ModuleCompiler

August 18, 2026

ModuleCompiler

Kind: Class

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

Part of: Core

`ModuleCompiler` normalizes static module classes and `DynamicModule` objects into a compiled `ModuleFactory`. It extracts the module type and dynamic metadata, then creates a stable token used by the injector and module container to register and resolve modules.

Methods

Method	Signature	Returns
`compile`	`compile(moduleClsOrDynamic:	Type
`extractMetadata`	`extractMetadata(moduleClsOrDynamic: Type	ForwardReference
`isDynamicModule`	`isDynamicModule(moduleClsOrDynamic: Type	DynamicModule

Where it refuses work

- `ModuleCompiler` stops the work with an early return when `!this.isDynamicModule(moduleClsOrDynamic)`.

Diagram

```
graph LR
  A[Module class or DynamicModule] --> B[ModuleCompiler.compile]
  B --> C[extractMetadata]
  C --> D[Module type]
  C --> E[Dynamic metadata]
  D --> F[Module token creation]
```

```
  E --> F
  F --> G[ModuleFactory]
```

Usage

```ts
import { ModuleCompiler } from '@nestjs/core/injector/compiler';

class AppModule {}

const compiler = new ModuleCompiler();

// Compile a standard module class.
const staticFactory = await compiler.compile(AppModule);

// Compile a dynamic module definition.
const dynamicFactory = await compiler.compile({
  module: AppModule,
  providers: [
    {
      provide: 'APP_CONFIG',
      useValue: { environment: 'production' },
    },
  ],
  exports: ['APP_CONFIG'],
});

console.log(dynamicFactory.type); // AppModule
console.log(dynamicFactory.token); // Generated module token
```

AI Coding Instructions

- Pass either a module class or a valid `DynamicModule` object containing a `module` property to `compile()`.
- Keep dynamic module metadata separate from the `module` class; `extractMetadata()` intentionally removes `module` from `dynamicMetadata`.
- Use the generated `ModuleFactory.token` as the module identity when integrating with the container or injector.
- Preserve the dynamic-module type guard when extending this class so static and dynamic module inputs remain distinguishable.
- Avoid manually generating module tokens; delegate token creation to the compiler and its token factory.

How it works

`ModuleCompiler` converts a module definition into a `ModuleFactory`: the resolved module class (`type`), optional dynamic-module metadata (`dynamicMetadata`), and a string key (`token`). `ModuleFactory` declares those three fields, with `dynamicMetadata` optional. [packages/core/injector/compiler.ts:8-12]