Google Angular, API Doc

* All copy in this doc is for placeholder purpose. Exact copy is TBD.

1. Content Structure

NgSwitch

Directive | Stability: Stable

Angular 2 for TypeScript

WHAT IT DOES

The `NgSwitch` directive is used to conditionally swap DOM structure on your template based on a scope expression. Elements within `NgSwitch` but without `NgSwitchWhen` or `NgSwitchDefault` directives will be preserved at the location as specified in the template.

HOW TO USE

```
<!-- platform directive - no import needed -->
<div [ng-switch]="conditionExpression">
  <template [ng-switch-when]="case1Exp">...</template>
  <template ng-switch-when="case2LiteralString">...</template>
  <template ng-switch-default>...</template>
</div>
```

SELECTORS

`[ng-switch]`

OUTPUTS

`ng-model-change` bound to `NgFormControl.update`

INPUTS

`ng-switch` bound to `NgSwitch.ngSwitch`

EXPORTED AS

`form`

CLASS OVERVIEW

```
class NgSwitch {
    ngSwitch
}
```

CLASS DESCRIPTION

`NgSwitch` simply chooses nested elements and makes them visible based on which element matches the value obtained from the evaluated expression. In other words, you define a container element (where you place the directive), place an expression on the `[ng-switch]="..."` attribute), define any inner elements inside of the directive and place a `[ng-switch-when]` attribute per element. The when attribute is used to inform NgSwitch which element to display when the expression is evaluated. If a matching expression is not found via a when attribute then an element with the default attribute is displayed.

live demo

```
<ANY [ng-switch]="expression">
  <template [ng-switch-when]="whenExpression1">...</template>
  <template [ng-switch-when]="whenExpression1">...</template>
  <template ng-switch-default>...</template>
</ANY>
```

ng-model

We can also use `ng-mobel` to bind a domain model to the form.

live demo

```
@Component({
  selector: 'login-comp',
  directives: [FORM_DIRICTIVES],
  template: "<input type='text' [ng-form-control]='loginControl'
[(ng-model)]='login'>"
  })
class LoginComp {
  loginControl: Control = new Control('');
  login:string;
}
```

CLASS DETAILS

NgSwitch

```
constructor(_validators:
                /* Array<Validator|Function> */ any[],
_asyncValidators:
                /* Array<Validator|Function> */ any[],
valueAccessors: ControlValueAccessor[])
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

```
form : Control
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

```
model : any
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

```
viewModel : any
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

```
onChanges(changes: {[key: string]: SimpleChange}) : void
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

```
path : string[]
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Injector

Class Stability: Stable

Angular 2 for TypeScript

WHAT IT DOES

A dependency injection container used for instantiating objects and resolving dependencies. An `Injector` is a replacement for a `new` operator, which can automatically resolve the constructor dependencies. In typical use, application code asks for the dependencies in the constructor and they are resolved by the `Injector`.

HOW TO USE

```
import {Injector} from 'angular2/core';

class MyComponent {
  constructor(injector: Injector) {
    var someTypeInstance = injector.get(SomeType);
  }
}
```

CLASS OVERVIEW

```
class Injector {iders)
  static resolveAndCreate
  static resolve(providers)

  constructor()

  get(token)
  getOptional(token)
  resolveAndCreateChild(providers)
  createChildFromResolved(providers)
  resolveAndInstantiate(provider)
  instantiateResolved(provider)
  displayName
  toString()
}
```

CLASS DESCRIPTION

The following example creates an `Injector` configured to create `Engine` and `Car`.

live demo

```
@Injectable()
class Engine {
}

@Injectable()
class Car {
  constructor(public engine:Engine) {}
}

var injector = Injector.resolveAndCreate([Car, Engine]);
var car = injector.get(Car);
expect(car instanceof Car).toBe(true);
expect(car.engine instanceof Engine).toBe(true);
```

Notice, we don't use the `new` operator because we explicitly want to have the `Injector` resolve all of the object's dependencies automatically.

