Rozproszone Systemy Informatyczne

Raport - Ćwiczenie 2b

Paweł Kluska, 260391

Katsiaryna Ziatsikava, 245891

System składa się z 3 modułów:

- GrpcClient

- GrpcInterface

- GrpcServer

GrpcInterface służy do sprecyzowania typów danych oraz metod w pliku proto, który odpowiada za komunikację pomiędzy serwerem, a klientem.

GrpcServer implementował metody, które zostały podane w GrpcInterface proto.

GrpcClient następujęł połączenie z serwerem, a następnie uruchomienie metod, które zostałe tam wystawiona.

Na początku odpowiednio przygotowaliśmy pliki konfiguracyjne pom.xml dla projektu i poszczególnych modułów.

Main pom.xml

W main pom.xml zdefiniowane moduły i wspólne komponenty takie jak properties i dependencies.

```xml
<modules>
    <module>GrpcClient</module>
    <module>GrpcInterface</module>
    <module>GrpcServer</module>
</modules>

<properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <grpc.version>1.43.2</grpc.version>
    <protobuf.maven.plugin.version>0.6.1</protobuf.maven.plugin.version>
    <protobuf.version>3.19.4</protobuf.version>
    <os.maven.plugin.version>1.6.2</os.maven.plugin.version>
</properties>
```

```xml
<dependencies>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-protobuf</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-stub</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-netty</artifactId>
        <version>${grpc.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.tomcat</groupId>
        <artifactId>annotations-api</artifactId>
        <version>6.0.53</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

GrpcInterface pom.xml

W pliku jest zdefiniowany moduł rodzica,os-maven-plugin i protobuf-maven-plugin.

```
<parent>
    <groupId>org.example</groupId>
    <artifactId>GrpcApp</artifactId>
    <version>1.0-SNAPSHOT</version>
</parent>
```

```
<build>
    <extensions>
        <extension>
            <groupId>kr.motd.maven</groupId>
            <artifactId>os-maven-plugin</artifactId>
            <version>${os.maven.plugin.version}</version>
        </extension>
    </extensions>
```

```
    <plugins>
        <plugin>
            <groupId>org.xolstice.maven.plugins</groupId>
            <artifactId>protobuf-maven-plugin</artifactId>
            <version>${protobuf.maven.plugin.version}</version>
            <configuration>
                <protocArtifact>com.google.protobuf:protoc:${protobuf.version}:exe:${os.detected.classifier}
                </protocArtifact>
                <pluginId>grpc-java</pluginId>
                <pluginArtifact>io.grpc:protoc-gen-grpc-java:${grpc.version}:exe:${os.detected.classifier}
                </pluginArtifact>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>compile-custom</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

GrpcServer pom.xml

W GrpcServer pom.xml jest zdefiniowany moduł rodzica, zależność do GrpcInterface i maven-assembly-plugin.

```
<parent>
    <groupId>org.example</groupId>
    <artifactId>GrpcApp</artifactId>
    <version>1.0-SNAPSHOT</version>
</parent>

<artifactId>GrpcServer</artifactId>

<properties>
    <maven.compiler.source>19</maven.compiler.source>
    <maven.compiler.target>19</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
    <dependency>
        <groupId>org.example</groupId>
        <artifactId>GrpcInterface</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

GrpcClient pom.xml

Posiada identyczną konfigurację jak GrpcServer.

Proto file

W module GrpcInterface w folderze main był stworzony plik GrpcInterface.proto, który umożliwia korzystanie z usług gRPC oraz komunikację między klientami gRPC i komunikatami serwera. W tym pliku definiujemy typy danych dla ImageRequest, ImageChunk i metodę rpc UploadImage i rpc DownloadImage.

```
syntax = "proto3";

option java_package = "com.example.grpc";
option java_outer_classname = "ImageServiceProto";

service ImageService {
  rpc UploadImage(stream ImageChunk) returns (UploadStatus) {}
  rpc DownloadImage(ImageRequest) returns (stream ImageChunk) {}
}

message ImageChunk {
  bytes data = 1;
}

message ImageRequest {
  string id = 1;
}

message UploadStatus {
  bool success = 1;
}
```

Mamy tutaj przedstawioną sygnaturę 2 metod, jedna jest odpowiedzialna za przesyłanie obrazu na serwer, druga za przesyłanie do klienta. Obie metody wykorzystują strumieniowe przesyłanie, po otrzymaniu żądania następuje przesyłanie strumienia bajtów.

GrpcServer class

Ta klasa jest odpowiedzialna za wystawienie serwera na określonym ip, porcie oraz zaimplementowania metod zdefiniowanych w pliku proto, które będzie dostępna na zewnątrz.

Metoda odpowiadająca za przesyłanie obrazków na serwer.

