Dubbo3.0入门教程与新特性介绍

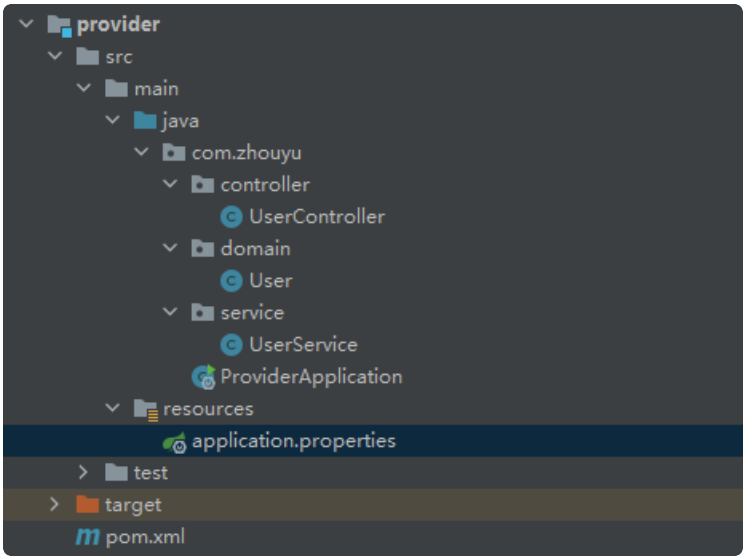
图灵学院–周瑜，大都督

如何将一个应用改造为一个Dubbo应用

首先，新建两个SpringBoot项目，一个叫consumer，一个叫provider

provider项目

项目结构



pom文件

Java 复制代码

```java
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>2.6.6</version>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.22</version>
    </dependency>
</dependencies>
```

ProviderApplication

Java 复制代码

```java
@SpringBootApplication
public class ProviderApplication {

    public static void main(String[] args) {
        SpringApplication.run(ProviderApplication.class);
    }
}
```

User

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
    private String uid;
    private String username;
}
```

UserService

```java
@Service
public class UserService {

    public User getUser(String uid) {
        User zhouyu = new User(uid, "zhouyu");
        return zhouyu;
    }
}
```

UserController

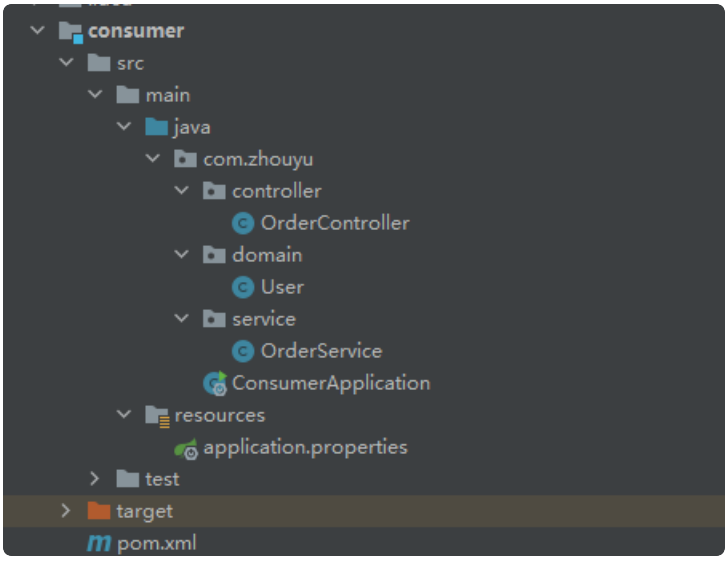
```java
@RestController
public class UserController {

    @Resource
    private UserService userService;

    @GetMapping("/user/{uid}")
    public User getUser(@PathVariable("uid") String uid) {
        return userService.getUser(uid);
    }
}
```

application.properties

```
server.port=8080
```

consumer项目

项目结构



pom文件

Java 复制代码

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>2.6.6</version>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.22</version>
    </dependency>
</dependencies>
```

ConsumerApplication

Java 复制代码

```
@SpringBootApplication
public class ConsumerApplication {

    @Bean
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }

    public static void main(String[] args) {
        SpringApplication.run(ConsumerApplication.class);
    }
}
```