CLASS DETAILS

`constructor(_proto: any, _parent?: Injector, _depProvider?: any, _debugContext?: Function)`

Private

`get(token: any) : any`

Retrieves an instance from the injector based on the provided token. Throws `NoProviderError` if not found.

live demo

```
var injector = Injector.resolveAndCreate([
  provide("validToken", {useValue: "Value"})
]);
expect(injector.get("validToken")).toEqual("Value");
expect(() => injector.get("invalidToken")).toThrowError();
```

`Injector` returns itself when given `Injector` as a token.

```
var injector = Injector.resolveAndCreate([]);
expect(injector.get(Injector)).toBe(injector);
```

`getOptional(token: any) : any`

Retrieves an instance from the injector based on the provided token. Returns null if not found.

live demo

```
var injector = Injector.resolveAndCreate([
  provide("validToken", {useValue: "Value"})
]);
expect(injector.getOptional("validToken")).toEqual("Value");
expect(() => injector.getOptional("invalidToken")).toBe(null);
```

`Injector` returns itself when given `Injector` as a token.

```
var injector = Injector.resolveAndCreate([]);
expect(injector.getOptional(Injector)).toBe(injector);
```

`parent : Injector`

Parent of this injector.

live demo

```
var parent = Injector.resolveAndCreate([]);
var child = parent.resolveAndCreateChild([]);
expect(child.parent).toBe(parent);
```

`resolveAndCreateChild(providers: Array<Type | Provider | any[]>) : Injector`

Resolves an array of providers and creates a child injector from those providers.
The passed-in providers can be an array of `Type`, `Provider`, or a recursive array of more providers.

live demo

```
class ParentProvider {}
class ChildProvider {}

var parent = Injector.resolveAndCreate([ParentProvider]);
var child = parent.resolveAndCreateChild([ChildProvider]);

expect(child.get(ParentProvider) instanceof
ParentProvider).toBe(true);
expect(child.get(ChildProvider) instanceof ChildProvider).toBe(true);
expect(child.get(ParentProvider)).toBe(parent.get(ParentProvider));
```

This function is slower than the corresponding `createChildFromResolved` because it needs to resolve the passed-in providers first. See `Injector` and `Injector`.

`createChildFromResolved(providers: ResolvedProvider[]) : Injector`

Creates a child injector from previously resolved providers.
This API is the recommended way to construct injectors in performance-sensitive parts.

live demo

```
class ParentProvider {}
class ChildProvider {}

var parent = Injector.resolve([ParentProvider]);
var childProviders = Injector.resolve([ChildProvider]);

var parent = Injector.fromResolvedProviders(parentProviders);
var child = parent.createChildFromResolved(childProviders);

expect(child.get(ParentProvider) instanceof
ParentProvider).toBe(true);
expect(child.get(ChildProvider) instanceof ChildProvider).toBe(true);
expect(child.get(ParentProvider)).toBe(parent.get(ParentProvider));
```

`resolveAndInstantiate(provider: Type | Provider) : any`

Resolves a provider and instantiates an object in the context of the injector.
The created object does not get cached by the injector.

live demo

```
@Injectable()
class Engine {
}

@Injectable()
class Car {
  constructor(public engine:Engine) {}
}

var injector = Injector.resolveAndCreate([Engine]);

var car = injector.resolveAndInstantiate(Car);
expect(car.engine).toBe(injector.get(Engine));
expect(car).not.toBe(injector.resolveAndInstantiate(Car));
```

`instantiateResolved(provider: ResolvedProvider) : any`

Instantiates an object using a resolved provider in the context of the injector.
The created object does not get cached by the injector.

live demo

```
@Injectable()
class Engine {
}

@Injectable()
class Car {
  constructor(public engine:Engine) {}
}

var injector = Injector.resolveAndCreate([Engine]);
var carProvider = Injector.resolve([Car])[0];
var car = injector.instantiateResolved(carProvider);
expect(car.engine).toBe(injector.get(Engine));
expect(car).not.toBe(injector.instantiateResolved(carProvider));
```

`displayName : string`

`toString() : string`

2. Interaction

Floating menu—above the fold

NgSwitch Directive

Angular 2 for TypeScript

WHAT IT DOES

The `NgSwitch` directive is used to conditionally swap DOM structure on your template based on a scope expression. Elements within `NgSwitch` but without `NgSwitchWhen` or `NgSwitchDefault` directives will be preserved at the location as specified in the template.

