Java-grpc 开发与配置

- 环境：Linux Centos7
- 安装 protobuf 相关库，https://github.com/google/protobuf.git

  安装完成之后在终端输入 protoc –version，如果有返回，说明安装成功，文档当前使用版本是 3.5.1。
- 编写 proto 文件，详细写法请查看相关文档
- 如果使用 Eclipse+Maven 插件的方式(推荐)
  - 新建一个 maven 工程，然后在 main 目录下创建一个资源目录 proto，里面存放编写号的 proto 文件。
  - 在 pom 文件中设置，具体查看文档附录。
  - 右键工程 run as 选择 maven install 运行后，会在 target 目录下生成对应的代码文件。

- 如果不想使用 maven 插件
  - 则需要先下载 grpc-java，地址 https://github.com/grpc/grpc-java.git
  - 解压缩户，进入目录，在进入 compile 目录，然后运行命令../gradlew java_pluginExecutable 进行编译。
  - 编译后会生成 protoc-gen-grpc-java 程序
  - 输入命令 protoc –plugin=protoc-gen-grpc-java={插件位置} -I{包含所有相关 proto 文件的目录} –grpc-java_out={生成文件目标位置} {proto 文件}

- 如果只是客户端，则不需要任何实现，直接调用即可，如果是服务端，则需要实现服务接口的业务逻辑，例子见附录

附录一：pom 文件

```
<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.rouies</groupId>
  <artifactId>etcd</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-all</artifactId>
            <version>1.12.0</version>
        </dependency>
  </dependencies>
  <build>
                <extensions>
                  <extension>
                    <groupId>kr.motd.maven</groupId>
                    <artifactId>os-maven-plugin</artifactId>
                    <version>1.5.0.Final</version>
                  </extension>
                </extensions>
      <plugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.5.1</version>
            <configuration>
              <source>1.8</source>
              <target>1.8</target>
            </configuration>
          </plugin>
           <plugin>
                          <groupId>org.xolstice.maven.plugins</groupId>
                          <artifactId>protobuf-maven-plugin</artifactId>
                          <version>0.5.1</version>
                          <configuration>
                            <protocArtifact>com.google.protobuf:protoc:3.5.1-
1:exe:${os.detected.classifier}</protocArtifact>
```

```
                                        <pluginId>grpc-java</pluginId>
                                        <pluginArtifact>io.grpc:protoc-gen-grpc-
java:1.12.0:exe:${os.detected.classifier}</pluginArtifact>

<protocExecutable>/usr/local/bin/protoc</protocExecutable>
                                      </configuration>
                                      <executions>
                                        <execution>
                                          <goals>
                                            <goal>compile</goal>
                                            <goal>compile-custom</goal>
                                          </goals>
                                        </execution>
                                      </executions>
                                    </plugin>
        </plugins>
    </build>
</project>
```

附录 2：客户端与服务器的代码

服务端：

```
package com.rouies.etcd;

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

import java.io.IOException;


public class TestServer {


        private class GreeterImpl extends GreeterGrpc.GreeterImplBase{
                @Override
                public void test(EtcdRequest request,
                                StreamObserver<EtcdResponse> responseObserver) {
                        String name = request.getName();
```

```
				System.out.println(name);
				EtcdResponse response = 
EtcdResponse.newBuilder().setMessage("hello," + name).build();
				responseObserver.onNext(response);
				responseObserver.onCompleted();

			}
		}


		private int port = 50051;
		private Server server;

		private void start() throws IOException {
			server = ServerBuilder.forPort(port)
					.addService(new GreeterImpl())
					.build()
					.start();

			Runtime.getRuntime().addShutdownHook(new Thread() {
				@Override
				public void run() {
					// Use stderr here since the logger may have been reset by its 
JVM shutdown hook.
					System.err.println("*** shutting down gRPC server since JVM 
is shutting down");
					TestServer.this.stop();
					System.err.println("*** server shut down");
				}
			});
		}

		private void stop() {
			if (server != null) {
				server.shutdown();
			}
		}

		/**
		 * Await termination on the main thread since the grpc library uses daemon threads.
		 */
		private void blockUntilShutdown() throws InterruptedException {
			if (server != null) {
				server.awaitTermination();
			}
		}

		/**
```

```
          * Main launches the server from the command line.
          */
         public static void main(String[] args) throws IOException, InterruptedException {
                   final TestServer server = new TestServer();
                   server.start();
                   server.blockUntilShutdown();
         }
}
```

客户端:

```
package com.rouies.etcd;

import java.util.concurrent.TimeUnit;

import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.StatusRuntimeException;

public class TestClient {
         private final ManagedChannel channel;
   private final GreeterGrpc.GreeterBlockingStub blockingStub;

   public TestClient(String host, int port) {
      channel = ManagedChannelBuilder.forAddress(host, port)
      .~~usePlaintext(true)~~
      .build();
      blockingStub = GreeterGrpc.newBlockingStub(channel);
   }

   public void shutdown() throws InterruptedException {
      channel.shutdown().awaitTermination(5, TimeUnit.SECONDS);
   }
   /** Say hello to server. */
   public String greet(String name) {

      EtcdRequest request = EtcdRequest.newBuilder().setName(name).build();
      EtcdResponse response;
      try {
         response = blockingStub.test(request);
         return response.getMessage();
      } catch (StatusRuntimeException e) {
         return null;
      }
   }
```

```
  /**
   * Greet server. If provided, the first element of {@code args} is the name to use in the
   * greeting.
   */
  public static void main(String[] args) throws Exception {
    TestClient client = new TestClient("localhost", 50051);
    try {
      String res = client.greet("zhangsan");
      System.out.println(res);
    } finally {
      client.shutdown();
    }
  }
}
```

附录三：proto 文件

```
syntax = "proto3";

option java_outer_classname = "EtcdClient";
option java_package = "com.rouies.etcd";
option java_multiple_files = true;

package etcd;

service Greeter {
  rpc Test (EtcdRequest) returns (EtcdResponse) {}
}

message EtcdRequest {
  string name = 1;
}

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