определим следующий компонент:

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

@Component({

selector: 'my-app',

template: `<p \*ngIf="condition">

Привет мир

</p>

<p \*ngIf="!condition">

Пока мир

</p>

<button (click)="toggle()">Toggle</button>`

})

export class AppComponent {

condition: boolean=true;

toggle(){

this.condition=!this.condition;

}

}

### ngFor

Директива ngFor позволяет перебрать в шаблоне элементы массива. Например:

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

@Component({

selector: 'my-app',

template: `<ul>

<li \*ngFor="let item of items">{{item}}</li>

</ul>`

})

export class AppComponent {

items =["Apple iPhone 7", "Huawei Mate 9", "Samsung Galaxy S7", "Motorola Moto Z"];

}

При переборе элементов нам доступен текущий индекс элемента через переменную index, которую мы также можем использовать. Например:

<div>

<p \*ngFor="let item of items; let i = index">{{i+1}}.{{item}}</p>

</div>

### ngSwitch

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

@Component({

selector: 'my-app',

template: `<div [ngSwitch]="count">

<ng-template \*ngSwitchCase="1">{{count \* 10}}</ng-template>

<ng-template \*ngSwitchCase="2">{{count \* 100}}</ng-template>

<ng-template ngSwitchDefault>{{count \* 1000}}</ng-template>

</div>`

})

export class AppComponent {

count: number = 5;

}

# Сервисы

Сервисы в Angular представляют довольно широкий спектр классов, которые выполняют некоторые специфические задачи, например, логгирование, работу с данными и т.д.

В отличие от компонентов и директив сервисы не работают с представлениями, то есть с разметкой html, не оказывают на нее прямого влияния. Они выполняют строго определенную и достаточно узкую задачу.

Стандартные задачи сервисов:

Предоставление данных приложению. Сервис может сам хранить данные в памяти, либо для получения данных может обращаться к какому-нибудь источнику данных, например, к серверу.

Сервис может представлять канал взаимодействия между отдельными компонентами приложения

Сервис может инкапсулировать бизнес-логику, различные вычислительные задачи, задачи по логгированию, которые лучше выносить из компонентов. Тем самым код компонентов будет сосредоточен непосредственно на работе с представлением. Кроме того, тем самым мы также можем решить проблему повторения кода, если нам потребуется выполнить одну и ту же задачу в разных компонентах и классах

По факту, сервис является классом, реализующим какой-либо функционал.

Для использования, сервис должен быть объявлен в секции providers модуля app.module.

# Pipe

Pipes представляют специальные инструменты, которые позволяют форматировать отображаемые значения. Например, нам надо вывести определенную дату:

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

@Component({

selector: 'my-app',

template: `<div>Без форматирования: {{myDate}}</div>

<div>С форматированием: {{myDate | date}}</div>`

})

export class AppComponent {

myDate = new Date(1961, 3, 12);

}

## Встроенные pipes

В Angular 6 есть ряд встроенных pipes. Основные из них:

CurrencyPipe: форматирует валюту

PercentPipe: форматирует проценты

UpperCasePipe: переводит строку в верхний регистр

LowerCasePipe: переводит строку в нижний регистр

DatePipe: форматирует дату

DecimalPipe: задает формат числа

SlicePipe: обрезает строку

## Параметры в pipes

Pipes могут получать параметры. Например, пайп SlicePipe, который обрезает строку, может получать в качестве параметра, начальный и конечный индексы подстроки, которую надо вырезать:

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

@Component({

selector: 'my-app',

template: `<div>{{welcome | slice:3}}</div>

<div>{{welcome | slice:6:11}}</div>`

})

export class AppComponent {

welcome: string = "Hello World!";

}

**Форматирование дат**

DatePipe в качестве параметра может принимать шаблон даты:

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

@Component({

selector: 'my-app',

template: `<div>{{myDate | date:"dd/MM/yyyy"}}</div>`

})

export class AppComponent {

myDate = Date.now();

}

## Создание своих pipes

Пример простого Pipe:

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'factorial'

})

export class FactorialPipe implements PipeTransform {

transform(value: number, args?: any): number {

if(value<=0) return 0;

let result = 1;

for(let i=1; i<=value; i++){

result = result \* i;

}

return result;

}

}

Не забываем, что Pipe должен быть описан в модуле приложения, в секции declarations:

declarations: [ AppComponent, FactorialPipe],

# Дополнительные материалы

<https://metanit.com/web/angular2/>

# Задание для закрепления материала

Переработать код приложения следующим образом:  
1. В новый сервис вынести задачу по автоматической расстановке кораблей. Требования:

а. Корабли не должны располагаться вплотную.

б. Расположение кораблей пока только вертикальное и горизонтальное, без изгибов

в. На текущий момент, сервис должен реализовывать расстановку кораблей и для игрока и для компьютера

2. Разработать новый сервис переключения ходов между игроком и компьютером.

3. Разработать сервис записи ходов противников.

4. Ячейки поля должны быть кликабельны.