HOW TO USE

```
<!-- platform directive - no import needed -->
<div [ng-switch]="conditionExpression">
  <template [ng-switch-when]="case1Exp">...</template>
  <template ng-switch-when="case2LiteralString">...</template>
  <template ng-switch-default>...</template>
</div>
```

SELECTORS `[ng-switch]`

Floating menu—below the fold


```
case1Exp">...</template>
  <template ng-switch-when="case2LiteralString">...</template>
  <template ng-switch-default>...</template>
</div>
```

SELECTORS `[ng-switch]`

OUTPUTS `ng-model-change` bound to `NgFormControl.update`

INPUTS `ng-switch` bound to `NgSwitch.ngSwitch`

EXPORTED AS `form`

CLASS OVERVIEW

```
class NgSwitch {
    ngSwitch
}
```

CLASS DESCRIPTION

`NgSwitch` simply chooses nested elements and makes them visible based on which element matches the value obtained from the evaluated expression. In other words, you define a container element (where you place the directive), place an expression on the `[ng-switch]="..."` attribute), define any inner elements inside of the directive and place

"Back to Top" fades in when the visible area is scrolled beneath the fold

Tooltip—hover over


```
case1Exp">...</template>
  <template ng-switch-when="case2LiteralString">...</template>
  <template ng-switch-default>...</template>
</div>
```

SELECTORS `[ng-switch]`

Tooltip text

OUTPUTS `ng-model-change` bound to `NgFormControl.update`

INPUTS `ng-switch` bound to `NgSwitch.ngSwitch`

EXPORTED AS `form`

CLASS OVERVIEW

```
class NgSwitch {
    ngSwitch
}
```

CLASS DESCRIPTION

`NgSwitch` simply chooses nested elements and makes them visible based on which element matches the value obtained from the evaluated expression. In other words, you define a container element (where you place the directive), place an expression on the `[ng-switch]="..."` attribute), define any inner elements inside of the directive and place

3. Spacing Guide

NgSwitch

Directive | Stability: Stable

Angular 2 for TypeScript

WHAT IT DOES

The `NgSwitch` directive is used to conditionally swap DOM structure on your template based on a scope expression. Elements within `NgSwitch` but without `NgSwitchWhen` or `NgSwitchDefault` directives will be preserved at the location as specified in the template.

HOW TO USE

```
<!-- platform directive - no import needed -->
<div [ng-switch]="conditionExpression">
  <template [ng-switch-when]="case1Exp">...</template>
  <template ng-switch-when="case2LiteralString">...</template>
  <template ng-switch-default>...</template>
</div>
```

SELECTORS

`[ng-switch]`

OUTPUTS

`ng-model-change` bound to `NgFormControl.update`

INPUTS

`ng-switch` bound to `NgSwitch.ngSwitch`

EXPORTED AS

`form`

CLASS OVERVIEW

```
class NgSwitch {
    ngSwitch
}
```

CLASS DESCRIPTION

`NgSwitch` simply chooses nested elements and makes them visible based on which element matches the value obtained from the evaluated expression. In other words, you define a container element (where you place the directive), place an expression on the `[ng-switch]="..."` attribute), define any inner elements inside of the directive and place a `[ng-switch-when]` attribute per element. The when attribute is used to inform NgSwitch which element to display when the expression is evaluated. If a matching expression is not found via a when attribute then an element with the default attribute is displayed.

live demo

```
<ANY [ng-switch]="expression">
  <template [ng-switch-when]="whenExpression1">...</template>
  <template [ng-switch-when]="whenExpression1">...</template>
  <template ng-switch-default>...</template>
</ANY>
```

ng-model

We can also use `ng-mobel` to bind a domain model to the form.

live demo

```
@Component({
  selector: 'login-comp',
  directives: [FORM_DIRICTIVES],
  template: "<input type='text' [ng-form-control]='loginControl'
[(ng-model)]='login'>"
  })
class LoginComp {
  loginControl: Control = new Control('');
  login:string;
}
```

