InstanceLinksHost

August 19, 2026

InstanceLinksHost

Kind: Class

Source: `packages/core/injector/instance-links-host.ts`

Part of: Core

`InstanceLinksHost` is Nest's internal registry for locating provider instance links across the application's modules. It indexes providers by injection token and returns the corresponding `InstanceLink`, which contains the resolved instance, wrapper metadata, and owning module context.

Methods

Method	Signature	Returns
`get`	`get(token: InjectionToken)`	`InstanceLink<T>`
`get`	`get(token: InjectionToken, options: { moduleId?: string; each?: boolean })`	`InstanceLink
`get`	`get(token: InjectionToken, options: { moduleId?: string; each?: boolean })`	`InstanceLink

Where it refuses work

- `InstanceLinksHost` stops the work with `UnknownElementException` when `!instanceLinksForGivenToken`.
- `InstanceLinksHost` stops the work with `UnknownElementException` when `!instanceLink`.
- `InstanceLinksHost` stops the work with an early return when `options.each`.

Diagram

```
graph LR
  Container[NestContainer] --> Modules[Registered Modules]
  Modules --> Providers[Provider Wrappers]
  Providers --> Links[InstanceLink entries]
  Links --> Host[InstanceLinksHost]
  Token[Injection Token] --> Host
  Host --> Result[InstanceLink / InstanceLink[]]
```

Usage

```
import { InstanceLinksHost } from '@nestjs/core/injector/instance-links-host';
import { NestContainer } from '@nestjs/core/injector/container';
import { CatsService } from './cats.service';

// Nest normally creates and manages these internal objects.
const container = new NestContainer();
const instanceLinksHost = new InstanceLinksHost(container);

// Look up the provider link registered for a token.
const link = instanceLinksHost.get<CatsService>(CatsService);

// The link exposes the resolved provider instance and wrapper metadata.
const catsService = link.instance as CatsService;
catsService.findAll();
```

AI Coding Instructions

- Treat `InstanceLinksHost` as internal Nest injector infrastructure; prefer public APIs such as `ModuleRef` or `app.get()` in application code.
- Use the same injection token used during provider registration when calling `get()`, including class constructors, strings, symbols, or custom tokens.
- Handle missing tokens carefully: lookups for unregistered providers can throw an unknown-element error.
- Remember that a token may exist in multiple modules; use the appropriate lookup mode when integration code needs every matching `InstanceLink`.
- Do not mutate returned link metadata or provider wrappers, because they are shared by Nest's dependency-injection container.