```java
public StreamObserver<ImageServiceProto.ImageChunk> uploadImage(StreamObserver<ImageServiceProto.UploadStatus> responseObserver) {
    try {
        outputStream = new FileOutputStream( name: "GrpcServer/src/main/resources/image/image_from_client.jpg");
    } catch (IOException e) {
        responseObserver.onError(e);
    }
    return new StreamObserver<>() {
        @Override
        public void onNext(ImageServiceProto.ImageChunk value) {
            try {
                outputStream.write(value.getData().toByteArray());
            } catch (IOException e) {
                onError(e);
            }
        }

        @Override
        public void onError(Throwable t) {
            t.printStackTrace();
            responseObserver.onError(t);
        }

        3 usages
        @Override
        public void onCompleted() {
            try {
                outputStream.close();
            } catch (IOException e) {
                onError(e);
            }
            ImageServiceProto.UploadStatus uploadStatus = ImageServiceProto.UploadStatus.newBuilder()
                    .setSuccess(true)
                    .build();
            responseObserver.onNext(uploadStatus);
            responseObserver.onCompleted();
        }
    };
```

Metoda przesyłająca obrazy do klienta.

```java
1 usage
public void downloadImage(ImageServiceProto.ImageRequest request, StreamObserver<ImageServiceProto.ImageChunk> responseObserver) {

    try {
        String fileName = "GrpcServer/src/main/resources/image/image2.jpg";
        FileInputStream input = new FileInputStream(fileName);
        byte[] buffer = new byte[1024];
        int bytesRead;

        while ((bytesRead = input.read(buffer)) != -1) {
            ImageServiceProto.ImageChunk chunk = ImageServiceProto.ImageChunk.newBuilder()
                    .setData(ByteString.copyFrom(buffer,  offset: 0, bytesRead))
                    .build();
            responseObserver.onNext(chunk);
        }
        responseObserver.onCompleted();
        input.close();
    } catch (IOException e) {
        e.printStackTrace();
        responseObserver.onError(e);
    }
}
```

Obie metody działają w podobny sposób. Wczytują obraz z podanej ścieżki, zamieniają na tablicę bajtów i następnie przesyłają na drugą stronę.

GrpcClient class

Ta klasa odpowiada za łączenie się z serwerem oraz wywołuje metodę z serwera

Metoda wysyłająca dane na serwer.

```
public static void startClient() throws IOException {
    String address = "localhost";
    int port = 50000;
    final Integer[] packageNumber = {0};
    ImageServiceGrpc.ImageServiceStub stub;
    ManagedChannel channel = ManagedChannelBuilder.forAddress(address, port)
            .usePlaintext().build();
    stub = ImageServiceGrpc.newStub(channel);
    StreamObserver<ImageServiceProto.ImageChunk> requestObserver = stub.uploadImage(new StreamObserver<ImageServiceProto.UploadStatus>() {

        @Override
        public void onNext(ImageServiceProto.UploadStatus value) {
            packageNumber[0]++;
            System.out.println("Numer paczki " + packageNumber[0]);
            System.out.println("Upload status: " + value.getSuccess());
        }

        @Override
        public void onError(Throwable t) { t.printStackTrace(); }

        3 usages
        @Override
        public void onCompleted() { System.out.println("Upload completed"); }
    });
    String fileName = "GrpcClient/src/main/resources/images/image.jpg";
    try (FileInputStream input = new FileInputStream(fileName)) {
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = input.read(buffer)) != -1) {
            ImageServiceProto.ImageChunk chunk = ImageServiceProto.ImageChunk.newBuilder()
                    .setData(ByteString.copyFrom(buffer, offset: 0, bytesRead))
                    .build();
            requestObserver.onNext(chunk);
        }
    }
    requestObserver.onCompleted();
    channel.shutdown();
```

Metoda pobierająca dane z serwera

