https://github.com/angularjs/angularJS/blob/master/src/ngCore/injector.js#L1-L1 https://github.com/angular/angular/blob/master/packages/core/src/di/injector.mod.ts#L1-L1

I want to every concept/element/pattern/feature/function/etc. in Angular's DI system explained in detail. Also, I want to know how it compares to other types of DI/IoC/SoC/SoD/etc. Also, I want to know what role does it have within the overall architecture of Angular's core/internals/etc. Also, I want to know what role does it have in the overall architecture of the function export function createInjector(modulesToLoad, strictDi = false) { var INSTANTIATING = {}, providerSuffix = 'Provider', path = [], loadedModules = new HashMap([[{}, true]]), providerCache = {$provide: { provider: supportObject(provider), factory: supportObject(factory), value: supportObject(value), constant: supportObject(constant), decorator: decorator }},

Does any other JS library/framework has similar functionality? Backbone/Ember/Knockout/Vue.js/React/Svelte/Preact/Inferno/Elm/Dojo/Dojo 2/Polymer/Aurelia/Stencil/Angular 1/Angular 2/Angular 4/Angular 5/Angular 6/Angular 7/Angular 8/Angular 9/Angular 10/Angular 11/Angular 12/Angular 13/Angular 14/Angular 15/Angular 16/Angular 17/Angular 18/Angular 19/Angular 20/Angular 21/Angular 22/Angular 23/Angular 24/Angular 25/Angular 26/Angular 27/Angular 28/Angular 29/Angular 30/Angular 31/Angular 32/Angular 33/Angular 34/Angular 35/Angular 36/Angular 37/Angular 38/Angular 39/Angular 40/Angular 41/Angular 42/Angular 43/Angular

- Normally, `Left` works only with class components or functions with `Component` or `Constructor`.

With `ExecuteLeft`:

- The parent's `Left` bypasses the child component and attaches directly to an internal DOM node inside the child.

2.3 Free Parent Access Child's DOM Nodes

Example

```
// ...
const ThirdButton = React.forwardRef(({ onClick, left, ref }
  ...
  ...
  ...
```

Angular defers the actual **binding** (association of tokens to resolved instances) until the application is running.

How Angular Implements Deferred Binding

- Angular doesn't resolve all dependencies at once. Instead, it resolves them lazily:
  - When a component, service, or directive requests a dependency, Angular resolves it only if it hasn't been instantiated already.
  - If a factory provider is used, Angular invokes the factory at runtime to create the instance.

Where Binding is Declared

Binding is declared in the following places:

1. **Constructor Parameters**:
   - When a dependency is declared in a constructor:

     typescript Copy code
     ```typescript
     constructor(private service: MyService) {}
     ```

     Angular resolves the `MyService` token at runtime when creating the class instance.
2. `inject()` **Function**:
   - If you use `inject()` directly:

     typescript Copy code
     ```typescript
     const myService = inject(MyService);
     ```

     Angular defers the resolution until `inject()` is called.
3. **Decorators**:
   - Angular decorators like `@Optional()`, `@Self()`, or `@SkipSelf()` influence the resolution process:

     typescript Copy code
     ```typescript
     constructor(@Optional() @SkipSelf() private service?: MyService) {}
     ```

Source Code Evidence

The binding process happens in Angular's DI system within the `resolveToken` and `createInstance` methods:

typescript Copy code
```typescript
function createInstance(record: Record, injector: Injector): any {
  const factory = record.factory;
  const instance = factory(injector); // Factory function called here
  record.instance = instance;        // Cached for future use
  return instance;
}
```

- **Deferred Execution**:
  - The factory is invoked **only when the token is resolved**.
  - This aligns with the deferred binding concept.

Summary

Angular DI as a Service Locator:

- Angular DI shares many traits of the **Service Locator Pattern**, particularly dynamic runtime resolution of tokens.
- However, its declarative and tree-shakable nature sets it apart.

Deferred Binding:

- In Angular, binding is deferred until runtime and is driven by explicit declarations in constructors or the `inject()` function.
- Dependencies are resolved only when requested, ensuring efficient resource usage.

This technique is a pragmatic blend of Service Locator functionality and declarative dependency injection, tailored for Angular's framework-level needs.

Previous 30 Days

Upgrade plan
More access to the best models

Message ChatGPT

ChatGPT can make mistakes. Check important info.