



Рафеенко Е.Д.

Web- программирование

Single Page Applications (SPA). Фреймворк Angular

Содержание

- ▶ Архитектура фреймворка Angular.
- ▶ Понятие одностраничного приложения (SPA).
- ▶ Создание Angular проекта.
- ▶ TypeScript.



Angular литература

1. <https://angular.io>
2. <https://www.ngdevelop.tech/>
3. Jeremy Wilken. Angular in Action. — Manning Publications Co, 2018.
4. Фримен А. Angular для профессионалов. — СПб.: Питер, 2018.



Версии Angular

Angular – это фреймворк для разработки приложений на клиентской стороне с использованием архитектуры **model–view–controller (MVC)** and **model–view–view-model (MVVM)**.

Были выпущены следующие версии:

AngularJS или *Angular 1.x*.

Код написан на JavaScript. Поддерживается корпорацией Google. Цели – облегчить разработку и тестирование приложений.

Последняя версия – 1.7.

Angular 2 - полностью переработанная версия (2016г). Код написан на TypeScript.

Angular 4, ..., Angular 15 (ноябрь 2022г.)



JavaEE + Angular

Цель изучения:

разработка web-приложений с использованием платформы Java EE на серверной стороне (backend) и фреймворка Angular на стороне клиента (frontend).



Single Page Applications

Одностраничные приложения (SPA) - это приложения, у которых есть одна точка входа - HTML-страница (может быть index.html); все содержимое приложения динамически добавляется и удаляется с этой страницы.

В одностраничных web-приложениях исходный HTML-документ отправляется в браузер.

Действия пользователя приводят к Ajax запросам небольших фрагментов HTML или данных, вставленных в существующий набор элементов, которые отображаются пользователю.



Angular Setup

Для разработки с использованием фреймворка Angular необходимо установить:

Node.js - среда выполнения для JavaScript, которая работает на серверах. В Node.js можно оперативно переходить на использование новых стандартов ECMAScript по мере их реализации на платформе (использование TypeScript).

npm - менеджер пакетов Node.js. Реестр npm имеет более полумиллиона open-source пакетов, которые может свободно использовать любой Node.js-разработчик.

TypeScript – язык, используемый в разработке Angular2 (>)

```
npm install -g typescript
```



Angular CLI

Angular CLI - используется для создания проектов, генерации кода приложений и библиотек, выполнения текущих задач разработки, таких как тестирование, связывание, развертывание.

Установить **Angular CLI** можно с помощью команды:

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

Команды **Angular CLI**:

