# RouterExecutionContext

August 18, 2026

## RouterExecutionContext

**Kind:** Class

**Source:** `packages/core/router/router-execution-context.ts`

**Part of:** Core

`RouterExecutionContext` builds the runtime execution layer for a router handler. It reflects handler metadata such as status codes, redirects, templates, headers, SSE configuration, and parameter bindings, then resolves request values and applies supported pipes before invoking the handler.

### Methods

| Method | Signature | Returns |
| --- | --- | --- |
| `create` | `create(instance: Controller, callback: (...args: any[]) => unknown, methodName: string, moduleKey: string, requestMethod: RequestMethod, contextId: undefined, inquirerId: string)` | `void` |
| `getMetadata` | `getMetadata(instance: Controller, callback: (...args: any[]) => any, methodName: string, moduleKey: string, requestMethod: RequestMethod, contextType: TContext)` | `HandlerMetadata` |
| `reflectRedirect` | `reflectRedirect(callback: (...args: unknown[]) => unknown)` | `RedirectResponse` |
| `reflectHttpStatusCode` | `reflectHttpStatusCode(callback: (...args: unknown[]) => unknown)` | `number` |
| `reflectRenderTemplate` | `reflectRenderTemplate(callback: (...args: unknown[]) => unknown)` | `string` |

| Method | Signature | Returns |
| --- | --- | --- |
| `reflectResponseHeaders` | `reflectResponseHeaders(callback: (...args: unknown[]) => unknown)` | `CustomHeader[]` |
| `reflectSse` | `reflectSse(callback: (...args: unknown[]) => unknown)` | `string` |
| `exchangeKeysForValues` | `exchangeKeysForValues(keys: string[], metadata: Record<number, RouteParamMetadata>, moduleContext: string, contextId: undefined, inquirerId: string, contextFactory: (args: unknown[]) => ExecutionContextHost)` | `ParamProperties[]` |
| `getParamValue` | `getParamValue(value: T, { | |

```
  metatype,
  type,
  data,
}: { metatype: unknown; type: RouteParamtypes; data: unknown }, pipes: PipeTransform[])` | `Pro
```

| `isPipeable` | `isPipeable(type: number | string)` | `boolean` |
| `createGuardsFn` | `createGuardsFn(guards: CanActivate[], instance: Controller, callback: (...args: any[]) => any, contextType: TContext)` | `((args: any[]) => Promise<void>) | null` |
| `createPipesFn` | `createPipesFn(pipes: PipeTransform[], paramsOptions: (ParamProperties & { metatype?: any })[])` | `void` |
| `createHandleResponseFn` | `createHandleResponseFn(callback: (...args: unknown[]) => unknown, isResponseHandled: boolean, redirectResponse: RedirectResponse, httpStatusCode: number)` | `HandleResponseFn` |

## Where it refuses work

- `RouterExecutionContext` stops the work with `ForbiddenException` when `!canActivate`.
- `RouterExecutionContext` stops the work with an early return when `cacheMetadata`.
- `RouterExecutionContext` stops the work with an early return when `!isEmpty(pipes)`.
- `RouterExecutionContext` stops the work with an early return when `renderTemplate`.
- `RouterExecutionContext` stops the work with an early return when `redirectResponse && isString(redirectResponse.url)`.
- `RouterExecutionContext` stops the work with an early return when `isSseHandler`.

## Diagram

```
graph LR
  A[Incoming HTTP Request] --> B[RouterExecutionContext]
  B --> C[getMetadata]
  C --> D[Reflect route metadata]
  D --> E[exchangeKeysForValues]
  E --> F[getParamValue]
  F --> G[Apply pipes when isPipeable]
  G --> H[Execute route handler]
  H --> I[Apply response metadata]
  I --> J[HTTP Response]
```

## Usage

```
import { RouterExecutionContext } from '@core/router/router-execution-context';

// The context is typically created by the router during route registration.
// Its dependencies, handler, controller instance, and metadata are configured
// before creating the request-time handler.
const executionContext = new RouterExecutionContext(
  /* framework dependencies */
);

const handler = executionContext.create();

// Register the generated callback with the underlying HTTP router.
router.get('/users/:id', handler);

// Internally, the generated handler can:
// - resolve @Param(), @Query(), @Body(), and other parameter values
// - run supported parameter pipes
// - invoke the controller method
// - apply redirect, status, header, template, or SSE metadata
```

## AI Coding Instructions

- Keep route-handler execution logic inside `RouterExecutionContext`; router adapters should only register the callback returned by `create()`.
- Use the `reflect*()` methods to read handler metadata instead of duplicating decorator or reflection lookups elsewhere.
- Resolve request arguments through `exchangeKeysForValues()` and `getParamValue()` so parameter decorators and pipes follow the same execution path.
- Check `isPipeable()` before applying transformation or validation pipes; not every resolved parameter value should be piped.

- Preserve response metadata behavior for redirects, HTTP status codes, templates, custom headers, and SSE responses when changing handler execution flow.