

# Cola-Admin 后端文档

---

## 序

---

### 项目地址

- 后端地址: <https://gitee.com/xiaolifeizei/cola-admin>
- 前端地址: <https://gitee.com/xiaolifeizei/cola-ui>

### 在线演示

- 演示地址: <http://www.cola-admin.vip>
- 默认用户: admin
- 默认密码: 123123

### 目录结构

```
1 | cola-admin
2 | └─cola-api           # api接口封装, entity/服务接口
3 | └─cola-common        # 全局服务公共模块
4 | └─cola-service       # 业务服务
5 |   └─cola-service-basics # 项目基础服务（系统服务和基础服务）
6 |   └─cola-service-common # 业务服务公共模块
7 |     └─cola-service-order # 订单服务（空服务）
8 | └─cola-web           # web接口服务
9 |   └─cola-web-auth    # 鉴权模块
10 |   └─cola-web-common  # web接口服务公共模块
11 |     └─cola-web-domain # web接口主服务
12 | └─doc                # 文档及脚本
```

## 环境要求

---

### 基础开发环境

- JDK1.8
- Maven 3.3 +
- Mysql 5.7+
- Redis 4.0+
- Nacos 2.1.0

### IDE插件

- Lombok Plugin (必须要装)

### 推荐IDE

- IntelliJ IDEA

## 环境准备

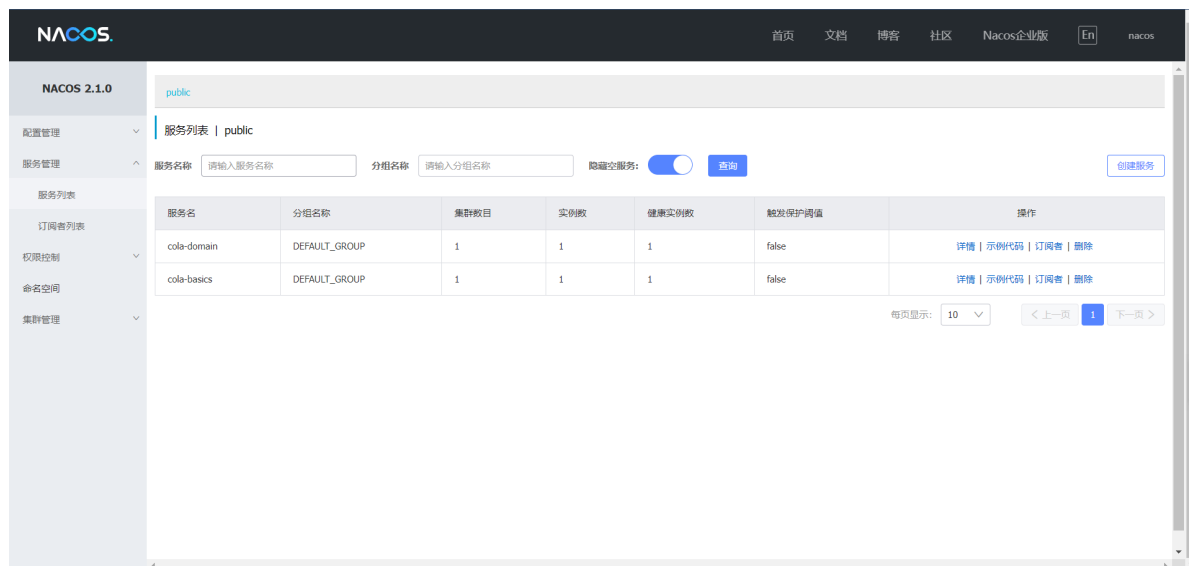
---

# Nacos安装

官方文档: <https://nacos.io/zh-cn/docs/quick-start.html>