CLASS DETAILS

NgSwitch

```
constructor(_validators:
                /* Array<Validator|Function> */ any[],
_asyncValidators:
                /* Array<Validator|Function> */ any[],
valueAccessors: ControlValueAccessor[])
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

`form : Control`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

`model : any`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

`viewModel : any`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

`onChanges(changes: {[key: string]: SimpleChange}) : void`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

`path : string[]`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

4. UI Components and Typography Guide

API Type Label

Directive

height:18px
padding-left: 8px
padding-right: 8px
background-color: #8eafda
border-radius: 4px;
font: Roboto, regular, 12pt, #014bac

Stability Label

8px
Stability: Stable

height: 18px
padding-left: 8px
padding-right: 8px
background-color: #8eafda
border-radius: 4px;
font: Roboto, regular, 12pt, #014bac

width: 12px
height: 12px
border-radius: 6px
background-color: #558b2f

Stability: Draft

width: 12px
height: 12px
border-radius: 6px
background-color: #FBC02D

Stability: Deprecated

width: 12px
height: 12px
border-radius: 6px
background-color: #E53935

Title

WHAT IT DOES — font: Roboto, regular, 16pt, #78909c

Tooltip Icon

8px
SELECTORS

width: 16px
height: 16px
color: #78909c
url download: https://www.google.com/design/icons/#ic_help

Tooltip Icon—hover state

SELECTORS

8px
Tooltip text

height: 22px
color: #78909c
opacity: 90%
border-radius: 2px
font: Roboto, Medium, 10px, #ffffff

Body Text

The `NgSwitch` directive is used to conditionally swap DOM structure on your template based on a scope expression. Elements within `NgSwitch` but without `NgSwitchWhen` or `NgSwitchDefault` directives will be preserved at the location as specified in the template.

font: Roboto, regular, 16pt, #5c7078
line-height: 28pt

Body Text—Link

NgSwitch

font: Roboto Mono, regular, 16pt, #2376ca
background-color: #eef1f2
border-radius: 2px
border-bottom: 0.5px, dotted, #2376ca

Body Text—Link, Hover

NgSwitch

border-bottom: 0.5px, solid, #2376ca

Body Text—Highlight without linking

[ng-switch] — background-color: #eef1f2, border-radius: 2px

Body Text—Code

```
<!-- platform directive - no import needed -->
<div [ng-switch]="conditionExpression">
  <template [ng-switch-when]="case1Exp">...</template>
  <template ng-switch-when="case2LiteralString">...</template>
  <template ng-switch-default>...</template>
</div>
```

Regular text

font: Roboto Mono, regular, 16pt, #5c7078
line-height: 28pt

Highlight 1

font: Roboto Mono, regular, 16pt, #9e9d24

Highlight 2

font: Roboto Mono, regular, 16pt, #d43669

Secondary punctuation — {}();

Code comments

font: Roboto Mono, regular, 16pt, #90a4ae

Sub-title

ng-model — font: Roboto, regular, 20pt, #5c7078

Code Snippet

background-color: #cfd8dc
height: 36px
The top-left and top-right round corner: 4px

font: Roboto, regular, 14pt, #2376ca
border-bottom: 0.5px, dotted, #2376ca
Hover state
border-bottom: 0.5px, solid, #2376ca

8px
live demo

width: 14px
height: 14px
color: #2376ca
url download: https://www.google.com/design/icons/#ic_exit_to_app

```
1.  @Component({
2.    selector: 'login-comp',
3.    directives: [FORM_DIRICTIVES],
4.    template: "<input type='text' [ng-form-control]='loginControl'
5.  [(ng-model)]='login'>"
6.    })
7.  class LoginComp {
8.    loginControl: Control = new Control('');
9.    login:string;
10. }
11.
```

background-color: #f5f6f7
padding: 16px
The bottom-left and bottom-right round corner: 4px

font: Roboto Mono, regular, 14pt, #90a4ae

Regular text

font: Roboto Mono, regular, 14pt, #5c7078
line-height: 26pt

Highlight 1

font: Roboto Mono, regular, 14pt, #9e9d24

Highlight 2

font: Roboto Mono, regular, 14pt, #d43669

Secondary punctuation — {}();

Code comments

font: Roboto Mono, regular, 14pt, #90a4ae

Horizontal rule

1px, solid, #eceff1

Floating menu

url download: https://www.google.com/design/icons/#ic_code
1px, solid, #cfd8dc
font: Roboto, regular, 14pt, #5c7078

8px
48px
View Cheat Sheet
8px 8px
View Source Code
Improve this Doc
48px
Back to Top
48px

width: 24px
height: 24px
color: #5c7078
url download:
https://www.google.com/design/icons/#ic_description

1px, solid, #cfd8dc
height: 24px

url download:
https://www.google.com/design/icons/#ic_create

url download:
https://www.google.com/design/icons/#ic_arrow_upward