MidEng GK861 gRPC Framework

author: Kacper Bohaczyk
version 27-02-2024

Tutorial: https://intuting.medium.com/implement-grpc-service-using-java-gradle-7a54258b60b8

Install protocol compiler using

```
$ apt install -y protobuf-compiler
```

Java

Most important code snippets from **HelloWorldServer***

```java

public class HelloWorldServer {
    private static final int PORT = 50051;
    private Server server;

    public void start() throws IOException {
        server = ServerBuilder.forPort(PORT)
                .addService(new HelloWorldServiceImpl())
                .build()
                .start();
    }

// startet den Server

    public void blockUntilShutdown() throws InterruptedException {
        if (server == null) {
            return;
        }

        server.awaitTermination();
    }

// Blockiert den Port bis der Server terminiert wird

```

```
    public static void main(String[] args)
            throws InterruptedException, IOException {
        HelloWorldServer server = new HelloWorldServer();
        server.start();
        server.blockUntilShutdown();
    }
}
```

Most important code snippets from **HelloWorldServiceImpl**

```
import io.grpc.stub.StreamObserver;

public class HelloWorldServiceImpl extends
HelloWorldServiceGrpc.HelloWorldServiceImplBase {

    @Override
    public void hello(
            Hello.HelloRequest request,
            StreamObserver<Hello.HelloResponse> responseObserver) {
        System.out.println(
                "Handling hello endpoint: " + request.toString());


        String text = request.getText() + " World";
        Hello.HelloResponse response =
                Hello.HelloResponse.newBuilder()
                        .setText(text).build();

        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

}
```

Gibt eine Nachricht zurück an den Client

Most important code snippets from **Client**

```

public class HelloWorldClient {

private final ManagedChannel channel;

private final HelloWorldServiceGrpc.HelloWorldServiceBlockingStub
blockingStub;
```

```
public HelloWorldClient(String host, int port) {

channel = ManagedChannelBuilder.forAddress(host, port)

.usePlaintext()

.build();

blockingStub = HelloWorldServiceGrpc.newBlockingStub(channel);

}

// Erstellt den eine verbindung zum Server

public void shutdown() throws InterruptedException {

channel.shutdown().awaitTermination(5,
java.util.concurrent.TimeUnit.SECONDS);

}

// Beendet die Connection

public void sayHello(String name) {

Hello.HelloRequest request =
Hello.HelloRequest.newBuilder().setText(name).build();

Hello.HelloResponse response = blockingStub.hello(request);

System.out.println("Response from server: " + response.getText());

}

  // Sendet eine Nachricht an den Server

public static void main(String[] args) throws InterruptedException {

HelloWorldClient client = new HelloWorldClient("localhost", 50051);

try {

String name = "User";
client.sayHello(name);

} finally {

client.shutdown();

}

}

}
```

! Notes!
Change the Version of java to 20 under the project structure
Change the gradle Version to 8.4 under gradle-wrapper.properties
Marke the source directories as source if not done automaticly

Most important code snippets from **python**

```
def say_hello(stub, name):

request = hello_pb2.HelloRequest(text=name)

response = stub.hello(request)

print("Response from server:", response.text)



def main():

channel = grpc.insecure_channel('localhost:50051')

stub = hello_pb2_grpc.HelloWorldServiceStub(channel)

try:

name = "Kacper_python"

say_hello(stub, name)

finally:

channel.close()
```

```
if __name__ == '__main__':

main()
```

Generated Files "hello_pb2_grpc.py" und "hello_pb2.py"

```
python3 -m grpc_tools.protoc -I/path/to/proto/directory --python_out=. --
grpc_python_out=. /path/to/proto/directory/hello.proto
```

path/to/proto/directory mit dem path auswähseln oder
in der proto directory ausführen

```
python3 -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=.
hello.proto
```

!!Notes!! python zu python3 im code wähseln.