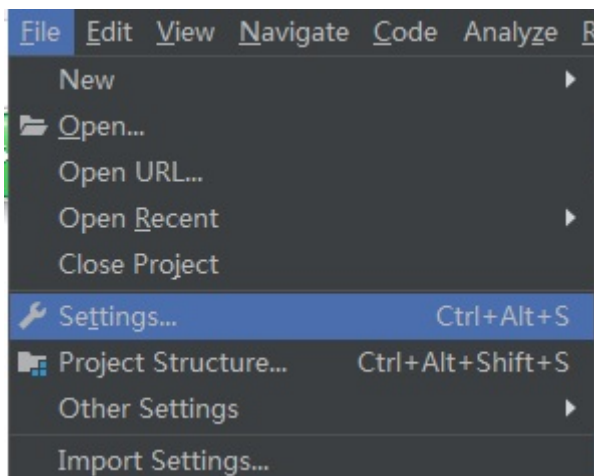
## Nacos界面

默认用户名和密码都是nacos

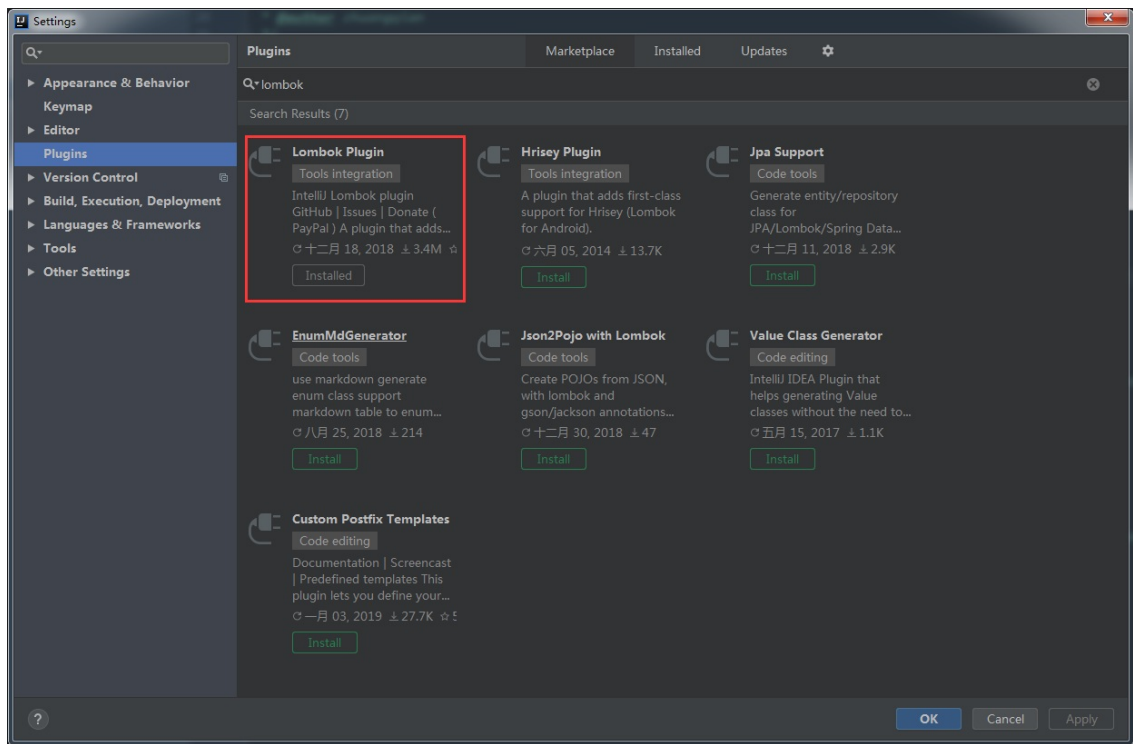


## IDEA 插件安装

1. 选择 File->Settings



2. 选择 Plugins 并搜索 Lombok



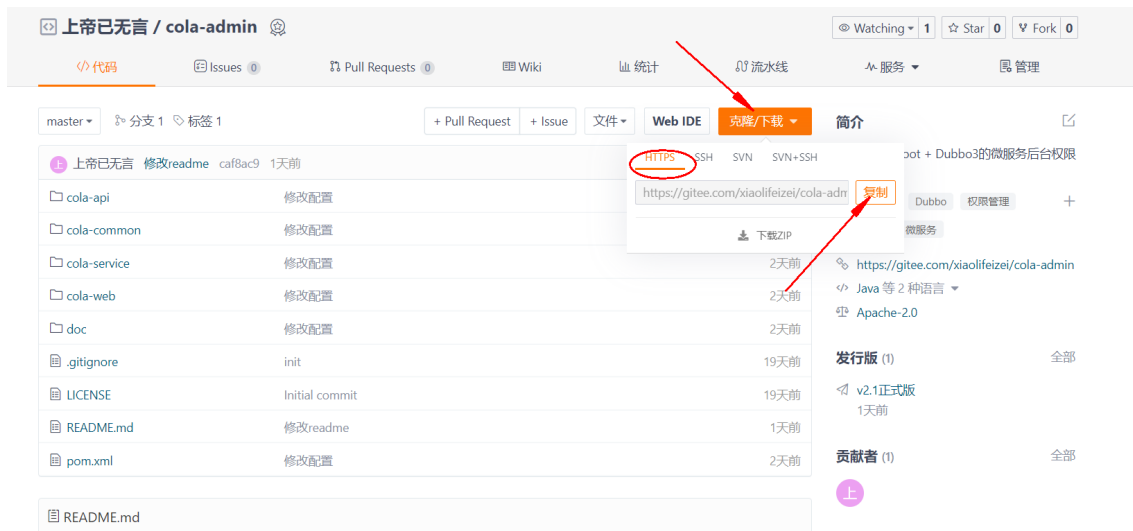
3. 点击 Install 按钮

4. 重启 idea 生效

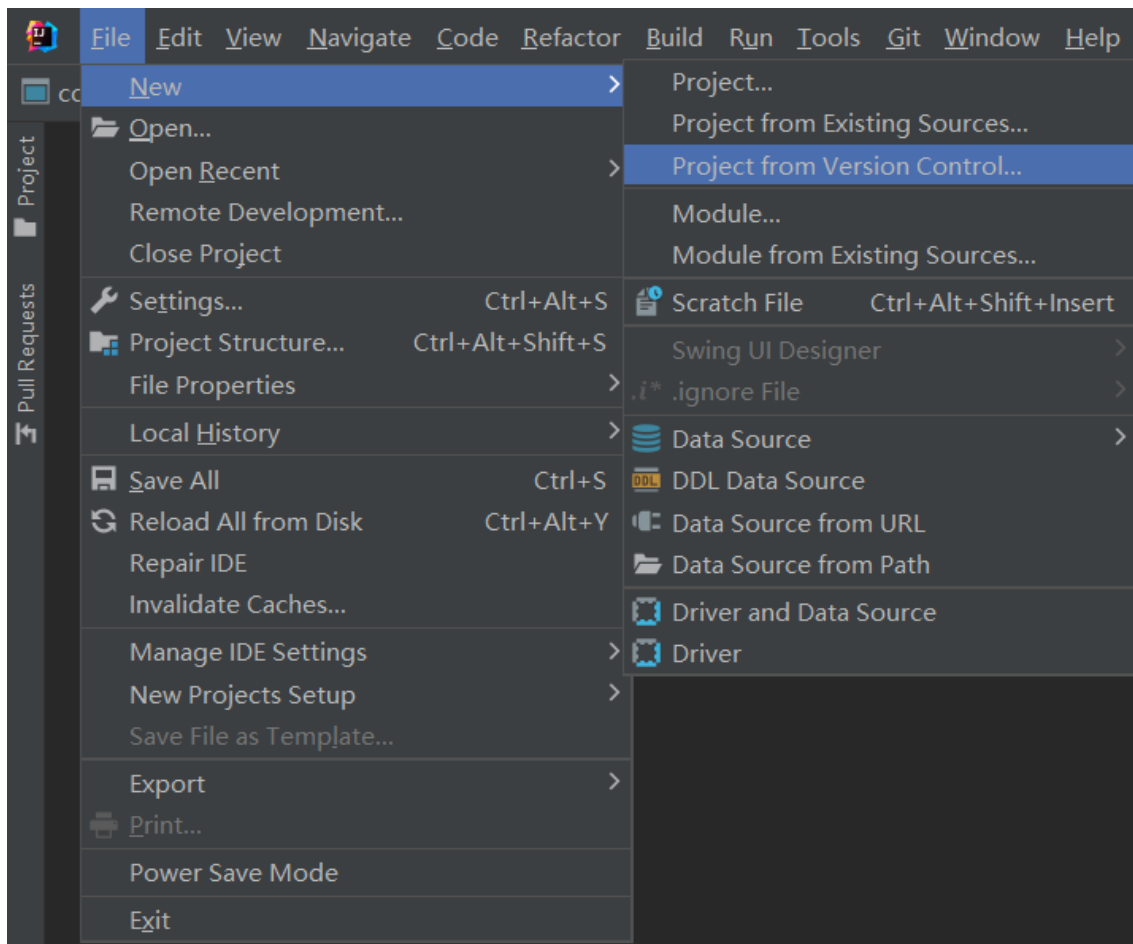
## 导入工程

1. 进入cola-admin项目首页<https://gitee.com/xiaolifeizei/cola-admin>