User

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
    private String uid;
    private String username;
}
```

OrderService

```java
@Service
public class OrderService {

    @Resource
    private RestTemplate restTemplate;

    public String createOrder(){
        User user = restTemplate.getForObject("http://localhost:8080/user/1", User.class);
        System.out.println("创建订单");

        return user.toString()+" succeeded in creating the order";
    }
}
```

OrderController

```java
@RestController
public class OrderController {

    @Resource
    private OrderService orderService;

    @GetMapping("/createOrder")
    public String createOrder() {
        return orderService.createOrder();
    }
}
```

application.properties

```java
server.port=8081
```

consumer中的OrderService会通过RestTemplate调用provider中的UserService。

改造成Dubbo

改造成Dubbo项目，有几件事情要做：

1. 添加dubbo核心依赖
2. 添加要使用的注册中心依赖
3. 添加要使用的协议的依赖
4. 配置dubbo相关的基本信息
5. 配置注册中心地址
6. 配置所使用的协议

增加依赖

Java 复制代码

```
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-spring-boot-starter</artifactId>
    <version>3.0.7</version>
</dependency>

<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-rpc-dubbo</artifactId>
    <version>3.0.7</version>
</dependency>

<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo-registry-zookeeper</artifactId>
    <version>3.0.7</version>
</dependency>
```

配置properties

Java 复制代码

```
dubbo.application.name=provider-application

dubbo.protocol.name=dubbo
dubbo.protocol.port=20880

dubbo.registry.address=zookeeper://127.0.0.1:2181
```

改造服务

consumer和provider中都用到了User类，所以可以单独新建一个maven项目用来存consumer和provider公用的一些类，新增一个common模块，把User类转移到这个模块中

要改造成Dubbo，得先抽象出来服务，用接口表示。

像UserService就是一个服务，不过我们得额外定义一个接口，我们把之前的UserService改为UserServiceImpl，然后新定义一个接口UserService，该接口表示一个服务，UserServiceImpl为该服务的具体实现。

Java 复制代码

```java
public interface UserService {

    public User getUser(String uid);
}
```

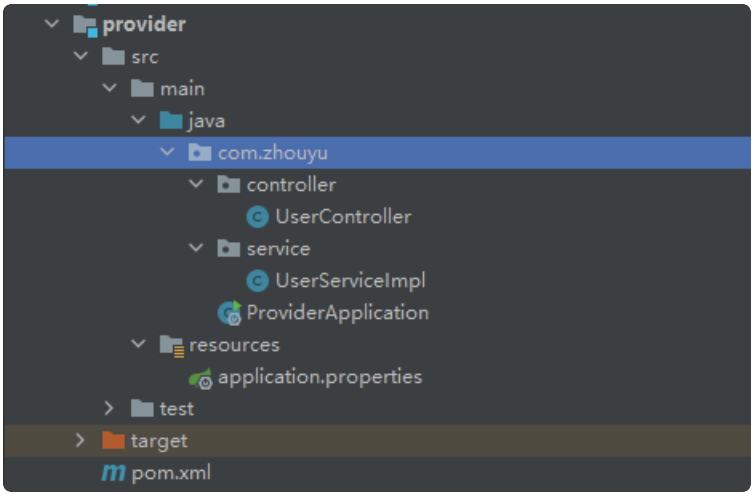
Java 复制代码

