

# Angular

---

- это чистовик - конспект
- вопросы для подготовки к собесу
- база знаний
- черновик - /frontend/framework/angular2.md

## Angular

### 1. сравнение версий 2-10

- общие для большинства версий изменения: версии NS/RxJS, производительность, Material компоненты
- 2 - typescript, dart
- 4 - renderer v2
- 5 - 2017 - HttpClientModule заменил http
- 6 - 2018 - angular elements(web custom elements), tree-shakable providers(providedIn:root), rxjs 6, Ivy
- 7 - drag and drop, node v10, service worker
- 8 - webworker, ES6 импорт модулей `()=>`
- 9 - Ivy по-умолчанию, ullTemplateTypeCheck, strictTemplates в tsc
- 10 - исправили 700 и посмотрели 2000 ошибок, компонент диапазона дат
- 11 - harnesses for all of the components, webpack 5 experimental, ng serve --hmr, tslint deprecated, IE9/10 deprecated
- 12 - nullish coalescing, tailwind css, webpack 5 prod, IE11 deprecated

### 2. Различие между AngularJS и Angular

- rxjs, DI

### 3. преимущества Angular

- поскольку вместо virtualDOM у нас shadowDOM и декларативный стиль HTML вместо императивного JS/аннотации
- не надо делать обёртки для сторонних UI компонентов
- Много библиотек с поддержкой вендора(обновления+наставления) вроде router/ngModule, покрывающих все типовые нужды SPA
- чёткие подробные наставления
- релиз каждые 6 мес с полной обратной совместимостью
- не нужно думать о совместимости версий внешних библиотек, всё включено, задокументировано
- DI - контроль количества экземпляров, изоляция, оптимизация

### 4. недостатки

- большой размер из-за Angular CLI, но после компиляции его убирают. В старом режиме JIT компилятор на борту также увеличивает объём.
- Приложение может быть сложным, Angular - это и полный framework, и platform с набором инструментов. Из-за этого его сложнее учить. В отличие от React тут включены по-умолчанию управление состоянием, сервисы, HTTP библиотека, DI, валидация форм, маршрутизация URL.

### 5. ngModule

- [declarations](#) - только компоненты, конвейеры и директивы
  - нельзя декларировать импортированные модули или уже декларированные компоненты
- `imports` - импортировать декларации из другого модуля
- `exports` - сделать видимыми для импорта свои декларации
- `providers` - сделать видимыми для экспорта сервисы и другие внедряемые зависимости

#### 6. веб-компоненты, `custom elements`

- элементы - реализации обёртки над веб-компонентами, расширение DOM API
- стандартный подход к компонентам без применения ангуляр специфичного синтаксиса
- можно написать кастомный элемент формы, например, кастомный `select`.
- Статические компоненты рисуются без участия ангуляр, на стороне браузера
- динамические компоненты без применения `dynamic components`  
`componentFactoryResolver.resolveComponentFactory`, т.е. меньше кода для встраивания логики в отрисовку.
- [https://developer.mozilla.org/en-US/docs/Web/Web\\_Components/Using\\_custom\\_elements](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_custom_elements)  
`customElements.define('word-count', WordCount, { extends: 'p' });`
- <https://angular.io/guide/elements>
- [Elements in v6 and Beyond - Rob Wormald](#)

#### 7. JIT [Deep Dive into the Angular Compiler | Alex Rickabaugh | 2019](#)

- перед чтением полей
- в декораторах
- простой тайпскрипт
- Больше объём приложения с Angular compiler.

#### 8. AOT

- компилирует код за декораторами `@`
- Лучше отрисовка с AOT. Браузер рисует без компиляции.
- Меньше XHR запросов. Компилятор сразу добавляет в код HTML+CSS.
- Меньше объём приложения без Angular compiler.
- Определяет ошибки шаблонов/связывания во время компилирования.
- Лучше ИБ из-за уменьшения загрузок и выполнения кода на клиенте
- нельзя использовать стрелочные функции и функциональные выражения внутри метаданных за декораторами `@`

