Inject

August 17, 2026

Inject

Kind: Function

Source: `packages/common/decorators/core/inject.decorator.ts`

Part of: Common

Decorator that marks a constructor parameter as a target for Dependency Injection (DI).

Any injected provider must be visible within the module scope (loosely speaking, the containing module) of the class it is being injected into. This can be done by:

- defining the provider in the same module scope
- exporting the provider from one module scope and importing that module into the module scope of the class being injected into
- exporting the provider from a module that is marked as global using the `@Global()` decorator

Injection tokens

Can be *types* (class names), *strings* or *symbols*. This depends on how the provider with which it is associated was defined. Providers defined with the `@Injectable()` decorator use the class name. Custom Providers may use strings or symbols as the injection token.

`@Inject()` marks a constructor parameter as a dependency to be resolved by Nest's dependency injection container. Use it when injecting providers by an explicit token, especially for custom providers registered with string or symbol tokens. The referenced provider must be visible in the module scope where the consuming class is declared.

Signature

```
function Inject(token: InjectionToken | ForwardReference): PropertyDecorator & ParameterDecorat
```

Parameters

Name	Type
`token`	`InjectionToken

Returns: `PropertyDecorator & ParameterDecorator`

Diagram

```
graph LR
  A[Module Provider Registration] -->|class, string, or symbol token| B[Nest DI Container]
  C[Consumer Constructor<br/>@Inject(TOKEN)] --> B
  B -->|resolves matching provider| D[Injected Dependency]
```

Usage

```
import { Injectable, Inject } from '@nestjs/common';

export const CACHE_CLIENT = Symbol('CACHE_CLIENT');

interface CacheClient {
  get(key: string): Promise<string | null>;
}

@Injectable()
export class UserService {
  constructor(
    @Inject(CACHE_CLIENT)
    private readonly cache: CacheClient,
  ) {}

  async findCachedUser(id: string) {
    return this.cache.get(`user:${id}`);
  }
}
```

```
import { Module } from '@nestjs/common';

@Module({
  providers: [
    UserService,
```

```
    {
      provide: CACHE_CLIENT,
      useValue: {
        get: async (key: string) => null,
      },
    },
  ],
})
export class UsersModule {}
```

AI Coding Instructions

- Use `@Inject(TOKEN)` when the provider uses a string or symbol token; class-based `@Injectable()` providers can usually be injected by type.
- Ensure the provider is declared in the same module, or exported from an imported module.
- Keep injection tokens in shared constants to avoid mismatched string literals or duplicate symbols.
- Prefer interfaces paired with explicit tokens for dependencies that have multiple implementations.
- Do not inject providers that are unavailable in the consuming module's scope; global modules should be used sparingly.

Used by

64 references from 64 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (64)

- `ROUTES` — `packages/core/router/router-module.ts`:9
- `CircularService` — `integration/injector/src/circular/circular.service.ts`:4
- `InputService` — `integration/injector/src/circular/input.service.ts`:4
- `CircularService` — `integration/injector/src/circular-modules/circular.service.ts`:4
- `InputService` — `integration/injector/src/circular-modules/input.service.ts`:4
- `CircularService` — `integration/injector/src/circular-properties/circular.service.ts`:4
- `InputService` — `integration/injector/src/circular-properties/input.service.ts`:4
- `DefaultsService` — `integration/injector/src/defaults/defaults.service.ts`:4

…and 56 more.