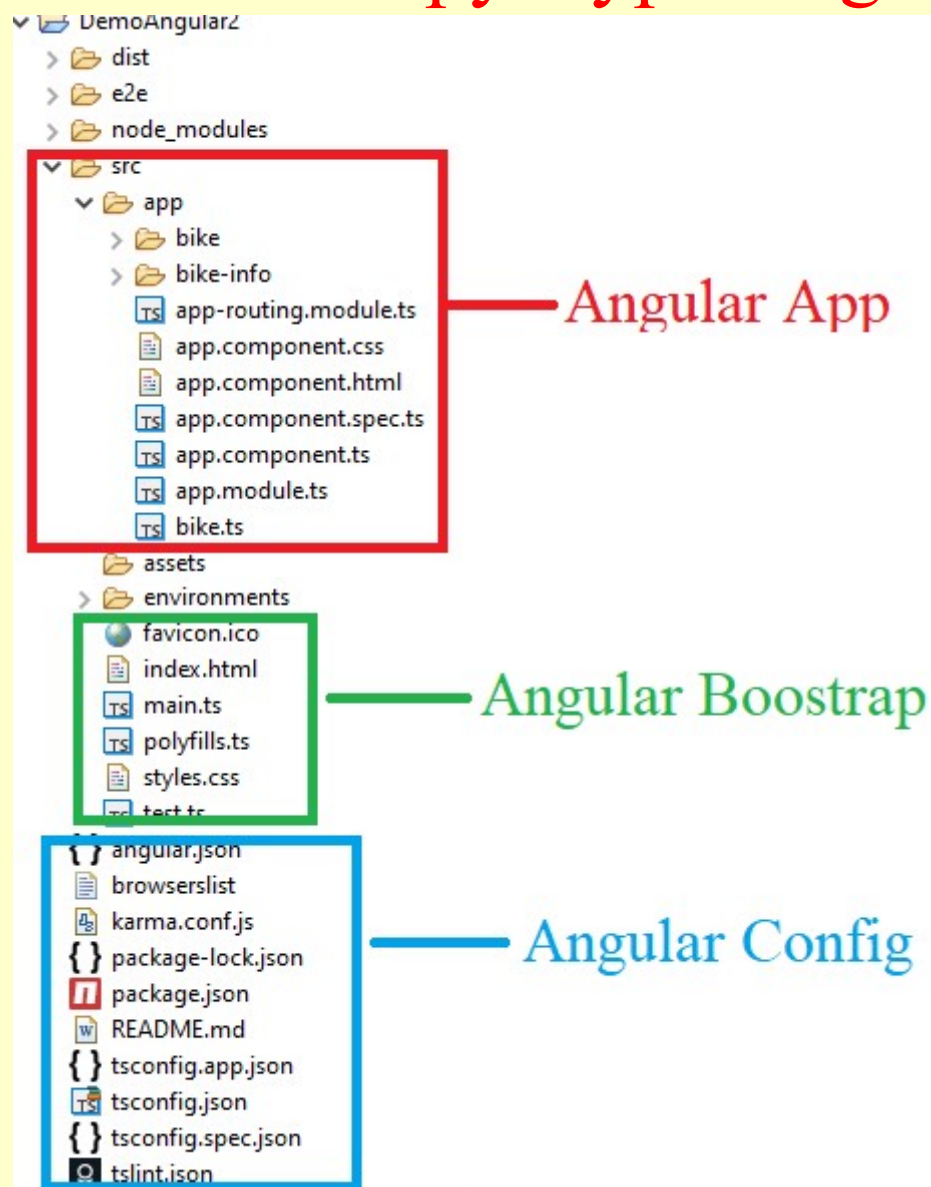
ng new my-app - создать новый Angular проект;

cd my-app

ng serve --open - запустить сервер (Angular CLI имеет встроенный http сервер), собрать приложение, развернуть его на сервере. После успешного выполнения команды приложение по умолчанию будет доступно по url <http://localhost:4200/>



Структура Angular проекта





Структура Angular проекта

Запуск приложения:

`main.ts` — точка входа

Загрузка корневого модуля: `app.module.ts`

Загрузка корневого компонента: `app.component.ts`

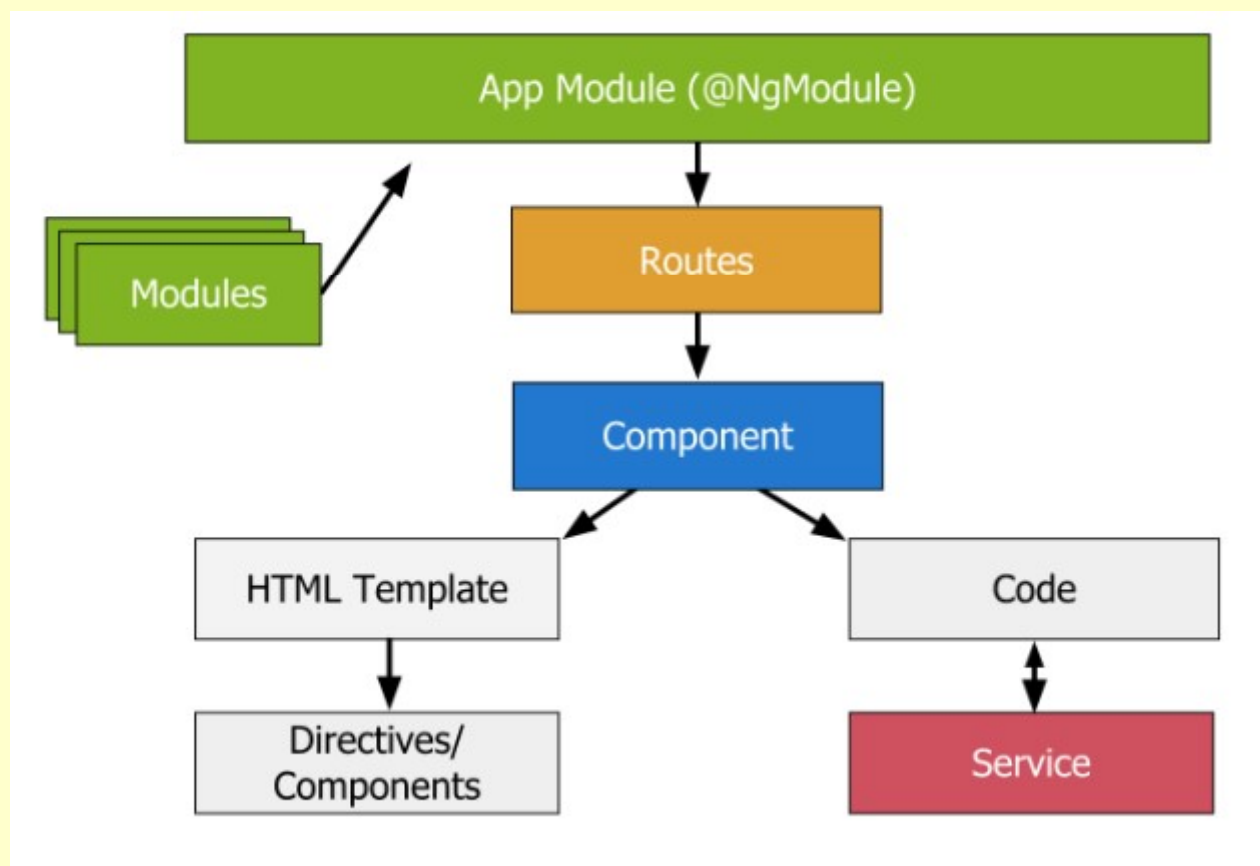
Загрузка отображения, управляемого корневым компонентом: `app.component.html`

