



Angular

Main Concepts

Точка входа в программу

Как происходит запуск фреймворка, что он делает, как происходит инициализация и с чего он начинает свою работу? Если Вы задавали себе хоть один такой вопрос, обратите внимание на файл **main.ts**.

```
}if (environment.production) {  
    enableProdMode();  
}  
  
platformBrowserDynamic().bootstrapModule(AppModule);
```

Важно понимать, что приложение не стартует одновременно всем скоупом, а точка входа в программу определяется **главным модулем** фреймворка.

Что происходит в браузере?

Ни один браузер не работает напрямую с **TypeScript**, а это значит что где-то происходит подключение сгенерированного скрипта к нашей странице.

Давайте посмотрим ближе на наш файл **index.html**, где вероятнее всего происходит подключение **js** файла.

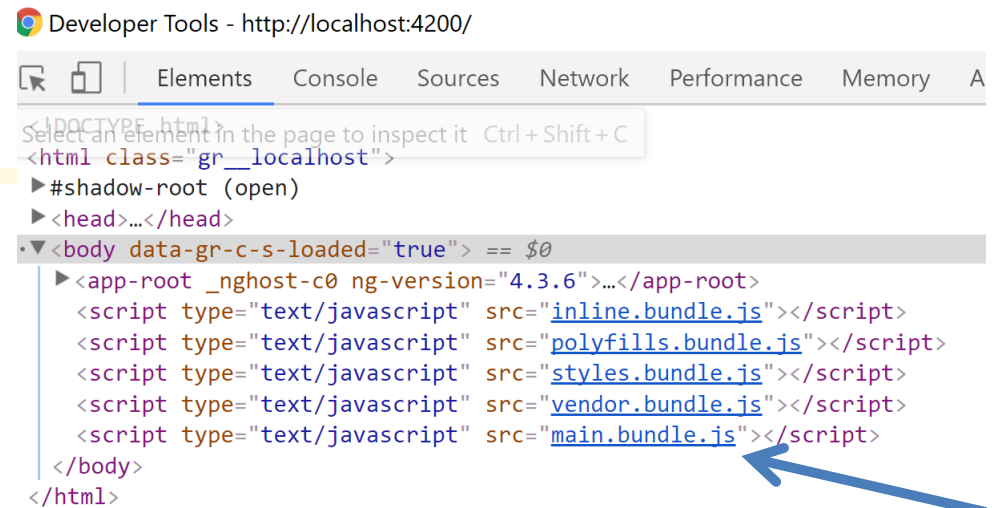
```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>FirstProject</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

А где же импорт? 😞

Давайте поищем в браузере!

main.bundle.js - и есть
наше приложение!



Главный модуль приложения

Главный модуль приложения представляет собой следующий файл:

```
import { BrowserModule } from '@angular/platform-browser';  
import { NgModule } from '@angular/core';
```

```
import { AppComponent } from './app.component';
```

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


Здесь мы говорим **Angular**, что принадлежит нашему приложению.

- **declaration** – включает в себя компоненты, директивы и пайпы, которые использует наше приложение
- **import** – определяет для Angular дополнительно подключаемые модули
- **providers** – определяет список сервисов для данного модуля

Селекторы

Одним из важнейших пониманий работы **Angular** – есть понимание роли **селектора компонента**.


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



Когда будет использован селектор на любой из доступных страниц, вместо него **Angular** подставит **html**, который ему соответствует.

При задании селектора, помните, что его **имя должно быть уникально!**

