ModuleRef

August 18, 2026

ModuleRef

Kind: Class

Source: `packages/core/injector/module-ref.ts`

Part of: Core

`ModuleRef` provides programmatic access to providers registered in the current NestJS module. Use it to retrieve existing singleton providers with `get()`, resolve request- or transient-scoped providers with `resolve()`, or instantiate classes dynamically with dependency injection through `create()`.

Extends: `AbstractInstanceResolver`

Methods

Method	Signature	Returns
`get`	`get(typeOrToken: Type	Function
`get`	`get(typeOrToken: Type	Function
`get`	`get(typeOrToken: Type	Function
`get`	`get(typeOrToken: Type	Function
`resolve`	`resolve(typeOrToken: Type	Function
`resolve`	`resolve(typeOrToken: Type	Function
`resolve`	`resolve(typeOrToken: Type	Function
`resolve`	`resolve(typeOrToken: Type	Function
`resolve`	`resolve(typeOrToken: Type	Function
`create`	`create(type: Type<T>, contextId: ContextId)`	`Promise<T>`
`introspect`	`introspect(token: Type	string

Method	Signature	Returns
`registerRequestByContextId`	`registerRequestByContextId(request: T, contextId: ContextId)`	`void`
`instantiateClass`	`instantiateClass(type: Type<T>, moduleRef: Module, contextId: ContextId)`	`Promise<T>`

Properties

Property	Type
`injector`	`Injector`

When something fails

- `ModuleRef` handles failure in 1 place: it logs it and continues in all 1.

Diagram

```
graph LR
  Consumer[Service / Controller] --> ModuleRef
  ModuleRef -->|get()| Singleton[Registered singleton provider]
  ModuleRef -->|resolve()| Scoped[Request or transient provider instance]
  ModuleRef -->|create()| Dynamic[New dynamically created class]
  ModuleRef --> Container[Nest dependency injection container]
```

Usage

```
import { Injectable } from '@nestjs/common';
import { ModuleRef } from '@nestjs/core';

@Injectable()
export class ReportsService {
  constructor(private readonly moduleRef: ModuleRef) {}

  async generateReport() {
    // Retrieve an existing singleton provider.
    const configService = this.moduleRef.get(ConfigService, {
      strict: false,
    });

    // Create or retrieve a scoped provider instance.
    const reportContext = await this.moduleRef.resolve(ReportContextService);
```

```
    // Instantiate a class while allowing Nest to inject its dependencies.
    const exporter = await this.moduleRef.create(CsvReportExporter);

    return exporter.export({
      timezone: configService.get('TIMEZONE'),
      context: reportContext,
    });
  }
}
```

AI Coding Instructions

- Use `get()` for providers that are already registered and safe to access as singleton instances.
- Use `resolve()` for request-scoped or transient providers; it may create a new instance for the current resolution context.
- Pass `{ strict: false }` to `get()` only when intentionally searching providers outside the current module boundary.
- Use `create()` when dynamically constructing a class that is not necessarily registered as a provider but still requires dependency injection.
- Prefer constructor injection for known dependencies; use `ModuleRef` only when runtime or lazy resolution is required.

How it works

`ModuleRef` is an abstract DI-container reference type. Each `Module` registers a resolved instance under the `ModuleRef` token; that instance is created from a module-specific subclass returned by `createModuleReferenceType()` and is hosted by that module. [packages/core/injector/module.ts:161-179] The subclass closes over its host `Module`, so its lookup and resolution operations can apply that module's ID and context. [packages/core/injector/module.ts:602-641]

The abstract class stores the `NestContainer`, creates an `Injector` from the container's preview, snapshot, and instance-decorator context options, and lazily builds an `InstanceLinksHost` from the container on first use. [packages/core/injector/module-ref.ts:26-47] `InstanceLinksHost` snapshots links for every module's providers, injectables, and controllers when constructed. [packages/core/injector/instance-links-host.ts:17-22][packages/core/injector/instance-links-host.ts:55-68]