AbstractHttpAdapter

August 17, 2026

AbstractHttpAdapter

Kind: Class

Source: `packages/core/adapters/http-adapter.ts`

Part of: Core

`AbstractHttpAdapter` defines the common HTTP server adapter contract used by NestJS platform integrations. It delegates middleware and route registration methods such as `use()`, `get()`, `post()`, and `head()` to the underlying HTTP framework instance, allowing Nest applications to work with different server implementations.

Implements: `HttpServer`

Methods

Method	Signature	Returns
`init`	`init()`	`void`
`use`	`use(args: any[])`	`void`
`get`	`get(handler: RequestHandler)`	`void`
`get`	`get(path: any, handler: RequestHandler)`	`void`
`get`	`get(args: any[])`	`void`
`post`	`post(handler: RequestHandler)`	`void`
`post`	`post(path: any, handler: RequestHandler)`	`void`
`post`	`post(args: any[])`	`void`
`head`	`head(handler: RequestHandler)`	`void`
`head`	`head(path: any, handler: RequestHandler)`	`void`
`head`	`head(args: any[])`	`void`

Method	Signature	Returns
`delete`	`delete(handler: RequestHandler)`	`void`
`delete`	`delete(path: any, handler: RequestHandler)`	`void`
`delete`	`delete(args: any[])`	`void`
`put`	`put(handler: RequestHandler)`	`void`
`put`	`put(path: any, handler: RequestHandler)`	`void`
`put`	`put(args: any[])`	`void`
`patch`	`patch(handler: RequestHandler)`	`void`
`patch`	`patch(path: any, handler: RequestHandler)`	`void`
`patch`	`patch(args: any[])`	`void`
`propfind`	`propfind(handler: RequestHandler)`	`void`
`propfind`	`propfind(path: any, handler: RequestHandler)`	`void`
`propfind`	`propfind(args: any[])`	`void`
`proppatch`	`proppatch(handler: RequestHandler)`	`void`
`proppatch`	`proppatch(path: any, handler: RequestHandler)`	`void`
`proppatch`	`proppatch(args: any[])`	`void`
`mkcol`	`mkcol(handler: RequestHandler)`	`void`
`mkcol`	`mkcol(path: any, handler: RequestHandler)`	`void`
`mkcol`	`mkcol(args: any[])`	`void`
`copy`	`copy(handler: RequestHandler)`	`void`
`copy`	`copy(path: any, handler: RequestHandler)`	`void`
`copy`	`copy(args: any[])`	`void`
`move`	`move(handler: RequestHandler)`	`void`
`move`	`move(path: any, handler: RequestHandler)`	`void`
`move`	`move(args: any[])`	`void`
`lock`	`lock(handler: RequestHandler)`	`void`
`lock`	`lock(path: any, handler: RequestHandler)`	`void`
`lock`	`lock(args: any[])`	`void`
`unlock`	`unlock(handler: RequestHandler)`	`void`

Method	Signature	Returns
`unlock`	`unlock(path: any, handler: RequestHandler)`	`void`
`unlock`	`unlock(args: any[])`	`void`
`all`	`all(handler: RequestHandler)`	`void`
`all`	`all(path: any, handler: RequestHandler)`	`void`
`all`	`all(args: any[])`	`void`
`search`	`search(handler: RequestHandler)`	`void`
`search`	`search(path: any, handler: RequestHandler)`	`void`
`search`	`search(args: any[])`	`void`
`options`	`options(handler: RequestHandler)`	`void`
`options`	`options(path: any, handler: RequestHandler)`	`void`
`options`	`options(args: any[])`	`void`

Properties

Property	Type
`httpServer`	`TServer`
`onRouteTriggered`	`

Diagram

```
graph LR
  Nest[Nest Application] --> Adapter[AbstractHttpAdapter]
  Adapter --> Instance[Underlying HTTP Framework Instance]
  Adapter --> Init[init()]
  Adapter --> Middleware[use()]
  Adapter --> Routes[get() / post() / head()]
  Middleware --> Instance
  Routes --> Instance
  Instance --> Server[HTTP Server]
```

Usage

```
import * as express from 'express';
import { NestFactory } from '@nestjs/core';
import { ExpressAdapter } from '@nestjs/platform-express';
```

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

async function bootstrap() {
  const expressApp = express();
  const httpAdapter = new ExpressAdapter(expressApp);

  // Register framework-native middleware through the adapter.
  httpAdapter.use((req, res, next) => {
    console.log(`${req.method} ${req.url}`);
    next();
  });

  // Register routes directly on the underlying platform.
  httpAdapter.get('/health', (_req, res) => {
    res.status(200).json({ status: 'ok' });
  });

  httpAdapter.head('/health', (_req, res) => {
    res.status(200).end();
  });

  const app = await NestFactory.create(AppModule, httpAdapter);
  await app.listen(3000);
}

bootstrap();
```

AI Coding Instructions

- Extend `AbstractHttpAdapter` only when integrating a new HTTP platform; use concrete adapters such as `ExpressAdapter` or `FastifyAdapter` in application code.
- Delegate route and middleware methods to the underlying framework instance so native platform behavior is preserved.
- Keep `get()`, `post()`, and `head()` overload behavior compatible with the target framework's route-registration API.
- Use `use()` for platform middleware registration, but prefer Nest middleware, guards, interceptors, and controllers for application-level concerns.
- Ensure the adapter is initialized and passed to `NestFactory.create()` when configuring a custom HTTP server instance.

How it works

`AbstractHttpAdapter<TServer, TRequest, TResponse>` is a public abstract base class for HTTP platform adapters. It implements the `HttpServer<TRequest, TResponse>` contract

and stores both a platform instance and, separately, an HTTP server value. Its type parameters default to `any`. `packages/core/adapters/http-adapter.ts:5-18`

Relationships

- IMPORTS → `HttpServer`
- IMPORTS → `RequestMethod`
- IMPORTS → `VersioningOptions`
- IMPORTS → `RequestHandler`
- IMPORTS → `VersionValue`
- IMPORTS → `NestApplicationOptions`

Used by

3 references from 3 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (3)

- `ExpressAdapter` — `packages/platform-express/adapters/express-adapter.ts`:51
- `FastifyAdapter` — `packages/platform-fastify/adapters/fastify-adapter.ts`:124
- `TestingModule` — `packages/testing/testing-module.ts`:26