`index.html`



Angular архитектура

Angular для организации кода использует *модульную* систему под названием NgModule.





Angular архитектура

Каждое Angular приложение имеет как минимум один класс с декоратором `@NgModule`. Это корневой модуль, обычно называемый `AppModule`:

```
@NgModule({  
  declarations: [  
    AppComponent,  
    BikeComponent,  
    BikeInfoComponent  
  ],  
  imports: [  
    BrowserModule,  
    FormsModule,  
    HttpClientModule  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```



Angular архитектура

Компонент является основным строительным блоком пользовательского интерфейса (UI).

Angular application - это дерево компонентов.

Каждый компонент сопоставляется с шаблоном (**template**).

Компонент содержит свойства, методы, конструктор, а также события ввода, события вывода и методы жизненного цикла, такие как `ngOnInit`, `ngOnDestroy` и т. д.

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  title = 'DemoAngular';  
  ...  
}
```



Angular архитектура

Шаблон (**template**) это HTML код , определяющий как отображать компонент на экране.

Шаблон помимо обычного HTML содержит директивы (directives), события (events), интерполяции (interpolation), привязку данных (data binding) и теги других компонентов.

app.component.html

```
<h1 class="display-3">Welcome to Bike-Shop</h1>
  <p class="lead">THE ONLINE MEGASTORE</p>
  <hr class="my-2">
<h2>Bikes</h2>
<ul>
  <li *ngFor="let bike of bikes" (click)="onSelect(bike)">
    {{bike.model}}
  </li>
</ul>
<app-bike-info [bike]="selectedBike"></app-bike-info>
```



Конфигурация проекта

package.json – файл конфигурации проекта

```
{  
  "dependencies": {
```

Пакеты, необходимые для работы приложения (включая Angular)

```
  },
```

```
  "devDependencies": {
```

Пакеты, используемые в процессе разработки (например, компиляторы и тестовые фреймворки)

```
  },
```

```
  "scripts": {
```

Команды, используемые для компиляции, тестирования и запуска приложения

```
  }
```

```
}
```



Конфигурация проекта

package.json – пример

```
{  
  "name": "DemoTypeScript",  
  "scripts": {  
    "start": "tsc -p tsconfig.json && node Ex2_rx.js"  
  },  
  "dependencies": {  
    "rxjs": "~7.8.0",  
    "tslib": "^2.3.0",  
  },  
  "devDependencies": {  
    "@types/node": "^12.11.1",  
    "ts-node": "~7.0.0",  
    "tslint": "~5.15.0",  
    "typescript": "~4.9.4"  
  }  
}
```

npm install — данная команда загружает пакеты, указанные в package.json, и устанавливает их в папку node_modules

npm start — запуск команды из раздела scripts



TypeScript

Angular как фреймворк стал популярным выбором для фронтенд-разработки, написан с использованием TypeScript. Синтаксис языка имеет некоторое сходство с Java, что упрощает работу Java-разработчиков.

