A basic tutorial introduction to gRPC in Java.

This tutorial provides a basic Java programmer's introduction to working with gRPC.

By walking through this example you'll learn how to:

- Define a service in a `.proto` file.
- Generate server and client code using the protocol buffer compiler.
- Use the Java gRPC API to write a simple client and server for your service.

It assumes that you have read the Introduction to gRPC and are familiar with protocol buffers. Note that the example in this tutorial uses the proto3 version of the protocol buffers language: you can find out more in the proto3 language guide and Java generated code guide.

Why use gRPC?

Our example is a simple route mapping application that lets clients get information about features on their route, create a summary of their route, and exchange route information such as traffic updates with the server and other clients.

With gRPC we can define our service once in a `.proto` file and generate clients and servers in any of gRPC's supported languages, which in turn can be run in environments ranging from servers inside a large data center to your own tablet — all the complexity of communication between different languages and environments is handled for you by gRPC. We also get all the advantages of working with protocol buffers, including efficient serialization, a simple IDL, and easy interface updating.

1. Create a New Maven Project in NetBeans:

1. **Open NetBeans** and go to **File** > **New Project**.
2. Select **Maven** under **Categories** and choose **Java Application** under **Projects**.
3. Click **Next** and configure your project by providing a Project Name (e.g., `grpc-java-example`), Group ID, and Artifact ID. Click **Finish**.

2. Add gRPC Dependencies:

1. In the **Projects** pane, find your project and open the `pom.xml` file.
2. Add the gRPC dependencies and the Protobuf Maven plugin to the `pom.xml` as described in the previous steps:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>grpc-java-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>21</maven.compiler.source>
    <maven.compiler.target>21</maven.compiler.target>
    <exec.mainClass>com.example.grpc</exec.mainClass>
  </properties>

  <dependencies>
  <dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-netty-shaded</artifactId>
    <version>1.58.0</version>
  </dependency>
  <dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-protobuf</artifactId>
    <version>1.58.0</version>
  </dependency>
  <dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-stub</artifactId>
```

```
      <version>1.58.0</version>
    </dependency>
    <dependency>
      <groupId>com.google.protobuf</groupId>
      <artifactId>protobuf-java</artifactId>
      <version>3.24.0</version>
    </dependency>
      <dependency>
        <groupId>org.realityforge.javax.annotation</groupId>
        <artifactId>javax.annotation</artifactId>
        <version>1.1.1</version>
        <type>jar</type>
      </dependency>
    </dependencies>

<build>
    <extensions>
      <extension>
        <groupId>kr.motd.maven</groupId>
        <artifactId>os-maven-plugin</artifactId>
        <version>1.7.0</version>
      </extension>
    </extensions>
    <plugins>
      <plugin>
        <groupId>org.xolstice.maven.plugins</groupId>
        <artifactId>protobuf-maven-plugin</artifactId>
        <version>0.6.1</version>
        <configuration>
```

```
<protocArtifact>com.google.protobuf:protoc:3.24.0:exe:${os.detected.classifier}</protocArtifact>
            <pluginId>grpc-java</pluginId>
            <pluginArtifact>io.grpc:protoc-gen-grpc-
java:1.58.0:exe:${os.detected.classifier}</pluginArtifact>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>compile</goal>
              <goal>compile-custom</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
</build>
</project>
```

3. Save the `pom.xml` file. NetBeans will automatically download the necessary dependencies.

4. Create the .proto File:

1. Right-click on the **Source Packages** folder in the Projects pane.
2. Select **New** > **Other**.
3. Choose **File** under the **General** category, name it `helloworld.proto`, and save it under `src/main/proto/`.
4. Add the following content to the `helloworld.proto` file:

```
syntax = "proto3";


option java_package = "com.example.grpc";
option java_outer_classname = "HelloWorldProto";
```

```
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply);
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

5. Generate Java Code from .proto File:

1. In NetBeans, open the **Projects** pane.
2. Right-click your project and select **Build**. This will run the Maven build process, generating the Java classes from the `.proto` file.

6. Implement the gRPC Service:

1. In the **Projects** pane, navigate to the `src/main/java/com/example/grpc` directory.
2. Right-click and create a new Java class named `HelloWorldServer`.
3. Implement the server code as follows:

```
package com.example.grpc;

/**
 *
 * @author kaylash.chaudhary
 */
import com.example.grpc.HelloWorldProto.HelloReply;
import com.example.grpc.HelloWorldProto.HelloRequest;
import io.grpc.Server;
```

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

import java.io.IOException;

public class HelloWorldServer {

   public static void main(String[] args) throws IOException, InterruptedException {
      Server server = ServerBuilder.forPort(50051).addService(new GreeterImpl()).build().start();

      System.out.println("Server started, listening on " + server.getPort());
      server.awaitTermination();
   }

   static class GreeterImpl extends GreeterGrpc.GreeterImplBase {
      @Override
      public void sayHello(HelloRequest req, StreamObserver<HelloReply> responseObserver) {
         HelloReply reply = HelloReply.newBuilder().setMessage("Hello " + req.getName()).build();
         responseObserver.onNext(reply);
         responseObserver.onCompleted();
      }
   }
}
```

7. Create the gRPC Client:

1. Right-click on the `com.example.grpc` package and create a new Java class named `HelloWorldClient`.
2. Implement the client code as follows:

```
package com.example.grpc;
```

```
/**
 *
 * @author kaylash.chaudhary
 */
import com.example.grpc.HelloWorldProto.HelloReply;
import com.example.grpc.HelloWorldProto.HelloRequest;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;


public class HelloWorldClient {
   public static void main(String[] args) throws InterruptedException {
      ManagedChannel channel = ManagedChannelBuilder.forAddress("localhost", 50051)
            .usePlaintext()
            .build();


      GreeterGrpc.GreeterBlockingStub stub = GreeterGrpc.newBlockingStub(channel);


      HelloReply response = stub.sayHello(HelloRequest.newBuilder().setName("World").build());


      System.out.println("Greeting: " + response.getMessage());


      channel.shutdown();
   }
}
```

8. Run the Server and Client:

1. **Run the Server:**
   - Right-click the `HelloWorldServer` class in NetBeans and select **Run**.
2. **Run the Client:**

- After the server is running, right-click the `HelloWorldClient` class and select **Run**.

You should see the server output "Server started, listening on 50051" and the client output "Greeting: Hello World".