RpcException

August 18, 2026

RpcException

Kind: Class

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

Part of: Microservices

`RpcException` represents errors that occur during microservice/RPC message handling. It wraps a string or object error payload, provides an `Error`-compatible message for logging, and preserves the original payload for transport-specific exception filters or serializers.

Extends: `Error`

Methods

Method	Signature	Returns
`initMessage`	`initMessage()`	`void`
`getError`	`getError()`	`string

Diagram

```
graph LR
  A[Microservice Handler] -->|throws| B[RpcException]
  B --> C[initMessage()]
  C --> D[Error.message]
  B --> E[getError()]
  E --> F[Original string or object payload]
  F --> G[RPC Exception Filter / Transport Response]
```

Usage

```
import { RpcException } from '@nestjs/microservices';

@Injectable()
export class UsersService {
  async findOne(id: string) {
    const user = await this.usersRepository.findById(id);

    if (!user) {
      throw new RpcException({
        statusCode: 404,
        message: `User ${id} was not found`,
        code: 'USER_NOT_FOUND',
      });
    }

    return user;
  }
}

// Exception filters can access the original payload.
const exception = new RpcException('Invalid request');
console.log(exception.message); // "Invalid request"
console.log(exception.getError()); // "Invalid request"
```

AI Coding Instructions

- Throw `RpcException` from microservice handlers when an error must be returned through an RPC transport rather than handled as a standard HTTP exception.
- Pass either a descriptive string or a structured object; use objects when consumers need fields such as `code`, `statusCode`, or validation details.
- Use `getError()` in custom RPC exception filters to retrieve the original payload instead of relying only on `message`.
- Avoid exposing internal stack traces, database errors, or sensitive implementation details in the exception payload.
- Ensure the selected microservice transport and exception filter serialize structured error objects consistently for downstream clients.

Relationships

- IMPORTS → `isObject`
- IMPORTS → `isString`

Used by

7 references from 7 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (7)

- `AppController` — `integration/microservices/src/app.controller.ts`:20
- `ConfigService` — `integration/microservices/src/app.module.ts`:18
- `GrpcController` — `integration/microservices/src/grpc/grpc.controller.ts`:21
- `NatsController` — `integration/microservices/src/nats/nats.controller.ts`:19
- `AppController` — `integration/microservices/src/tcp-tls/app.controller.ts`:22
- `ConfigService` — `integration/microservices/src/tcp-tls/app.module.ts`:29
- `ExceptionFilter` — `sample/03-microservices/src/common/filters/rpc-exception.filter.ts`:5