ClientRedis

August 17, 2026

ClientRedis

Kind: Class

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

Part of: Microservices

`ClientRedis` is a Redis-backed microservice client that publishes request messages and listens for correlated reply messages. It manages the Redis connection lifecycle—including connection, reconnection, readiness, errors, and shutdown—while applying request and reply channel naming conventions.

Extends: `ClientProxy`

Methods

Method	Signature	Returns
`getRequestPattern`	`getRequestPattern(pattern: string)`	`string`
`getReplyPattern`	`getReplyPattern(pattern: string)`	`string`
`close`	`close()`	`void`
`connect`	`connect()`	`Promise<any>`
`createClient`	`createClient()`	`Redis`
`registerErrorListener`	`registerErrorListener(client: Redis)`	`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`

Method	Signature	Returns
	`=> void; })`	
`handleClose`	`handleClose()`	`void`
`getClientOptions`	`getClientOptions()`	`Partial<RedisOptions['options']>`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`unwrap`	`unwrap()`	`T`
`createRetryStrategy`	`createRetryStrategy(times: number)`	`undefined
`createResponseCallback`	`createResponseCallback()`	`( channel: string, buffer: string, ) => Promise<void>`
`publish`	`publish(partialPacket: ReadPacket, callback: (packet: WritePacket) => any)`	`() => void`
`dispatchEvent`	`dispatchEvent(packet: ReadPacket)`	`Promise<any>`
`unsubscribeFromChannel`	`unsubscribeFromChannel(channel: string)`	`void`

Properties

Property	Type
`logger`	`any`
`subscriptionsCount`	`any`
`pubClient`	`Redis`
`subClient`	`Redis`
`connectionPromise`	`Promise<any>`
`isManuallyClosed`	`any`
`wasInitialConnectionSuccessful`	`any`
`pendingEventListeners`	`Array<{ event: keyof RedisEvents; callback: RedisEvents[keyof RedisEvents]; }>`

Where it refuses work

- `ClientRedis` stops the work with `Error` when `!this.pubClient || !this.subClient` — “Not initialized. Please call the "connect" method first.”.

- `ClientRedis` stops the work with an early return when `this.isManuallyClosed`, in 3 places.
- `ClientRedis` stops the work with an early return when `this.pubClient && this.subClient`.
- `ClientRedis` stops the work with an early return when `isDisposed || err`.

When something fails

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

Diagram

```
graph LR
  App[Application Service] --> ClientRedis[ClientRedis]
  ClientRedis -->|publish: pattern_request| Redis[(Redis)]
  Redis -->|subscribe: pattern_response| ClientRedis
  ClientRedis -->|response/error| App

  ClientRedis --> Connection[Redis Client Connection]
  Connection --> Ready[Ready Listener]
  Connection --> Reconnect[Reconnect Listener]
  Connection --> Errors[Error Listener]
  Connection --> End[End Listener]
```

AI Coding Instructions

- Use `connect()` before sending messages when the client lifecycle is managed manually; reuse a connected client instead of creating one per request.
- Keep request patterns aligned with the receiving Redis microservice transport configuration; `ClientRedis` derives separate request and reply channels from the pattern.
- Use `send()` for request-response flows and ensure the downstream service returns a response on the corresponding reply channel.
- Call `close()` during application shutdown so Redis subscriptions and connections are released cleanly.
- Do not bypass lifecycle listeners when changing connection behavior; error, reconnect, ready, and end handlers are required for reliable Redis transport operation.

Relationships

- IMPORTS → `Logger`
- IMPORTS → `loadPackage`