DependenciesScanner

August 18, 2026

DependenciesScanner

Kind: Class

Source: `packages/core/scanner.ts`

Part of: Core

`DependenciesScanner` discovers an application's module tree and reflects Nest metadata for imports, providers, controllers, exports, and dynamic module configuration. It populates the `NestContainer` during application initialization so the dependency injection system can resolve modules and their registered components.

Methods

Method	Signature	Returns
`scan`	`scan(module: Type<any>, options: { overrides?: ModuleOverride[] })`	`void`
`scanForModules`	`scanForModules({	

```
moduleDefinition,
lazy,
scope = [],
ctxRegistry = [],
overrides = [],
```

}: ModulesScanParameters) | Promise<Module[]> | | insertModule | insertModule(moduleDefinition: any, scope: Type[]) | Promise< | { moduleRef: Module; inserted: boolean; } | undefined > | | scanModulesForDependencies | scanModulesForDependencies(modules: Map<string, Module>) | void | | reflectImports | reflectImports(module: Type, token: string, context: string) | void | | reflectProviders | reflectProviders(module: Type, token: string) | void | | reflectControllers | reflectControllers(module: Type, token:

string) | void | | reflectDynamicMetadata | reflectDynamicMetadata(cls: Type, token: string) | void | | reflectExports | reflectExports(module: Type, token: string) | void | | reflectInjectables | reflectInjectables(component: Type, token: string, metadataKey: string) | void | | reflectParamInjectables | reflectParamInjectables(component: Type, token: string, metadataKey: string) | void | | reflectKeyMetadata | reflectKeyMetadata(component: Type, key: string, methodKey: string) | { methodKey: string; metadata: any } | undefined | | calculateModulesDistance | calculateModulesDistance() | void | | insertImport | insertImport(related: any, token: string, context: string) | void | | isCustomProvider | isCustomProvider(provider: Provider) | provider is | ClassProvider | ValueProvider | FactoryProvider | ExistingProvider | | insertProvider | insertProvider(provider: Provider, token: string) | void | | insertInjectable | insertInjectable(injectable: Type | object, token: string, host: Type, subtype: EnhancerSubtype, methodKey: string) | void | | insertExportedProviderOrModule | insertExportedProviderOrModule(toExport: ForwardReference | DynamicModule | Type, token: string) | void | | insertController | insertController(controller: Type, token: string) | void | | reflectMetadata | reflectMetadata(metadataKey: string, metatype: Type) | T[] | | registerCoreModule | registerCoreModule(overrides: ModuleOverride[]) | void | | addScopedEnhancersMetadata | addScopedEnhancersMetadata() | void | | applyApplicationProviders | applyApplicationProviders() | void | | getApplyProvidersMap | getApplyProvidersMap() | { [type: string]: Function } | | getApplyRequestProvidersMap | getApplyRequestProvidersMap() | { [type: string]: Function } | | isDynamicModule | isDynamicModule(module: Type | DynamicModule) | module is DynamicModule` |

Where it refuses work

- `DependenciesScanner` stops the work with `UndefinedModuleException` when `innerModule === undefined`.
- `DependenciesScanner` stops the work with `InvalidModuleException` when `!innerModule`.
- `DependenciesScanner` stops the work with `InvalidClassModuleException` when `this.isInjectable(moduleToAdd)`.
- `DependenciesScanner` stops the work with `InvalidClassModuleException` when `this.isController(moduleToAdd)`.
- `DependenciesScanner` stops the work with `InvalidClassModuleException` when `this.isExceptionFilter(moduleToAdd)`.

- `DependenciesScanner` stops the work with `CircularDependencyException` when `isUndefined(related)`.

Diagram

```
graph LR
  A[Root Module] --> B[DependenciesScanner.scan]
  B --> C[scanForModules]
  C --> D[insertModule]
  D --> E[NestContainer]
  B --> F[scanModulesForDependencies]
  F --> G[reflectImports]
  F --> H[reflectProviders]
  F --> I[reflectControllers]
  F --> J[reflectDynamicMetadata]
  F --> K[reflectExports]
  F --> L[reflectInjectables]
  G --> E
  H --> E
  I --> E
  J --> E
  K --> E
  L --> E
```

Usage

```
import { DependenciesScanner } from '@nestjs/core/scanner';
import { NestContainer } from '@nestjs/core/injector/container';
import { MetadataScanner } from '@nestjs/core/metadata-scanner';
import { GraphInspector } from '@nestjs/core/inspector/graph-inspector';
import { ApplicationConfig } from '@nestjs/core/application-config';

import { AppModule } from './app.module';

// These dependencies are normally created and managed by Nest's core bootstrap flow.
declare const container: NestContainer;
declare const metadataScanner: MetadataScanner;
declare const graphInspector: GraphInspector;
declare const applicationConfig: ApplicationConfig;

async function scanApplicationModules() {
  const scanner = new DependenciesScanner(
    container,
    metadataScanner,
    graphInspector,
    applicationConfig,
  );
```

```
  await scanner.scan(AppModule);

  // The container now includes discovered modules, providers, and controllers.
}
```

AI Coding Instructions

- Treat `DependenciesScanner` as framework bootstrap infrastructure; application code should normally rely on `NestFactory` rather than instantiate it directly.
- Preserve the scanning order: register modules before reflecting imports, providers, controllers, exports, and injectable metadata.
- When adding metadata reflection behavior, ensure discovered components are registered through `NestContainer` so dependency resolution remains consistent.
- Account for dynamic modules and module overrides when changing `scanForModules()` or `insertModule()`.
- Avoid duplicate module registration; use the `inserted` result from `insertModule()` when subsequent work should only run for newly added modules.

Relationships

- IMPORTS → `DynamicModule`
- IMPORTS → `ForwardReference`
- IMPORTS → `Provider`
- IMPORTS → `CATCH_WATERMARK`
- IMPORTS → `CONTROLLER_WATERMARK`
- IMPORTS → `ENHANCER_KEY_TO_SUBTYPE_MAP`
- IMPORTS → `EXCEPTION_FILTERS_METADATA`
- IMPORTS → `EnhancerSubtype`
- IMPORTS → `GUARDS_METADATA`
- IMPORTS → `INJECTABLE_WATERMARK`
- IMPORTS → `INTERCEPTORS_METADATA`
- IMPORTS → `MODULE_METADATA`
- IMPORTS → `PIPES_METADATA`
- IMPORTS → `ROUTE_ARGS_METADATA`
- IMPORTS → `CanActivate`
- IMPORTS → `ClassProvider`

- IMPORTS → `Controller`
- IMPORTS → `ExceptionFilter`
- IMPORTS → `ExistingProvider`
- IMPORTS → `FactoryProvider`
- IMPORTS → `Injectable`
- IMPORTS → `InjectionToken`
- IMPORTS → `NestInterceptor`
- IMPORTS → `PipeTransform`
- IMPORTS → `Scope`
- IMPORTS → `Type`
- IMPORTS → `ValueProvider`
- IMPORTS → `isFunction`
- IMPORTS → `isNil`
- IMPORTS → `isUndefined`

Used by

1 reference from 1 file. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (1)

- `TestingModuleOptions` — `packages/testing/testing-module.builder.ts`:29