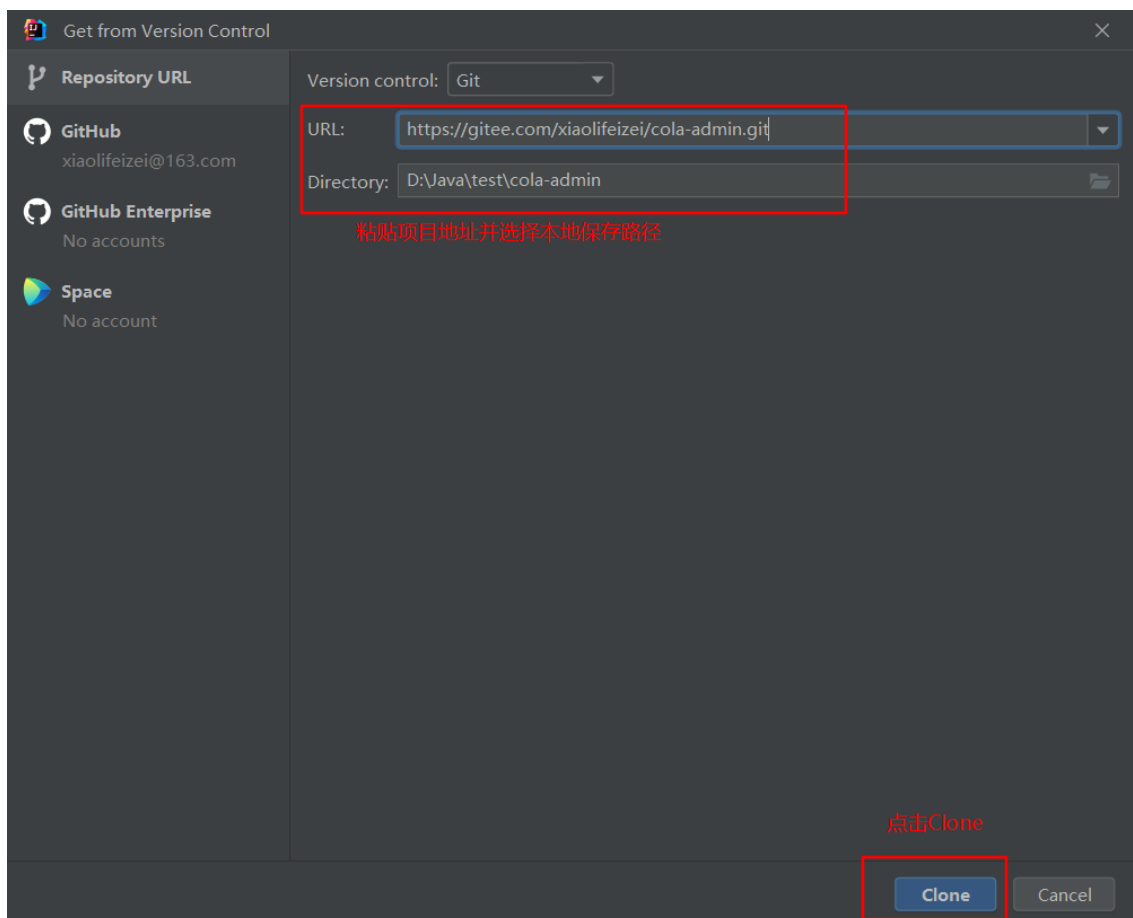
2. 复制项目地址



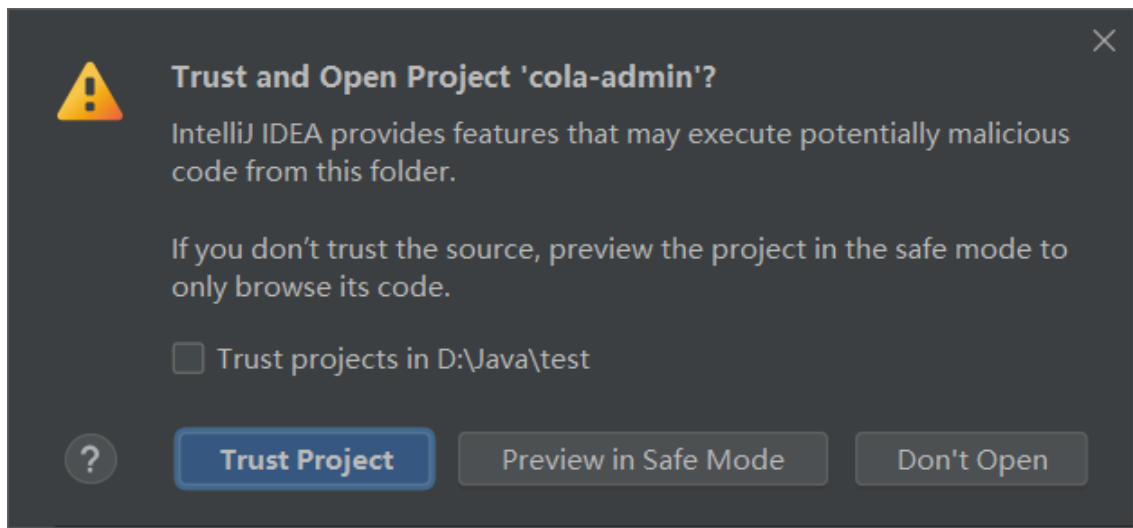
3. 打开IDEA，依次选择：File->New->Project from Version Control



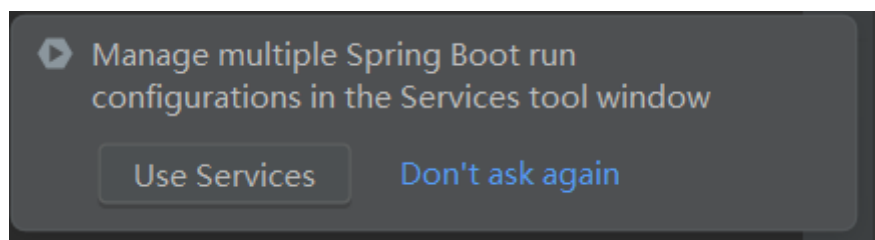
4. 在弹出的对话框中粘贴复制的项目地址



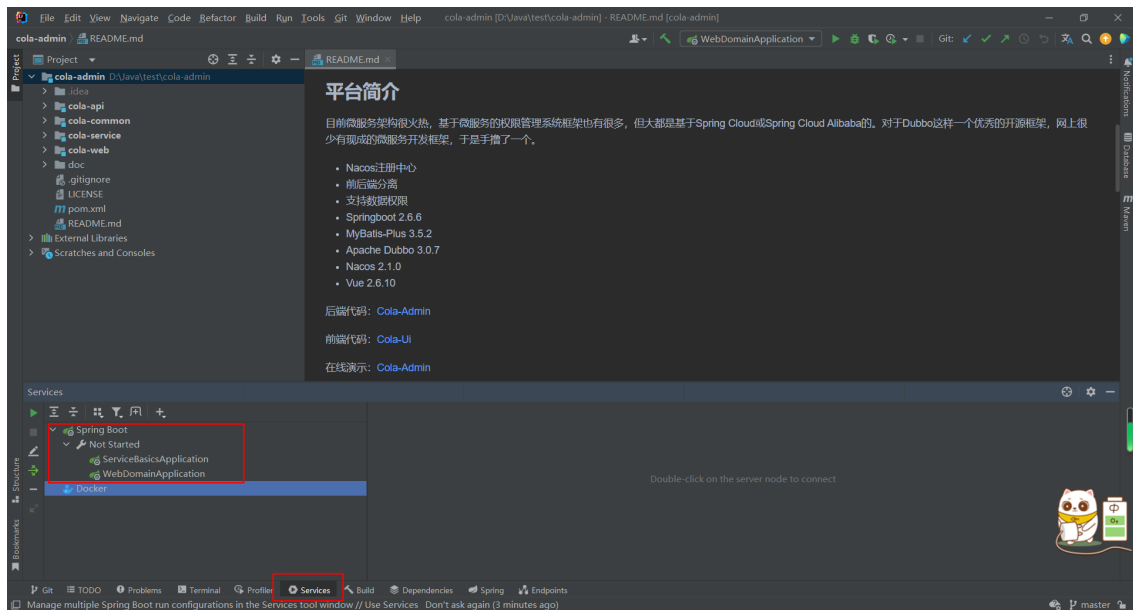
5. IDEA可能会弹出对话框提示，点击“Trust Project”



