Core Concepts, architecture & lifecycle

Overview

* Service definition

⇒ gRPC lets you define four kind of service method:

① Unary RPCs

- Client sends a single request to the server and gets a single response back.
- Just like a normal function call.

rpc SayHello (HelloRequest) return (HelloResponse);

② Server streaming RPCs

- Client sends a request to the server & gets a stream to read a sequence of messages back.
- gRPC guarantees message ordering within an individual RPC call.

rpc LotsOfReply (HelloRequest) return (stream HelloResponse)

(3) Client streaming RPCs

→ Client writes a sequence of messages & sends them to Server.

→ Once the client has finished writing the messages, it waits for the server to read them & return its response.

→ Again gRPC guarantees message ordering within an individual RPC call.

rpc LotsOfGreeting (stream HelloRequest)
returns (HelloResponse)

(4) Bidirectional Streaming RPCs

→ Both sides send a sequence of messages using a read-write stream.

rpc BidiHello (stream HelloRequest) returns (stream HelloResponse)

* Using the API

⇒ Starting from a service definition in a .proto file, gRPC provides protocol buffer compiler plugins that generates client-and Server-side code.

→ gRPC users typically call these APIs on the client side & implement the corresponding API on the server side.

⇒ On the Server Side:

- Server implements the methods declared by the service & runs gRPC server to handle client calls.
- gRPC infrastructure decodes incoming requests, executes service methods, and encodes service responses.

⇒ On the Client Side:

- Client has a local object known as stub that implements the same method as the service.
- The client can then just call those methods on the local object, wrapping the parameters for the call in the appropriate protocol buffer message type.
- gRPC looks after sending the request(s) to the server and returning the server's protocol buffer response(s).

* Synchronous vs Asynchronous

⇒ The gRPC programming API in most language comes in both synchronous & asynchronous flavours.

RPC life Cycle

★ Unary RPC

1. Once the Client calls a stub method, the Server is notified that the RPC has been invoked with the client's metadata for this call, the method name, and the specified deadline if applicable.

2. The Server can then either send back its own initial metadata (before any response) straight away, or wait for the client's request message.

   ↳ Which happens first is application Specific.

3. Once the server has the client's request message, it does whatever work is necessary to create & populate a response.

   ↳ The response is then returned to the client together with status details & optional trailing metadata.

4. If the response status is OK, then client gets the response, which completes the call on the client side.

★ Deadlines / Timeouts

⇒ gRPC allows clients to specify how long they are willing to wait for an RPC to complete before the RPC is terminated with a DEADLINE_EXCEEDED error.

⇒ Servers can query to see if a particular RPC has timed out, or how much time is left to complete RPC.

★ Cancelling an RPC

⇒ Either the client or the Server can cancel an RPC at any time.

⇒ A Cancellation terminates the RPC immediately So no further work is done.

⇒ Changes made before Cancellation are not rolled back.

★ Metadata

⇒ Metadata is information about a particular RPC call in the form of a list of key-value pairs.

{ Example authentication details }

★ Channels

⇒ A gRPC channel provides a connection to a gRPC server on a specified host and port.

↳ It is used when creating a client stub.

⇒ A channel has state including connected & idle.