链滴

SpringBoot 整合 gRPC

作者：96XL

原文链接：https://ld246.com/article/1666681606876

来源网站：链滴

简介

  gRPC是google开源的一个高性能、跨语言的RPC框架，基于HTTP2协议，于protobuf 3.x，基于Netty 4.x +。

对于开发者而言：

1）需要使用protobuf定义接口，即.proto文件。

2）然后使用compile工具生成特定语言的执行代码，比如JAVA、C/C++、Python等。类似于thrift为了解决跨语言问题。

3）启动一个Server端，server端通过侦听指定的port，来等待Client链接请求，通常使用Netty来构，GRPC内置了Netty的支持。

4）启动一个或者多个Client端，Client也是基于Netty，Client通过与Server建立TCP常链接，并发送求；Request与Response均被封装成HTTP2的stream Frame，通过Netty Channel进行交互。

proto3

编写proto3文件

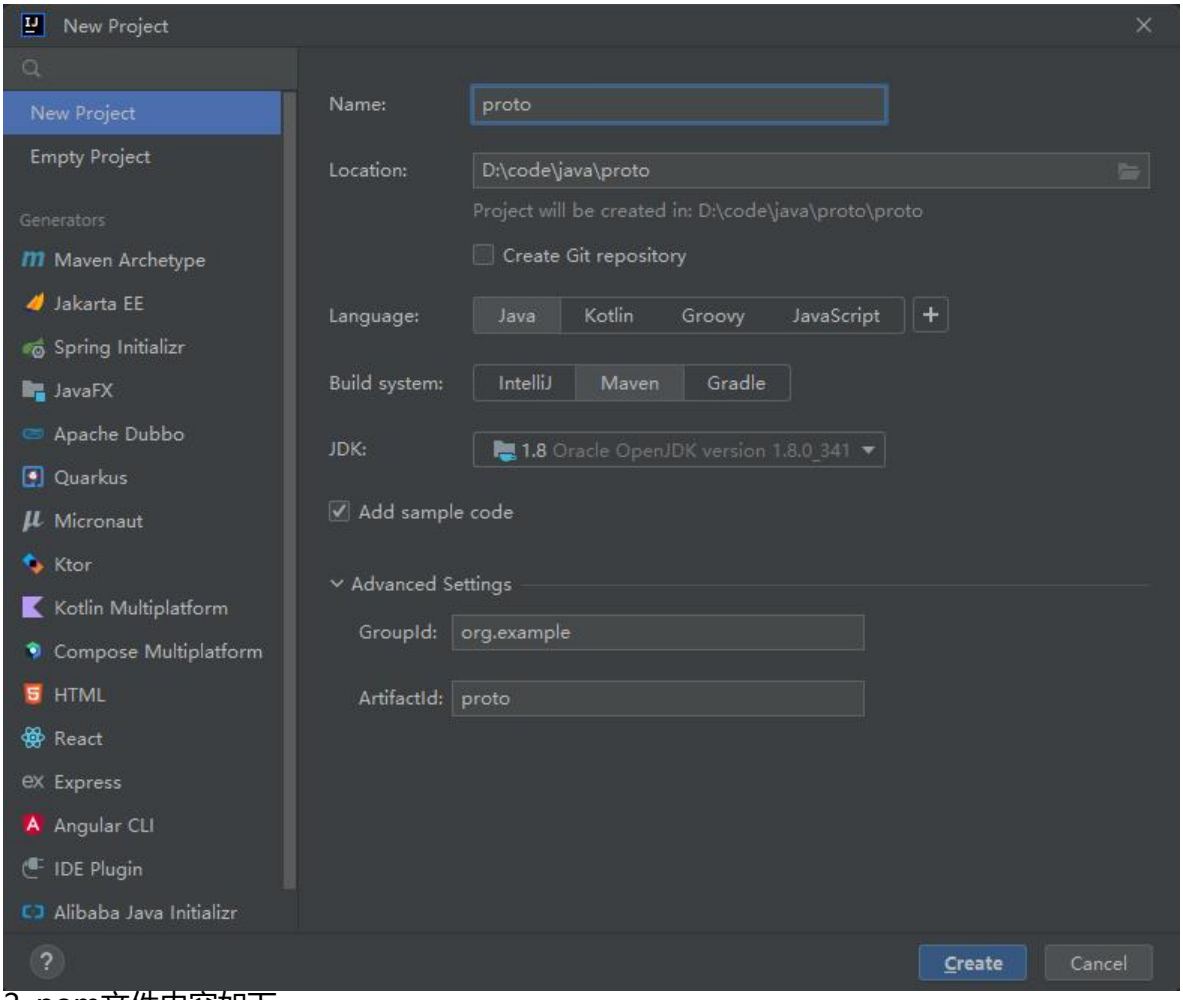
文件命名为pb.proto。

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "org.example.entity";
option java_outer_classname = "UserServiceProto";
```

```
service UserService {
  // 注册接口
  rpc register(RegisterReq) returns (RegisterRsp);
}

message RegisterReq {
  // 手机号
  string Phone = 1;
  // 用户ID
  string UserId = 2;
}

message RegisterRsp {
  // 业务状态码
  int32 code = 1;
  // 信息
  string msg = 2;
}
```

根据proto生成代码

1. 创建一个空的项目。



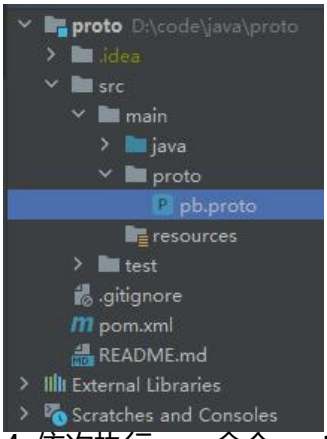
2. pom文件内容如下。

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

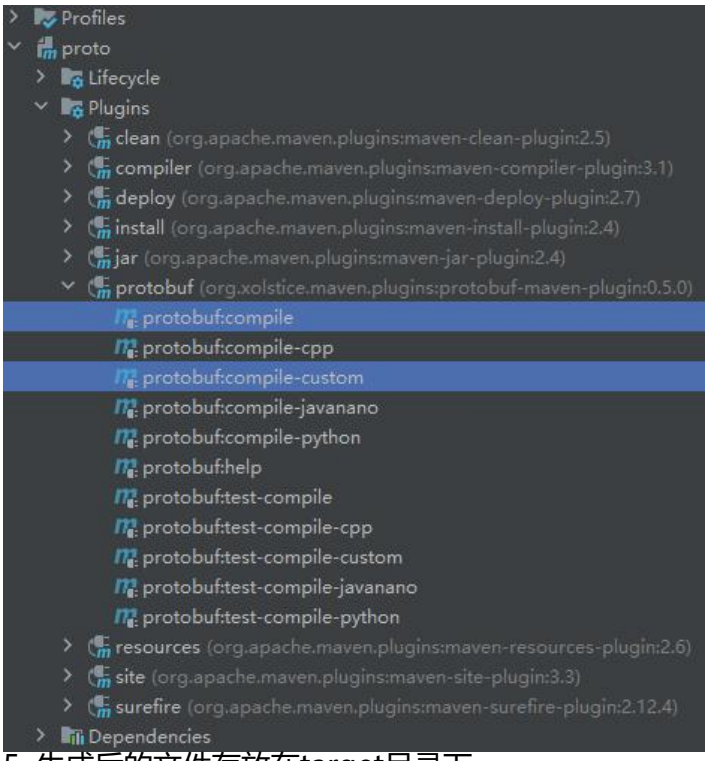
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>proto</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <jackson.version>2.8.3</jackson.version>
        <grpc.version>1.6.1</grpc.version>
        <os.plugin.version>1.5.0.Final</os.plugin.version>
        <protobuf.plugin.version>0.5.0</protobuf.plugin.version>
        <protoc.version>3.3.0</protoc.version>
        <grpc.netty.version>4.1.14.Final</grpc.netty.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-netty</artifactId>