#### 9. ivy

- Ivy - новый компилятор и отрисовщик. По-умолчанию с 9 версии
- локальная компиляция классов для лучшего деревотрясения. Компонент может быть вне модуля
- <https://github.com/angular/angular/blob/master/packages/compiler/design/architecture.md>
- [готовность ivy](#)
- генерирует меньше кода, он более понятный, его легче отлаживать - ссылки в консоли на шаблон
- [angular compiler](#)
- [lazyLoad ivy](#)
- [Angular In Depth workshop on dynamic rendering with Ivy](#)
- <https://medium.com/ngx/angular-ivy-renderer-%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C-93-20e0aff2e861>

# DI

## 1. Иерархия Injector'ов, какие бывают, сколько их может быть?

- <https://angular.io/guide/hierarchical-dependency-injection>
- первая: ModuleInjector иерархия через @NgModule() или @Injectable()
- вторая: ElementInjector иерархия для каждого DOM элемента. ElementInjector пустой пока нет providedIn в @Directive() или @Component()
- иерархия NullInjector()-->platformModuleInjector(Renderer, Sanitizer)-->rootModuleInjector(services)-->componentInjector(ElementRef, ChangeDetectorRef)
- @Injectable() предпочтительнее @NgModule() providers, т.к. позволяют tree-shaking
- The `platformBrowserDynamic()` method creates an `platformModuleInjector` for platform/browser-specific dependencies like domsanitizer. Configured by `platformBrowser().extraProviders`
- можно создать дочерние ModuleInjectors при ленивой загрузке
- стратегии разрешения зависимостей: @Optional(), @Self(), @SkipSelf(), @Host()  
<https://angular.io/guide/dependency-injection-in-action#make-a-dependency-optional-and-limit-search-with-host>
  - @Optional() присваивает null не найденным провайдерам сервисов. Нужен для игнорирования ошибок
  - @Self() ищет в текущем ElementInjector component или directive
  - @SkipSelf() ищет в родительском ElementInjector
  - @Host() ищет в component и ниже по вложенному дереву

## 2. Управление экземплярами <https://angular.io/guide/dependency-injection-in-action>

- песочница, отдельные экземпляры: регистрируем в providers аннотациях компонентов  
<https://angular.io/guide/dependency-injection-in-action#multiple-service-instances-sandboxing>
- 

## 3. Сколько у нас инжекторов, у одного модуля и трёх компонентов?

## 4. Dependency injection

- **провайдер**
  - Объект, который реализует один из интерфейсов **Provider**
  - предоставляет инжектору порядок разрешения зависимости, связанной с токеном/идентификатором
  - может предоставлять разные реализации одной и той же зависимости
- **токен**
  - объект, который реализует интерфейс **InjectionToken**
- **инжектор**
  - Объект, который находит именованную зависимость в своём кэше, либо создаёт её используя провайдер
  - Предоставляет и внедряет синглтон
  - Создаются автоматом для модулей в ходе bootstrap и наследуются в иерархии компонентов
- зачем

- для работы с сервисами и модулями
  - меньше кода в конструкторах
  - облегчение рефакторинга, автоматическая инъекция зависимостей по всей цепочке
  - облегчение юнит-тестирования сервисов
  - переиспользование сервисов
- синглтон <https://angular.io/guide/architecture-services#providing-services>
    - для всего приложения: в аннотации компонента `@Injectable({providedIn: 'root',})` <https://angular.io/api/core/Injectable#injectable>
      - root: для приложения
      - platform - для всех приложений
    - для модуля: в модуле `@NgModule({providers: [...]`
    - для компонента: в аннотации компонента `@Component({...providers: [HeroService ],...})`
  - используется
    - в маршрутизаторе для хранения по одной копии состояния маршрута
  - forRoot: модуль с providers
  - forChild: отдельный экземпляр модуля без providers для ленивой загрузки
  - нельзя внедрять интерфейсы <https://angular.io/guide/dependency-injection-providers#interfaces-and-dependency-injection> , но можно абстрактные классы <https://angular.io/guide/dependency-injection-in-action#class-interface>
  - **внедрение лёгких токенов** для уменьшения размера библиотек

```
@Component({selector: 'lib-header',...,})
class LibHeaderComponent {}

@Component({selector: 'lib-card',...,})
class LibCardComponent {
  @ContentChild(LibHeaderComponent)
  header: LibHeaderComponent|null = null;
}
```

- **внедрение классов**

```
// более многословный вариант стандартного внедрения
{ provide: HeroService, useClass: HeroService },
// подмена класса
{ provide: LoggerService, useClass: DateLoggerService }
```

