Token, classe et injection

Temps de lecture : 3 minutes

Portée des dépendances

Angular propose plusieurs manières de fournir des dépendances selon leur portée et leur usage :

Au niveau racine avec `providedIn`

Utiliser `providedIn: 'root'` dans le décorateur `@Injectable` est la manière préférée pour fournir un service globalement.

Cela permet à Angular d'optimiser le code (*tree-shaking*) en supprimant les services inutilisés.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class GestionUtilisateurService {
  obtenirUtilisateur() {
    return { nom: 'Alice', role: 'Admin' };
  }
}
```

Ce service peut être injecté n'importe où dans l'application car il est fournit au niveau racine.

Au niveau d'un composant

Un service peut être limité à un composant en l'ajoutant au champ `providers` du décorateur `@Component` .

Chaque instance du composant obtient sa propre instance du service.

```
import { Component } from '@angular/core';
import { GestionUtilisateurService } from './gestion-utilisateu
r.service';

@Component({
```

```
  selector: 'app-profil-utilisateur',
  template: '<p>Utilisateur : {{ utilisateur.nom }}</p>',
  providers: [GestionUtilisateurService],
})
export class ProfilUtilisateurComponent {
  utilisateur = this.utilisateurService.obtenirUtilisateur();

  constructor(private utilisateurService: GestionUtilisateurServ
ice) {}
}
```

Chaque `ProfilUtilisateurComponent` aura une instance indépendante de `GestionUtilisateurService`.

Au niveau de la configuration globale avec `ApplicationConfig`

La configuration globale permet de déclarer des dépendances à l'échelle de l'application.

```
import { ApplicationConfig } from '@angular/core';
import { GestionUtilisateurService } from './gestion-utilisateu
r.service';

export const appConfig: ApplicationConfig = {
  providers: [GestionUtilisateurService],
};
```

Injection d'objets avec `InjectionToken`

Pour fournir des valeurs ou des objets non classes, Angular utilise des `InjectionToken`.

```
import { InjectionToken } from '@angular/core';

export const CONFIG_APP = new InjectionToken('Configuration de
l'application');

export const appConfig = {
  titre: 'Mon Application Angular',
};

export const appProviders = [
```

```
  { provide: CONFIG_APP, useValue: appConfig },
];
```

Cela permet d'injecter `appConfig` dans n'importe quel composant ou service.

Consommation des dépendances

Injection via constructeur

La méthode la plus ancienne consiste à déclarer les dépendances dans le constructeur. Nous vous la présentons car vous pouvez encore la rencontrer fréquemment.

Chaque dépendance est passée en argument au constructeur (il est obligatoire de la typer pour que cela fonctionne) et l'injecteur d'Angular pourra les détecter et les fournir :

```
import { Component } from '@angular/core';
import { GestionUtilisateurService } from './gestion-utilisateu
r.service';

@Component({
  selector: 'app-profil',
  template: '<p>{{ utilisateur.nom }}</p>',
})
export class ProfilComponent {
  utilisateur = this.utilisateurService.obtenirUtilisateur();

  constructor(private utilisateurService: GestionUtilisateurServ
ice) {}
}
```

Injection avec `inject`

Pour une injection hors constructeur, utilisez la fonction `inject`.

```
import { Component, inject } from '@angular/core';
import { GestionUtilisateurService } from './gestion-utilisateu
r.service';

@Component({
  selector: 'app-profil',
  template: '<p>{{ utilisateur.nom }}</p>',
})
export class ProfilComponent {
  utilisateur = inject(GestionUtilisateurService).obtenirUtilisa
```

```
teur();
}
```

Types de fournisseurs

Cette partie est vraiment avancée, vous pouvez passer rapidement et y revenir si vous avez des besoins spécifiques.

Fournisseur de classe (`useClass`)

Permet de substituer une implémentation par une autre.

```
import { Injectable } from '@angular/core';

@Injectable()
export class Logger {
  log(message: string) {
    console.log('Log : ', message);
  }
}

@Injectable()
export class LoggerAmeliore extends Logger {
  log(message: string) {
    super.log(`Message amélioré : ${message}`);
  }
}

export const appProviders = [
  { provide: Logger, useClass: LoggerAmeliore },
];
```

Fournisseur d'alias (`useExisting`)

Crée un alias pour une dépendance existante.

```
export const appProviders = [
  Logger,
  { provide: 'AliasLogger', useExisting: Logger },
];
```

Création dynamique (`useFactory`)

Permet de créer dynamiquement une valeur en utilisant une fonction.

```
import { Injectable } from '@angular/core';

@Injectable()
export class AuthService {
  estConnecte() {
    return true;
  }
}

export const appProviders = [
  {
    provide: 'Autorisation',
    useFactory: (auth: AuthService) => auth.estConnecte(),
    deps: [AuthService],
  },
];
```

Fournisseur de valeur (`useValue`)

Permet de fournir une valeur statique comme dépendance.

```
export const appProviders = [
  { provide: 'ModeDebug', useValue: true },
];
```

Exemple de la vidéo

`src/app/shared/dummy.ts`

```
import { Injectable, InjectionToken } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class MyClass {
  test = 123;
}

export const MY_CONFIG_TOKEN = new InjectionToken('je suis un to
ken valide');
```

```ts
export const config = {
  title: 'Angular',
};
```

src/app/components/dyma.component.ts

```ts
import { Component, inject } from '@angular/core';
import { MY_CONFIG_TOKEN, MyClass } from '../shared/dummy';

@Component({
  selector: 'app-dyma',
  template: ``,
})
export class DymaComponent {
  myclass = inject(MyClass);
  myconfig = inject(MY_CONFIG_TOKEN);

  constructor() {
    console.log(this.myclass);
    console.log(this.myconfig);
  }
}
```

src/app/app.config.ts

```ts
import {
  ApplicationConfig,
  inject,
  provideExperimentalZonelessChangeDetection,
} from '@angular/core';
import { config, MY_CONFIG_TOKEN, MyClass } from './shared/dumm
y';

export const appConfig: ApplicationConfig = {
  providers: [
    provideExperimentalZonelessChangeDetection(),
    {
      provide: MY_CONFIG_TOKEN,
      useFactory: () => {
        const myclass = inject(MyClass);
        console.log('deps : ', myclass);
        return config;
      },
```

```
      deps: [MyClass],
    },
  ],
};
```