```
@Component ({  
  selector: 'app-user-component',  
  templateUrl: './user-component.component.html',  
  styleUrls: ['./user-component.component.css']  
})
```



Angular добавляет префикс, чтобы случайно не переопределить селектор **html** или библиотеки

Несколько заданий

Задание 1: удалите файл `app.component.html`, `app.component.css` и изученным ранее способом объявите стилизованный **h1** для нашей главной страницы

Задание 2: верните исходную структуру приложения (восстановите файлы `app.component.html` и `app.component.css`).

Создайте стилизованный **h1** элемент для главного компонента. **Создайте новый компонент**, добавьте в него стилизованный текст и используйте созданный компонент в главном.

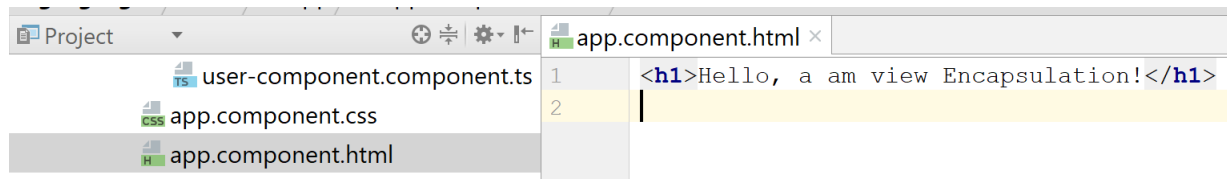
View Encapsulation

ViewEncapsulation – концепт используемый в **Angular**.

Это в свою очередь заставляет **Angular** эмулировать концепт, который называется **ShadowDOM**. Это значит что каждый компонент имеет свой **DOM**.

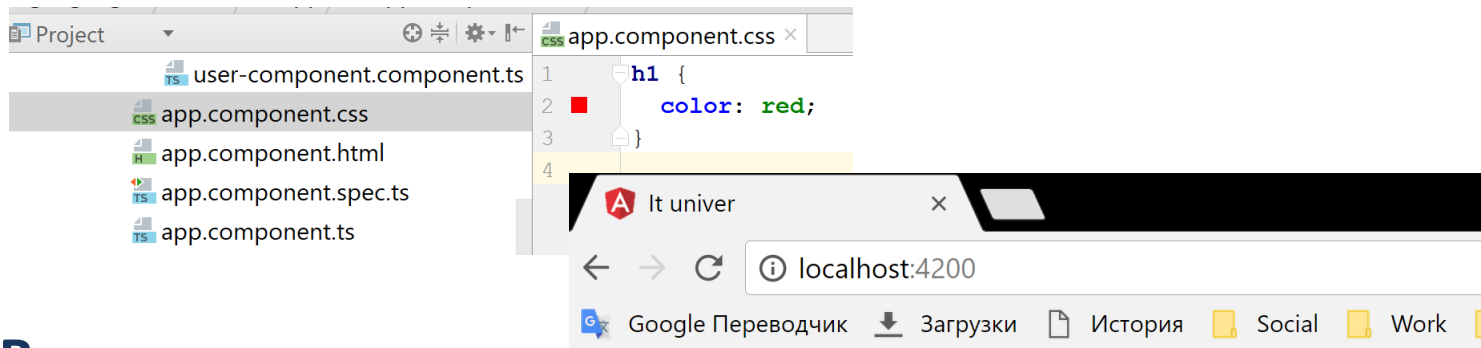
Давайте посмотрим на один из стилей, созданный нами и обработанным в **Angular**.

Для начала создадим в главном компоненте заголовок **h1**.



```
1 <h1>Hello, a am view Encapsulation!</h1>
```

И раскрасим его в **красный** цвет.