```java
@DubboService
public class UserServiceImpl implements UserService {

    public User getUser(String uid) {
        User zhouyu = new User(uid, "zhouyu");
        return zhouyu;
    }
}
```

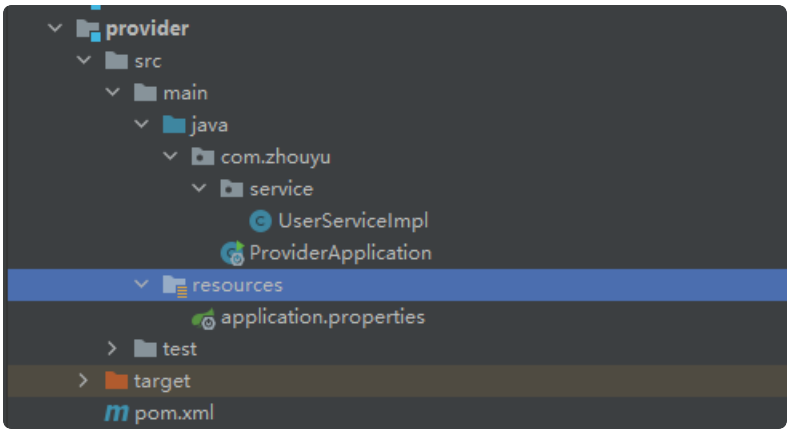
注意：要把Spring中的@Service注解替换成Dubbo中的@DubboService注解。

然后把UserService接口也转移到common模块中去，在provider中依赖common。

改造之后的provider为：



其实UserController也可以去掉，去掉之后provider就更加简单了



此时就可以启动该Provider了，注意先启动zookeeper（**高版本的Zookeeper启动过程中不仅会占用2181，也会占用8080，所以可以把provider的端口改为8082**）

开启Dubbo

在ProviderApplication上加上**@EnableDubbo(scanBasePackages = "com.zhouyu.service")**，表示Dubbo会去扫描某个路径下的@DubboService，从而对外提供该Dubbo服务。

```java
@SpringBootApplication
@EnableDubbo(scanBasePackages = "com.zhouyu.service")
public class ProviderApplication {

    public static void main(String[] args) {
        SpringApplication.run(ProviderApplication.class);
    }
}
```

调用Dubbo服务

引入依赖

在consumer中如果想要调用Dubbo服务，也要引入相关的依赖：

1. 引入common，主要是引入要用调用的接口
2. 引入dubbo依赖
3. 引入需要使用的协议的依赖
4. 引入需要使用的注册中心的依赖

```java
        <dependency>
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo-spring-boot-starter</artifactId>
            <version>3.0.7</version>
        </dependency>

        <dependency>
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo-rpc-dubbo</artifactId>
            <version>3.0.7</version>
        </dependency>

        <dependency>
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo-registry-zookeeper</artifactId>
            <version>3.0.7</version>
        </dependency>

        <dependency>
            <groupId>com.zhouyu</groupId>
            <artifactId>common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
```

引入服务

通过@DubboReference注解来引入一个Dubbo服务。

```java
@Service
public class OrderService {

    @DubboReference
    private UserService userService;

    public String createOrder(){
        User user = userService.getUser("1");
        System.out.println("创建订单");

        return user.toString()+" succeeded in creating the order";
    }
}
```

这样就不需要用RestTemplate了。

配置properties

```java
dubbo.application.name=consumer-application
dubbo.registry.address=zookeeper://127.0.0.1:2181
```

调用

如果User没有实现Serializable接口，则会报错。

总结

自此，Dubbo的改造就完成了，总结一下：

1. 添加pom依赖
2. 配置dubbo应用名、协议、注册中心

3. 定义服务接口和实现类
4. 使用@DubboService来定义一个Dubbo服务
5. 使用@DubboReference来使用一个Dubbo服务
6. 使用@EnableDubbo开启Dubbo

Dubbo3.0新特性介绍

注册模型的改变

在服务注册领域，市面上有两种模型，一种是应用级注册，一种是接口级注册，在Spring Cloud中，一个应用是一个微服务，而在Dubbo2.7中，一个接口是一个微服务。

所以，Spring Cloud在进行服务注册时，是把应用名以及应用所在服务器的IP地址和应用所绑定的端口注册到注册中心，相当于key是应用名，value是ip+port，而在Dubbo2.7中，是把接口名以及对应应用的IP地址和所绑定的端口注册到注册中心，相当于key是接口名，value是ip+port。

所以在Dubbo2.7中，一个应用如果提供了10个Dubbo服务，那么注册中心中就会存储10对keyvalue，而Spring Cloud就只会存一对keyvalue，所以以Spring Cloud为首的应用级注册是更加适合的。

所以Dubbo3.0中将注册模型也改为了应用级注册，提升效率节省资源的同时，通过统一注册模型，也为各个微服务框架的互通打下了基础。

新一代RPC协议

定义了全新的 RPC 通信协议 — Triple，一句话概括 Triple：它是基于 HTTP/2 上构建的 RPC 协议，完全兼容 gRPC，并在此基础上扩展出了更丰富的语义。 使用 Triple 协议，用户将获得以下能力

- 更容易到适配网关、Mesh架构，Triple 协议让 Dubbo 更方便的与各种网关、Sidecar 组件配合工作。
- 多语言友好，推荐配合 Protobuf 使用 Triple 协议，使用 IDL 定义服务，使用 Protobuf 编码业务数据。
- 流式通信支持。Triple 协议支持 Request Stream、Response Stream、Bi-direction Stream

当使用Triple协议进行RPC调用时，支持多种方式来调用服务，只不过在服务接口中要定义不同的方法，比如：

