ServerMqtt

August 17, 2026

ServerMqtt

Kind: Class

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

Part of: Microservices

`ServerMqtt` is NestJS's MQTT transport server implementation for microservices. It creates and manages an MQTT client connection, subscribes to message patterns, parses incoming packets, and dispatches them to registered message handlers. It also provides publishing support for request-response messaging and manages connection shutdown.

Extends: `Server`

Methods

Method	Signature	Returns
`listen`	`listen(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`void`
`start`	`start(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`void`
`bindEvents`	`bindEvents(mqttClient: MqttClient)`	`void`
`close`	`close()`	`void`
`createMqttClient`	`createMqttClient()`	`MqttClient`
`getMessageHandler`	`getMessageHandler(pub: MqttClient)`	`void`
`handleMessage`	`handleMessage(channel: string, buffer: Buffer, pub: MqttClient, originalPacket: Record<string, any>)`	`Promise<any>`
`getPublisher`	`getPublisher(client: MqttClient, context: MqttContext, id: string)`	`any`

Method	Signature	Returns
`parseMessage`	`parseMessage(content: any)`	`ReadPacket & PacketId`
`matchMqttPattern`	`matchMqttPattern(pattern: string, topic: string)`	`void`
`getHandlerByPattern`	`getHandlerByPattern(pattern: string)`	`MessageHandler
`removeHandlerKeySharedPrefix`	`removeHandlerKeySharedPrefix(handlerKey: string)`	`void`
`getRequestPattern`	`getRequestPattern(pattern: string)`	`string`
`getReplyPattern`	`getReplyPattern(pattern: string)`	`string`
`registerErrorListener`	`registerErrorListener(client: MqttClient)`	`void`
`registerReconnectListener`	`registerReconnectListener(client: MqttClient)`	`void`
`registerDisconnectListener`	`registerDisconnectListener(client: MqttClient)`	`void`
`registerCloseListener`	`registerCloseListener(client: MqttClient)`	`void`
`registerConnectListener`	`registerConnectListener(client: MqttClient)`	`void`
`unwrap`	`unwrap()`	`T`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`initializeSerializer`	`initializeSerializer(options: MqttOptions['options'])`	`void`

Properties

Property	Type
`transportId`	`TransportId`
`url`	`string`
`mqttClient`	`MqttClient`
`pendingEventListeners`	`Array<{ event: keyof MqttEvents; callback: MqttEvents[keyof MqttEvents]; }>`

Where it refuses work

- `ServerMqtt` stops the work with `Error` when `!this.mqttClient` — “Not initialized. Please call the "listen"/"startAllMicroservices" method before accessing…”.
- `ServerMqtt` stops the work with an early return when `isUndefined((packet as IncomingRequest).id)`.
- `ServerMqtt` stops the work with an early return when `!currentTopic && currentPattern !== MQTT_WILDCARD_ALL`.
- `ServerMqtt` stops the work with an early return when `patternChar === MQTT_WILDCARD_ALL`.
- `ServerMqtt` stops the work with an early return when `patternChar !== MQTT_WILDCARD_SINGLE && currentPattern !== currentTopic`.
- `ServerMqtt` stops the work with an early return when `this.messageHandlers.has(route)`.

When something fails

- `ServerMqtt` 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[MQTT Broker] --> B[ServerMqtt]
  B --> C[MQTT Client]
  C --> D[bindEvents]
  D --> E[Incoming MQTT Packet]
  E --> F[parseMessage]
  F --> G[matchMqttPattern]
  G --> H[getMessageHandler]
  H --> I[handleMessage]
  I --> J[getPublisher]
  J --> A
```

Usage

```
import { Controller } from '@nestjs/common';
import { NestFactory } from '@nestjs/core';
import {
  MessagePattern,
  ServerMqtt,
} from '@nestjs/microservices';
```

```
import { AppModule } from './app.module';

@Controller()
class DeviceController {
  @MessagePattern('devices/+/temperature')
  handleTemperature(payload: { value: number; unit: string }) {
    return {
      received: true,
      temperature: payload.value,
    };
  }
}

async function bootstrap() {
  const mqttServer = new ServerMqtt({
    url: 'mqtt://localhost:1883',
  });

  const app = await NestFactory.createMicroservice(AppModule, {
    strategy: mqttServer,
  });

  await app.listen();
}

bootstrap();
```

AI Coding Instructions

- Keep MQTT topic patterns aligned with `@MessagePattern()` handlers; wildcard topics must be compatible with MQTT matching rules.
- Use `ServerMqtt` through NestJS microservice configuration rather than manually calling internal methods such as `bindEvents()` or `handleMessage()`.
- Ensure the MQTT broker URL and connection options are supplied through the transport options, including authentication or TLS settings when required.
- Preserve packet parsing and publisher behavior when modifying request-response flows, as MQTT responses depend on packet identifiers and reply topics.
- Always allow the NestJS application lifecycle to call `close()` so the MQTT client disconnects cleanly.

Relationships

- IMPORTS → `isObject`
- IMPORTS → `isUndefined`