- **внедрение псевдонимов**

```
providers: [  
  NewLogger,  
  // Alias OldLogger w/ reference to NewLogger  
  { provide: OldLogger, useExisting: NewLogger}  
]
```

- внедрение/разрыв циклической зависимости

```
providers: [{ provide: Parent, useExisting: forwardRef(() =>  
  AlexComponent) }],
```

- внедрение не классов

```
// src/app/app.config.ts  
import { InjectionToken } from '@angular/core';  
export const APP_CONFIG = new InjectionToken<AppConfig>  
( 'app.config' );  
  
// src/app/providers.component.ts  
providers: [{ provide: APP_CONFIG, useValue: HERO_DI_CONFIG  
}]  
  
// src/app/app.component.ts  
constructor(@Inject(APP_CONFIG) config: AppConfig) {  
  this.title = config.title;  
}
```

- внедрение по условию

```
// src/app/heroes/hero.service.ts  
constructor(  
  private logger: Logger,  
  private isAuthorized: boolean  
) { }  
  
getHeroes() {  
  const auth = this.isAuthorized ? 'authorized ' :  
'unauthorized';  
  this.logger.log(`Getting heroes for ${auth} user.`);  
  return HEROES.filter(hero => this.isAuthorized ||  
!hero.isSecret);  
}  
  
// src/app/heroes/hero.service.provider.ts  
const heroServiceFactory = (logger: Logger, userService:  
UserService) => {
```

```
        return new HeroService(logger,
userService.user.isAuthenticated);
    };

    // src/app/heroes/hero.service.provider.ts
    export let heroServiceProvider = {
        provide: HeroService,
        useFactory: heroServiceFactory,
        deps: [Logger, UserService]
    };
};
```

- внедрение в DOM через ElementRef

```
@Directive({selector: '[appHighlight]'})
export class HighlightDirective {

    @Input('appHighlight') highlightColor: string;

    private el: HTMLElement;

    constructor(el: ElementRef) {
        this.el = el.nativeElement;
    }

    @HostListener('mouseenter') onMouseEnter() {
        this.highlight(this.highlightColor || 'cyan');
    }

    @HostListener('mouseleave') onMouseLeave() {
        this.highlight(null);
    }

    private highlight(color: string) {
        this.el.style.backgroundColor = color;
    }
}

// HTML
<div appHighlight="yellow">
```

## Конвейеры, директивы и компоненты

### 1. связывание данных

```
<!-- значение переменной, присвоение в свойство -->
<img [src]="itemImageUrl">
<!-- просто строка, присвоение в атрибут -->
<app-item-detail childItem="parentItem"></app-item-detail>
<!-- значение переменной, присвоение в атрибут -->
```

```

<app-item-detail childItem="{{parentItem}}"></app-item-detail>
<tr><td [colSpan]="1 + 1">Three-Four</td></tr>
<p> is the <i>interpolated</i> image.
</p>
<!-- присвоение в атрибут интерполированного значения, некоторые
атрибуты этого требуют-->
<p> is the <i>interpolated</i>
image.</p>
<p><img [src]="itemImageUrl"> is the <i>property bound</i> image.
</p>
<p><span>"{{interpolationTitle}}" is the <i>interpolated</i>
title.</span></p>
<p>"<span [innerHTML]="propertyTitle"></span>" is the <i>property
bound</i> title.</p>

```

## 2. защита от null/undefined <https://angular.io/guide/template-expression-operators>

```

<!-- Assert color is defined, even if according to the `Item` type
it could be undefined. -->
<p>The item's color is: {{item.color!.toUpperCase()}}</p>
<p>The item's undeclared best by date is:
{{ $any(item).bestByDate }}</p>
<p>The item's undeclared best by date is:
{{ $any(this).bestByDate }}</p>

```

## 3. Какие обязательные props для Component

- template
- style

## 4. Разница поведения между ng-if и style: hidden

- **ngif** удаляет элемент из DOM
- можно прятать структурные директивы в ng-container