```
1 h1 {  
2   color: red;  
3 }  
4
```

It univer

localhost:4200

В итоге получим:

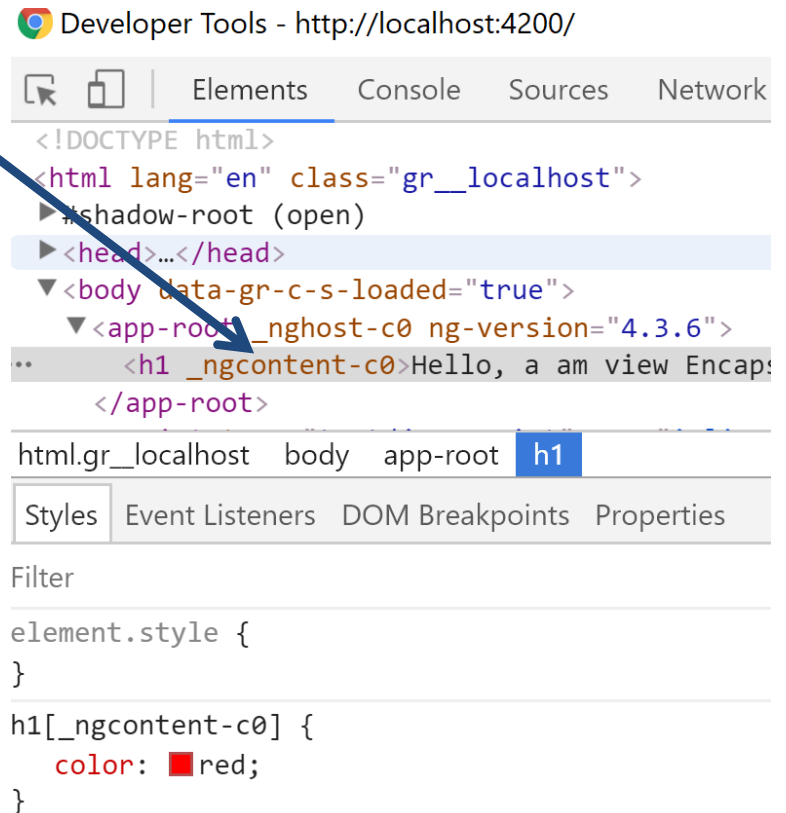
Hello, a am view Encapsulation!

View Encapsulation

Давайте посмотрим на наш стиль используя инструменты разработчика в браузере.

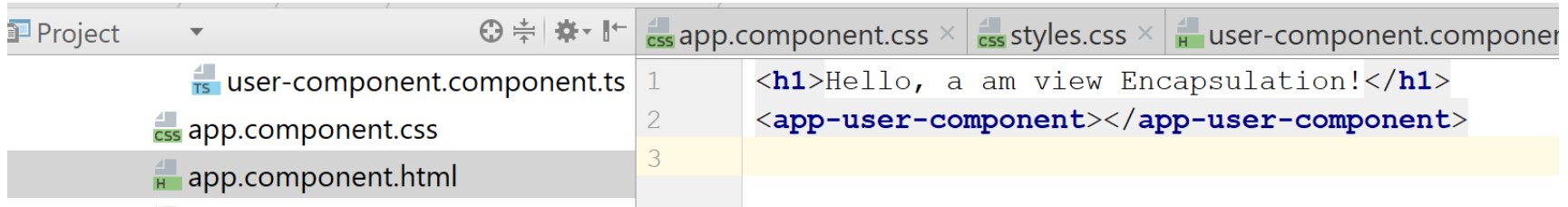
Стоп, кто объявлял такой стиль `h1[_ngcontent-c0]`?

Дело в том, что **Angular** эмулирует **ShadowDOM** отдельно для каждого компонента, таким образом инкапсулируя стили.



View Encapsulation

Давайте создадим ещё один компонент, определим в нем заголовок **h1**.



Вопрос: будет ли стиль главного компонента, изменять цвет **h1** компонента **<app-user-component>**?

Правильный ответ – **нет!** Каждый из компонентов имеет свой **уникальный селектор** этого стиля! В данной ситуации – второй компонент не имеет стиля в принципе.

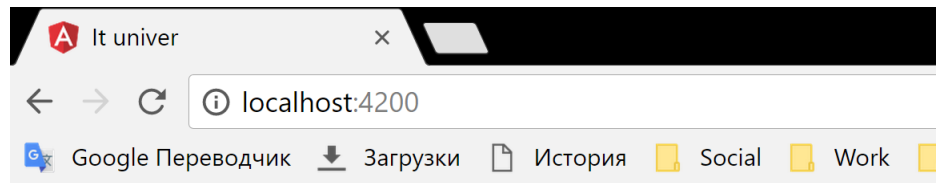
```
h1[_ngcontent-c0] {
  color: red;
}
```

View Encapsulation

Ввиду данной концепции возникает вопрос – а что если есть одинаковые стили для нескольких компонентов?

Для решения этой ситуации существует файл **style.css** и стили определенные в нем, будут применены ко всем компонентам.

Таким образом, если мы **удалим** все ранее определенные стили с **главного и дочернего компонента**, и определим **h1** в файле **style.css** – увидим следующее.



Hello, a am view Encapsulation!

I am user defined component!

Вопрос: что будет, если определить стиль **h1** в любом из используемых компонентов?


View Encapsulation

Существует 3 типа инкапсуляции стилей в Angular:

- **Emulated** (default) (уникальные (emulated) id + global style.css)
- **None** – отсутствие любого **ShadowDOM**, а соответственно и инкапсуляции стилей (все файлы стилей, **равны**, не важно это файл **style.css** или **component.css**).
- **Native** – использование родного **ShadowDOM** браузера (использование **css** только этого компонента, **полная** изоляция внутри компонентов).

Изменение типа **ViewEncapsulation** происходит, дополнением метаданных компонента еще одним параметром.

