Лекция 22. Обзор Angular.

Angular1, Angular2+, Angular CLI, Shadow Dom, Components, Modules

[**Angular 1 (AngularJS). Обзор**](#_l4cz0ibm7bhp) **3**

[Основные особенности и преимущества AngularJS](#_35m66kwszcgj) 3

[Проблемные места AngularJS](#_q57methvhupx) 4

[**Angular 2+**](#_70h2usk637ko) **5**

[Обзор](#_gs4yemu1cd47) 5

[Angular CLI](#_xsky4hx5gmue) 5

[Shadow DOM & Virtual DOM](#_kq3t6xnsaxjp) 6

[Virtual DOM](#_t6ghc1m4xqa) 6

[Shadow DOM](#_45iktx8pnsey) 6

[Shadow DOM и производительность](#_l0o21nvdfptd) 7

[Modules](#_9nc0jzye2adk) 7

[Components](#_lph56ns6uqyb) 8

[Шаблоны компонента](#_wp7bpz4tfong) 10

[Отображение данных](#_gyvateso7raw) 11

[Переменные как атрибуты тэга](#_8af71pudblww) 11

[Двусторонняя связность](#_7ftcllbo9obu) 11

[Обработка событий элементов](#_wph7rg80k9sy) 12

[Стили компонента](#_kow5oujorxhu) 12

[**Дополнительные материалы**](#_em2hsibkszmj) **14**

[**Задание для закрепления материала**](#_wz24g9bclrud) **15**

# Angular 1 (AngularJS). Обзор

AngularJS был разработан одним из разработчиков интерфейса Gmail. AngularJS был предложен команде Gmail в качестве расширения возможностей обычного HTML. Разработка настолько понравилась команде, что была предложена как основной инструмент разработки и с тех пор поддерживается компанией Google.

Как уже говорилось, AngularJS был разработан для расширения возможностей HTML, и основным инструментом для этого были предложены директивы. Директива может выполнена в качестве атрибута для тэга или же нового тэга. Функционал расширения может быть абсолютно любой.

## Основные особенности и преимущества AngularJS

**Расширяемость**

Основной идеей AngularJS было расширить стандартные возможности HTML и это получилось более чем. Расширить любой функционал достаточно просто. Просто добавить новый атрибут, просто добавить новый тег, очень просто использовать и расширять контроллеры.

**Использование нативного JS**

AngularJS использует стандартные возможности Javascript, никаких es6, никакого TypeScript.

**Двусторонняя связанность данных**

Основной фишкой AngularJS была и остается двусторонняя связанность между переменной с данными и полем для редактирования данных. Т.е. никаких дополнительных действий предпринимать не нужно, чтобы данные из поля формы сразу оказались со связанной переменной в JS.

**Грамотное разделение на MVC**

Данные, обработка этих данных и отображение грамотно разделены и не перемешиваются.

**Простая валидация форм**

Валидация форм с AngularJS действительно стала очень простой.

<https://docs.angularjs.org/guide/forms>

**Повторное использование**

Любую директиву можно использовать повторно, т.к. она не зависит от других компонент и директив, легко выносится в отдельный файл и легко подключается к проекту.

**Локализация проектов**

Перевод проекта на другой язык выполняется очень просто при помощи подключаемых модулей.

**Пригодность к тестированию**

Приложения AngularJS хорошо пригодны для автоматического тестирования, что повышает общую надежность кода.

**Легкость для внедрения**

AngularJS очень легко вставить в проект, даже на отдельную страницу проекта.

Достаточно подключить 2 JS файла: один с библиотекой, второй с приложением.

**Большое количество расширений и модулей**

Самыми интересными из множества дополнений являются: AngularUI и NgResource.

## Проблемные места AngularJS

Основной проблемой AngularJS считается отображение больших объемов данных, при количестве отображаемых элементов более 100, приложение начинает заметно притормаживать.

# Angular 2+

Angular 2 появился как развитие AngularJS, он преследует несколько иную цель, нежели AngularJS. Тогда как основной идеей AngularJS являлось расширение возможностей HTML, Angular2 это полноценный фреймворк для разработки приложений, не только Web, но и десктопных приложений для Windows и Mac, для мобильных приложений и тд.

## Обзор

После выхода Angular2 разработчиками было принято решение чаще выпускать major билды, которые не полностью совместимы с предыдущими версиями, это было сделано для того, чтобы обеспечить ускоренное развитие проекта.

Основными преимуществами Angular2+ считаются:

**Кроссплатформенность**

Angular позволяет разрабатывать кроссплатформенные приложение, ориентированные не только на разный бэкэнд (NodeJS, PHP, .NET, JAVA), но также позволяет разрабатывать мобильные приложения, и десктопные приложения.

**Скорость разработки**

Angular использует различные техники для ускорения разработки: Разделение кода на компоненты, загружаемые только в момент использования, Автоматическую генерацию кода, но основным фактором, позволяющим сократить время на разработку является кроссплатформенность.

**Продуктивность**

Отделение шаблонов от основного кода позволяет быстро собрать внешний вид для приложения.

