README

gRPC in Springboot Java

Author: Rahul Gupta
Date: Feb, 14

Explanation of the Project

Getting started

Building the project

In order to utilize gRPC with Springboot we need to use gradle and define the necassary code blocks inside the **build.gradle** file. With spring initializr0 you can initialize a basic spring boot project. Project --> Gradle - Groovy, Language --> Java, Spring Boot --> 3.2.2, Packaging --> Jar, Java --> 17.
After that I have searched the web for the necessary dependencies and plugins. This is how the working **build.gradle** looked in the end.

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.2.2'
    id 'io.spring.dependency-management' version '1.1.4'
    id "com.google.protobuf" version "0.9.4"
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'

java {
    sourceCompatibility = '17'
}

repositories {
    mavenCentral()
}

dependencies {
    // https://mvnrepository.com/artifact/com.google.protobuf/protobuf-java
    implementation group: 'com.google.protobuf', name: 'protobuf-java',
version: '3.25.2'
    implementation 'org.springframework.boot:spring-boot-starter'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    runtimeOnly 'io.grpc:grpc-netty-shaded:1.61.0'
    implementation 'io.grpc:grpc-protobuf:1.61.0'
    implementation 'io.grpc:grpc-stub:1.61.0'
    implementation 'net.devh:grpc-spring-boot-starter:2.15.0.RELEASE'
    compileOnly 'org.apache.tomcat:annotations-api:6.0.53' // necessary for
Java 9+
}

tasks.named('test') {
    useJUnitPlatform()
}

sourceSets {
    main {
       java {
          srcDirs 'build/generated/source/proto/main/java', 'src/main/java'
       }
    }
}

protobuf {
    protoc {
       artifact = "com.google.protobuf:protoc:3.25.1"
    }
    plugins {
       grpc {
          artifact = 'io.grpc:protoc-gen-grpc-java:1.61.0'
       }
    }
    generateProtoTasks {
       all()*.plugins {
          grpc {
             outputSubDir = 'java'
          }
       }
    }
}
```

Generating Java files out of .proto file

`./gradlew generateProto`

Implementing gRPC Server and Client in Springboot application

Helloworld example

HelloworldServer.java:

```java
import io.grpc.Server;
import io.grpc.ServerBuilder;

import java.io.IOException;


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();
    }

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

    public static void main(String[] args) throws InterruptedException,
IOException {
        HelloWorldServer server = new HelloWorldServer();
        System.out.println( "HelloWorld Service is running!");
        server.start();
        server.blockUntilShutdown();
    }

}
```

HelloWorldClient.java:

```java
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
```

```
public class HelloWorldClient {

    public static void main(String[] args) {

        ManagedChannel channel =
ManagedChannelBuilder.forAddress("localhost", 50051)
                .usePlaintext()
                .build();

        HelloWorldServiceGrpc.HelloWorldServiceBlockingStub stub =
HelloWorldServiceGrpc.newBlockingStub(channel);

        Hello.HelloResponse helloResponse =
stub.hello(Hello.HelloRequest.newBuilder()
                .setFirstname("Max")
                .setLastname("Mustermann")
                .build());
        System.out.println( helloResponse.getText() );
        channel.shutdown();

    }

}
```

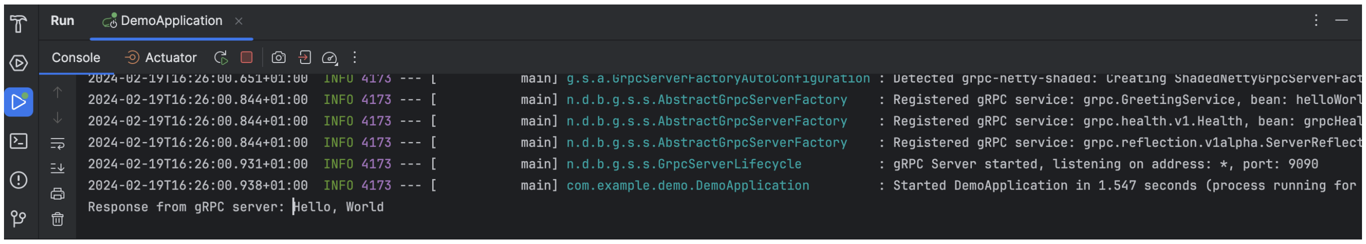
HelloWorldServiceImpl.java:

```
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 = "Hello World, " + request.getFirstname() + " " +
request.getLastname();
        Hello.HelloResponse response =
Hello.HelloResponse.newBuilder().setText(text).build();