Особенности языка:

- types
- classes
- decorators
- imports
- language utilities



TypeScript

ECMAScript (ES) - это стандартная спецификация, которую реализует JavaScript.

ES5 – это “regular” JavaScript;

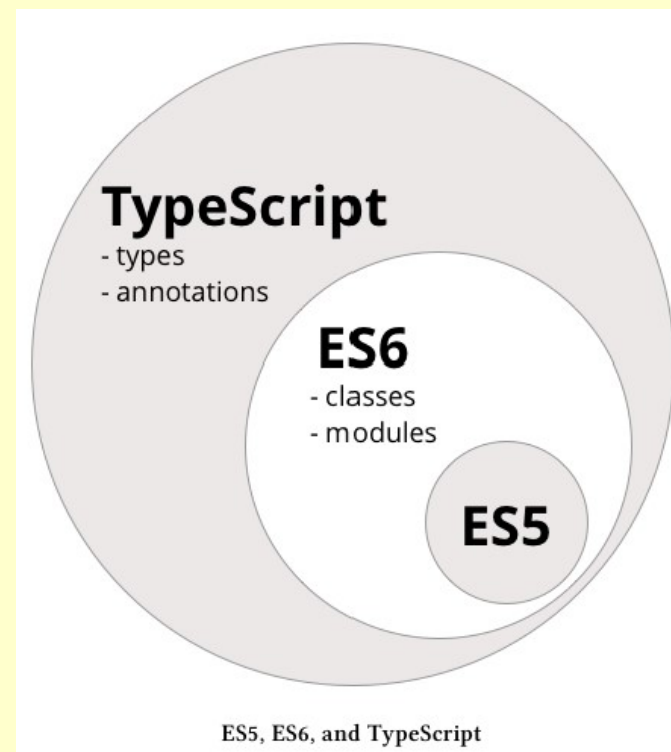
ES6 пытается превратить JavaScript в зрелый язык, устраняя его недостатки;

TypeScript - решение с открытым исходным кодом, надмножество JavaScript.

*TypeScript is a typed superset
of JavaScript that compiles
to plain JavaScript*

(<https://www.typescriptlang.org/>)

Для конвертации TypeScript в ES5
есть единый транспилятор.





TypeScript

vs.

Java Script

```
class Person {  
  name: string;  
  age: number;  
  constructor (name: string, age: number) {  
    this.name = name;  
    this.age = age;  
  }  
  show() {  
    console.log(`Hello ${this.name} is  
    ${this.age} years old`);  
  }  
}  
let p = new Person('bob',35);  
p.show();
```

```
var Person = /** @class */ (function () {  
  function Person(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  Person.prototype.show = function () {  
    console.log("Hello " + this.name +  
    " is " + this.age + " years old");  
  };  
  return Person;  
})();  
var p = new Person('bob', 35);  
p.show();
```

<https://www.typescriptlang.org/#download-links>



TSUN - TypeScript Upgraded Node

TSUN, a TypeScript Upgraded Node – пакет для Node.js.

Поддерживает интерактивную строку REPL(Read Evaluate Print Loop) и интерпретатор TypeScript.

Установить TSUN:

```
npm install -g tsun
```

Выполнить .ts файл:

```
tsun <имя_файла.ts>
```



tsconfig.json

tsconfig.json - конфигурационный файл для TypeScript компилятора. Он определяет как преобразовать (transpile) TypeScript файл в Javascript.

