NestContainer

August 18, 2026

NestContainer

Kind: Class

Source: `packages/core/injector/container.ts`

Part of: Core

`NestContainer` is NestJS's internal registry for application modules, dynamic module metadata, global modules, and HTTP adapter references. During bootstrap, it compiles and stores modules so the injector and dependency-scanning pipeline can resolve providers, imports, and global dependencies.

Methods

Method	Signature	Returns
`setHttpAdapter`	`setHttpAdapter(httpAdapter: any)`	`void`
`getHttpAdapterRef`	`getHttpAdapterRef()`	`void`
`getHttpAdapterHostRef`	`getHttpAdapterHostRef()`	`void`
`addModule`	`addModule(metatype: ModuleMetatype, scope: ModuleScope)`	`Promise<
`replaceModule`	`replaceModule(metatypeToReplace: ModuleMetatype, newMetatype: ModuleMetatype, scope: ModuleScope)`	`Promise<
`addDynamicMetadata`	`addDynamicMetadata(token: string, dynamicModuleMetadata: Partial<DynamicModule>, scope: Type<any>[])`	`void`
`addDynamicModules`	`addDynamicModules(modules: any[], scope: Type<any>[])`	`void`
`isGlobalModule`	`isGlobalModule(metatype: Type<any>, dynamicMetadata:`	`boolean`

Method	Signature	Returns
	`Partial<DynamicModule>)`	
`addGlobalModule`	`addGlobalModule(module: Module)`	`void`
`getModules`	`getModules()`	`ModulesContainer`
`getModuleCompiler`	`getModuleCompiler()`	`ModuleCompiler`
`getModuleByKey`	`getModuleByKey(moduleKey: string)`	`Module
`getInternalCoreModuleRef`	`getInternalCoreModuleRef()`	`Module
`addImport`	`addImport(relatedModule: Type	DynamicModule, token: string)`
`addProvider`	`addProvider(provider: Provider, token: string, enhancerSubtype: EnhancerSubtype)`	`string
`addInjectable`	`addInjectable(injectable: Provider, token: string, enhancerSubtype: EnhancerSubtype, host: Type<Injectable>)`	`void`
`addExportedProviderOrModule`	`addExportedProviderOrModule(toExport: Type	DynamicModule, token: string)`
`addController`	`addController(controller: Type<any>, token: string)`	`void`
`clear`	`clear()`	`void`
`replace`	`replace(toReplace: any, options: { scope: any[]	null })`
`bindGlobalScope`	`bindGlobalScope()`	`void`
`bindGlobalsToImports`	`bindGlobalsToImports(moduleRef: Module)`	`void`
`bindGlobalModuleToModule`	`bindGlobalModuleToModule(target: Module, globalModule: Module)`	`void`
`getDynamicMetadataByToken`	`getDynamicMetadataByToken(token: string)`	`Partial<DynamicModule>`
`getDynamicMetadataByToken`	`getDynamicMetadataByToken(token: string, metadataKey: K)`	`DynamicModule[K]`
`getDynamicMetadataByToken`	`getDynamicMetadataByToken(token: string, metadataKey: Exclude<keyof DynamicModule, 'global'	'module'>)`

Method	Signature	Returns
`registerCoreModuleRef`	`registerCoreModuleRef(moduleRef: Module)`	`void`
`getModuleTokenFactory`	`getModuleTokenFactory()`	`ModuleOpaqueKeyFactory`
`registerRequestProvider`	`registerRequestProvider(request: T, contextId: ContextId)`	`void`

Where it refuses work

- `NestContainer` stops the work with `UnknownModuleException` when `!this.modules.has(token)`, in 3 places.
- `NestContainer` stops the work with `UndefinedForwardRefException` when `!metatype`.
- `NestContainer` stops the work with `UndefinedForwardRefException` when `!metatypeToReplace || !newMetatype`.
- `NestContainer` stops the work with `CircularDependencyException` when `!provider`.
- `NestContainer` stops the work with `UnknownModuleException` when `!moduleRef`.
- `NestContainer` stops the work with an early return when `!this.internalProvidersStorage.httpAdapterHost`.

Diagram

```
graph LR
  Bootstrap[Nest application bootstrap] --> Container[NestContainer]
  Container --> Compiler[ModuleCompiler]
  Compiler --> Modules[ModulesContainer]
  Container --> DynamicMetadata[Dynamic module metadata]
  Container --> GlobalModules[Global module registry]
  Container --> Adapter[HTTP adapter reference]
  Modules --> Injector[Dependency injector]
  GlobalModules --> Injector
```

Usage

```
import { NestContainer } from '@nestjs/core/injector/container';
import { ExpressAdapter } from '@nestjs/platform-express';

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

// NestContainer is typically created internally by NestFactory.
// This example demonstrates the underlying module registration flow.
```

```
async function registerApplicationModule() {
  const container = new NestContainer();

  container.setHttpAdapter(new ExpressAdapter());

  const result = await container.addModule(AppModule);

  if (result?.inserted) {
    console.log(`Registered module: ${result.moduleRef.metatype.name}`);
  }

  const modules = container.getModules();
  console.log(`Registered modules: ${modules.size}`);

  return container;
}
```

AI Coding Instructions

- Treat `NestContainer` as framework infrastructure; application code should generally use `NestFactory` instead of constructing it directly.
- Register modules through `addModule()` so module compilation, dynamic metadata handling, and token generation remain consistent.
- Use `replaceModule()` only for controlled overrides, such as testing or platform-level module replacement.
- When adding dynamic modules, preserve their metadata through `addDynamicMetadata()` and `addDynamicModules()` rather than mutating the module registry directly.
- Mark and register global modules through the container flow so their exported providers are available across the application.

Relationships

- IMPORTS → `DynamicModule`
- IMPORTS → `Provider`
- IMPORTS → `EnhancerSubtype`
- IMPORTS → `GLOBAL_MODULE_METADATA`
- IMPORTS → `Injectable`
- IMPORTS → `Type`
- IMPORTS → `NestApplicationContextOptions`

Used by

9 references from 9 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (9)

- `ExceptionFiltersContext` — `packages/microservices/context/exception-filters-context.ts`:16
- `ListenersController` — `packages/microservices/listeners-controller.ts`:45
- `MicroservicesModule` — `packages/microservices/microservices-module.ts`:23
- `NestMicroservice` — `packages/microservices/nest-microservice.ts`:35
- `TestingInjector` — `packages/testing/testing-injector.ts`:23
- `TestingModuleOptions` — `packages/testing/testing-module.builder.ts`:29
- `TestingModule` — `packages/testing/testing-module.ts`:26
- `ExceptionFiltersContext` — `packages/websockets/context/exception-filters-context.ts`:10

…and 1 more.