RmqContext

August 17, 2026

RmqContext

Kind: Class

Source: `packages/microservices/ctx-host/rmq.context.ts`

Part of: Microservices

`RmqContext` provides access to RabbitMQ-specific metadata during a microservice message handler invocation. It exposes the raw incoming message, the underlying AMQP channel, and the matched message pattern so handlers can inspect payloads and acknowledge or reject deliveries.

Extends: `BaseRpcContext`

Methods

Method	Signature	Returns
`getMessage`	`getMessage()`	`void`
`getChannelRef`	`getChannelRef()`	`void`
`getPattern`	`getPattern()`	`void`

Diagram

```
graph LR
  RabbitMQ[RabbitMQ Queue] --> Handler[Message Handler]
  Handler --> Context[RmqContext]
  Context --> Message[getMessage()]
  Context --> Channel[getChannelRef()]
  Context --> Pattern[getPattern()]
  Channel --> Ack[channel.ack(message)]
```

Usage

```ts
import { Controller } from '@nestjs/common';
import {
  Ctx,
  MessagePattern,
  Payload,
  RmqContext,
} from '@nestjs/microservices';

@Controller()
export class OrdersController {
  @MessagePattern('orders.created')
  handleOrderCreated(
    @Payload() order: { id: string; total: number },
    @Ctx() context: RmqContext,
  ) {
    const message = context.getMessage();
    const channel = context.getChannelRef();
    const pattern = context.getPattern();

    console.log(`Received ${pattern} for order ${order.id}`);
    console.log(`Delivery tag: ${message.fields.deliveryTag}`);

    channel.ack(message);
  }
}
```

AI Coding Instructions

- Inject `RmqContext` with `@Ctx()` in RabbitMQ message handlers decorated with `@MessagePattern()` or `@EventPattern()`.
- Use `getMessage()` when access to raw AMQP properties, fields, headers, or the delivery tag is required.
- Use `getChannelRef()` to manually call `ack()`, `nack()`, or `reject()` when the RabbitMQ transport is configured with `noAck: false`.
- Do not acknowledge a message before all required processing succeeds; use rejection or retry handling for recoverable failures.
- Use `getPattern()` for logging, tracing, or behavior that depends on the matched routing pattern.

How it works

`RmqContext` is a public, exported RabbitMQ RPC context class that extends `BaseRpcContext` with a three-element argument tuple: a message record, a channel reference, and a string pattern. [packages/microservices/ctx-host/rmq.context.ts:3-10]

Its constructor accepts that tuple and passes it to `BaseRpcContext`, which stores it as the protected, read-only `args` value. [packages/microservices/ctx-host/rmq.context.ts:8-10] [packages/microservices/ctx-host/base-rpc.context.ts:4-5]

It exposes:

- `getMessage()`, which returns tuple element `0`: the original message. [packages/microservices/ctx-host/rmq.context.ts:13-18]
- `getChannelRef()`, which returns tuple element `1`: the original RabbitMQ channel reference. [packages/microservices/ctx-host/rmq.context.ts:20-25]
- `getPattern()`, which returns tuple element `2`: the pattern string. [packages/microservices/ctx-host/rmq.context.ts:27-32]
- The inherited `getArgs()` and `getArgByIndex(index)` methods, which return the full stored tuple or an element selected by index. [packages/microservices/ctx-host/base-rpc.context.ts:7-20]

`ServerRMQ.handleMessage()` constructs this context after deserializing an incoming message. It supplies the received message object, the channel passed to the handler, and a pattern that is left unchanged when already a string or is JSON-stringified otherwise. [packages/microservices/server/server-rmq.ts:290-304] The server passes the context to event handling and to matched message handlers. [packages/microservices/server/server-rmq.ts:305-306] [packages/microservices/server/server-rmq.ts:328-345]

The class contains no visible argument validation, error throwing, message acknowledgement, channel operation, or mutation beyond storing the supplied tuple through its base-class constructor. [packages/microservices/ctx-host/rmq.context.ts:8-32] [packages/microservices/ctx-host/base-rpc.context.ts:4-20]

Used by

3 references from 3 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (3)

- `RMQFanoutExchangeConsumerController` — `integration/microservices/src/rmq/fanout-exchange-consumer-rmq.controller.ts`:4
- `RMQController` — `integration/microservices/src/rmq/rmq.controller.ts`:16
- `RMQTopicExchangeController` — `integration/microservices/src/rmq/topic-exchange-rmq.controller.ts`:12