```java
public interface DemoService {

    // UNARY
    String sayHello(String name);

    // SERVER_STREAM
    default void sayHelloServerStream(String name, StreamObserver<String> response) {
    }

    // CLIENT_STREAM / BI_STREAM
    default StreamObserver<String> sayHelloStream(StreamObserver<String> response) {
        return response;
    }

}
```

UNARY

unary，就是正常的调用方法

服务实现类对应的方法：

```java
// UNARY
@Override
public String sayHello(String name) {
    return "Hello " + name;
}
```

服务消费者调用方式：

```java
String result = demoService.sayHello("zhouyu");
```

SERVER_STREAM

服务实现类对应的方法：

```java
// SERVER_STREAM
@Override
public void sayHelloServerStream(String name, StreamObserver<String>
response) {
    response.onNext(name + " hello");
    response.onNext(name + " world");
    response.onCompleted();
}

```

服务消费者调用方式：

```java
demoService.sayHelloServerStream("zhouyu", new StreamObserver<String>() {
    @Override
    public void onNext(String data) {
        // 服务端返回的数据
        System.out.println(data);
    }

    @Override
    public void onError(Throwable throwable) {}

    @Override
    public void onCompleted() {
        System.out.println("complete");
    }
});
```

CLIENT_STREAM

服务实现类对应的方法：

Java | 复制代码

```java
// CLIENT_STREAM
@Override
public StreamObserver<String> sayHelloStream(StreamObserver<String>
response) {
    return new StreamObserver<String>() {
        @Override
        public void onNext(String data) {
            // 接收客户端发送过来的数据，然后返回数据给客户端
            response.onNext("result: " + data);
        }

        @Override
        public void onError(Throwable throwable) {}

        @Override
        public void onCompleted() {
            System.out.println("completed");
        }
    };
}
```

服务消费者调用方式：

```java
StreamObserver<String> streamObserver = demoService.sayHelloStream(new StreamObserver<String>() {
    @Override
    public void onNext(String data) {
        System.out.println("接收到响应数据: "+ data);
    }

    @Override
    public void onError(Throwable throwable) {}

    @Override
    public void onCompleted() {
        System.out.println("接收到响应数据完毕");
    }
});

// 发送数据
streamObserver.onNext("request zhouyu hello");
streamObserver.onNext("request zhouyu world");
streamObserver.onCompleted();
```

BI_STREAM

和CLIENT_STREAM一样

Dubbo3.0跨语言调用

在工作中，我们用Java语言通过Dubbo提供了一个服务，另外一个应用（也就是消费者）想要使用这个服务，如果消费者应用也是用Java语言开发的，那没什么好说的，直接在消费者应用引入Dubbo和服务接口相关的依赖即可。

但是，如果消费者应用不是用Java语言写的呢，比如是通过python或者go语言实现的，那就至少需要满足两个条件才能调用Java实现的Dubbo服务：

1. Dubbo一开始是用Java语言实现的，那现在就需要一个go语言实现的Dubbo框架，也就是现在的dubbo-go，然后在go项目中引入dubbo-go，从而可以在go项目中使用dubbo，比如使用go语言去

暴露和使用Dubbo服务。

2. 我们在使用Java语言开发一个Dubbo服务时，会把服务接口和相关类，单独抽象成为一个Maven项目，实际上就相当于一个单独的jar包，这个jar能被Java项目所使用，但不能被go项目所使用，所以go项目中该如何使用Java语言所定义的接口呢？直接用是不太可能的，只能通过间接的方式来解决这个问题，除开Java语言之外，那有没有其他技术也能定义接口呢？并且该技术也是Java和go都支持，这就是protobuf。

protobuf

我们可以通过protobuf来定义接口，然后通过protobuf的编译器将接口编译为特定语言的实现。

在provider项目中定义一个userservice.proto文件，路径为src/main/proto/userservice.proto:

Java | 复制代码