AngularCLI позволяет генерировать код, автотесты, приложения и деплотить приложение в несколько команд.

Поддержка IDE, позволяет более аккуратно и точно использовать возможности Angular и TypeScript.

## Angular CLI

Angular CLI это консольное приложение, для обеспечения поддержки разработчика. Этот инструмент позволяет быстро генерировать приложение, создавать контроллы, классы, тесты и тд.

**Установка:**

**npm install -g @angular/cli**

следите чтобы перед названием npm пакета обязательно стоял символ @, без этого символа поставляется другой, устаревший пакет.

**Создание нового приложения:  
ng new {app\_name}**

после создания нового приложения в папке с проектом вы можете увидеть базовую структуру приложения с уже настроенными конфигами, все это делает за вас Angular CLI.

**Запуск проекта**

**ng serve --open**

эта команда запустит сервер и откроет проект на главной странице.

## Shadow DOM & Virtual DOM

### Virtual DOM

Virtual DOM - это отказ от ненужных изменений в DOM, которые являются дорогостоящими по производительности, потому что изменения в DOM обычно вызывают повторную визуализацию страницы. Virtual DOM также позволяет собирать сразу несколько изменений, поэтому не каждое изменение вызывает повторную визуализацию, но вместо этого повторное рендеринг происходит только один раз после того, как набор изменений был применен к DOM.

### Shadow DOM

Shadow dom в основном касается инкапсуляции реализации. Один пользовательский элемент может реализовать более или менее сложную логику в сочетании с более или менее сложным DOM. Все веб-приложение произвольной сложности может быть добавлено на страницу с помощью импорта <body> <my-app> </ my-app>, но также более простые многоразовые и составные компоненты могут быть реализованы как пользовательские элементы, где внутреннее представление скрыто в теневой DOM, например <date-picker> </ date-picker>.

Инкапсуляция стиля Shadow DOM также предотвращает случайное применение стилей к элементам, которые разработчик не намеревался, например, потому, что используемая вами библиотека CSS или компонентов изменила селектор, который теперь применяется к другим элементам, которые используют одни и те же имена классов CSS. Стили, добавленные к компонентам, привязаны к этому компоненту и предотвращаются кровотечение или стили.

### Shadow DOM и производительность

Несмотря на то, что Shadow DOM не имеет прямого отношения к производительности, он имеет свои последствия для производительности. Поскольку стили ограничены, браузер может сделать предположения о некоторых изменениях влиять только на ограниченную область страницы (теневое DOM настраиваемого элемента), которое может ограничить повторный рендеринг в области такого компонента, вместо повторного рендеринга всей страницы.

Angular2 использует преимущества обоих миров.

Он использует однонаправленный поток данных и запускает обнаружение изменений только на модели. Если он обнаруживает изменения, это приводит к обновлению DOM путем обновления привязок и создания структурных директив, таких как \* ngFor, \* ngIf, etc. Поэтому DOM обновляется только тогда, когда модель действительно изменилась.

Angular2 использует shadow DOM (только с ViewEncapsulation.Native, который в настоящее время не является стандартным), чтобы использовать возможности инкапсуляции стиля, предоставляемые браузером, или (текущий по умолчанию) просто эмулировать инкапсуляцию стиля путем переписывания стилей, добавленных в компоненты, в качестве обходного пути до собственной тени DOM и переменные CSS (для динамических изменений глобального стиля) становятся широко доступными.

## Modules

Приложение Angular состоит из отдельных модулей. Как правило, приложения состоят из нескольких модулей. И каждое приложение Angular как минимум имеет один корневой модуль (root module), который, согласно условностям, называется **AppModule.**

Для работы модуля ему необходимы ряд библиотек, поэтому в начале файла идет их подключение. Имя каждой библиотеки Angular начинается с префикса @angular.

Библиотеки устанавливаются через пакетный менеджер npm и импортируются с помощью директивы import. Например, импорт функциональности декоратора NgModule из библиотеки @angular/core:

**import { NgModule } from '@angular/core';**

В целом, стандартная ситуация, которую мы обсуждали в разделе ES6.

В Angular модуль это не просто класс. Каждый модуль должен определяться с декоратором @NgModule.

NgModule представляет функцию-декоратора, которая принимает объект, свойства которого описывают метаданные модуля. Наиболее важные свойства:

**declarations**: классы представлений (view classes), которые принадлежат модулю. Angular имеет три типа классов представлений: компоненты (components), директивы (directives), каналы (pipes)

**exports**: набор классов представлений, которые должны использоваться в шаблонах компонентов из других модулей

**imports**: другие модули, классы которых необходимы для шаблонов компонентов из текущего модуля

**providers**: классы, создающие сервисы, используемые модулем

**bootstrap**: корневой компонент, который вызывается по умолчанию при загрузке приложения

Пример:

@NgModule({

imports: [ BrowserModule, FormsModule ],

declarations: [ AppComponent ],

bootstrap: [ AppComponent ],

exports: [ ],

providers: [ ]

})

## Components

Компоненты являются основой любого Angular приложения. Следуя принципу единой ответственности, следует делать компоненты как можно меньше и реализовывать в них один функционал. При этом не запрещается подключать к компоненту более мелкие.