## 5. В чем разница между Directive и Component.

- компонент - частный случай директивы, с шаблоном
- Структурные директивы — манипуляции с DOM: **ngif**, **ngfor**
  - Обязательно импортируют: Input, TemplateRef, and ViewContainerRef symbols
  - this.viewContainer.createEmbeddedView(this.templateRef);
- Атрибутные директивы — манипулирует: element, component, directive. **ngStyle**, **ngClass**

```

@Directive({
  selector: '[appHighlight]'
})

```

## 6. Директивы

- <https://angular.io/guide/built-in-directives>
- `ngClass`
  - <https://angular.io/api/common/NgClass#description>
  - `ngClass: string | string[] | Set<string> | { [klass: string]: any; }`

```
<div [ngClass]="isSpecial ? 'special' : ''">
  <div [ngClass]="currentClasses">
    <some-element [ngClass]='first second'>...</some-element>
    <some-element [ngClass]='["first", "second"]'>...</some-
element>
    <some-element [ngClass]='{"first": true, "second": true,
'third': false}'>...</some-element>
    <some-element [ngClass]="stringExp|arrayExp|objExp">...
</some-element>
    <some-element [ngClass]='{"class1 class2 class3" : true}'>...
</some-element>
```

```
currentClasses: {};
// src/app/app.component.ts
setCurrentClasses() {
  // CSS classes: added/removed per current state of
component properties
  this.currentClasses = {
    saveable: this.canSave,
    modified: !this.isUnchanged,
    special: this.isSpecial
  };
}
```

- `ngStyle`

```
<div [ngStyle]="currentStyles">
```

```
currentStyles: {};
/* . . . */
setCurrentStyles() {
  // CSS styles: set per current state of component
properties
```



```
        this.currentStyles = {
          'font-style': this.canSave      ? 'italic' : 'normal',
          'font-weight': !this.isUnchanged ? 'bold'  : 'normal',
          'font-size':   this.isSpecial   ? '24px'   : '12px'
        };
      }
    }
```

- 

## 7. NG-FOR. TrackBy зачем нужен, преимущества.

- для более быстрого вычисления признака замены значений вложенного объекта в ячейке массива, быстрота перерисовки

## 8. LifeCycle Hooks. Назовите основные, которые используете в приложении.

- ngOnChanges()

```
ngOnChanges(changes: SimpleChanges) {
  for (let propName in changes) {
    let chng = changes[propName];
    let cur  = JSON.stringify(chng.currentValue);
    let prev = JSON.stringify(chng.previousValue);
  }
}
```

- ngOnInit()
- ngDoCheck()
- ngAfterContentInit()
- ngAfterContentChecked()
- ngAfterViewInit()
- ngAfterViewChecked()
- ngOnDestroy() - Отписка от Observables, DOM events, остановка таймеров, Unregister all callbacks that the directive registered with global or application services.

## 9. Template variables. Как объявить. Зачем нужны?

- # или ref- в html, для доступа к свойствам элемента внутри html шаблона
- ссылается на: DOM element, directive (which contains a component), an element, ElementRef, or a web component.
- Область видимости - весь шаблон, потому что одинаковые переменные будут содержать непредсказуемые ссылки.

## 10. ElementRef, ElementRef, в чем разница?

- DOM/JS ?

## 11. В чем отличие ContentChild vs ViewChild.

- **ContentChild**
  - ng-content внешний HTML, transclusion, content projection
  - доступ после ngContentInit
  - прямой доступ в DOM нежелательно использовать при рендеринге на сервере из-за ИБ
  - можно ссылаться на директивы по имени и типу, на шаблонные переменные #
  - можно читать/писать в @Input компонента
- Property decorator that configures a view query. The change detector looks for the first element or the directive matching the selector in the view DOM. The following selectors are supported:
  - Any class with the @Component or @Directive decorator
  - A template reference variable as a string (e.g. query <my-component #cmp> with @ViewChild('cmp'))
  - Any provider defined in the child component tree of the current component (e.g. @ViewChild(SomeService) someService: SomeService)
  - Any provider defined through a string token (e.g. @ViewChild('someToken') someTokenVal: any)
  - A TemplateRef (e.g. query with @ViewChild(TemplateRef) template;)
- **ViewChild** для доступа к DOM после рендеринга afterViewInit
  - <https://angular.io/api/core/ViewChild>
  - <https://angular.io/guide/glossary#view-hierarchy>
  - <https://angular.io/guide/lifecycle-hooks#responding-to-view-changes>
  - доступ после ngAfterViewInit
  - прямой доступ в DOM нежелательно использовать при рендеринге на сервере из-за ИБ
  - можно ссылаться на директивы по имени и типу, на шаблонные переменные #
  - можно читать/писать в @Input компонента

```
// Accessing DOM element with JavaScript
let domReference = document.getElementById("someElement");

// Access DOM element using Angular @ViewChild
@ViewChild("someElement") domReference;
@ViewChild(NgModel) userNameReference: NgModel;
@ViewChild("userInformation") childComponentReference: any;

ngAfterViewInit(): void {
  this.domReference.nativeElement.focus();
  this.userNameReference.valueChanges.subscribe(() => {
this.executeOtherFunction() })
  // Accessing Property of Child Component
  this.childComponentReference.userName = "Updated Name";

  // Accessing Functions of Child Component
  this.childComponentReference.updateUserName();
}
```