```
syntax = "proto3";

package api;

option go_package = "./;api";

option java_multiple_files = true;
option java_package = "com.zhouyu";
option java_outer_classname = "UserServiceProto";

service UserService {
  rpc GetUser (UserRequest) returns (User) {}
}

// The response message containing the greetings
message UserRequest {
  string uid = 1;
}

// The response message containing the greetings
message User {
  string uid = 1;
  string username = 2;
}
```

相当于定义了一个HelloService服务，并且定义了一个getUser方法，接收UserRequest类型的参数，返回User类型的对象。

编译成Java

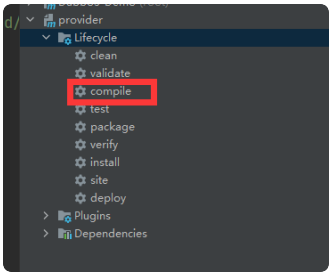
在provider项目中的pom文件中添加相关maven插件：

Java | 复制代码

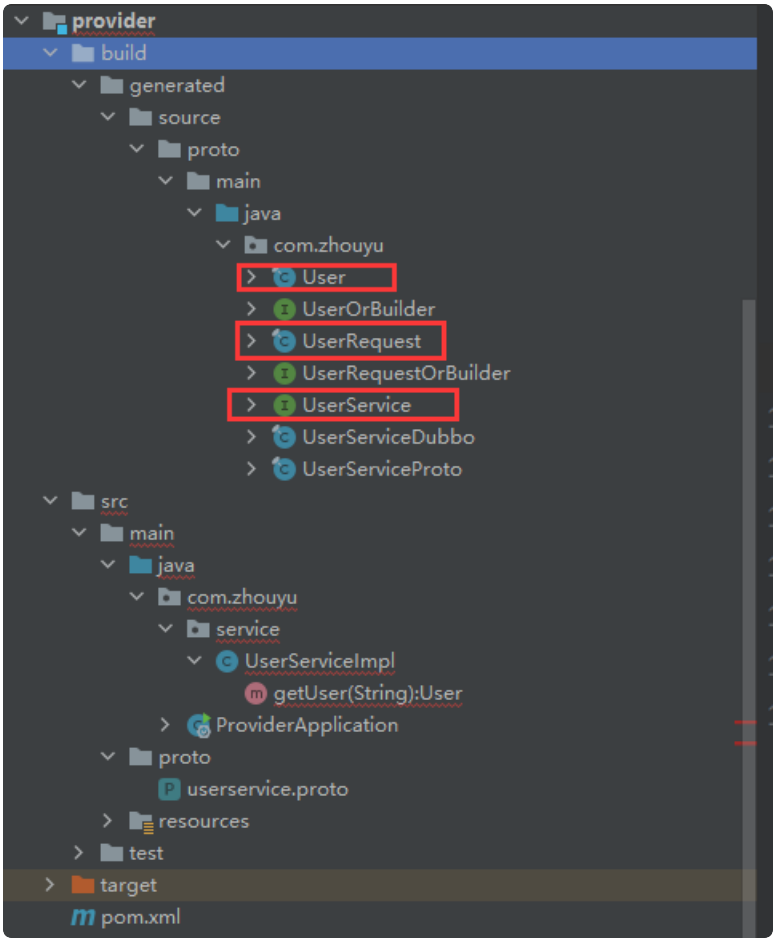
```
<build>
        <extensions>
            <extension>
                <groupId>kr.motd.maven</groupId>
                <artifactId>os-maven-plugin</artifactId>
                <version>1.6.1</version>
            </extension>
        </extensions>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</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>0.6.1</version>
                <configuration>

