ServerGrpc

August 18, 2026

ServerGrpc

Kind: Class

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

Part of: Microservices

`ServerGrpc` is the NestJS microservices transport server responsible for exposing message handlers as gRPC service methods. It initializes the gRPC server, binds configured services and RPC handlers, applies keepalive options, and translates incoming gRPC requests into Nest message handler calls.

Extends: `Server`

Methods

Method	Signature	Returns
`listen`	`listen(callback: (err?: unknown, ...optionalParams: unknown[]) => void)`	`void`
`start`	`start(callback: () => void)`	`void`
`bindEvents`	`bindEvents()`	`void`
`getServiceNames`	`getServiceNames(grpcPkg: any)`	`{ name: string; service: any }[]`
`getKeepaliveOptions`	`getKeepaliveOptions()`	`void`
`createService`	`createService(grpcService: any, name: string)`	`void`
`getMessageHandler`	`getMessageHandler(serviceName: string, methodName: string, streaming: GrpcMethodStreamingType, grpcMethod: { path?: string })`	`MessageHandler`

Method	Signature	Returns
`createPattern`	`createPattern(service: string, methodName: string, streaming: GrpcMethodStreamingType)`	`string`
`createServiceMethod`	`createServiceMethod(methodHandler: Function, protoNativeHandler: any, streamType: GrpcMethodStreamingType)`	`Function`
`createUnaryServiceMethod`	`createUnaryServiceMethod(methodHandler: Function)`	`Function`
`createStreamServiceMethod`	`createStreamServiceMethod(methodHandler: Function)`	`Function`
`unwrap`	`unwrap()`	`T`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`createRequestStreamMethod`	`createRequestStreamMethod(methodHandler: Function, isResponseStream: boolean)`	`void`
`createStreamCallMethod`	`createStreamCallMethod(methodHandler: Function, isResponseStream: boolean)`	`void`
`close`	`close()`	`Promise<void>`
`deserialize`	`deserialize(obj: any)`	`any`
`addHandler`	`addHandler(pattern: unknown, callback: MessageHandler, isEventHandler: undefined)`	`void`
`createClient`	`createClient()`	`void`
`lookupPackage`	`lookupPackage(root: any, packageName: string)`	`void`
`loadProto`	`loadProto()`	`any`

Properties

Property	Type
`transportId`	`TransportId`
`url`	`string`
`grpcClient`	`GrpcServer`

Where it refuses work

- `ServerGrpc` stops the work with an early return when `hasDrained`, in 2 places.
- `ServerGrpc` stops the work with an early return when `!isObject(this.options.keepalive)`.
- `ServerGrpc` stops the work with an early return when `streamType === GrpcMethodStreamingType.PT_STREAMING`.
- `ServerGrpc` stops the work with an early return when `!writing`.
- `ServerGrpc` stops the work with an early return when `!isObject(grpcDefinition)`.
- `ServerGrpc` stops the work with an early return when `name.length === 0`.

When something fails

- `ServerGrpc` handles failure in 3 places: it logs it and continues in 1, turns it into a return value in 1, and lets it reach the caller in 1.

Diagram

```
graph LR
  Client[gRPC Client] -->|RPC request| GrpcServer[ServerGrpc]
  GrpcServer -->|bindEvents| Services[gRPC Service Definitions]
  Services -->|createServiceMethod| Handlers[Nest Message Handlers]
  Handlers -->|getMessageHandler| Controllers[Controller Methods]
  Controllers -->|response / stream| Client
```

Usage

```
import { NestFactory } from '@nestjs/core';
import { MicroserviceOptions, Transport } from '@nestjs/microservices';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.createMicroservice<MicroserviceOptions>(
    AppModule,
    {
      transport: Transport.GRPC,
      options: {
        package: 'users',
        protoPath: 'proto/users.proto',
        url: '0.0.0.0:50051',
      },
    },
  );
```

```
  // Internally creates and starts a ServerGrpc instance.
  await app.listen();
}

bootstrap();
```

AI Coding Instructions

- Configure `ServerGrpc` through Nest's `Transport.GRPC` microservice options rather than manually instantiating it in application code.
- Ensure `package`, `protoPath`, and service names match the `.proto` definition exactly; mismatches prevent handlers from being bound.
- Use controller methods decorated with `@GrpcMethod()` or `@GrpcStreamMethod()` so `bindEvents()` can map RPC methods to Nest message handlers.
- Keep unary and streaming RPC implementations distinct; `createUnaryServiceMethod()` handles request/response calls, while streaming methods require stream-aware handlers.
- When changing connection behavior, preserve the expected gRPC keepalive option format used by `getKeepaliveOptions()`.

How it works

`ServerGrpc`

`ServerGrpc` is the gRPC transport implementation of the abstract `Server` base class. It identifies itself as `Transport.GRPC` and stores the bound gRPC server instance in `grpcClient`. packages/microservices/server/server-grpc.ts:59-63

Relationships

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