AbstractInstanceResolver

August 18, 2026

AbstractInstanceResolver

Kind: Class

Source: `packages/core/injector/abstract-instance-resolver.ts`

Part of: Core

`AbstractInstanceResolver` is the internal base class responsible for locating and resolving provider instances from Nest's dependency injection container. It supports synchronous lookup through `get()` and `find()`, plus context-aware asynchronous resolution through `resolvePerContext()` for request-scoped or transient providers. It is extended by container-facing APIs such as `ModuleRef` rather than instantiated directly.

Methods

Method	Signature	Returns
`get`	`get(typeOrToken: Type	Function
`find`	`find(typeOrToken: Type	Abstract
`resolvePerContext`	`resolvePerContext(typeOrToken: Type	Abstract

Properties

Property	Type
`instanceLinksHost`	`InstanceLinksHost`
`injector`	`Injector`

Where it refuses work

- `AbstractInstanceResolver` stops the work with `InvalidClassScopeException` when `wrapperRef.scope === Scope.REQUEST || wrapperRef.scope === Scope.TRANSIENT ||`

`!wrapperRef···`.

- `AbstractInstanceResolver` stops the work with `UnknownElementException` when `!instance`.
- `AbstractInstanceResolver` stops the work with an early return when `Array.isArray(instanceLinkOrArray)`, in 2 places.
- `AbstractInstanceResolver` stops the work with an early return when `wrapperRef.isDependencyTreeStatic() && !wrapperRef.isTransient`.

Diagram

```
graph LR
  A[Application Code] --> B[ModuleRef]
  B --> C[AbstractInstanceResolver]
  C --> D[InstanceLinksHost]
  D --> E[Provider Wrapper]
  E --> F[Singleton Instance]
  E --> G[Request-scoped Instance]
  E --> H[Transient Instance]

  C -->|get / find| F
  C -->|resolvePerContext| G
  C -->|resolvePerContext| H
```

Usage

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

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

  getSharedLogger() {
    // Uses the resolver's synchronous lookup flow for singleton providers.
    return this.moduleRef.get(AppLogger, { strict: false });
  }

  async createRequestHandler(request: Request) {
    const contextId = ContextIdFactory.getByRequest(request);

    // Uses context-aware resolution for request-scoped providers.
    return this.moduleRef.resolve(RequestHandler, contextId, {
      strict: false,
    });
```

```
  }
}
```

AI Coding Instructions

- Do not instantiate `AbstractInstanceResolver` directly; use or extend container APIs such as `ModuleRef`.
- Use synchronous `get()`-style resolution only for providers that are already available in the current static context, typically singleton providers.
- Use context-aware resolution for request-scoped or transient providers so each request receives the correct instance.
- Pass `{ strict: false }` only when a provider may be declared in another module; prefer strict lookup when module boundaries should be enforced.
- Avoid resolving dependencies manually when constructor injection is sufficient; use runtime resolution primarily for dynamic or context-specific dependencies.

Relationships

- IMPORTS → `Abstract`
- IMPORTS → `Scope`
- IMPORTS → `Type`
- IMPORTS → `GetOrResolveOptions`