            <version>${grpc.version}</version>
        </dependency>
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-protobuf</artifactId>
            <version>${grpc.version}</version>
        </dependency>
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-stub</artifactId>
            <version>${grpc.version}</version>
        </dependency>
        <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-common</artifactId>
            <version>${grpc.netty.version}</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-annotations</artifactId>
            <version>${jackson.version}</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-core</artifactId>
            <version>${jackson.version}</version>
        </dependency>
        <dependency>
```

```
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson.version}</version>
        </dependency>
    </dependencies>

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

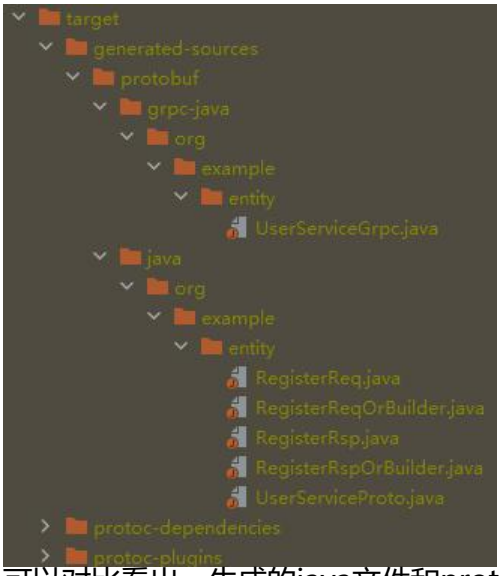
3. java同级目录下添加proto目录，将编写好的proto文件放到该目录下。



4. 依次执行mvn命令protobuf:compile和protobuf:compile-custom编译生成java文件。



5. 生成后的文件存放在target目录下。



可以对比看出，生成的java文件和proto文件中定义的service和message是对应的。其中 option java multiple_files = true; 配置决定是将message定义为一个文件还是内部类，将配置修改为false后再执行mvn命令后会发现生成的文件变成以下这种，这就是把message定义为了内部类。不过这两种对能上并无影响，开发者可以自行选择。

原文链接：SpringBoot 整合 gRPC

推荐将生成代码的项目单独存放，不要合并到开发项目中。因为pom中引入的jar和插件只是为了生代码，后续在SpringBoot中使用gRPC并不需要这些jar。示例项目放到了96XL/proto: proto文件生成(github.com)，需要的可以自取。

SpringBoot整合

生成代码的项目、服务端项目、客户端项目三者可以完全分离，单独使用哪个项目都可以。例如使用务端给外部系统提供gRPC接口，或者使用客户端调用外部系统的gRPC接口。

服务端

1. 添加maven依赖。

