SỬ DỤNG gRPC TRONG PROJECT JAVA

“Đừng so sánh bản thân với người khác, nếu bạn làm vậy thì bạn đang hạ thấp bản thân"

Bill Gates

I. CHUẨN BỊ SOURCE VÀ LIBRARY

1. Build protobuf và gRPC từ file .proto

- Giả sử ta có file helloworld.proto với nội dung như sau:

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "io.grpc.examples.helloworld";
option java_outer_classname = "HelloWorldProto";
option objc_class_prefix = "HLW";

package helloworld;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Hình 1: Cấu trúc file helloworld.proto

- Thực hiện build protobuf và gRPC theo hướng dẫn, ta được các file:
  + GreeterGrpc.java
  + HelloReply.java
  + HelloReplyOrBuilder.java

+ HelloRequest.java
+ HelloRequestOrBuilder.java
+ HelloWorldProto.java

2. Chuẩn bị các library cần thiết

- Đối với Protobuf Compiler, cần kiểm tra version đã cài trên máy để tải library phù hợp. Giả sử trên máy đang cài **Protobuf Compiler version 3.1.0** thì tải file **protobuf-java-3.1.0.jar**.
- Đối với gRPC, ta xem tại trang hướng dẫn chính thức Google về gRPC Java (http://www.grpc.io/docs/quickstart/java.html).

  Hiện tại, theo hướng dẫn, ta sẽ được dẫn đến trang https://github.com/grpc/grpc-java

  Download tất cả các file JAR theo hướng dẫn:

Hình 2: Download các file JAR

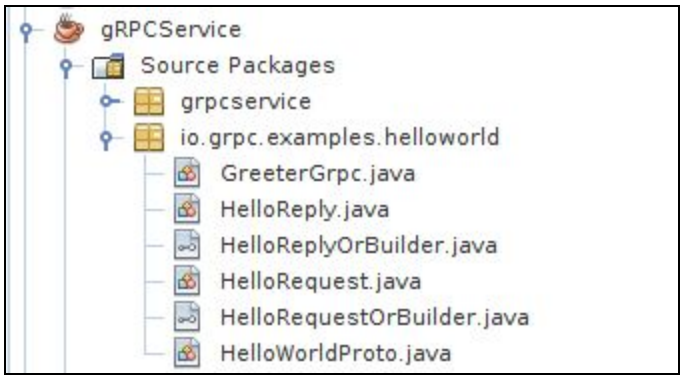
Ngoài ra, để có thể chạy được một client hoặc server gRPC, ta còn cần các thư viện: **guava, netty** và **netty-tcnative-boringssl-static** (nếu cần dùng SSL).

Tại đây, có cung cấp sẵn plugin gencode cho gRPC Java, ta có thể tải về và sử dụng mà không cần build plugin từ source.

II. TẠO gRPC SERVER

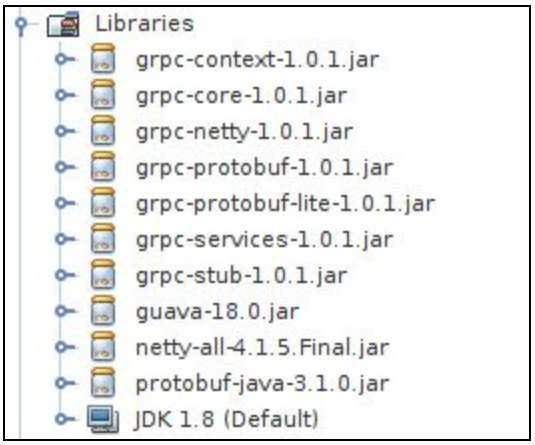
1. Tạo project

- Tạo một project Java trên Netbeans, giả sử đặt tên project là **gRPCService**.
- Import các file .java đã build được từ file .proto vào project. Lưu ý: phải thêm các file này vào đúng package như đã khai báo trong option **java_pakage** của file helloworld.proto.



Hình 3: Cấu trúc project

- Import các file JAR cần thiết của grpc vào project. Lưu ý: nếu cần sử dụng kết nối SSL thì thêm netty-tcnative-boringssl-static.



Hình 4: Các file JAR

2. Implement

- Extends class GreeterImplBase và Override các phương thức cần thiết để handle request từ client.
- Lưu ý: HelloRequest là protobuf do Client gửi lên, HelloReply là protobuf mà client sẽ nhận về. Như vậy client chỉ cần thao tác với hai đối tượng trên mà không cần quan tâm nhiều đến hoạt động network.

```
public class GreeterGrpcHandler extends GreeterGrpc.GreeterImplBase {

    @Override
    final public void sayHello(HelloRequest request,
            StreamObserver<HelloReply> responseObserver) {

        HelloReply reply = this.handleRequest(request);

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

Hình 5: Extends Class GreeterImplBase

3. Start server

Sử dụng ServerBuilder để build và start server. Cần chỉ định rõ port listen và service mà server có thể xử lý.

```
public class ZServer {

    private Server server;
    private GreeterGrpcHandler requestHandler;

    private void start() throws IOException {
        server = ServerBuilder.forPort(8079)
                .addService(requestHandler)
                .build()
                .start();
    }
```

Hình 6: Build và start server

III. TẠO gRPC CLIENT

1. Tạo project

Tạo mới project và import các class cùng các file JAR tương tự server. Giả sử đặt trên project là **gRPCStub**.

2. Implement

- Tạo channel và stub để gọi các service của server. Cần chỉ định rõ host, port. Mặc định, gRPC sử dụng SSL, do đó để nhận dữ liệu dạng thường ta cần thêm chỉ định **usePlainText(true)**

```
public class ZStub {

    private final ManagedChannel channel;
    private final GreeterBlockingStub blockingStub;

    public ZStub() throws Exception {
        channel = ManagedChannelBuilder.forAddress("localhost", 8079)
                    .usePlaintext(true)
                    .build();
        blockingStub = GreeterGrpc.newBlockingStub(channel);
    }
```

Hình 7: Tạo Stub gRPC

- Client sử dụng giao thức SSL.

```
public class ZStub {

    private final ManagedChannel channel;
    private final GreeterBlockingStub blockingStub;

    public ZStub() throws Exception {
        channel = ManagedChannelBuilder.forAddress("localhost", 8079)
                    .build();

        blockingStub = GreeterGrpc.newBlockingStub(channel);
    }
```

Hình 8: Tạo Stub sử dụng giao thức SSL

- Gọi phương thức của service thông qua stub đã tạo và đợi để nhận response hoặc exception.

```
public HelloReply sayHello() {
    if (channel.isShutdown() || channel.isTerminated()) {
        System.err.println("Connection is closed. Reconnecting ...");
        return null;
    }

    HelloRequest request = HelloRequest.newBuilder()
            .setName("grpcStub").build();

    try {
        HelloReply response = blockingStub.sayHello(request);
        if (response == null) {
            System.err.println("ERR: get response null");
        }
        return response;
    } catch (StatusRuntimeException e) {
        System.err.println("ERR: RPC throw exception: " + e.getStatus());
    } catch (Exception ex) {
        System.err.println("ERR: process response string throw exception: "
                + ex.getMessage());
    }
    return null;
}
```

Hình 9: Gọi phương thức của service thông qua stub