```
@Component ({  
  selector: 'app-user-component',  
  templateUrl: './user-component.component.html',  
  styleUrls: ['./user-component.component.css'],  
  encapsulation: ViewEncapsulation.Native  
})
```



Совет: старайтесь использовать **ViewEncapsulation.Emulated (by default!)**, так как это принцип, который предоставляет Angular. В остальных случаях, могут возникнуть проблемы с отображением и ожидаемым поведением.

View Encapsulation

В данных **заданиях**, работа происходит со стилизацией **h1** элементов, как и в предыдущих примерах. **Ваша задача** – ответить, какой стиль будет применен в каждом компоненте.

Состояние проекта:

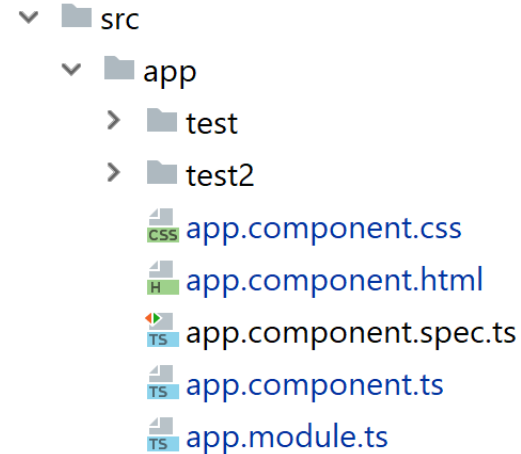
ViewEncapsulation.Emulated:

Состояние:

- **style.css**: color:red;
- **app.component.css**: color: black
- **test.component.css**: color: blue
- **test2.component.css**: color: yellow

Результат:

- **app.component.html** – black
- **test.component.html** – blue
- **test2.component.css** – yellow



View Encapsulation

ViewEncapsulation.Emulated:

Состояние:

- **style.css:** color:red;
- **app.component.css:** color: black
- **test.component.css:** -
- **test2.component.css:** color: yellow

Результат:

- **app.component.html** – black
- **test.component.html** – red
- **test2.component.css** – yellow

ViewEncapsulation.Emulated:

Состояние:

- **style.css:** color:red **!important;**
- **app.component.css:** color: black
- **test.component.css:** color: blue
- **test2.component.css:** color: yellow

Результат:

- **app.component.html** – red
- **test.component.html** – red
- **test2.component.css** – red

View Encapsulation

ViewEncapsulation.Native:

Состояние:

- **style.css**: color:red;
- **app.component.css**: color: black
- **test.component.css**: color: blue
- **test2.component.css**: color: yellow

Результат:

- **app.component.html** – black
- **test.component.html** – blue
- **test2.component.css** – yellow

ViewEncapsulation.Native:

Состояние:

- **style.css**: color:red;
- **app.component.css**: color: black
- **test.component.css**: color: blue **!important**
- **test2.component.css**: color: yellow

Результат:

- **app.component.html** – black
- **test.component.html** – blue
- **test2.component.css** – yellow

View Encapsulation

ViewEncapsulation.Native:

Состояние:

- **style.css: color:red !important;**
- **app.component.css: color: black**
- **test.component.css: color: blue**
- **test2.component.css: color: yellow**

Результат:

- **app.component.html – black**
- **test.component.html – blue**
- **test2.component.css – yellow**

ViewEncapsulation.Native:

Состояние:

- **style.css: color:red;**
- **app.component.css: color: black !important**
- **test.component.css: color: blue**
- **test2.component.css: color: yellow**

Результат:

- **app.component.html – black**
- **test.component.html – blue**
- **test2.component.css – yellow**

View Encapsulation

ViewEncapsulation.None:

Состояние:

- **style.css:** color:red;
- **app.component.css:** color: black
- **test.component.css:** color: blue
- **test2.component.css:** color: yellow

Результат:

- **app.component.html** – yellow
- **test.component.html** – yellow
- **test2.component.css** – yellow

