Skip to main content

UpgradeModule NGMODULE

An `NgModule`, which you import to provide AngularJS core services, and has an instance method used to bootstrap the hybrid upgrade application.

See more...

```
class UpgradeModule {
  $injector: any
  injector: Injector
  ngZone: NgZone
bootstrap(element: Element, modules: string[] = [],
config?: any): any
}
```

Description

Part of the upgrade/static library for hybrid upgrade apps that support AOT compilation

The `upgrade/static` package contains helpers that allow AngularJS and Angular components to be used together inside a hybrid upgrade

pports AOT compilation.

Specifically, the classes and functions in the `upgrade/static` module allow the following:

1. Creation of an Angular directive that wraps and exposes an AngularJS component so that it can be used in an Angular template. See `UpgradeComponent`.
2. Creation of an AngularJS directive that wraps and exposes an Angular component so that it can be used in an AngularJS template. See `downgradeComponent`.
3. Creation of an Angular root injector provider that wraps and exposes an AngularJS service so that it can be injected into an Angular context. See `UpgradeModule#upgrading-an-angular-1-service` below.
4. Creation of an AngularJS service that wraps and exposes an Angular injectable so that it can be injected into an AngularJS context. See `downgradeInjectable`.
5. Bootstrapping of a hybrid Angular application which contains both of the frameworks coexisting in a single application.

Further information is available in the Usage Notes...

Skip to main content

Properties

Property	Description
`$injector: any`	The AngularJS `$injector` for the upgrade application.
`injector: Injector`	The Angular Injector *
`ngZone: NgZone`	The bootstrap zone for the upgrade application

Methods

Skip to main content

Bootstrap an AngularJS application from this NgModule

```
bootstrap(element: Element, modules: string[] = [],
config?: any): any
```

Parameters

element	Element	the element on which to bootstrap the AngularJS application
modules	string[]	the AngularJS modules to bootstrap for this application Optional. Default is `[]`.
config	any	optional extra AngularJS bootstrap configuration Optional. Default is `undefined`.

Returns

any : The value returned by angular.bootstrap().

Providers

Provider
angular1Providers

Skip to main content

Usage notes

```
import {UpgradeModule} from '@angular/upgrade/static';
```

See also the `UpgradeModule#examples` below.

Mental Model

When reasoning about how a hybrid application works it is useful to have a mental model which describes what is happening and explains what is happening at the lowest level.

Skip to main content

dependent frameworks running in a single
framework treats the other as a black box.

2. Each DOM element on the page is owned exactly by one framework. Whichever framework instantiated the element is the owner. Each framework only updates/interacts with its own DOM elements and ignores others.
3. AngularJS directives always execute inside the AngularJS framework codebase regardless of where they are instantiated.
4. Angular components always execute inside the Angular framework codebase regardless of where they are instantiated.
5. An AngularJS component can be "upgraded"" to an Angular component. This is achieved by defining an Angular directive, which bootstraps the AngularJS component at its location in the DOM. See `UpgradeComponent`.
6. An Angular component can be "downgraded" to an AngularJS component. This is achieved by defining an AngularJS directive, which bootstraps the Angular component at its location in the DOM. See `downgradeComponent`.
7. Whenever an "upgraded"/"downgraded" component is instantiated the host element is owned by the framework doing the instantiation. The other framework then instantiates and owns the view for that component.
    a. This implies that the component bindings will always follow the semantics of the instantiation framework.
    b. The DOM attributes are parsed by the framework that owns the current template. So attributes in AngularJS templates must use kebab-case, while AngularJS templates must use camelCase.
    c. However the template binding syntax will always use the Angular style, e.g. square brackets (`[...]`) for property binding.
8. Angular is bootstrapped first; AngularJS is bootstrapped second. AngularJS always owns the root component of the application.
9. The new application is running in an Angular zone, and therefore it no longer needs calls to `$apply()`.

`odule` class

This class is an `NgModule`, which you import to provide AngularJS core services, and has an instance method used to bootstrap the hybrid upgrade application.

- Core AngularJS services

  Importing this `NgModule` will add providers for the core AngularJS services to the root injector.

- Bootstrap

  The runtime instance of this class contains a `UpgradeModule#bootstrap` method, which you use to bootstrap the top level AngularJS module onto an element in the DOM for the hybrid upgrade app.

  It also contains properties to access the `UpgradeModule#injector`, the bootstrap `NgZone` and the AngularJS $injector.

Examples

Import the `UpgradeModule` into your top level Angular `NgModule`.

```
e represents the Angular pieces of the
application
@NgModule({
  declarations: [Ng2HeroesComponent,
Ng1HeroComponentWrapper],
  providers: [
    HeroesService,
    // Register an Angular provider whose value is the
"upgraded" AngularJS service
    {provide: TextFormatter, useFactory: (i: any) =>
i.get('textFormatter'), deps: ['$injector']},
  ],
  // We must import `UpgradeModule` to get access to
the AngularJS core services
  imports: [BrowserModule, UpgradeModule],
})
export class Ng2AppModule {
}
```

Then inject `UpgradeModule` into your Angular `NgModule` and use it to bootstrap the top level AngularJS module in the `ngDoBootstrap()` method.

```
export class Ng2AppModule {
  constructor(private upgrade: UpgradeModule) {}

  ngDoBootstrap() {
    // We bootstrap the AngularJS app.
    this.upgrade.bootstrap(document.body,
[ng1AppModule.name]);
  }
}
```

Finally, kick off the whole process, by bootstrapping your top level Angular `NgModule`.

```
the Angular module as we would do in a
app.
// (We are using the dynamic browser platform as this
example has not been compiled AOT.)
platformBrowserDynamic().bootstrapModule(Ng2AppModule);
```

Upgrading an AngularJS service

There is no specific API for upgrading an AngularJS service. Instead you should just follow the following recipe:

Let's say you have an AngularJS service:

```
export class TextFormatter {
  titleCase(value: string) {
    return value.replace(/((^|\s)[a-z])/g, (_, c) =>
c.toUpperCase());
  }
}

// This AngularJS service will be "upgraded" to be used
in Angular
ng1AppModule.service('textFormatter', [TextFormatter]);
```

Then you should define an Angular provider to be included in your `NgModule` `providers` property.

```
// Register an Angular provider whose value is the
"upgraded" AngularJS service
{provide: TextFormatter, useFactory: (i: any) =>
i.get('textFormatter'), deps: ['$injector']},
```

Then you can use the "upgraded" AngularJS service by injecting it into an Angular component or service.

```
constructor(textFormatter: TextFormatter) {
  // Change all the hero names to title case, using the
"upgraded" AngularJS service
  this.heroes.forEach((hero: Hero) => (hero.name =
textFormatter.titleCase(hero.name)));
}
```