6. 等待代码下载完成，同时IDEA会自动导入依赖
7. 此时出现“Manage multiple Spring Boot run”对话框，点击Use Services

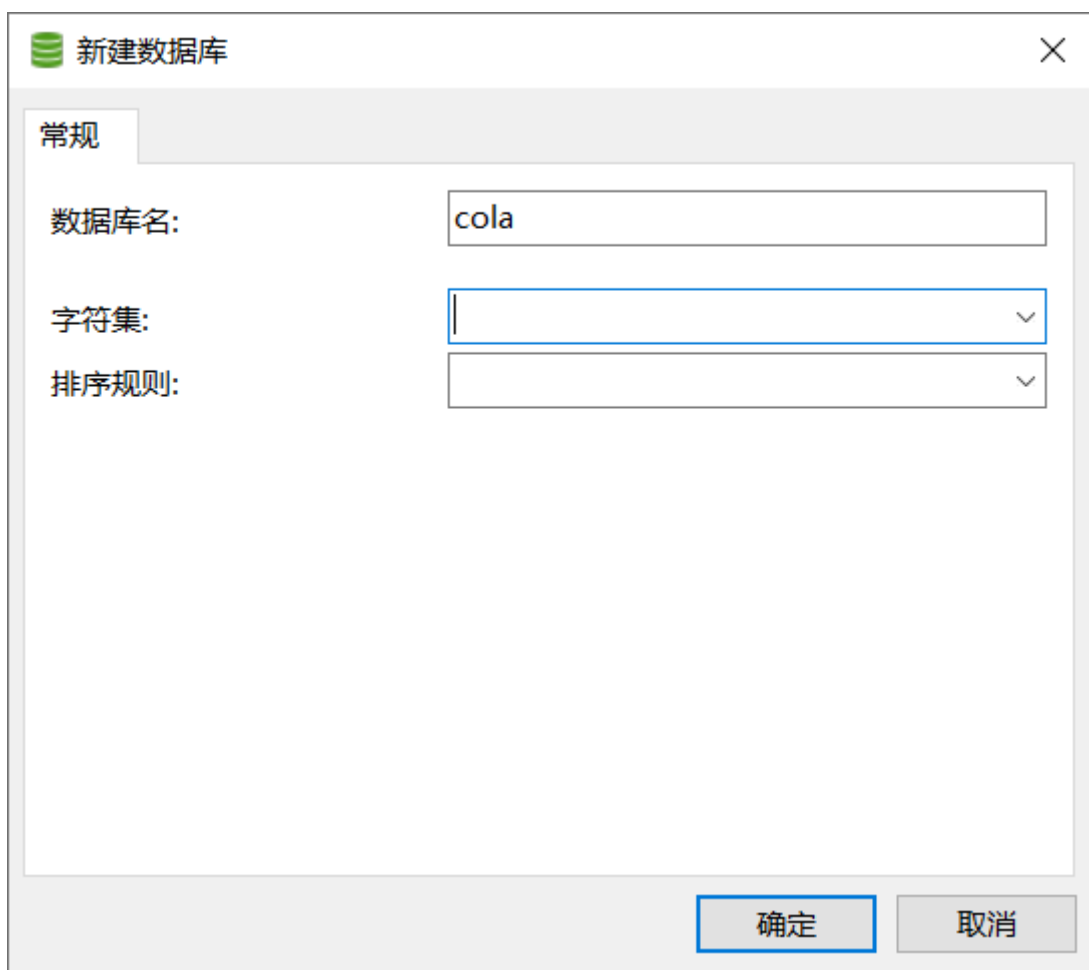
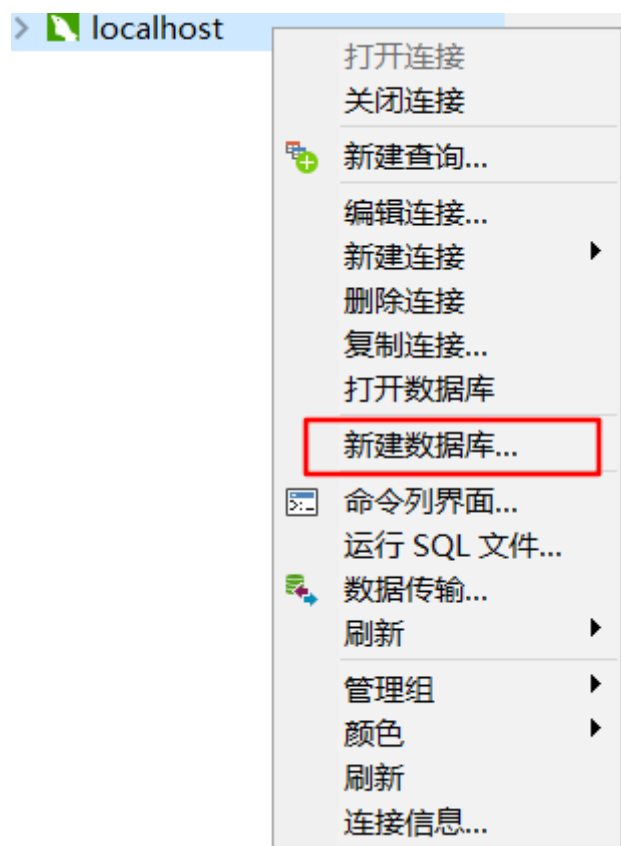


