RpcExceptionsHandler

August 18, 2026

RpcExceptionsHandler

Kind: Class

Source: `packages/microservices/exceptions/rpc-exceptions-handler.ts`

Part of: Microservices

`RpcExceptionsHandler` handles exceptions raised while processing microservice RPC requests. It first attempts to delegate errors to registered custom RPC exception filters, then falls back to the framework's default RPC exception handling behavior when no filter applies.

Extends: `BaseRpcExceptionFilter`

Methods

Method	Signature	Returns
`handle`	`handle(exception: Error	RpcException, host: ArgumentsHost)`
`setCustomFilters`	`setCustomFilters(filters: RpcExceptionFilterMetadata[])`	`void`
`invokeCustomFilters`	`invokeCustomFilters(exception: T, host: ArgumentsHost)`	`Observable

Where it refuses work

- `RpcExceptionsHandler` stops the work with `InvalidExceptionFilterException` when `!Array.isArray(filters)`.
- `RpcExceptionsHandler` stops the work with an early return when `filterResult$`.
- `RpcExceptionsHandler` stops the work with an early return when `isEmpty(this.filters)`.

Diagram

```
graph LR
  A[RPC handler throws exception] --> B[RpcExceptionsHandler.handle]
  B --> C[invokeCustomFilters]
  C -->|Matching filter found| D[Custom RPC exception filter]
  C -->|No matching filter| E[BaseRpcExceptionFilter.catch]
  D --> F[Observable RPC error response]
  E --> F
```

Usage

```
import { RpcExceptionsHandler } from '@nestjs/microservices';
import { RpcException } from '@nestjs/microservices';
import { Observable, throwError } from 'rxjs';

const exceptionsHandler = new RpcExceptionsHandler();

exceptionsHandler.setCustomFilters([
  {
    catch(exception: Error): Observable<never> {
      console.error('RPC error:', exception.message);

      return throwError(
        () => new RpcException('A service-specific error occurred'),
      );
    },
  },
]);

const response$ = exceptionsHandler.handle(
  new Error('Unable to process payment'),
);

response$.subscribe({
  error: error => console.error(error),
});
```

AI Coding Instructions

- Register custom filters through `setCustomFilters()` when a microservice needs transport- or domain-specific error responses.
- Ensure custom filters return an `Observable`; RPC exception handling is asynchronous and must preserve the observable contract.

- Let `handle()` remain the main entry point so custom filters are evaluated before default exception serialization.
- Make custom filters target specific exception types where possible to avoid unintentionally handling unrelated RPC errors.
- Keep fallback behavior intact: if `invokeCustomFilters()` returns `null`, delegate to the base RPC exception handler.

How it works

- `RpcExceptionsHandler` is an exported public class that extends `BaseRpcExceptionFilter`. It maintains a private array of RPC exception-filter metadata, initially empty. [packages/microservices/exceptions/rpc-exceptions-handler.ts:10-14]
- `handle(exception, host)` first calls `invokeCustomFilters`. If that call returns a truthy observable, `handle` returns it; otherwise it delegates to the inherited `catch(exception, host)` method. Its declared inputs are an `Error` or `RpcException` and an `ArgumentsHost`, and its declared return type is `Observable<any>`. [packages/microservices/exceptions/rpc-exceptions-handler.ts:16-25]
- A configured filter is selected only when its `exceptionMetatypes` array is empty or the thrown value is an `instanceof` at least one listed metatype. Selection uses `find`, so only the first matching metadata entry is invoked. The selected entry's `func` is called with `(exception, host)` and its observable is returned. [packages/microservices/exceptions/rpc-exceptions-handler.ts:34-44] [packages/common/utils/select-exception-filter-metadata.util.ts:3-13] The metadata contract consists of a filter `catch` function and an array of exception metatypes; that function returns an observable. [packages/common/interfaces/exceptions/rpc-exception-filter-metadata.interface.ts:4-7] [packages/common/interfaces/exceptions/rpc-exception-filter.interface.ts:11-19]
- If no filters have been set, or the stored array has no items, `invokeCustomFilters` returns `null`. It also returns `null` when no configured filter matches, causing `handle` to use inherited handling. [packages/microservices/exceptions/rpc-exceptions-handler.ts:38-43] [packages/common/utils/shared.utils.ts:48-50]
- `setCustomFilters(filters)` requires its runtime argument to be an array. A non-array throws `InvalidExceptionFilterException`; an array replaces the handler's stored filter array. [packages/microservices/exceptions/rpc-exceptions-handler.ts:27-32] That exception extends `RuntimeException` and is initialized with the message `Invalid`

`exception filters (@UseFilters()).` . [packages/core/errors/exceptions/invalid-exception-filter.exception.ts:4-7] [packages/core/errors/messages.ts:260-263]

- In the inherited fallback, a `RpcException` produces an RxJS error observable: if `getError()` returns an object, that object is emitted as the error; otherwise the emitted error is `{ status: 'error', message: <getError result> }` . [packages/microservices/exceptions/base-rpc-exception-filter.ts:21-29] An exception that is not a `RpcException` produces an error observable with `{ status: 'error', message: 'Internal server error' }` ; it is logged unless it is an `IntrinsicException` . [packages/microservices/exceptions/base-rpc-exception-filter.ts:31-40] [packages/core/constants.ts:5-8]
- In the microservices execution path, `ExceptionFiltersContext.create` instantiates this handler, builds filter metadata for a controller callback, and, when that set is non-empty, stores the reversed array through `setCustomFilters` . [packages/microservices/context/exception-filters-context.ts:24-45] `RpcProxy` sends both rejected observables and thrown callback errors to `handle` , creating an `ExecutionContextHost` from the call arguments and marking its type as `rpc` . [packages/microservices/context/rpc-proxy.ts:11-23] [packages/microservices/context/rpc-proxy.ts:27-35]

Relationships

- IMPORTS → `RpcExceptionFilterMetadata`
- IMPORTS → `ArgumentsHost`
- IMPORTS → `selectExceptionFilterMetadata`
- IMPORTS → `isEmpty`
- IMPORTS → `InvalidExceptionFilterException`