- viewChildren - Директива @ViewChild отличается от @ViewChildren тем, что первая всегда вернет вам только один элемент, в то время как вторая позволяет вам находить несколько элементов, возвращая вам объект типа QueryList.

```
@ViewChildren("NgModel") domReference: QueryList<NgModel>;
@ViewChildren("UserDetailComponent") userDetailReferences:
QueryList<NgModel>;
@ViewChildren("userName, userAge, userDesignation")
userInfoReference: QueryList<NgModel>;

ngAfterViewInit(): void {
    console.log("Element List: " + this.domReference.length);
    console.log("Element List: " +
this.userDetailReferences.length);
}
```

## 12. динамическое создание компонентов

### 13. Как получить доступ к HTML Element из компонента.

- <https://angular.io/guide/component-interaction>
- шаблонные переменные в родителе дают доступ в HTML к свойствам компонента

```
<button (click)="timer.start()">Start</button>
<button (click)="timer.stop()">Stop</button>
<div class="seconds">{{timer.seconds}}</div>
<app-countdown-timer #timer></app-countdown-timer>
```

- @ViewChild даёт доступ к экземпляру другого компонента

```
@Component({
  selector: 'app-countdown-parent-vc',
  template: `
    <button (click)="start()">Start</button>
    <button (click)="stop()">Stop</button>
    <div class="seconds">{{ seconds() }}</div>
    <app-countdown-timer></app-countdown-timer>
  `,
  styleUrls: ['./assets/demo.css']
})
export class CountdownViewChildParentComponent implements
AfterViewInit {

  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;
```

```
seconds() { return 0; }

ngAfterViewInit() {
  // Redefine `seconds()` to get from the
  `CountdownTimerComponent.seconds` ...
  // but wait a tick first to avoid one-time devMode
  // unidirectional-data-flow-violation error
  setTimeout(() => this.seconds = () =>
this.timerComponent.seconds, 0);
}

start() { this.timerComponent.start(); }
stop() { this.timerComponent.stop(); }
}
```

- ElementRef даёт доступ к собственному DOM
- <https://angular.io/guide/testing-components-basics#nativeelement>
- <https://angular.io/guide/attribute-directives>
- <https://angular.io/guide/dependency-injection-in-action#inject-the-components-dom-element>

```
@Directive({selector: '[appHighlight]'})
export class HighlightDirective {
  constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'yellow';
  }
}
```

#### 14. ViewEncapsulation. Какая бывает, зачем нужна?

- Emulated - CSS обёртка для эмуляции стандартного поведения. если не объявлены templates/templateUrls переключается в None.
- None - для наследования общих стилей
- shadowDom - для прямого доступа к DOM [https://developer.mozilla.org/en-US/docs/Web/Web\\_Components/Using\\_shadow\\_DOM](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM)

#### 15. Change detection. Какие есть стратегии, для чего используются? Какие есть методы, чтобы запустить детектор?

- <https://angular.io/guide/zone#fundamentals-of-change-detection>
- <https://angular.io/guide/glossary#change-detection>
- <https://web.dev/faster-angular-change-detection/>
- <https://angular.io/api/core/ChangeDetectorRef#usage-notes>
- ангуляр манкипатчит browser API - mouse/input/keyb events, xhr, promise, async/await, webworkers,