8. 点击Service面板可以看到下图的启动项则说明导入成功

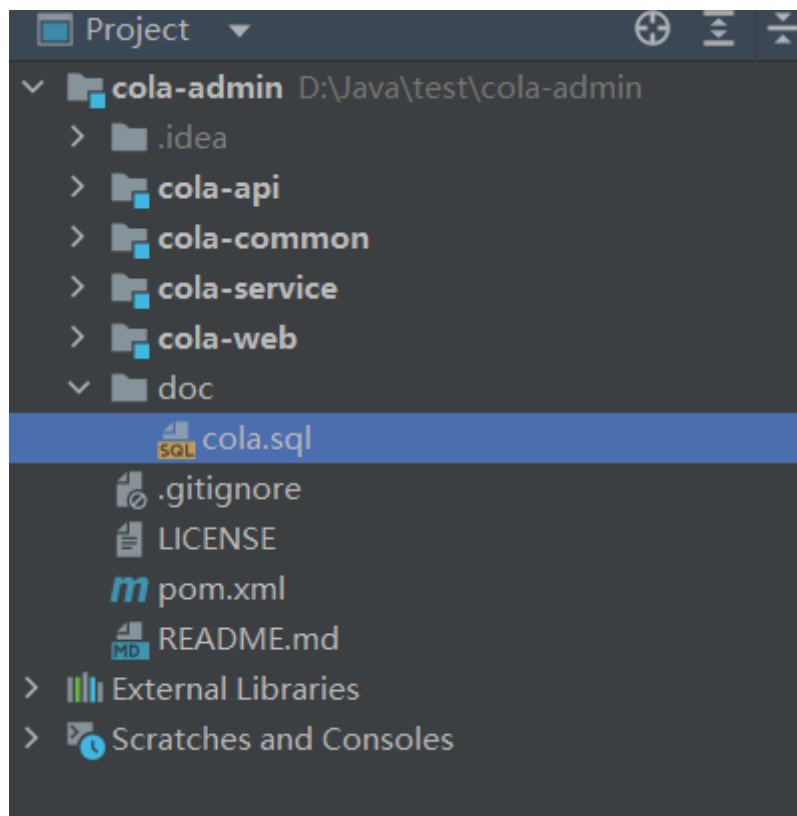


## 创建数据库

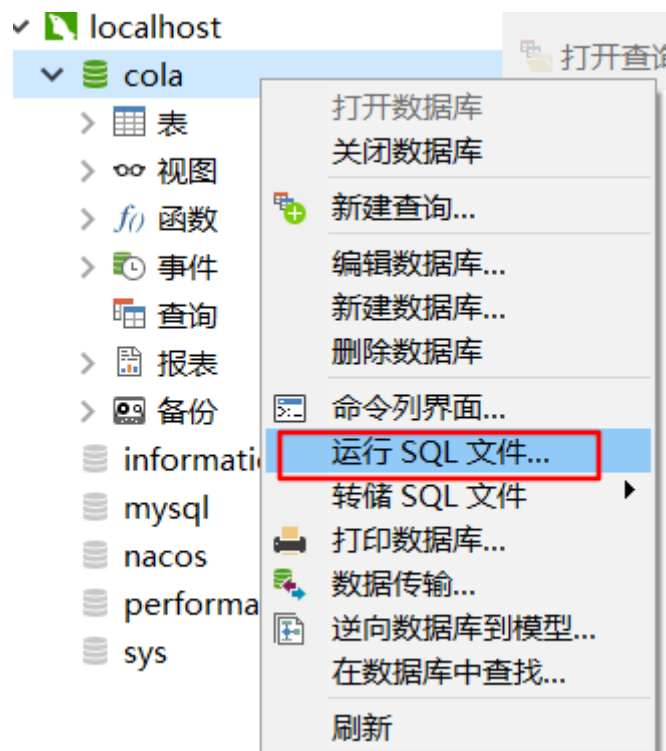
