ServerRedis

August 19, 2026

ServerRedis

Kind: Class

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

Part of: Microservices

`ServerRedis` is the Redis transport server implementation for NestJS microservices. It creates Redis publisher/subscriber clients, subscribes to message patterns, dispatches incoming requests to registered handlers, and publishes responses or events back through Redis.

Extends: `Server`

Methods

Method	Signature	Returns
`listen`	`listen(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`void`
`start`	`start(callback: () => void)`	`void`
`bindEvents`	`bindEvents(subClient: Redis, pubClient: Redis)`	`void`
`close`	`close()`	`void`
`createRedisClient`	`createRedisClient()`	`Redis`
`getMessageHandler`	`getMessageHandler(pub: Redis)`	`void`
`handleMessage`	`handleMessage(channel: string, buffer: string, pub: Redis, pattern: string)`	`void`
`getPublisher`	`getPublisher(pub: Redis, pattern: any, id: string, ctx: RedisContext)`	`void`

Method	Signature	Returns
`parseMessage`	`parseMessage(content: any)`	`Record<string, any>`
`getRequestPattern`	`getRequestPattern(pattern: string)`	`string`
`getReplyPattern`	`getReplyPattern(pattern: string)`	`string`
`registerErrorListener`	`registerErrorListener(client: any)`	`void`
`registerReconnectListener`	`registerReconnectListener(client: { on: (event: string, fn: () => void) => void; })`	`void`
`registerReadyListener`	`registerReadyListener(client: { on: (event: string, fn: () => void) => void; })`	`void`
`registerEndListener`	`registerEndListener(client: { on: (event: string, fn: () => void) => void; })`	`void`
`getClientOptions`	`getClientOptions()`	`Partial<RedisOptions['options']>`
`createRetryStrategy`	`createRetryStrategy(times: number)`	`undefined
`unwrap`	`unwrap()`	`T`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`

Properties

Property	Type
`transportId`	`TransportId`
`subClient`	`Redis`
`pubClient`	`Redis`
`isManuallyClosed`	`any`
`wasInitialConnectionSuccessful`	`any`
`pendingEventListeners`	`Array<{ event: keyof RedisEvents; callback: RedisEvents[keyof RedisEvents]; }>`

Where it refuses work

- `ServerRedis` stops the work with `Error` when `!this.pubClient || !this.subClient` — “Not initialized. Please call the "listen"/"startAllMicroservices" method before accessing…”.
- `ServerRedis` stops the work with an early return when `this.isManuallyClosed`, in 3 places.
- `ServerRedis` stops the work with an early return when `isUndefined((packet as IncomingRequest).id)`.

When something fails

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

Diagram

```
graph LR
  Client[Redis Microservice Client] -->|Publish request/event| Redis[(Redis)]
  Redis -->|Subscribed channel| ServerRedis[ServerRedis]
  ServerRedis -->|Parse message| Handler[Registered Message Handler]
  Handler -->|Response| ServerRedis
  ServerRedis -->|Publish response| Redis
  Redis --> Client
```

AI Coding Instructions

- Register message handlers with `addHandler()` before calling `listen()` so Redis subscriptions can route incoming patterns correctly.
- Keep request patterns stable and serializable; `getRequestPattern()` uses the pattern to determine the Redis subscription channel.
- Ensure Redis connection options match the deployed Redis instance, including authentication, TLS, retry behavior, and database selection where required.
- Always call `close()` during application shutdown to disconnect both publisher and subscriber Redis clients cleanly.
- Avoid placing long-running synchronous work in handlers; Redis message processing should delegate expensive work to asynchronous services or workers.

Relationships

- IMPORTS → `isUndefined`