Код компонента состоит из нескольких блоков:

**Блок импорта**

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

import { Phrase } from '../phrase';

import { PhrasesService } from '../services/phrases.service';

import { PhrasesDisplayComponent } from '../phrases-display/phrases-display.component'

**Блок описания компонента**

@Component({

selector: 'app-phrase-edit-popup', // селектор

templateUrl: './phrase-edit-popup.component.html', // шаблон (внешний файл)

styleUrls: ['./phrase-edit-popup.component.css'] // стили

template : `<div>...</div>` // шаблон

})

**Непосредственно, код самого компонента**

export class PhraseEditPopupComponent implements OnInit {

@Input() displayComponent: PhrasesDisplayComponent;

/\*\*

\* @prop boolean

\*/

showForm = false;

/\*\*

\* @prop Array

\*/

phraseTypes:Array<string> = [];

/\*\*

\* @prop Phrase

\*/

phrase:Phrase = new Phrase();

/\*\*

\*

\* @param DataProvider PhrasesService

\*/

constructor(private DataProvider:PhrasesService) { }

/\*\*

\*

\*/

ngOnInit() {

this.phraseTypes = this.DataProvider.getTypesList();

}

/\*\*

\*

\* @param id number

\*/

showPopup(id:number = null) {

if(id) {

this.phrase = this.DataProvider.findById(id);

}

this.showForm = true;

}

}

### Шаблоны компонента

Следуя принципу разделения бизнес-логики и отображения, ангуляр предоставляет возможность подключить шаблон к каждому компоненту отдельно. Есть 2 варианта добавить шаблон к компоненту: Inline-овый и внешний файл.

**Шаблон в блоке описания компонента**

@Component({

selector: 'app-my-component', // селектор

styleUrls: ['./app-my-component.component.css'] // стили

template : `<div> // шаблон

<h1>Заголовок для нового компонента</h1>

<p>{{myComponentData}}</p>

</div>`

})

Обратите внимание, что многострочные шаблоны описываются внутри ``(клавиша тильда) блока.

**Шаблон во внешнем файле**

@Component({

selector: 'app-my-component', // селектор

templateUrl: './app-my-component.component.html', // шаблон (внешний файл)

styleUrls: ['./app-my-component.component.css'] // стили

})

в данном случае, подключается внешний файл в качестве шаблона компонента.

#### Отображение данных

Для отображения данных из компонента используется конструкция {{variable\_name}}, где variable\_name это имя переменной внутри кода компонента, пример:

@Component({

selector: 'app-my-component', // селектор

styleUrls: ['./app-my-component.component.css'] // стили

template : `<div> // шаблон

<h1>Заголовок для нового компонента</h1>

<p>{{my\_var}}</p>

</div>`

})

export class MyComponent{

public my\_var: string = ‘Displaying data...’;

}

В данном случае, при любом изменении переменной my\_var поменяется и отображение страницы.

#### Переменные как атрибуты тэга

В случае, если необходимо передать данные компонента, как атрибут в тег (таким же образом передаются данные в дочерние компоненты) используется следующая конструкция:

<input type="text" [value]="name" />

атрибут, в который необходимо передать данные из компонента оборачивается в [] (квадратные скобки), а переменная передается без каких-либо дополнительных манипуляций. В примере выше текст для инпута будет взят из переменной name компонента. Существуют другие методы задать значение атрибута, но этот является рекомендованным.

#### Двусторонняя связность

Двусторонняя связность является главным преимуществом ангуляра. Эта особенность позволяет отображать данные на странице, если они изменились в результате какого-либо действия в коде, также значение переменной в коде меняется, если изменить связанный с переменной инпут. Все это происходит автоматически, без каких-либо усилий со стороны разработчика.

<input [(ngModel)]="name" />

#### Обработка событий элементов

Angular позволяет привязать событие элемента к обработчику на стороне компонента.

@Component({

selector: 'my-app',

template: `<p>Количество кликов {{count}}</p>

<button (click)="increase()">Click</button>`

})

export class AppComponent {

public count: number=0;

increase() : void {

this.count++;

}

}

### Стили компонента

Как и шаблоны, стили можно прописывать двумя способами: внутри блока описания компонента, или во внешнем файле:

@Component({

selector: 'app-my-component', // селектор

templateUrl: './app-my-component.component.html', // шаблон (внешний файл)

styleUrls: ['./app-my-component.component.css'], // стили во внешнем файле

styles: [`

h1{color:red; font-size:1.2em}

p{margin: 20px 10px;}

`]

})

Как и в шаблонах, многострочные стили помещаются внутрь ``(клавиша тильда)

## 

# Дополнительные материалы

<https://angularjs.org/>

<https://angular.io/>

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

<https://www.youtube.com/playlist?list=PLIcAMDxr6tpprBS29b8IJMhZVcymPr-lM>

# Задание для закрепления материала

На основе разработанной архитектуры разработать структуру компонент для приложения Angular6. Из созданных компонент необходимо реализовать только поля для игры (вместе с ячейками).