```
<h1>App Component</h1>
<app-test></app-test>
<app-test2></app-test2>
```

ViewEncapsulation.None:

Состояние:

- **style.css:** color:red;
- **app.component.css:** color: black
- **test.component.css:** color: blue **!important**
- **test2.component.css:** color: yellow

Результат:

- **app.component.html** – blue
- **test.component.html** – blue
- **test2.component.css** – blue

Databinding

Databinding – означает взаимодействие с данными.

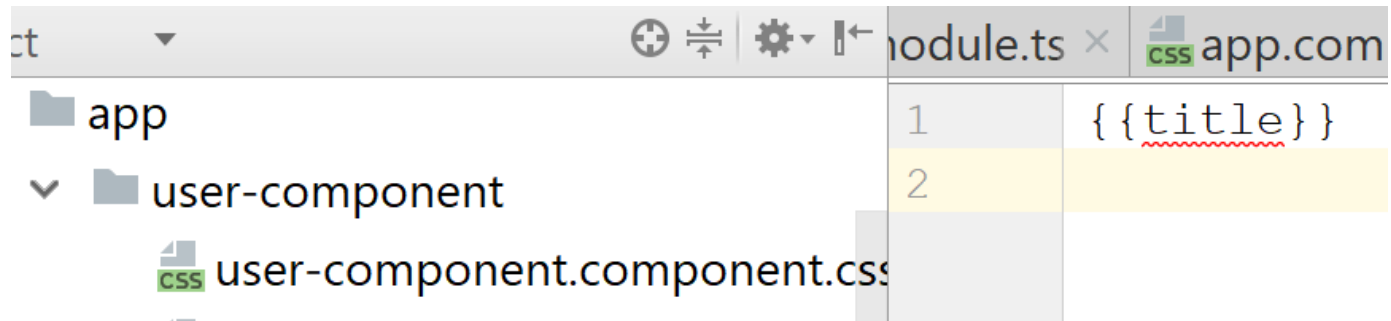


Существует несколько способов передачи данных:

➤ String interpolation

```
private title = 'Hello, a am view Encapsulation!';
```


JIT – OK
AOT - ERROR



Databinding

- **Property binding** – отправка значения в DOM (по сути html код)


```
<button [disabled]="true">Click me!</button>
```



здесь может быть переменная или **любое** выражение, которое удовлетворяет **необходимый тип** для данного **property binding**

- **Event binding** – некая противоположность Property binding. Здесь мы ожидаем что **некоторое значение отдаст сам элемент** (сгенерируется event)

```
<button (click)="onClick()">Click me!</button>
```



Пытаемся обработать соответствующее действие, выполненное над **html** элементом

- **Two-way binding** – позволяет изменять данные в обе стороны (view меняет данные в компоненте, данные в компоненте меняют view). Для использования **нужен импорт FormsModule**.

```
<input [(ngModel)]="inputValue">
```

Property & Event Binding

- **DOM properties & events** – означает использование **встроенных** возможностей DOM.

```
<img [src]="...">  
<img (click)="...">
```

- **Directive properties** – использование встроенных проперти Angular

```
<div [ngClass]="..."></div>  
<button (onSubmit)="..."></button>
```

- **Component properties** – проперти объявленные в компоненте (user defined)

Создание собственных проперти возможно благодаря следующим метаданным:

@Input() propertyName: string;

@Output() eventName = new EventEmitter<>();

Жизненный цикл компонента

