FullStack.Cafe - Kill Your Tech Interview

Q1: Explain the difference between `Promise` and `Observable` in Angular? ☆☆☆

Topics: Angular

Answer:

Promises:

- return a single value
- not cancellable
- more readable code with try/catch and async/await

Observables:

- work with multiple values over time
- cancellable
- support map, filter, reduce and similar operators
- use Reactive Extensions (RxJS)
- an array whose items arrive asynchronously over time

Q2: What is the difference between `@Component` and `@Directive` in Angular? ☆☆☆

Topics: Angular

Answer:

A @Component requires a view whereas a @Directive does not.

Directives

I liken a @Directive to an Angular 1.0 directive with the option `restrict: 'A'` (Directives aren't limited to attribute usage.) Directives add behaviour to an existing DOM element or an existing component instance. One example use case for a directive would be to log a click on an element.

Components

A component, rather than adding/modifying behaviour, actually creates its own view (hierarchy of DOM elements) with attached behaviour. An example use case for this might be a contact card component:

Write a component when you want to create a reusable set of DOM elements of UI with custom behaviour. Write a directive when you want to write reusable behaviour to supplement existing DOM elements.

Q3: Why should `ngOnInit` be used, if we already have a `constructor`? ☆☆☆

Topics: Angular

Answer:

- The `Constructor` is a default method of the class that is executed when the class is instantiated and ensures proper initialization of fields in the class and its subclasses.

- `ngOnInit` is a life cycle hook called by Angular2 to indicate that Angular is done creating the component.

Mostly we use `ngOnInit` for all the initialization/declaration and avoid stuff to work in the constructor. The constructor should only be used to initialize class members but shouldn't do actual "work". So you should use `constructor()` to setup Dependency Injection and not much else. ngOnInit() is better place to "start" - it's where/when components' bindings are resolved.

Q4: What is difference between `declarations`, `providers` and `import` in *NgModule*? ☆☆☆

Topics: Angular

Answer:

- `imports` makes the exported declarations of other modules available in the current module
- `declarations` are to make directives (including components and pipes) from the current module available to other directives in the current module. Selectors of directives, components or pipes are only matched against the HTML if they are declared or imported.
- `providers` are to make services and values known to DI. They are added to the root scope and they are injected to other services or directives that have them as dependency.

A special case for `providers` are lazy loaded modules that get their own child injector. `providers` of a lazy loaded module are only provided to this lazy loaded module by default (not the whole application as it is with other modules).

Q5: What's new in Angular 6 and why shall we upgrade to it? ☆☆☆

Topics: Angular

Answer:

- **Angular Elements** - Angular Elements is a project that lets you wrap your Angular components as Web Components and embed them in a non-Angular application.
- **New Rendering Engine: Ivy** - increases in speed and decreases in application size.
- **Tree-shakeable providers** - a new, recommended, way to register a provider, directly inside the @Injectable() decorator, using the new `providedIn` attribute
- **RxJS 6** - Angular 6 now uses RxJS 6 internally, and requires you to update your application also. RxJS released a library called rxjs-compat, that allows you to bump RxJS to version 6.0 even if you, or one of the libraries you're using, is still using one of the "old" syntaxes.
- **ElementRef**`<T>` - in Angular 5.0 or older, is that the said ElementRef had its nativeElement property typed as any. In Angular 6.0, you can now type ElementRef more strictly.
- **Animations** - The polyfill web-animations-js is not necessary anymore for animations in Angular 6.0, except if you are using the AnimationBuilder.
- **i18n** - possibility to have "runtime i18n", without having to build the application once per locale.

Q6: What is *Protractor*? ☆☆☆

Topics: Angular

Answer:

Protractor is an end-to-end **test framework** for Angular and AngularJS applications. Protractor is a Node.js program built on top of WebDriverJS. Protractor runs tests against your application running in a real browser, interacting with it as a user would.

Q7: What is the use of *Codelyzer*? ☆☆☆

Topics: Angular

Answer:

All enterprise applications follows a set of coding conventions and guidelines to maintain code in better way. **Codelyzer** is an open source tool to run and check **whether the pre-defined coding guidelines has been followed or not**. Codelyzer does only static code analysis for angular and typescript project.

Codelyzer runs on top of tslint and its coding conventions are usually defined in tslint.json file. Codelyzer can be run via angular cli or npm directly. Editors like Visual Studio Code and Atom also supports codelyzer just by doing a basic settings.

Q8: How to inject `base href`? ☆☆☆

Topics: Angular

Answer:

When building you can modify base tag (`<base href="/">`) in your `index.html` with `--base-href your-url` option.

```
# Sets base tag href to /myUrl/ in your index.html
ng build --base-href /myUrl/
```

Q9: When would you use *eager module loading*? ☆☆☆

Topics: Angular

Answer:

Eager loading: To load a feature module *eagerly* we need to import it into application module using `imports` metadata of `@NgModule` decorator.

Eager loading is useful in small size applications. In eager loading, all the feature modules will be loaded before the application starts. Hence the subsequent request to the application will be faster.

Q10: What is the purpose of *Wildcard route*? ☆☆☆

Topics: Angular

Answer:

If the URL doesn't match any predefined routes then it causes the router to throw an error and crash the app. In this case, you can use **wildcard route**. A wildcard route has a path consisting of two asterisks to match every URL.

For example, you can define PageNotFoundComponent for wildcard route as below

```
{ path: '**', component: PageNotFoundComponent }
```

Q11: What are dynamic components? ☆☆☆

Topics: Angular

Answer:

Dynamic components are the components in which components location in the application is **not defined at build time** .i.e, they are not used in any angular template. But the component is instantiated and placed in the application at runtime.