```
<dependency>
    <groupId>net.devh</groupId>
    <artifactId>grpc-server-spring-boot-starter</artifactId>
    <version>2.13.1.RELEASE</version>
</dependency>
```

2. 服务端代码（记得将生成的类加到项目中）。

```
@Slf4j
@GrpcService
public class UserGrpcServer extends UserServiceGrpc.UserServiceImplBase {

    public void register(UserServiceProto.RegisterReq request, StreamObserver<UserServicePro
o.RegisterRsp> responseObserver) {
        // 接收到的参数
        log.info(request.toString());
        // 响应
        UserServiceProto.RegisterRsp reply = UserServiceProto.RegisterRsp.newBuilder().setCode
0).build();
        responseObserver.onNext(reply);
        responseObserver.onCompleted();
    }
}
```

3. 配置文件添加配置。

```
grpc:
  server:
    port: 7777
```

客户端

1. 添加maven依赖。

```
<dependency>
    <groupId>net.devh</groupId>
    <artifactId>grpc-client-spring-boot-starter</artifactId>
    <version>2.13.1.RELEASE</version>
</dependency>
```

2. 客户端代码（记得将生成的类加到项目中）。

```
@Service
public class UserGrpcClient {

    @GrpcClient("userServiceGrpc")
    private UserServiceGrpc.UserServiceBlockingStub userServiceBlockingStub;

    public void register(String phone, String userId) {
        UserServiceProto.RegisterReq registerReq = UserServiceProto.RegisterReq.newBuilder().s
tPhone(phone).setUserId(userId).build();
        userServiceBlockingStub.register(registerReq);
    }
}
```

3. 配置文件添加配置。这里的IP就是服务端的IP，端口号要和服务端指定的端口号保持一致。userServ ceGrpc就是客户端代码@GrpcClient("userServiceGrpc") 中指定的值，两者保持一致即可，和服务 没有关系。

```
grpc:
  client:
    userServiceGrpc:
      address: static://localhost:7777
      negotiationType: PLAINTEXT
```

4. 接口中正常调用方法即可。