 <protocArtifact>com.google.protobuf:protoc:3.7.1:exe:${os.detected.classifier}</protocArtifact>

 <outputDirectory>build/generated/source/proto/main/java</outputDirectory>
                    <clearOutputDirectory>false</clearOutputDirectory>
                    <protocPlugins>
                        <protocPlugin>
                            <id>dubbo</id>
                            <groupId>org.apache.dubbo</groupId>
                            <artifactId>dubbo-compiler</artifactId>
                            <version>0.0.3</version>

 <mainClass>org.apache.dubbo.gen.dubbo.Dubbo3Generator</mainClass>
                        </protocPlugin>
                    </protocPlugins>
                </configuration>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
```

```
                                <goal>test-compile</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
                <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>build-helper-maven-plugin</artifactId>
                    <version>3.0.0</version>
                    <executions>
                        <execution>
                            <phase>generate-sources</phase>
                            <goals>
                                <goal>add-source</goal>
                            </goals>
                            <configuration>
                                <sources>

    <source>build/generated/source/proto/main/java</source>
                                </sources>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
```

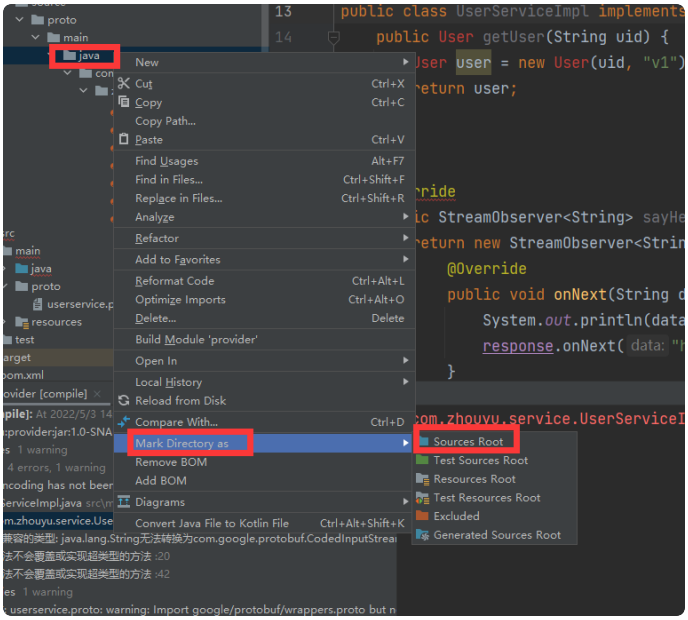
并且可以把common依赖去掉，然后运行provider中lifecycle的compile，就会进行编译了，



并且会编译出对应的接口等信息，编译完成后，会生成一些类：



如果Java没有蓝色的，就



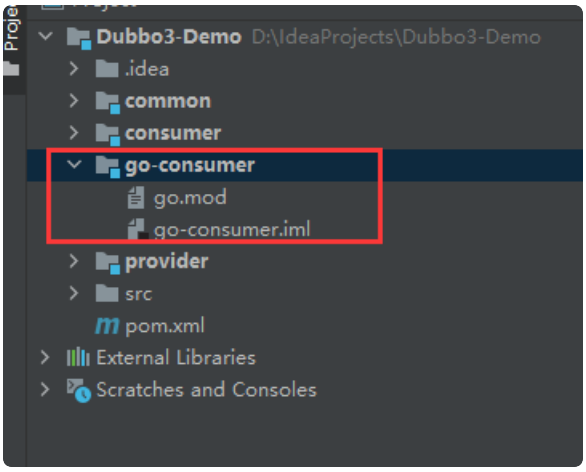
其中就包括了一个UserService接口，所以我们的UserServiceImpl就可以实现这个接口了：

```java
@DubboService
public class UserServiceImpl implements UserService {

    public User getUser(UserRequest userRequest) {
        User user = User.newBuilder().setUid(userRequest.getUid()).setUsername("zhouyu").build();
        return user;
    }

