

## 26. gRPC (Introduction)

- Основы работы с gRPC сервисами
- Пример создания клиентской и серверной частей в среде Java
- Отправка запросов и получение ответа от сервера

Pom.xml (добавление gRPC библиотек, и плагин для генерации proto кода в другие языки программирования):

```
<dependencies>
  <dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-netty-shaded</artifactId>
    <version>1.24.0</version>
  </dependency>
  <dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-protobuf</artifactId>
    <version>1.24.0</version>
  </dependency>
  <dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-stub</artifactId>
    <version>1.24.0</version>
  </dependency>
  <dependency>
    <groupId>javax.annotation</groupId>
    <artifactId>javax.annotation-api</artifactId>
    <version>1.3.1</version>
  </dependency>
</dependencies>

<build>
  <extensions>
    <extension>
      <groupId>kr.motd.maven</groupId>
      <artifactId>os-maven-plugin</artifactId>
      <version>1.6.2</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:3.9.0:exe:${os.detected.classifier}</
protocArtifact>
        <pluginId>grpc-java</pluginId>
        <pluginArtifact>io.grpc:protoc-gen-grpc-
java:1.24.0:exe:${os.detected.classifier}</pluginArtifact>
      </configuration>
      <executions>
        <execution>
          <goals>
            <goal>compile</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```

        <goal>compile-custom</goal>
      </goals>
    </execution>
  </executions>
</plugin>

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
  </configuration>
</plugin>
</plugins>
</build>

```

Структура файла greetingService.proto (добавление блоков message, инициализация сервиса в блоке service, с помощью ключевого слова rpc):

```

syntax = "proto3";
package com.example.grpc;

message HelloRequest {
  string name = 1;
  repeated string hobbies = 2;
}

message HelloResponse {
  string greeting = 1;
}

service GreetingService{
  rpc greeting(HelloRequest) returns (HelloResponse);
}

```

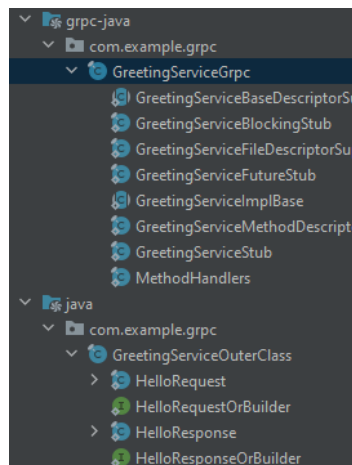
Вариант с определением потоков (ключевое слово stream):

```

service GreetingService{
  rpc greeting(HelloRequest) returns (stream HelloResponse);
}

```

Сборка проекта в maven, и генерация функциональных классов на основе proto файла:



## Реализация серверной части

Класс GreetingServiceImpl (наследование от класса GreetingServiceImplBase, переопределение метода greeting(), определение response, responseObserver, методы onNext(), onCompleted()):

```
public class GreetingServiceImpl extends
GreetingServiceGrpc.GreetingServiceImplBase {
    @Override
    public void greeting(GreetingServiceOuterClass.HelloRequest request,
        StreamObserver<GreetingServiceOuterClass.HelloResponse>
responseObserver) {
        System.out.println(request);
        GreetingServiceOuterClass.HelloResponse response =
GreetingServiceOuterClass.HelloResponse
            .newBuilder()
            .setGreeting("Hello from server, " + request.getName()).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```

Класс Runner (определение порта сервера, добавление функциональных сервисов, и запуск):

```
public class Runner {
    public static void main(String[] args) throws IOException,
InterruptedException {
        Server server = ServerBuilder.forPort(8080).addService(new
GreetingServiceImpl()).build();
        server.start();
        System.out.println("Server started");
        server.awaitTermination();
    }
}
```

## Вариант с использованием потоков:

```
@Override
public void greeting(GreetingServiceOuterClass.HelloRequest request,
    StreamObserver<GreetingServiceOuterClass.HelloResponse>
responseObserver) {
    for (int i=0; i<10000; i++){
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        GreetingServiceOuterClass.HelloResponse response =
GreetingServiceOuterClass.HelloResponse
            .newBuilder()
            .setGreeting("Hello from server, " + request.getName())
            .build();
        responseObserver.onNext(response);
    }
    responseObserver.onCompleted();
}
```

## Реализация клиентской части

Класс Client (создание канала, определение объекта Stub, конфигурация запроса и вызов методом greeting()):

```
public class Client {
    public static void main(String[] args) {
        ManagedChannel channel =
ManagedChannelBuilder.forTarget("localhost:8080").usePlaintext().build();
        GreetingServiceGrpc.GreetingServiceBlockingStub stub =
GreetingServiceGrpc.newBlockingStub(channel);
        GreetingServiceOuterClass.HelloRequest request =
GreetingServiceOuterClass.HelloRequest.newBuilder()
            .setName("Linar").build();
        GreetingServiceOuterClass.HelloResponse response =
stub.greeting(request);
        System.out.println(response);
        channel.shutdown();
    }
}
```

## Вариант с использованием потоков:

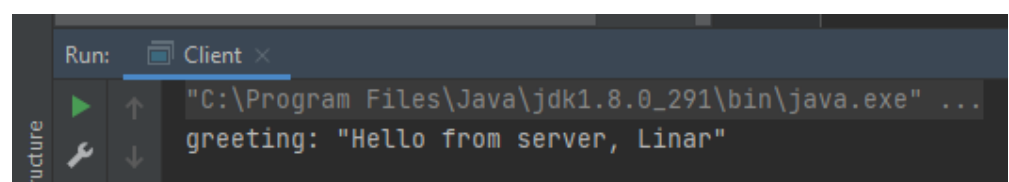
```
public class ClientStream {
    public static void main(String[] args) {
        ManagedChannel channel =
ManagedChannelBuilder.forTarget("localhost:8080").usePlaintext().build();

        GreetingServiceGrpc.GreetingServiceBlockingStub stub =
GreetingServiceGrpc.newBlockingStub(channel);
        GreetingServiceOuterClass.HelloRequest request =
GreetingServiceOuterClass.HelloRequest.newBuilder()
            .setName("Linar").build();

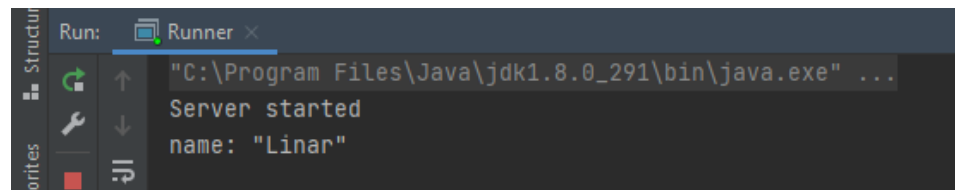
        Iterator<GreetingServiceOuterClass.HelloResponse> response =
stub.greeting(request);
        while (response.hasNext())
            System.out.println(response.next());
        channel.shutdown();
    }
}
```

## Результат

Отправка запроса через клиент и ответ от сервера:



Считывание данных запроса сервером, в данном случае параметр name:



The image shows a screenshot of an IDE's Run console. On the left, there is a vertical toolbar with icons for Run (a green play button), Debug (a blue bug), and Stop (a red square). Above these icons, the word 'Run:' is visible. The main area of the console displays the output of a Java application. The first line is the command to run Java: `"C:\Program Files\Java\jdk1.8.0_291\bin\java.exe" ...`. The subsequent lines are `Server started` and `name: "Linar"`.

```
Run: Runner X
"C:\Program Files\Java\jdk1.8.0_291\bin\java.exe" ...
Server started
name: "Linar"
```