ServerRMQ

August 17, 2026

ServerRMQ

Kind: Class

Source: `packages/microservices/server/server-rmq.ts`

Part of: Microservices

`ServerRMQ` is the RabbitMQ transport server used by the microservices package to consume queue messages and dispatch them to registered message or event handlers. It manages the RabbitMQ connection and channel lifecycle, configures queue consumption, sends replies for request-response patterns, and exposes the underlying channel through `unwrap()` when needed.

Extends: `Server`

Methods

Method	Signature	Returns
`listen`	`listen(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`Promise<void>`
`close`	`close()`	`Promise<void>`
`start`	`start(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`void`
`createClient`	`createClient()`	`T`
`setupChannel`	`setupChannel(channel: Channel, callback: Function)`	`void`
`handleMessage`	`handleMessage(message: Record<string, any>, channel: any)`	`Promise<void>`
`handleEvent`	`handleEvent(pattern: string, packet: ReadPacket, context:`	`Promise<any>`

Method	Signature	Returns
	`RmqContext)`	
`sendMessage`	`sendMessage(message: T, replyTo: any, correlationId: string, context: RmqContext)`	`void`
`unwrap`	`unwrap()`	`T`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`getHandlerByPattern`	`getHandlerByPattern(pattern: string)`	`MessageHandler
`initializeSerializer`	`initializeSerializer(options: RmqOptions['options'])`	`void`
`initializeWildcardHandlersIfExist`	`initializeWildcardHandlersIfExist()`	`void`

Properties

Property	Type
`transportId`	`TransportId`
`server`	`AmqpConnectionManager
`channel`	`ChannelWrapper
`connectionAttempts`	`any`
`urls`	`string[]
`queue`	`string`
`noAck`	`boolean`
`queueOptions`	`any`
`wildcardHandlers`	`any`
`pendingEventListeners`	`Array<{ event: keyof RmqEvents; callback: RmqEvents[keyof RmqEvents]; }>`

Where it refuses work

- `ServerRMQ` stops the work with `Error` when `!this.server` — “Not initialized. Please call the "listen"/"startAllMicroservices" method before accessing…”.
- `ServerRMQ` stops the work with an early return when `this.channel`.

- `ServerRMQ` stops the work with an early return when `maxConnectionAttempts === INFINITE_CONNECTION_ATTEMPTS || isReconnecting`.
- `ServerRMQ` stops the work with an early return when `isNil(message)`.
- `ServerRMQ` stops the work with an early return when `isUndefined((packet as IncomingRequest).id)`.
- `ServerRMQ` stops the work with an early return when `!this.options.wildcards`.

When something fails

- `ServerRMQ` handles failure in 2 places: it logs it and continues in 1, and turns it into a return value in 1.

Diagram

```
graph LR
  A[RabbitMQ Broker] --> B[ServerRMQ]
  B --> C[Create Client Connection]
  C --> D[Setup Channel & Queue]
  D --> E[Consume Messages]
  E --> F{Message Type}
  F -->|Request/Response| G[handleMessage]
  F -->|Event| H[handleEvent]
  G --> I[sendMessage Reply]
  H --> J[Application Event Handler]
  B --> K[unwrap Channel]
```

Usage

```
import { Controller } from '@nestjs/common';
import { MessagePattern, NestFactory } from '@nestjs/microservices';
import { AppModule } from './app.module';
import { Transport } from '@nestjs/microservices';

@Controller()
class MathController {
  @MessagePattern({ cmd: 'sum' })
  sum(numbers: number[]) {
    return numbers.reduce((total, value) => total + value, 0);
  }
}

async function bootstrap() {
  const app = await NestFactory.createMicroservice(AppModule, {
    transport: Transport.RMQ,
```

```
    options: {
      urls: ['amqp://guest:guest@localhost:5672'],
      queue: 'math_queue',
      queueOptions: {
        durable: true,
      },
    },
  });

  // Internally starts a ServerRMQ instance.
  await app.listen();
}

bootstrap();
```

AI Coding Instructions

- Configure RabbitMQ through `Transport.RMQ` options rather than manually managing connections unless implementing a custom transport strategy.
- Keep queue names, broker URLs, durability settings, and acknowledgement behavior consistent between producers and consumers.
- Use message patterns for request-response communication and event patterns for fire-and-forget events; `ServerRMQ` handles them through separate message flows.
- Call `close()` during application shutdown so RabbitMQ channels and connections are released cleanly.
- Use `unwrap()` only when direct access to the RabbitMQ channel is required, such as for advanced acknowledgements or broker-specific operations.

How it works

`ServerRMQ` is the RabbitMQ microservice server transport. It extends the generic `Server` base class with RMQ event/status types and identifies itself as `Transport.RMQ`; `ServerFactory` constructs it when the selected transport is RMQ.
packages/microservices/server/server-rmq.ts:61-62
packages/microservices/server/server-factory.ts:20-40

Relationships

- IMPORTS → `isNil`
- IMPORTS → `isString`
- IMPORTS → `isUndefined`