```java
public static void startClient2() throws IOException {
    String address = "localhost";
    int port = 50000;
    final Integer[] packageNumber = {0};
    ImageServiceGrpc.ImageServiceStub stub;
    ManagedChannel channel = ManagedChannelBuilder.forAddress(address, port)
            .usePlaintext().build();
    stub = ImageServiceGrpc.newStub(channel);
    ImageServiceProto.ImageRequest request = ImageServiceProto.ImageRequest.newBuilder().build();
    FileOutputStream fileOutputStream = new FileOutputStream( name: "GrpcClient/src/main/resources/images/image_from_server.jpg");
    stub.downloadImage(request, new StreamObserver<ImageServiceProto.ImageChunk>() {
        @Override
        public void onNext(ImageServiceProto.ImageChunk response) {
            try {
                packageNumber[0]++;
                System.out.println("Numer paczki " + packageNumber[0]);
                fileOutputStream.write(response.getData().toByteArray());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        @Override
        public void onError(Throwable t) { t.printStackTrace(); }

        3 usages
        @Override
        public void onCompleted() {
            try {
                fileOutputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            channel.shutdown();
        }
    });
}
```

Menu główne

```java
public static void main(String[] args) throws IOException {
    MyData.getData();
    Scanner scanner = new Scanner(System.in);

    while (true) {
        System.out.println("Wybierz tryb:");
        System.out.println("1. Download from client");
        System.out.println("2. Download from server");
        System.out.println("0. Exit");

        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                startClient();
                break;
            case 2:
                startClient2();
                break;
            case 0:
                System.exit( status: 0);
            default:
                System.out.println("Nieprawidłowy wybór. Spróbuj ponownie.");
        }
    }
}
```

Na powyższych zrzutach ekranu jest widoczne wykorzystanie metody info z klasy MyDate, która została zaimportowana do projektu przy użyciu paczki jar.

Działanie systemu

System uruchomiliśmy w konfiguracji dwu maszynowej, na jednej maszynie klient a na drugiej serwer. Oba komputery zostały podłączone do tej samej sieci lokalnej. Działanie systemu prezentuje się następująco:

Serwer

```
Paweł Kluska, 260391
Katya Zyatikava, 245891
4 kwietnia 08:33:41
19.0.1
pawelk
Linux
192.168.43.162
Server started...
Numer paczki 1
Numer paczki 2
Numer paczki 3
Numer paczki 4
Numer paczki 5
Numer paczki 6
Numer paczki 7
Numer paczki 8
Numer paczki 9
Numer paczki 10
Numer paczki 11
```

```
Numer paczki 2193
Numer paczki 2194
Numer paczki 2195
Numer paczki 2196
Numer paczki 2197
Numer paczki 2198
Numer paczki 2199
Numer paczki 2200
Numer paczki 2201
```

W tym przypadku przesyłaliśmy obrazek o rozmiarze około 2,3 MB. Rozmiar paczki wynosił 1024 bajty, więc po przeprowadzeniu prostych obliczeń można stwierdzić, że ilość paczek się zgadza.

Po zwiększeniu rozmiaru paczki dwukrotnie ilość paczek się zmniejszyła.

```
Numer paczki 1093
Numer paczki 1094
Numer paczki 1095
Numer paczki 1096
Numer paczki 1097
Numer paczki 1098
Numer paczki 1099
Numer paczki 1100
Numer paczki 1101
```

Klient

MyDate.info();

```
Katsiaryna Ziatsikava - 245891
Paweł Kluska - 260391
4 APRIL 04 April 08:34:46
19.0.2+7-44
Windows 10
192.168.56.1
```

W tym przypadku przesyłaliśmy obrazek o rozmiarze około 4,79 MB (5,028,733 bytes). Rozmiar paczki wynosił 2048 bajty, więc po przeprowadzeniu prostych obliczeń można stwierdzić, że ilość paczek się zgadza.

```
Wybierz tryb:
1. Download from client to server
2. Download from server to client
0. Exit
Numer paczki 1
Numer paczki 2
Numer paczki 3
Numer paczki 4
Numer paczki 5
Numer paczki 6
Numer paczki 7
```

```
Numer paczki 2451
Numer paczki 2452
Numer paczki 2453
Numer paczki 2454
Numer paczki 2455
Numer paczki 2456
Apr 04, 2023 8:35:28 AM GrpcClient$2 onCompleted
INFO: Image stream completed.
```

Bibliografia

https://www.baeldung.com/java-grpc-streaming

https://grpc.io/docs/languages/java/basics/