Q12: Explain how *Custom Elements* works internally? ☆☆☆

Topics: Angular

Answer:

Below are the steps in an order about custom elements functionality:

1. **App registers custom element with browser:** Use the `createCustomElement()` function to convert a component into a class that can be registered with the browser as a custom element.
2. **App adds custom element to DOM:** Add custom element just like a built-in HTML element directly into the DOM.
3. **Browser instantiate component based class:** Browser creates an instance of the registered class and adds it to the DOM.
4. **Instance provides content with data binding and change detection:** The content with in template is rendered using the component and DOM data.

Q13: What are *Custom elements*? ☆☆☆

Topics: Angular

Answer:

Custom elements (or Web Components) are a Web Platform feature which extends HTML by allowing you to define a tag whose content is created and controlled by JavaScript code. The browser maintains a `CustomElementRegistry` of defined custom elements, which maps an instantiable JavaScript class to an HTML tag. Currently this feature is supported by Chrome, Firefox, Opera, and Safari, and available in other browsers through polyfills.

Q14: How do you perform *error handling* in `Observable`s? ☆☆☆

Topics: Angular

Answer:

You can handle errors by specifying an **error callback** on the observer instead of relying on try/catch which are ineffective in asynchronous environment. For example, you can define error callback as below:

```
myObservable.subscribe({
      next(num) { console.log('Next num: ' + num)},
      error(err) { console.log('Received an errror: ' + err)}
});
```

Q15: What is *Multicasting*? ☆☆☆

Topics: Angular

Answer:

Multi-casting is the practice of **broadcasting to a list of multiple subscribers in a single execution**. Let's demonstrate the multi-casting feature:

```
var source = Rx.Observable.from([1, 2, 3]);
var subject = new Rx.Subject();
var multicasted = source.multicast(subject);

// These are, under the hood, `subject.subscribe({...})`:
multicasted.subscribe({
    next: (v) => console.log('observerA: ' + v)
});
multicasted.subscribe({
    next: (v) => console.log('observerB: ' + v)
});
```

Q16: What is the difference between `promise` and `observable` ? ☆☆☆

Topics: Angular

Answer:

Below are the list of differences between `promise` and `observable` ,

Observable	Promise
Declarative: Computation does not start until subscription so that they can be run whenever you need the result	Execute immediately on creation
Provide multiple values over time	Provide only one
Subscribe method is used for error handling which makes centralized and predictable error handling	Push errors to the child promises
Provides chaining and subscription to handle complex applications	Uses only `.then()` clause

Q17: How do you perform *Error Handling* for `HttpClient` ? ☆☆☆

Topics: Angular

Answer:

If the request fails on the server or failed to reach the server due to network issues then `HttpClient` will return an error object instead of a successful reponse. In this case, you need to handle in the component by passing error object as a second callback to `subscribe()` method.

Let's see how it can be handled in the component with an example,

```
fetchUser() {
    this.userService.getProfile()
        .subscribe(
            (data: User) => this.userProfile = {
                ...data
            }, // success path
            error => this.error = error // error path
        );
}
```

It is always a good idea to give the user some meaningful feedback instead of displaying the raw error object returned from `HttpClient`.

Q18: How do you categorize data binding types? ☆☆☆

Topics: Angular

Answer:

Binding types can be grouped into three categories distinguished by the direction of data flow. They are listed as below:

1. From the **source-to-view**
2. From **view-to-source**
3. **View-to-source-to-view**

The possible binding syntax can be tabularized as below,

Data direction	Syntax	Type
From the source-to-view(One-way)	1. {{expression}} 2. [target]="expression" 3. bind-target="expression"	Interpolation, Property, Attribute, Class, Style
From view-to-source(One-way)	1. (target)="statement" 2. on-target="statement"	Event
View-to-source-to-view(Two-way)	1. [(target)]="expression" 2. bindon-target="expression"	Two-way

Q19: What is the option to choose between *Inline* and *External template* file? ☆☆☆

Topics: Angular

Answer:

You can store your component's template in one of two places.

- You can define it **inline** using the **template** property, or

  ```
  import {Component} from 'angular2/core';
  @Component({
    selector: 'my-app',
    template: `
    <h1>My First Angular 2 multiline template</h1>
    <p>Second line</p>
    `
  })
  export class AppComponent { }
  ```

- you can define the template in a **separate HTML** file and link to it in the component metadata using the `@Component` decorator's `templateUrl` property.

```
import {Component} from 'angular2/core';
@Component({
    selector: 'my-app',
    templateUrl: 'multi-line.html'
})
export class AppComponent { }
```

The choice between inline and separate HTML is a matter of taste, circumstances, and organization policy. But normally we use inline template for small portion of code and external template file for bigger views. By default, the Angular CLI generates components with a template file. But you can override that with the below command,

```
ng generate component hero -it
```

Q20: What is Angular *Ivy*? ☆☆☆

Topics: Angular

Answer:

A big part of Angular is its compiler: it takes all your HTML and generates the necessary JS code. This compiler (and the runtime) has been completely rewritten over the last year, and this is what Ivy is about. The last rewrite was done in Angular 4.0.

Ivy is a complete rewrite of the compiler (and runtime) in order to:

- reach better build times (with a more incremental compilation)
- reach better build sizes (with a generated code more compatible with tree-shaking)
- unlock new potential features (metaprogramming or higher order components, lazy loading of component instead of modules, a new change detection system not based on zone.js...)