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

    }
}
```

helloworld.proto:

```
syntax = "proto3";

package grpc;

// Der GreetingService definiert einen RPC-Dienst mit einer Methode
sayHello.
service GreetingService {
  // sayHello ist eine Methode, die eine HelloRequest-Nachricht nimmt und
eine HelloReply-Nachricht zurückgibt.
  rpc sayHello (HelloRequest) returns (HelloReply) {}
}

// HelloRequest ist die Anfrage-Nachricht mit einem Namen.
message HelloRequest {
  string name = 1;
}

// HelloReply ist die Antwort-Nachricht mit einer Begrüßung.
message HelloReply {
  string message = 1;
}
```

Output:



```
2024-02-19T16:26:00.651+01:00  INFO 4173 --- [           main] g.s.a.GrpcServerFactoryAutoConfiguration : Detected grpc-netty-shaded: Creating ShadedNettyGrpcServerFact
2024-02-19T16:26:00.844+01:00  INFO 4173 --- [           main] n.d.b.g.s.s.AbstractGrpcServerFactory    : Registered gRPC service: grpc.GreetingService, bean: helloWorl
2024-02-19T16:26:00.844+01:00  INFO 4173 --- [           main] n.d.b.g.s.s.AbstractGrpcServerFactory    : Registered gRPC service: grpc.health.v1.Health, bean: grpcHeal
2024-02-19T16:26:00.844+01:00  INFO 4173 --- [           main] n.d.b.g.s.s.AbstractGrpcServerFactory    : Registered gRPC service: grpc.reflection.v1alpha.ServerReflect
2024-02-19T16:26:00.931+01:00  INFO 4173 --- [           main] n.d.b.g.s.s.GrpcServerLifecycle          : gRPC Server started, listening on address: *, port: 9090
2024-02-19T16:26:00.938+01:00  INFO 4173 --- [           main] com.example.demo.DemoApplication         : Started DemoApplication in 1.547 seconds (process running for
Response from gRPC server: Hello, World
```

Warehouse example

```
syntax = "proto3";

package tradearea;

// Definition für ProductData
message ProductData {
  string productId = 1;
  string productName = 2;
  string productCategory = 3;
  string productAmount = 4;
  string productUnit = 5;
}

// Definition für WarehouseData
message WarehouseData {
  string warehouseID = 1;
  string warehouseName = 2;
  string timestamp = 3;
  string warehouseCountry = 4;
  string warehouseCity = 5;
  string address = 6;
  repeated ProductData productData = 7;
}

// Service-Definition
service WarehouseService {
  rpc GetWarehouseData (WarehouseRequest) returns (WarehouseData) {}
}

// Anfrageformat für WarehouseData
message WarehouseRequest {
  string warehouseID = 1;
}
```

```
package tradearea.warehouse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import tradearea.model.WarehouseData;

@Service
public class WarehouseService {

    private WarehouseService service;

    public String getGreetings( String inModule ) {
        return "Greetings from " + inModule;
    }

    public WarehouseData getWarehouseData( String inID ) {
        WarehouseSimulation simulation = new WarehouseSimulation();
        return simulation.generateRandomWarehouseData();

    }
    public static WarehouseData getWarehouseData() {

        WarehouseSimulation simulation = new WarehouseSimulation();
        return simulation.generateRandomWarehouseData();

    }

}
```

```
package tradearea.warehouse;

import io.grpc.stub.StreamObserver;
import tradearea.Warehouse.WarehouseRequest;
import tradearea.Warehouse.WarehouseData;
import tradearea.WarehouseServiceGrpc.WarehouseServiceImplBase;

public class WarehouseServiceImpl extends WarehouseServiceImplBase {

    @Override
    public void getWarehouseData(WarehouseRequest request,
StreamObserver<WarehouseData> responseObserver) {
        // Hier können Sie die Logik hinzufügen, um die Daten zu generieren
oder abzurufen
        // Zum Beispiel könnte es so aussehen:        WarehouseData data =
WarehouseData.newBuilder()
                .setWarehouseID(request.getWarehouseID())
                .setWarehouseName("Pandabuy Warehouse")
                .setWarehouseCity("Warehouse City")
                .setWarehouseCountry("Warehouse Country")
                .setAddress("Warehouse Address")
                .setTimestamp("Timestamp")
                .build();

        responseObserver.onNext(data);
        responseObserver.onCompleted();
    }
}
```

