UsePipes

August 18, 2026

UsePipes

Kind: Function

Source: `packages/common/decorators/core/use-pipes.decorator.ts`

Part of: Common

Decorator that binds pipes to the scope of the controller or method, depending on its context.

When `@UsePipes` is used at the controller level, the pipe will be applied to every handler (method) in the controller.

When `@UsePipes` is used at the individual handler level, the pipe will apply only to that specific method.

`@UsePipes()` binds one or more NestJS pipes to a controller class or individual route handler. Pipes transform or validate incoming request data before it reaches the handler, with controller-level pipes applying to all handlers and method-level pipes applying only to that route.

Signature

```
function UsePipes(pipes: (PipeTransform | Function)[]): ClassDecorator & MethodDecorator
```

Parameters

Name	Type
`pipes`	`(PipeTransform

Returns: `ClassDecorator & MethodDecorator`

Diagram

```
graph LR
  Request[Incoming Request] --> Controller{Controller Scope?}
  Controller -->|@UsePipes on controller| GlobalPipe[Apply pipes to every handler]
  Controller -->|@UsePipes on method| MethodPipe[Apply pipes only to selected handler]
  GlobalPipe --> Handler[Route Handler]
  MethodPipe --> Handler
  Handler --> Response[Response]
```

Usage

```
import {
  Body,
  Controller,
  Get,
  Param,
  ParseIntPipe,
  Post,
  UsePipes,
  ValidationPipe,
} from '@nestjs/common';

class CreateUserDto {
  name: string;
  email: string;
}

@Controller('users')
@UsePipes(new ValidationPipe({ transform: true }))
export class UsersController {
  @Post()
  create(@Body() createUserDto: CreateUserDto) {
    return createUserDto;
  }

  @Get(':id')
  findOne(
    @Param('id', ParseIntPipe) id: number,
  ) {
    return { id };
  }
}
```

AI Coding Instructions

- Apply `@UsePipes()` at the controller level when the same validation or transformation behavior should affect every route in that controller.

- Apply it at the method level for route-specific pipe behavior; method-level configuration should not be assumed to affect sibling handlers.
- Pass pipe classes, instances, or multiple pipe arguments as supported by NestJS, such as `@UsePipes(ValidationPipe)` or `@UsePipes(new ValidationPipe())`.
- Prefer DTO-based validation with `ValidationPipe` for request bodies, and use parameter pipes such as `ParseIntPipe` for targeted route parameter conversion.
- Ensure validation pipes are configured consistently with application-wide pipe settings to avoid unexpected differences between routes.