- default ("CheckAlways") - the change detector goes through the view hierarchy on each VM turn to check every data-bound property in the template. In the first phase, it compares the current state of the dependent data with the previous state, and collects changes. In the second phase, it updates the page DOM to reflect any new data values.
- OnPush ("CheckOnce") - ручная проверка doCheck, поменялась @Input ссылка(не значение), DOM event(input) для связанных свойств, async pipe(rxjs, promise)

```
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

#### 16. Что такое zone.js, как он работает.

- <https://medium.com/@overthesanity/zone-js-от-а-до-я-fdb995917968>
- <https://angular.io/guide/zone#zones-and-execution-contexts>
- <https://github.com/angular/angular/blob/master/packages/zone.js/MODULE.md>
- <https://github.com/angular/angular/blob/master/packages/zone.js/README.md>
- [https://youtu.be/3IqtmUscE\\_U?t=116](https://youtu.be/3IqtmUscE_U?t=116)
- портирован из Dart
- контекст исполнения
- манкипатчинг и освобождение ресурсов
- профилирование отладка(трассировка) связывания HTML-JS
- заглушки для тестирования Jasmine/Mocha
- принудительный патчинг асинхронных API, не перехватываемых по-умолчанию

```
this.ngZone.run(() => {
  someNewAsyncAPI(() => {
    // update the data of the component
  });
});
```

#### 17. Как запустить код за пределами Angular.

- <https://angular.io/guide/zone#ngzone-run-and-runoutsideofangular>

```
export class AppComponent implements OnInit {
  constructor(private ngZone: NgZone) {}
  ngOnInit() {
    // You know no data will be updated,
    // so you don't want to trigger change detection in this
    // specified operation. Instead, call
    ngZone.runOutsideAngular()
    this.ngZone.runOutsideAngular(() => {
      setTimeout(() => {
        // update component data
        // but don't trigger change detection.
      });
    });
  }
}
```

```
});
});
}
}
```

- принудительное выключение перехватчиков по типу

```
// disable patching requestAnimationFrame
(window as any).__Zone_disable_requestAnimationFrame = true;

// disable patching specified eventNames
(window as any).__zone_symbol__UNPATCHED_EVENTS = ['scroll',
'mousemove'];
```

- принудительное выключение перехватчиков

```
// import 'zone.js/dist/zone'; // Included with Angular CLI.
platformBrowserDynamic().bootstrapModule(AppModule, { ngZone:
'noop' })
.catch(err => console.error(err));
```

## 18. pipe конвеер

- `<p>Message: {{ message$ | async }}</p>` - rxjs/promise без подписок
- `{{ amount | currency:'EUR':'Euros '}}`
- 

```
import { Pipe, PipeTransform } from '@angular/core';
/*
 * Raise the value exponentially
 * Takes an exponent argument that defaults to 1.
 * Usage:
 *   value | exponentialStrength:exponent
 * Example:
 *   {{ 2 | exponentialStrength:10 }}
 *   formats to: 1024
 */
@Pipe({
  name: 'exponentialStrength',
  pure: true // default
})
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent?: number): number {
    return Math.pow(value, isNaN(exponent) ? 1 : exponent);
  }
}
```

```
<p>Super power boost: {{2 | exponentialStrength: 10}}</p>
```

19. Сколько может быть router-outlet в компоненте.

- несколько именованных <https://angular.io/api/router/RouterOutlet#description>

20. Зачем нужны resolvers.

- <https://angular.io/guide/lazy-loading-ngmodules#preloading-component-data>
- <https://angular.io/guide/router-tutorial-toh#preloading-background-loading-of-feature-areas>
- <https://codeburst.io/understanding-resolvers-in-angular-736e9db71267>
- <https://medium.com/first-byte/resolvers-in-angular-simplified-85becdd6932b>
- блокируют загрузку страницы до окончания отрисовки компонента

```
// service
export class CrisisDetailResolverService implements Resolve<> {
  resolve(route: ActivatedRouteSnapshot, state:
RouterStateSnapshot): Observable<> {
    // your logic goes here
  }
}

// route
{
  path: '/your-path',
  component: YourComponent,
  resolve: {
    crisis: YourResolverService
  }
}

// component
ngOnInit() {
  this.route.data
    .subscribe((your-parameters) => {
      // your data-specific code goes here
    });
}
```

21. Зачем нужен OnInit если есть Constructor?

- для работы с начальными @Input значениями

22. Pure pipes / pipes

- By default, pipes are defined as pure so that Angular executes the pipe only when it detects a pure change to the input value. A pure change is either a change to a primitive input value (such as String, Number, Boolean, or Symbol), or a changed object reference (such as Date, Array, Function, or Object). A pure pipe must use a pure function, which is one that processes

inputs and returns values without side effects. In other words, given the same input, a pure function should always return the same output.

- Angular executes an impure pipe every time it detects a change with every keystroke or mouse movement.

