ExecutionContextHost

August 18, 2026

ExecutionContextHost

Kind: Class

Source: `packages/core/helpers/execution-context-host.ts`

Part of: Core

`ExecutionContextHost` is the concrete implementation of Nest's execution context abstraction. It stores the current invocation arguments, controller class, handler function, and transport type, then exposes transport-specific views for HTTP, RPC, and WebSocket execution.

Implements: `ExecutionContext`

Methods

Method	Signature	Returns
`setType`	`setType(type: TContext)`	`void`
`getType`	`getType()`	`TContext`
`getClass`	`getClass()`	`Type<T>`
`getHandler`	`getHandler()`	`Function`
`getArgs`	`getArgs()`	`T`
`getArgByIndex`	`getArgByIndex(index: number)`	`T`
`switchToRpc`	`switchToRpc()`	`RpcArgumentsHost`
`switchToHttp`	`switchToHttp()`	`HttpArgumentsHost`
`switchToWs`	`switchToWs()`	`WsArgumentsHost`

Diagram

```
graph LR
  A[Invocation Arguments] --> B[ExecutionContextHost]
  C[Controller Class] --> B
  D[Handler Method] --> B

  B --> E[getArgs / getArgByIndex]
  B --> F[getClass / getHandler]
  B --> G[setType / getType]

  B --> H[switchToHttp]
  B --> I[switchToRpc]
  B --> J[switchToWs]

  H --> K[HttpArgumentsHost]
  I --> L[RpcArgumentsHost]
  J --> M[WsArgumentsHost]
```

Usage

```
import { ExecutionContextHost } from '@nestjs/core/helpers/execution-context-host';

class UsersController {
  findOne() {
    return { id: 1 };
  }
}

const request = { params: { id: '1' } };
const response = {};
const next = () => undefined;

const context = new ExecutionContextHost(
  [request, response, next],
  UsersController,
  UsersController.prototype.findOne,
);

context.setType('http');

const http = context.switchToHttp();

console.log(context.getType()); // "http"
console.log(context.getClass()); // UsersController
console.log(context.getHandler()); // findOne
console.log(http.getRequest().params.id); // "1"
```

AI Coding Instructions

- Preserve argument ordering when creating a context: HTTP uses `[request, response, next]`, while RPC and WebSocket adapters use transport-specific argument positions.
- Call `setType()` when manually constructing a host so guards, interceptors, and metadata consumers can identify the active transport.
- Use `switchToHttp()`, `switchToRpc()`, or `switchToWs()` instead of directly indexing arguments when writing transport-aware integrations.
- Treat `getClass()` and `getHandler()` as metadata lookup targets for decorators, reflection, guards, and interceptors.
- Avoid reusing one `ExecutionContextHost` instance across unrelated requests or message handlers.

How it works

`ExecutionContextHost` is a concrete `ExecutionContext` implementation that stores a handler argument array, an optional class reference, an optional handler reference, and a mutable context-type string. Its initial context type is `'http'`. [packages/core/helpers/execution-context-host.ts:10-17]

- Its constructor accepts `args`, `constructorRef`, and `handler`; the latter two default to `null`. [packages/core/helpers/execution-context-host.ts:13-17]
- `getArgs()` returns the stored argument array, and `getArgByIndex(index)` reads the element at that index. [packages/core/helpers/execution-context-host.ts:35-41]
- `getClass()` returns the stored class reference and `getHandler()` returns the stored handler reference. Both use TypeScript non-null assertions, although their constructor values may be `null` when omitted. [packages/core/helpers/execution-context-host.ts:13-17] [packages/core/helpers/execution-context-host.ts:27-33]
- `setType(type)` replaces the stored context type only when `type` is truthy; a falsy value leaves the existing type unchanged. `getType()` returns the stored value. [packages/core/helpers/execution-context-host.ts:19-25]
- `switchToRpc()` assigns `getData()` and `getContext()` methods onto the host itself and returns that same object; those methods read argument indexes `0` and `1`, respectively. [packages/core/helpers/execution-context-host.ts:43-48]
- `switchToHttp()` assigns `getRequest()`, `getResponse()`, and `getNext()` onto the same host and maps them to argument indexes `0`, `1`, and `2`. [packages/core/helpers/execution-context-host.ts:50-56]

- `switchToWs()` assigns `getClient()` and `getData()` onto the same host for argument indexes `0` and `1`; `getPattern()` reads the final element of the current argument array. [packages/core/helpers/execution-context-host.ts:58-64]
- The switch methods use `Object.assign(this, ...)`; therefore, each call has the side effect of adding or replacing those accessor methods on the existing instance rather than creating a separate adapter object. [packages/core/helpers/execution-context-host.ts:43-64]
- The class contains no explicit runtime validation, bounds checks, or thrown errors for the argument array, indexes, class reference, handler reference, or context type. [packages/core/helpers/execution-context-host.ts:13-64]

Core code creates this host for guards with the invocation arguments, `instance.constructor`, and callback, then sets the optional guard context type before passing the host to each guard's `canActivate`. [packages/core/guards/guards-consumer.ts:18-22] [packages/core/guards/guards-consumer.ts:37-46] Shared context helpers also construct it from invocation arguments and optionally set its type for custom parameter factories. [packages/core/helpers/context-utils.ts:77-85] [packages/core/helpers/context-utils.ts:87-97]

Relationships

- IMPORTS → `ExecutionContext`
- IMPORTS → `Type`
- IMPORTS → `ContextType`
- IMPORTS → `HttpArgumentsHost`
- IMPORTS → `RpcArgumentsHost`
- IMPORTS → `WsArgumentsHost`

Used by

5 references from 5 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (5)

- `RpcHandlerMetadata` — `packages/microservices/context/rpc-context-creator.ts`:35
- `RpcProxy` — `packages/microservices/context/rpc-proxy.ts`:6
- `ListenersController` — `packages/microservices/listeners-controller.ts`:45
- `WsHandlerMetadata` — `packages/websockets/context/ws-context-creator.ts`:34

- `WsProxy` — `packages/websockets/context/ws-proxy.ts`:6