}
```

而对于想要调用UserService服务的消费者而言，其实也是一样的改造，只需要使用同一份userservice.proto进行编译就可以了，比如现在有一个go语言的消费者。

go消费者调用java服务

首先，在IDEA中新建一个go模块：



然后把userservice.proto复制到go-consumer/proto下，然后进行编译，编译成为go语言对应的服务代码，只不过go语言中没有maven这种东西可以帮助我们编译，我们只能用原生的protobuf的编译器进行编译。

这就需要大家在机器上下载、安装protobuf的编译器：protoc

1. 下载地址：https://github.com/protocolbuffers/protobuf/releases/download/v3.20.1/protoc-3.20.1-win64.zip
2. 解压之后，把protoc-3.20.1-win64\bin添加到环境变量中去
3. 在cmd中执行protoc --version，能正常看到版本号即表示安装成功

另外还需要安装go：

1. 下载地址：https://studygolang.com/dl/golang/go1.18.1.windows-amd64.msi
2. 然后直接下一步安装
3. 在cmd中（新开一个cmd窗口）执行go version，能正常看到版本号即表示安装成功

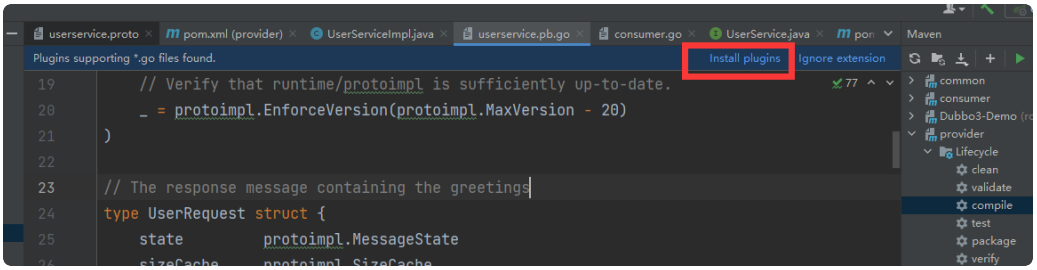
然后在go-consumer下新建文件夹api，进入到go-consumer/proto下，运行：

Java 复制代码

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://goproxy.cn,direct


go get -u github.com/dubbogo/tools/cmd/protoc-gen-go-triple
go install github.com/golang/protobuf/protoc-gen-go
go install github.com/dubbogo/tools/cmd/protoc-gen-go-triple

protoc -I. userservice.proto --go_out=../api --go-triple_out=../api
```

这样就会在go-consumer/api下生成一个userservice.pb.go文件和userservice_triple.pb.go文件

如果IDEA中提示要安装插件，就安装一下：



安装完之后，代码可能会报错，可以在**go-consumer目录**下执行命令下载依赖：

Java 复制代码

```
go mod tidy
```

然后就可以写go语言的服务消费者了，新建一个consumer.go，内容为：

Java | 复制代码

```java
package main

import (
    "context"
    "dubbo.apache.org/dubbo-go/v3/common/logger"
    "dubbo.apache.org/dubbo-go/v3/config"
    _ "dubbo.apache.org/dubbo-go/v3/imports"
    "go-consumer/api"
)

var userServiceImpl = new(api.UserServiceClientImpl)

// export DUBBO_GO_CONFIG_PATH=conf/dubbogo.yml
func main() {
    config.SetConsumerService(userServiceImpl)
    config.Load()

    logger.Info("start to test dubbo")
    req := &api.UserRequest{
        Uid: "1",
    }

    user, err := userServiceImpl.GetUser(context.Background(), req)

    if err != nil {
        logger.Error(err)
    }

    logger.Infof("client response result: %v\n", user)
}