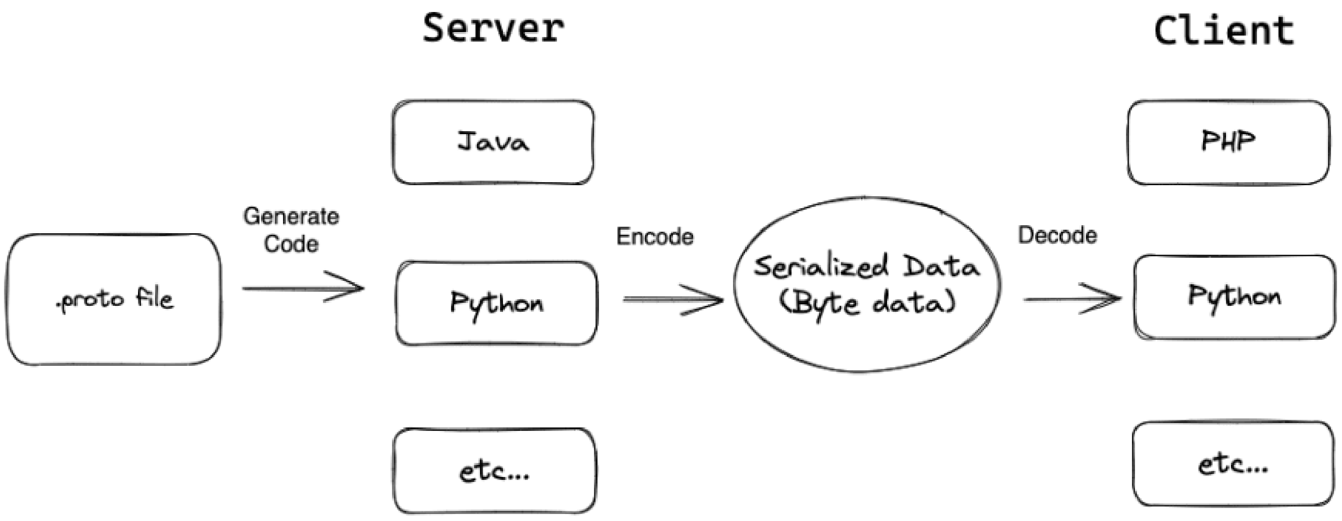
Questions

- What is gRPC and why does it work accross languages and platforms?
  gRPC is a Remote Procedure Call- framework developed by Google. It uses HTTP/2 for transport, Protocol Buffers as the interface description language, and provides features such as authentication, load balancing, and flow control. It works across languages and platforms because it uses Protocol Buffers for serialization and deserialization of structured data, which is language-agnostic, and HTTP/2 for transport, which is supported by most modern programming languages and platforms.
- Describe the RPC life cycle starting with the RPC client?

The RPC life cycle begins with the client invoking a remote method. This is done by creating a stub, a local representation of the remote service. The stub provides a method with the same signature as the remote method. The client can call this method as if it were a local function. The stub serializes the method parameters into a binary format using Protocol Buffers and sends the method parameters to the server over an HTTP/2 connection

Schau https://grpc.io/docs/what-is-grpc/core-concepts/

- Describe the workflow of Protocol Buffers?
  Define the message structure in a .proto file. Compile the .proto file and get a language-specific code for serialization and deserialization. Use the generated code in your application to serialize and deserialize data according to the defined message structure.



- What are the benefits of using protocol buffers?
  Smaller data size
  Fast encoding and decoding
  Automatically generate code
  Support a variety of object oriented programming languages
- When is the use of protocol not recommended? of the serialized data
  For very simple data structures
  In situations where human readability is more important than efficiency.
  For non-object-oriented languages
- List 3 different data types that can be used with protocol buffers?
  Integer
  String
  Double

Fehler, die aufgetreten sind:

1. Verwendung der falschen Gradle-Version (Kotlin statt Groovy).

   Lösung: Stellen Sie sicher, dass die richtige Gradle-Version verwendet wird.
2. Verwendung der falschen Version von gRPC (1.29.0 statt 1.61.0).

   Lösung: Überprüfen Sie die neueste Version von gRPC auf mvnrepository.com und aktualisieren Sie die Abhängigkeit entsprechend.

3. Ändern des Kompilierungsschritts von `compile` zu `implementation` in der `build.gradle`-Datei.

   Lösung: Ändern Sie den Kompilierungsschritt in der `build.gradle`-Datei von `compile` zu `implementation`.
4. Die Build-Verzeichnisse wurden nicht als Quellverzeichnis markiert.

   Lösung: Markieren Sie die Build-Verzeichnisse als Quellverzeichnis in Ihrem Projekt.
5. Importanweisungen fehlten im Code.

   Lösung: Fügen Sie die benötigten Importanweisungen zum Code hinzu.

Quellen

https://grpc.io/

https://www.makeuseof.com/grpc-what-why-use/

https://dev.to/koich1/what-are-protocol-buffers-ec9

https://protobuf.dev/overview/