```
@Pipe({
  name: 'flyingHeroesImpure',
  pure: false
})
```

## 23. ng-content

- точка сборки для вложенных компонентов
- transclusion или content projection

## Router

1. Как подгрузить отдельную библиотеку по требованию, не используя роутер?
2. Как загрузить по требованию какую-то часть приложения. Lazy-loading.
  - To lazy load Angular modules, use loadChildren (instead of component) in your AppRoutingModule routes configuration as follows.

```
const routes: Routes = [{
  path: 'items',
  loadChildren: () => import('./items/items.module').then(m =>
m.ItemsModule)
}];
```

3. Разница между root и forChild routes
  - forChild создаётся отдельный экземпляр для lazyLoad импортов, нет providers
4. Query params vs matrix params, в чем преимущества и недостатки.
5. Текущий маршрут

```
constructor(private route: ActivatedRoute){
  this.id = route.snapshot.paramMap.get('id')
  this.id$ = route.paramMap.map(params=>params.get('id'))
}
```

## Формы



### 1. NgForm, преимущество реактивных форм над template-driven.

- удобные валидаторы
- меньше кода в HTML

### 2. template vs reactive

- в шаблонных хуже покрытие линтерами
- state
  - value: pristine, dirty
  - validity: valid, errors: `errorName:any[]`
  - visited: touched, untouched
- для обоих одинаковые модели форм, но они по-разному создаются
- fromControl
  - `<input>`
  - 
  -
- formGroup
  - `<form>`
  - `.reset()`
  - `.setControl('formControlName', any[])`
- шаблонные
  - в шаблон: form, input, связывание со свойствами, правила валидации, ошибки валидации
  - в класс: свойства, методы
  - модель генерируется автоматом
  - доступ к formGroup возможен через шаблонные переменные
  - директивы: ngForm, ngModel, ngModelGroup
  - не подходят для:
    - отложить валидацию до конца ввода значений
    - анализа вводимого текста на лету
    - динамического добавления полей ввода
    - валидация по условию
    - иммутабельные данные
- реактивные
  - в шаблон: form, input, связывание с моделью
  - в класс: правила валидации, ошибки валидации, свойства, методы, модель
  - директивы: formGroup, formControl, formControlName, formGroupName, formArrayName
  - доступ:
    - `formGroup.controls.controlName`
    - `formGroup.get('controlName')`
  - запись:
    - `formGroup.setValues(valuesObj)` - полный список полей
    - `formGroup.patchValues(valuesObj)` - неполный список полей

### 3. валидаторы

- `formControl.clearValidators();`
- `formControl.updateValueAndValidity();`

- для валидации нескольких полей можно сделать валидатор для вложенной группы

```
myVal(c:AbstractControl): {[key:string]: boolean}|null{
    if (c.firstInput.dirty && c.secondInput.dirty &&
c.firstInput.value !== c.secondInput.value) { return
{'notSame':true}}
    return null;
}
this.formGroup = this.fb.group({
  g: this.fb.group({
    firstInput: '',
    secondInput: '',
  }, {validator: myVal})
})
```

- 
- 
- 4.
- 
- 
- 
- 
- 
- 
- 5.
- 
- 
- 
- 
- 
- 
- 6.
- 
- 
- 
- 
- 
- 

## Angular Material и SDK.

### HTTP/websocket

- angular in memory web api <https://angular.io/tutorial/toh-pt6#simulate-a-data-server>
- create - post - можно много записей за раз, не идемпотентный(разный результат)
- read - get
- update - put - подходит для конкретного экземпляра, идемпотентный

- delete - delete

## RxJS

- [public/kbo/kb/frontend/angular/rxjs.md](#)

## NGRX

- [public/kbo/kb/frontend/angular/ngrx.md](#)