DiscoveryService

August 18, 2026

DiscoveryService

Kind: Class

Source: `packages/core/discovery/discovery-service.ts`

Part of: Core

`DiscoveryService` provides runtime access to registered NestJS modules, providers, and controllers. It also creates typed discoverable decorators and retrieves metadata associated with those decorators, enabling convention-based discovery patterns such as plugin registries or handler scanning.

Methods

Method	Signature	Returns
`createDecorator`	`createDecorator()`	`DiscoverableDecorator<T>`
`getProviders`	`getProviders(options: DiscoveryOptions, modules: Module[])`	`InstanceWrapper[]`
`getControllers`	`getControllers(options: DiscoveryOptions, modules: Module[])`	`InstanceWrapper[]`
`getMetadataByDecorator`	`getMetadataByDecorator(decorator: T, instanceWrapper: InstanceWrapper, methodKey: string)`	`T extends DiscoverableDecorator ? R`
`getModules`	`getModules(options: DiscoveryOptions)`	`Module[]`

Where it refuses work

- `DiscoveryService` stops the work with an early return when `methodKey`.

Diagram

```
graph LR
  A[DiscoveryService] --> B[ModulesContainer]
  B --> C[Module]
  C --> D[Providers<br/>InstanceWrapper]
  C --> E[Controllers<br/>InstanceWrapper]
  A --> F[createDecorator]
  F --> G[DiscoverableDecorator]
  G --> H[Metadata on classes]
  A --> I[getMetadataByDecorator]
  I --> H
```

Usage

```
import { Injectable, OnModuleInit } from '@nestjs/common';
import {
  DiscoveryService,
  InstanceWrapper,
} from '@nestjs/core';

// Create a reusable decorator for classes that implement application jobs.
export const JobHandler = DiscoveryService.createDecorator<{
  queue: string;
}>();

@JobHandler({ queue: 'emails' })
@Injectable()
export class SendEmailJob {
  async execute() {
    // Send email work
  }
}

@Injectable()
export class JobRegistry implements OnModuleInit {
  constructor(private readonly discoveryService: DiscoveryService) {}

  onModuleInit() {
    const providers = this.discoveryService.getProviders();

    const jobHandlers = providers
      .map((provider: InstanceWrapper) => ({
        provider,
        metadata: this.discoveryService.getMetadataByDecorator(
          JobHandler,
          provider,
        ),
      }))
```

```
      .filter(({ metadata }) => metadata !== undefined);

    for (const { provider, metadata } of jobHandlers) {
      console.log(
        `Registered ${provider.name} for the "${metadata.queue}" queue.`,
      );
    }
  }
}
```

AI Coding Instructions

- Use `createDecorator<T>()` to define typed metadata contracts for discoverable providers or controllers.
- Call discovery methods after module initialization (for example, in `onModuleInit`) so the NestJS container has been populated.
- Always handle `undefined` from `getMetadataByDecorator()`; not every discovered wrapper has the requested decorator.
- Inspect `InstanceWrapper.metatype` or `InstanceWrapper.instance` carefully, since wrappers can represent aliases, factories, or unresolved providers.
- Use `getProviders()`, `getControllers()`, and `getModules()` for framework integration and registry-building logic rather than maintaining duplicate manual registries.

Used by

3 references from 3 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (3)

- `NonAppliedDecorator` — `integration/discovery/src/decorators/non-applied.decorator.ts`:9
- `Webhook` — `integration/discovery/src/decorators/webhook.decorators.ts`:3
- `WebhooksExplorer` — `integration/discovery/src/webhooks.explorer.ts`:5