NestFactory

August 17, 2026

NestFactory

Kind: Constant

Source: `packages/core/nest-factory.ts`

Part of: Core

Use NestFactory to create an application instance.

Specifying an entry module

Pass the required *root module* for the application via the module parameter. By convention, it is usually called `ApplicationModule`. Starting with this module, Nest assembles the dependency graph and begins the process of Dependency Injection and instantiates the classes needed to launch your application.

`NestFactory` creates and initializes Nest application instances from a root module. It starts the dependency-injection graph, configures the selected runtime adapter, and returns an application object that can be configured and started.

Definition

```
new NestFactoryStatic()
```

Value

```
new NestFactoryStatic()
```

Diagram

```
graph LR
  A[ApplicationModule] --> B[NestFactory]
  B --> C[Dependency Injection Container]
  C --> D[Providers and Controllers]
  B --> E[Nest Application Instance]
  E --> F[Configure Middleware / Pipes / Guards]
  F --> G[app.listen()]
```

Usage

```
import { NestFactory } from '@nestjs/core';
import { ApplicationModule } from './application.module';

async function bootstrap() {
  const app = await NestFactory.create(ApplicationModule);

  app.setGlobalPrefix('api');
  await app.listen(3000);
}

bootstrap();
```

AI Coding Instructions

- Pass the application's root module (commonly `ApplicationModule` or `AppModule`) as the first argument to `NestFactory.create()`.
- Await application creation before configuring global middleware, pipes, guards, interceptors, or filters.
- Configure the returned application instance before calling `app.listen()` so startup behavior is consistent.
- Use the appropriate factory method for the runtime: `create()` for HTTP applications, `createMicroservice()` for microservices, and `createApplicationContext()` for standalone contexts.