ClientGrpcProxy

August 18, 2026

ClientGrpcProxy

Kind: Class

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

Part of: Microservices

`ClientGrpcProxy` is the NestJS microservices client implementation for communicating with gRPC services. It loads protobuf definitions, creates gRPC client instances for configured services, and exposes service methods as RxJS `Observable` streams for unary, server-streaming, client-streaming, and bidirectional-streaming RPCs.

Extends: `ClientProxy`

Implements: `ClientGrpc`

Methods

Method	Signature	Returns
`getService`	`getService(name: string)`	`T`
`getClientByServiceName`	`getClientByServiceName(name: string)`	`T`
`createClientByServiceName`	`createClientByServiceName(name: string)`	`void`
`getKeepaliveOptions`	`getKeepaliveOptions()`	`void`
`createServiceMethod`	`createServiceMethod(client: any, methodName: string)`	`(...args: unknown[]) => Observable<unknown>`
`createStreamServiceMethod`	`createStreamServiceMethod(client: unknown, methodName: string)`	`(...args: any[]) => Observable<any>`
`createUnaryServiceMethod`	`createUnaryServiceMethod(client: any, methodName: string)`	`(...args: any[]) => Observable<any>`

Method	Signature	Returns
`createClients`	`createClients()`	`any[]`
`loadProto`	`loadProto()`	`any`
`lookupPackage`	`lookupPackage(root: any, packageName: string)`	`void`
`close`	`close()`	`void`
`connect`	`connect()`	`Promise<any>`
`send`	`send(pattern: any, data: TInput)`	`Observable<TResult>`
`getClient`	`getClient(name: string)`	`any`
`publish`	`publish(packet: any, callback: (packet: any) => any)`	`any`
`dispatchEvent`	`dispatchEvent(packet: any)`	`Promise<any>`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`unwrap`	`unwrap()`	`T`

Properties

Property	Type
`logger`	`any`
`clients`	`any`
`url`	`string`
`grpcClients`	`GrpcClient[]`

Where it refuses work

- `ClientGrpcProxy` stops the work with `InvalidGrpcServiceException` when `!clientRef`, in 2 places.
- `ClientGrpcProxy` stops the work with an early return when `isClientCanceled`, in 2 places.
- `ClientGrpcProxy` stops the work with an early return when `!isObject(this.options.keepalive)`.
- `ClientGrpcProxy` stops the work with an early return when `call.finished`.

- `ClientGrpcProxy` stops the work with an early return when `isRequestStream && isUpstreamSubject`.
- `ClientGrpcProxy` stops the work with an early return when `error`.

When something fails

- `ClientGrpcProxy` handles failure in 1 place: it lets it reach the caller in all 1.

Diagram

```
graph LR
  A[Application Service] --> B[ClientGrpcProxy]
  B --> C[loadProto]
  C --> D[Proto Definition]
  B --> E[createClients]
  E --> F[gRPC Service Client]
  A --> G[getService serviceName]
  G --> F
  F --> H[Unary / Streaming RPC Method]
  H --> I[Observable Response]
```

Usage

```
import { Injectable, OnModuleInit } from '@nestjs/common';
import { ClientGrpc, ClientProxyFactory, Transport } from '@nestjs/microservices';
import { Observable } from 'rxjs';

interface UsersService {
  findOne(data: { id: string }): Observable<{ id: string; name: string }>;
}

@Injectable()
export class UsersClient implements OnModuleInit {
  private usersService: UsersService;

  private readonly client: ClientGrpc = ClientProxyFactory.create({
    transport: Transport.GRPC,
    options: {
      package: 'users',
      protoPath: 'proto/users.proto',
      url: 'localhost:5000',
    },
  });

  onModuleInit() {
    // ClientGrpcProxy resolves the named gRPC service from the loaded proto.
```

```
    this.usersService = this.client.getService<UsersService>('UsersService');
  }

  findUser(id: string) {
    return this.usersService.findOne({ id });
  }
}
```

AI Coding Instructions

- Use `getService<T>(serviceName)` after application initialization to retrieve a typed gRPC service client.
- Configure `package`, `protoPath`, and `url` consistently with the server-side gRPC transport configuration.
- Treat RPC method results as RxJS `Observable` values; use operators or `firstValueFrom()` rather than assuming promises.
- Preserve protobuf service and method names exactly, including the service name passed to `getService()`.
- Use the configured client factory or dependency injection instead of manually calling internal methods such as `loadProto()` or `createClients()`.

Relationships

- IMPORTS → `Logger`
- IMPORTS → `loadPackage`
- IMPORTS → `isFunction`
- IMPORTS → `isObject`

Used by

1 reference from 1 file. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (1)

- `GrpcController` — `integration/microservices/src/grpc/grpc.controller.ts`:21