RequestMappingMetadata

August 17, 2026

RequestMappingMetadata

Kind: Interface

Source: `packages/common/decorators/http/request-mapping.decorator.ts`

Part of: Common

`RequestMappingMetadata` describes the routing information attached to an HTTP request handler. It combines one or more URL paths with a `RequestMethod`, allowing request-mapping decorators and routing infrastructure to register controller endpoints consistently.

Properties

Property	Type
`path`	`string
`method`	`RequestMethod`

Diagram

```
graph LR
  A[RequestMappingMetadata] --> B[path: string | string[]]
  A --> C[method: RequestMethod]
  B --> D[Single route path]
  B --> E[Multiple route paths]
  C --> F[HTTP verb]
  F --> G[Router registration]
```

Usage

```
import { RequestMethod } from '@nestjs/common';
import type { RequestMappingMetadata } from './request-mapping.decorator';
```

```ts
const mapping: RequestMappingMetadata = {
  path: ['/users', '/accounts'],
  method: RequestMethod.GET,
};

// Example decorator metadata consumed by the HTTP router.
function registerRoute(metadata: RequestMappingMetadata) {
  for (const path of Array.isArray(metadata.path)
    ? metadata.path
    : [metadata.path]) {
    console.log(`Registering ${RequestMethod[metadata.method]} ${path}`);
  }
}

registerRoute(mapping);
```

AI Coding Instructions

- Use `path` as a string for a single route or a string array when the same handler supports multiple routes.
- Always provide a valid `RequestMethod` enum value; do not use raw HTTP method strings unless converted first.
- Normalize `path` to an array before iterating over route mappings in router integration code.
- Keep this metadata focused on route path and HTTP method; add unrelated handler metadata through separate interfaces or decorators.

How it works

- `RequestMappingMetadata` is an exported TypeScript interface used as the `metadata` argument type for the `RequestMapping` method-decorator factory. It declares two optional fields: `path`, a string or string array, and `method`, a `RequestMethod` enum member. packages/common/decorators/http/request-mapping.decorator.ts:4-7 packages/common/decorators/http/request-mapping.decorator.ts:14-16
- `path` corresponds to the `PATH_METADATA` key, whose string value is `'path'`; `method` corresponds to `METHOD_METADATA`, whose string value is `'method'`. packages/common/constants.ts:10 packages/common/constants.ts:17
- Its `method` type is the `RequestMethod` enum. The enum contains `GET`, `POST`, `PUT`, `DELETE`, `PATCH`, `ALL`, `OPTIONS`, `HEAD`, `SEARCH`, and WebDAV-related members through `UNLOCK`. packages/common/enums/request-method.enum.ts:1-18

- When passed to `RequestMapping`, a non-empty `path` value is retained; an omitted path, empty string, or empty array becomes `'/'`. An omitted or falsy `method` becomes `RequestMethod.GET`. packages/common/decorators/http/request-mapping.decorator.ts:9-19 The decorator writes the resulting path and request method as reflection metadata on the decorated method's function (`descriptor.value`). packages/common/decorators/http/request-mapping.decorator.ts:21-29
- The interface contains no runtime validation or error handling itself. Its observable runtime effect occurs only through `RequestMapping`, which calls `Reflect.defineMetadata` for both keys. packages/common/decorators/http/request-mapping.decorator.ts:4-7 packages/common/decorators/http/request-mapping.decorator.ts:26-27
- During controller route discovery, the stored path and method metadata are read from the prototype method. A string path is converted to a one-element path array after adding a leading slash; an array is mapped the same way per element. If no path metadata exists, that method is not treated as a route. packages/core/router/paths-explorer.ts:53-82