TcpContext

August 17, 2026

TcpContext

Kind: Class

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

Part of: Microservices

`TcpContext` provides TCP-specific metadata to NestJS microservice message handlers. It exposes the underlying client socket and the pattern associated with the incoming message, allowing handlers to inspect connection details when needed.

Extends: `BaseRpcContext`

Methods

Method	Signature	Returns
`getSocketRef`	`getSocketRef()`	`void`
`getPattern`	`getPattern()`	`void`

Diagram

```
graph LR
  Client[TCP Client] --> Server[NestJS TCP Server]
  Server --> Handler[Message Handler]
  Handler --> Context[TcpContext]
  Context --> Socket[getSocketRef()]
  Context --> Pattern[getPattern()]
```

Usage

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

```
@Controller()
export class MathController {
  @MessagePattern('sum')
  sum(
    data: number[],
    @Ctx() context: TcpContext,
  ): number {
    const socket = context.getSocketRef();
    const pattern = context.getPattern();

    console.log(`Received "${pattern}" from ${socket.remoteAddress}`);

    return data.reduce((total, value) => total + value, 0);
  }
}
```

AI Coding Instructions

- Use `TcpContext` only in handlers invoked through the NestJS TCP microservice transport.
- Inject the context with `@Ctx() context: TcpContext` alongside `@MessagePattern()` handlers.
- Use `getSocketRef()` for connection-level metadata such as remote address or port; avoid mutating or closing the socket unless the transport lifecycle requires it.
- Use `getPattern()` when logging, tracing, or routing behavior based on the incoming message pattern.

How it works

`TcpContext` is a public context class for TCP message handling. It extends `BaseRpcContext` with an argument tuple whose first item is a `TcpSocket` and whose second item is a string pattern. tcp.context.ts:4-11

- Construct it with `[socket, pattern]`; its constructor passes that tuple directly to `BaseRpcContext`. tcp.context.ts:4-12
- `getSocketRef()` returns the tuple's first item—the same `TcpSocket` reference supplied at construction. tcp.context.ts:17-19
- `getPattern()` returns the tuple's second item—the supplied pattern string. tcp.context.ts:24-26
- Through `BaseRpcContext`, it also inherits `getArgs()` to return the complete tuple and `getArgByIndex(index)` to return an item at the requested index. base-rpc.context.ts:4-20

`ServerTCP.handleMessage()` deserializes an incoming message, converts a non-string packet pattern with `JSON.stringify`, constructs `TcpContext` from `[socket, pattern]`, and passes that context to either event handling or the matched request handler. server-tcp.ts:93-101 server-tcp.ts:114-120

There is no runtime validation, error throwing, socket I/O, or mutation in `TcpContext` itself; its constructor stores the supplied tuple via the base class, and its declared methods only read tuple elements. tcp.context.ts:9-26