DeepHashedModuleOpaqueKeyFactory

August 18, 2026

DeepHashedModuleOpaqueKeyFactory

Kind: Class

Source: `packages/core/injector/opaque-key-factory/deep-hashed-module-opaque-key-factory.ts`

Part of: Core

`DeepHashedModuleOpaqueKeyFactory` creates stable opaque keys used by the NestJS container to identify module instances. Static modules receive a key based on a cached module ID and name, while dynamic modules include deeply serialized metadata and a hash so differently configured instances remain distinct.

Implements: `ModuleOpaqueKeyFactory`

Methods

Method	Signature	Returns
`createForStatic`	`createForStatic(moduleCls: Type)`	`string`
`createForDynamic`	`createForDynamic(moduleCls: Type<unknown>, dynamicMetadata: Omit<DynamicModule, 'module'>)`	`string`
`getStringifiedOpaqueToken`	`getStringifiedOpaqueToken(opaqueToken: object	undefined)`
`getModuleId`	`getModuleId(metatype: Type<unknown>)`	`string`
`getModuleName`	`getModuleName(metatype: Type<any>)`	`string`

Where it refuses work

- `DeepHashedModuleOpaqueKeyFactory` stops the work with an early return when `this.moduleTokenCache.has(key)`.
- `DeepHashedModuleOpaqueKeyFactory` stops the work with an early return when `moduleId`.
- `DeepHashedModuleOpaqueKeyFactory` stops the work with an early return when `isClass`.

- `DeepHashedModuleOpaqueKeyFactory` stops the work with an early return when `isSymbol(value)`.

Diagram

```
graph LR
  A[Module class] --> B[getModuleId]
  A --> C[getModuleName]
  B --> D[Static module key]
  C --> D

  E[Dynamic module metadata] --> F[getStringifiedOpaqueToken]
  B --> G[Opaque token]
  C --> G
  E --> G
  G --> F
  F --> H[Hash]
  H --> I[Dynamic module key]
```

Usage

```
import { DeepHashedModuleOpaqueKeyFactory } from '@nestjs/core/injector/opaque-key-factory/deep

class FeatureModule {}

const keyFactory = new DeepHashedModuleOpaqueKeyFactory();

// Produces a stable key for this module class during the factory lifetime.
const staticKey = keyFactory.createForStatic(FeatureModule);

// Produces a hash that includes dynamic module configuration.
const dynamicKey = keyFactory.createForDynamic(FeatureModule, {
  providers: [{ provide: 'FEATURE_OPTIONS', useValue: { enabled: true } }],
  exports: ['FEATURE_OPTIONS'],
});

console.log({ staticKey, dynamicKey });
```

AI Coding Instructions

- Use `createForStatic()` for regular module classes and `createForDynamic()` whenever module metadata affects container identity.
- Preserve the cached module ID behavior; generating a new ID per call would prevent equivalent module references from resolving consistently.

- Ensure dynamic metadata is serializable through the factory's opaque-token stringification logic; functions and symbols require consistent representations.
- Do not rely on generated opaque keys as public application identifiers; they are internal container keys and may change between process runs.
- When extending module metadata, include identity-relevant configuration so distinct dynamic module registrations produce distinct keys.

Relationships

- IMPORTS → `DynamicModule`
- IMPORTS → `Type`
- IMPORTS → `Logger`
- IMPORTS → `randomStringGenerator`
- IMPORTS → `isFunction`
- IMPORTS → `isSymbol`