RpcProxy

August 18, 2026

RpcProxy

Kind: Class

Source: `packages/microservices/context/rpc-proxy.ts`

Part of: Microservices

`RpcProxy` wraps RPC handler callbacks with consistent asynchronous execution and error handling. It converts thrown errors and observable stream errors into RPC-compatible error responses, including unwrapping `RpcException` payloads for microservice transports.

Methods

Method	Signature	Returns
`create`	`create(targetCallback: (...args: unknown[]) => Promise<Observable<any>>, exceptionsHandler: RpcExceptionsHandler)`	`(...args: unknown[]) => Promise<Observable<unknown>>`
`handleError`	`handleError(exceptionsHandler: RpcExceptionsHandler, args: unknown[], error: T)`	`Observable<unknown>`

When something fails

- `RpcProxy` handles failure in 1 place: it turns it into a return value in all 1.

Diagram

```
graph LR
  A[Microservice transport] --> B[RpcProxy.create]
  B --> C[RPC handler callback]
  C --> D{Returns or throws}
```

```
  D -->|Observable result| E[Attach catchError]
  D -->|Thrown error| F[handleError]
  E --> G[RPC response Observable]
  F --> G
```

Usage

```ts
import { Observable, of } from 'rxjs';
import { RpcException } from '@nestjs/microservices';
import { RpcProxy } from '@nestjs/microservices/context/rpc-proxy';

const rpcProxy = new RpcProxy();

const handler = async (id: string): Promise<Observable<unknown>> => {
  if (!id) {
    throw new RpcException('A user id is required');
  }

  return of({ id, name: 'Ada Lovelace' });
};

const proxiedHandler = rpcProxy.create(handler);

const response$ = await proxiedHandler('user-123');

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

AI Coding Instructions

- Wrap microservice request handlers with `RpcProxy.create()` when adding transport-level execution paths that need standardized RPC error behavior.
- Return `Observable` values from wrapped handlers; errors emitted by the observable are routed through `handleError()`.
- Use `RpcException` for expected client-facing RPC failures so its payload can be sent through the configured transport.
- Do not manually duplicate `try/catch` and RxJS `catchError` logic around handlers when `RpcProxy` is already responsible for error normalization.
- Preserve the asynchronous callback signature when integrating with new transports or context creators.

How it works

`RpcProxy`

`RpcProxy` is a class that wraps an RPC callback with exception handling. Its `create()` method accepts a callback and an `RpcExceptionsHandler`, then returns an asynchronous function that accepts and forwards arbitrary arguments. [packages/microservices/context/rpc-proxy.ts:6-10]

Relationships

- IMPORTS → `ExecutionContextHost`