```
package tradearea.warehouse;
```

```
import io.grpc.Server;
import io.grpc.ServerBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.io.IOException;

@Configuration
public class GrpcServerConfig {

    @Bean(destroyMethod = "shutdown")
    public Server grpcServer() throws IOException {
        return ServerBuilder.forPort(50051)
                .addService(new WarehouseServiceImpl())
                .build()
                .start();
    }
}
```

```
package tradearea;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import tradearea.Warehouse.WarehouseRequest;
import tradearea.Warehouse.WarehouseData;
import tradearea.Warehouse.ProductData;
import tradearea.WarehouseServiceGrpc.WarehouseServiceBlockingStub;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;

@SpringBootApplication
public class WarehouseApplication {

    public static void main(String[] args) {
        SpringApplication.run(WarehouseApplication.class, args);
    }

    @Bean
    public CommandLineRunner run() {
        return args -> {
            ManagedChannel channel =
ManagedChannelBuilder.forAddress("localhost", 50051)
                    .usePlaintext()
```

```
                      .build();

              try {
                  WarehouseServiceBlockingStub stub = 
WarehouseServiceGrpc.newBlockingStub(channel);
                  WarehouseRequest request = 
WarehouseRequest.newBuilder().setWarehouseID("001").build();
                  WarehouseData response = stub.getWarehouseData(request);

                  // Ausgabe der Antwort mit mehr Details
                  System.out.println("Response from Warehouse gRPC server:");
                  System.out.println("Warehouse ID: " + 
response.getWarehouseID());
                  System.out.println("Warehouse Name: " + 
response.getWarehouseName());
                  System.out.println("Warehouse Country: " + 
response.getWarehouseCountry());
                  System.out.println("Warehouse City: " + 
response.getWarehouseCity());
                  System.out.println("Warehouse Address: " + 
response.getAddress());
                  System.out.println("Timestamp: " + response.getTimestamp());

                  // Ausgabe der Produktdaten
                  for (ProductData product : response.getProductDataList()) {
                      System.out.println("Product ID: " + 
product.getProductId());
                      System.out.println("Product Name: " + 
product.getProductName());
                      System.out.println("Product Category: " + 
product.getProductCategory());
                      System.out.println("Product Amount: " + 
product.getProductAmount());
                      System.out.println("Product Unit: " + 
product.getProductUnit());
                  }
              } finally {
                  channel.shutdownNow();
              }
          };
      }
}
```

Common Issues and Solutions

1. **Importing the Generated Class Issue**: The generated `PersonOuterClass` couldn't be imported. This was resolved by adjusting the `sourceSets` in `build.gradle` to include the directory of the generated sources, so the IDE could recognize them as part of the project. In the `build.gradle` file, it was:

```
sourceSets {
    main {
       java {
          srcDirs 'build/generated/source/proto/main/java', 'src/main/java'
       }
    }
}
```

With this, `PersonOuterClass` could be imported with: `import com.example.proto.PersonOuterClass;`

2. **`generateProto` Execution Error**: Attempting to run the `generateProto` task failed due to an unresolved dependency. This was caused by using a release candidate version (`4.26.0-RC2`) of `protobuf-java`, which might have been incompatible or not available. The solution was to switch to a stable version, which was **3.25.2** in this case, and restarting the IDE resolved the issue. It also turned out that the mavenCentral repository is not always up to date with the latest releases of plugins and dependencies.
3. **Missing Package Declaration**: Initially, the generated class couldn't be imported because the `.proto` file lacked a package declaration, causing the generated class to default to the default package. The issue was fixed by adding a `package` statement to the `.proto` file and re-running the `generateProto` task to regenerate the classes with the correct package name. For this case, it was **package com.example.proto;**

```
syntax = "proto3";

package com.example.proto;

message Person {
  optional string name = 1;
  optional int64 id = 2;
  optional string email = 3;
}
```

4. **GRPC-Service Annotation not working** The @GRPC-Service annotation that was essential for combining a gRPC service with Springboot was not resolvable. The fix was to add the necessary dependency with `implementation 'net.devh:grpc-spring-boot-starter:2.15.0.RELEASE'` in the build.gradle file and restart the IDE.

5. **Spring Boot Application terminates after starting it** This was caused by a dependency error I was facing. The description of the exception was: `Correct the classpath of your application so that it contains compatible versions of the classes io.grpc.internal.AutoConfiguredLoadBalancerFactory$AutoConfiguredLoadBalancer and io.grpc.LoadBalancer`.