NestApplication

August 17, 2026

NestApplication

Kind: Class

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

Part of: Core

`NestApplication` is the core runtime class behind a Nest HTTP application. It coordinates HTTP server creation, module registration, WebSocket integration, parser middleware, and application lifecycle initialization. Applications are typically created through `NestFactory`, which configures and returns a `NestApplication` instance.

Extends: `NestApplicationContext`

Implements: `INestApplication`

Methods

Method	Signature	Returns
`dispose`	`dispose()`	`Promise<void>`
`getHttpAdapter`	`getHttpAdapter()`	`AbstractHttpAdapter`
`registerHttpServer`	`registerHttpServer()`	`void`
`getUnderlyingHttpServer`	`getUnderlyingHttpServer()`	`T`
`applyOptions`	`applyOptions()`	`void`
`createServer`	`createServer()`	`T`
`registerModules`	`registerModules()`	`void`
`registerWsModule`	`registerWsModule()`	`void`
`init`	`init()`	`Promise<this>`
`registerParserMiddleware`	`registerParserMiddleware()`	`void`

Method	Signature	Returns
`registerRouter`	`registerRouter()`	`void`
`registerRouterHooks`	`registerRouterHooks()`	`void`
`connectMicroservice`	`connectMicroservice(microserviceOptions: T, hybridAppOptions: NestHybridApplicationOptions)`	`INestMicroservice`
`getMicroservices`	`getMicroservices()`	`INestMicroservice[]`
`getHttpServer`	`getHttpServer()`	`void`
`startAllMicroservices`	`startAllMicroservices()`	`Promise<this>`
`use`	`use(args: [any, any?])`	`this`
`useBodyParser`	`useBodyParser(args: [any, any?])`	`this`
`enableCors`	`enableCors(options: any)`	`void`
`enableVersioning`	`enableVersioning(options: VersioningOptions)`	`this`
`listen`	`listen(port: number	string)`
`listen`	`listen(port: number	string, hostname: string)`
`listen`	`listen(port: number	string, args: any[])`
`getUrl`	`getUrl()`	`Promise<string>`
`setGlobalPrefix`	`setGlobalPrefix(prefix: string, options: GlobalPrefixOptions)`	`this`
`useWebSocketAdapter`	`useWebSocketAdapter(adapter: WebSocketAdapter)`	`this`
`useGlobalFilters`	`useGlobalFilters(filters: ExceptionFilter[])`	`this`
`useGlobalPipes`	`useGlobalPipes(pipes: PipeTransform<any>[])`	`this`
`useGlobalInterceptors`	`useGlobalInterceptors(interceptors: NestInterceptor[])`	`this`
`useGlobalGuards`	`useGlobalGuards(guards: CanActivate[])`	`this`
`useStaticAssets`	`useStaticAssets(options: any)`	`this`
`useStaticAssets`	`useStaticAssets(path: string, options: any)`	`this`

Method	Signature	Returns
`useStaticAssets`	`useStaticAssets(pathOrOptions: any, options: any)`	`this`
`setBaseViewsDir`	`setBaseViewsDir(path: string`	string[])`
`setViewEngine`	`setViewEngine(engineOrOptions: any)`	`this`

Properties

Property	Type
`logger`	`any`

Where it refuses work

- `NestApplication` stops the work with an early return when `!this.appOptions || !this.appOptions.cors`.
- `NestApplication` stops the work with an early return when `!passCustomOptions`.
- `NestApplication` stops the work with an early return when `!this.socketModule`.
- `NestApplication` stops the work with an early return when `this.isInitialized`.
- `NestApplication` stops the work with an early return when `originalCallbackArgs[0] instanceof Error`.
- `NestApplication` stops the work with an early return when `platform() === 'win32'`.

Diagram

```
graph LR
  A[NestFactory.create()] --> B[NestApplication]
  B --> C[createServer()]
  B --> D[registerModules()]
  B --> E[registerWsModule()]
  B --> F[registerParserMiddleware()]
  B --> G[init()]
  G --> H[HTTP Adapter]
  H --> I[Underlying HTTP Server]
  B --> J[dispose()]
```

Usage

```ts
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

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

  // Initialize registered modules, middleware, and adapters.
  await app.init();

  const httpAdapter = app.getHttpAdapter();
  const server = app.getUnderlyingHttpServer();

  console.log(`Using adapter: ${httpAdapter.getType()}`);
  console.log(`Server available: ${Boolean(server)}`);

  await app.listen(3000);

  process.on('SIGTERM', async () => {
    await app.close();
    await app.dispose();
  });
}

bootstrap();
```

AI Coding Instructions

- Create application instances through `NestFactory.create()` rather than constructing `NestApplication` directly.
- Preserve the initialization order: register modules and infrastructure before calling `init()`.
- Use `getHttpAdapter()` for adapter-specific integrations; avoid assuming Express or Fastify APIs are always available.
- Use `getUnderlyingHttpServer()` only when direct access to the native HTTP server is required.
- Ensure shutdown paths clean up resources through the application lifecycle, including disposal when appropriate.

Relationships

- IMPORTS → `CanActivate`
- IMPORTS → `ExceptionFilter`
- IMPORTS → `HttpServer`

- IMPORTS → `INestApplication`
- IMPORTS → `INestMicroservice`
- IMPORTS → `NestHybridApplicationOptions`
- IMPORTS → `NestInterceptor`
- IMPORTS → `PipeTransform`
- IMPORTS → `VersioningOptions`
- IMPORTS → `VersioningType`
- IMPORTS → `WebSocketAdapter`
- IMPORTS → `GlobalPrefixOptions`
- IMPORTS → `NestApplicationOptions`
- IMPORTS → `Logger`
- IMPORTS → `loadPackage`
- IMPORTS → `addLeadingSlash`
- IMPORTS → `isFunction`
- IMPORTS → `isObject`
- IMPORTS → `isString`
- IMPORTS → `SocketModule`
- IMPORTS → `MicroservicesModule`
- IMPORTS → `-nestjs-microservices`

Used by

2 references from 2 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (2)

- `TestingModule` — `packages/testing/testing-module.ts` :26
- `BaseWsInstance` — `packages/websockets/adapters/ws-adapter.ts` :8