```

然后在go-consumer下新建conf/dubbogo.yml，用来配置注册中心：

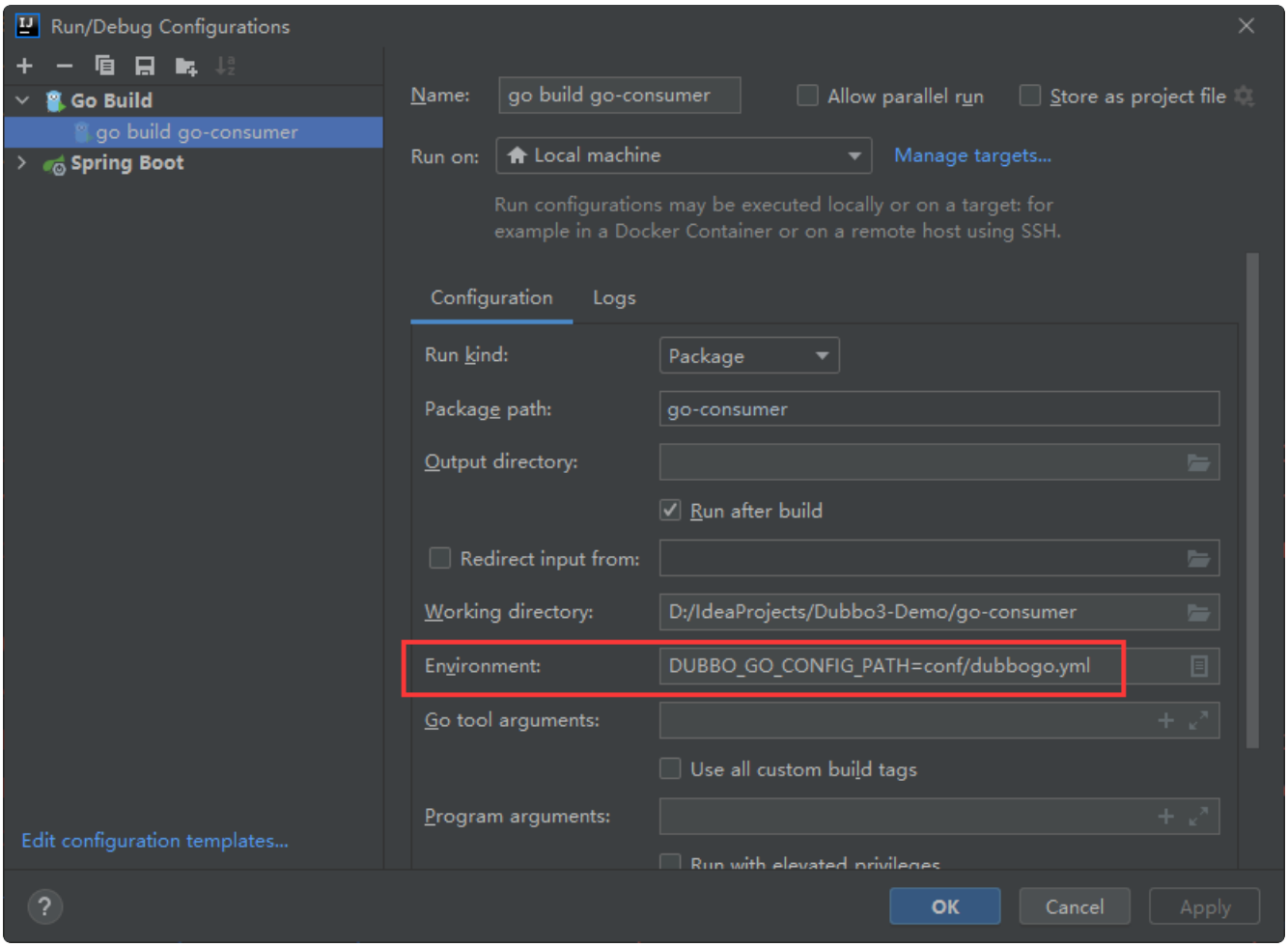
Java | 复制代码

```yaml
dubbo:
  registries:
    demoZK:
      protocol: zookeeper
      address: 127.0.0.1:2181
  consumer:
    references:
      UserServiceClientImpl:
        protocol: tri
        interface: com.zhouyu.UserService
```

注意这里配置的协议为tri，而不是dubbo，在provider端也得把协议改为tri：

```
server.port=8082

dubbo.application.name=provider-application

dubbo.protocol.name=tri
dubbo.protocol.port=20880

dubbo.registry.address=zookeeper://127.0.0.1:2181
```

然后就可以运行consumer.go了，只不过需要在environment中添加一个参数：



运行成功：