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true
  },
  "exclude": [ "node_modules" ]
}
```



tsconfig.json

compilerOptions – настройки компилятора

"target": "es5" - версия JavaScript для компилятора. Компилятор преобразует TypeScript в простой код JavaScript, использующий только возможности заданной версии. Значение es5 соответствует стандарту ES5, поддерживаемому большинством браузеров;

"module": "commonjs" - формат создания модулей JavaScript; значение должно соответствовать загрузчику, используемому в проекте;

"moduleResolution": "node" - режим обработки команд import компилятором. Со значением node пакеты ищутся в папке node_modules, где их размещает NPM;

"emitDecoratorMetadata": true - Со значением true компилятор включает информацию о декораторе, к которой можно обратиться при помощи пакета reflect-metadata;

"experimentalDecorators": true
},

"exclude": ["node_modules"] - параметр сообщает компилятору, какие каталоги следует игнорировать
}




TypeScript

TypeScript основывается на принципе строгой типизации данных

JavaScript:

```
function sum(a, b) {  
    return a + b;  
}  
  
sum( a: 1, b: '2'); // результат: "12"
```

TypeScript:

```
698  
699 function sum(a: number, b: number): number {  
700     return a + b;  
701 }  
702   
703 sum( a: 1, b: '2');  
704
```

5: Argument of type '"2"' is not assignable to parameter of type 'number'.



TypeScript

Преимущества TypeScript перед JavaScript - обеспечивает лучшую типизацию данных, структуру и читабельность кода, а также обладает обратной совместимостью с JS.

Недостатки - типы контролируются только до этапа компиляции, в результате которого весь код превращается в JavaScript.

Это означает, что если с сервера будут переданы аргументы неверного типа — код об этом не сообщит.



TypeScript - настройка проекта

Глобально установить компилятор TypeScript:

```
npm install -g typescript
```

Скомпилировать TypeScript код в файле index.ts в JavaScript:

```
tsc index
```

TypeScript сообщает об ошибках в текстовом редакторе, но он все равно всегда скомпилирует код - в независимости от того, есть в нем ошибки, или нет.

Создать файл конфигурации TS:

```
tsc -init
```

В результате выполнения команды в корне проекта появится файл tsconfig.json.



TypeScript – типы данных

Примитивные типы:

string, number, boolean, bigint, undefined, null, symbol.

Явная аннотация типа:

```
let firstname : string = 'Danny'
```

Избыточно, т.к. TypeScript автоматически присваивает тип переменной (вывод типа):

```
let firstname = 'Danny'
```



TypeScript – типы данных

Динамические типы:

`any` — указывает на то, что переменная/параметр могут быть чем угодно.

```
let age: any = 100;  
age = true;
```

Union Types

Переменной может быть присвоено несколько типов:

```
let age : number | string;  
age = 22;  
age = '22';
```



TypeScript – типы данных

Литералы:

```
let season: 'winter' | 'summer';  
season = 'winter';
```

Объекты - объявление:

```
let employee: object;  
или  
let employee: {  
    firstName: string;  
    lastName: string;  
    age: number;  
};
```

Объекты – создание:

```
employee = {  
    firstName: 'John',  
    lastName: 'Doe',  
    age: 25  
};
```



TypeScript – типы данных

Arrays:

```
let skills: string[];  
skills.push('Software Design');
```

Tuple – подобны массиву, но количество элементов фиксировано, типы элементов известны:

```
let skill: [string, number];  
skill = ['Programming', 5];
```

Тип `never` не содержит значения, представляет возвращаемый тип функции, которая всегда выдает ошибку, или функции, содержащей бесконечный цикл.



DOM и приведение типов

TypeScript не имеет доступа к DOM. Это означает, что при обращении к DOM-элементам TypeScript не может быть уверен в том, что они существуют.

С оператором ненулевого подтверждения ! можно сказать компилятору, что выражение не равно null или undefined:

```
const link = document.querySelector('a')!;
```

Если нам надо найти DOM-элемент по его классу или id, надо сообщить TypeScript, что этот элемент существует, и что он типа HTMLElement. Для этого используется приведение типов (ключевое слово as):

```
const form =  
document.getElementById('signup-form') as HTMLElement;
```



Функции в TypeScript

Для функции необходимо указать какие типы должны быть у аргументов функции и какой тип должен быть у возвращаемого из функции значения.

```
function circle(diam: number): string {  
    return 'Длина окружности: ' + Math.PI * diam;  
}
```

То же самое, но со стрелочной функцией ES6:

```
const circle = (diam: number): string => {  
    return 'Длина окружности: ' + Math.PI * diam;  
}
```

После параметра можно добавить `?`, чтобы сделать его необязательным.