1. 打开Navicat（此处可以选择其他的客户端），新建一个数据库cola

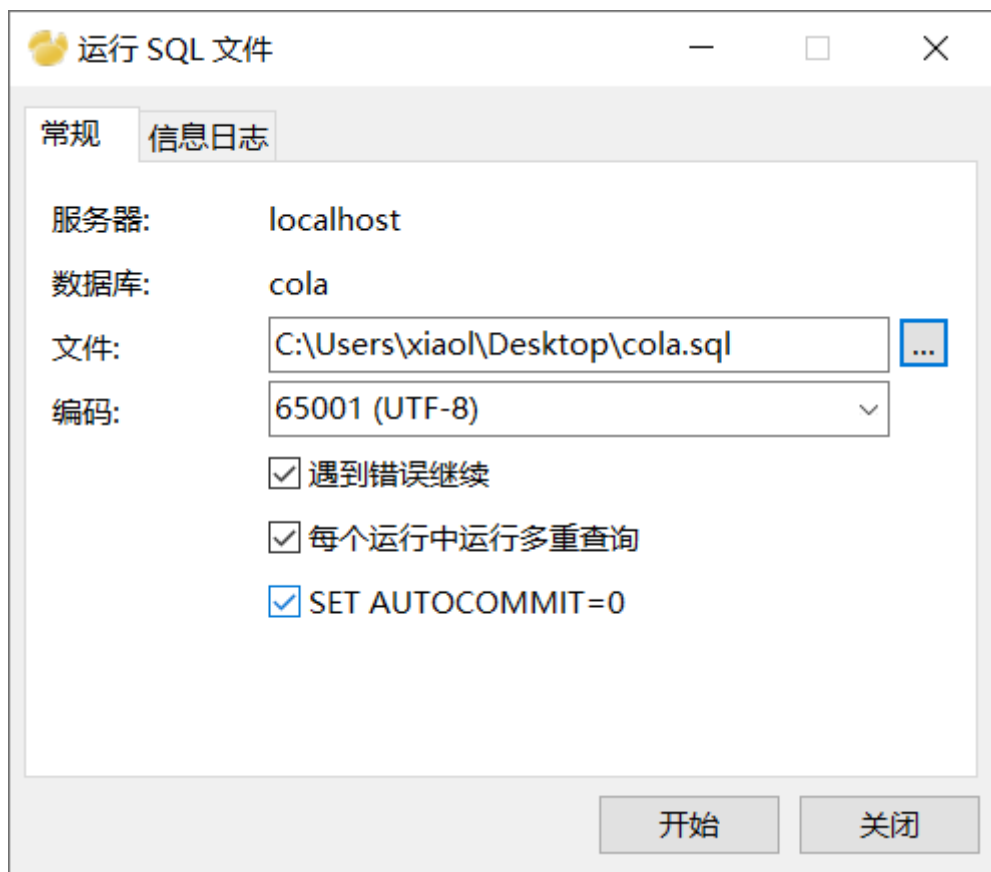


2. 找到cola-admin工程->doc->cola.sql



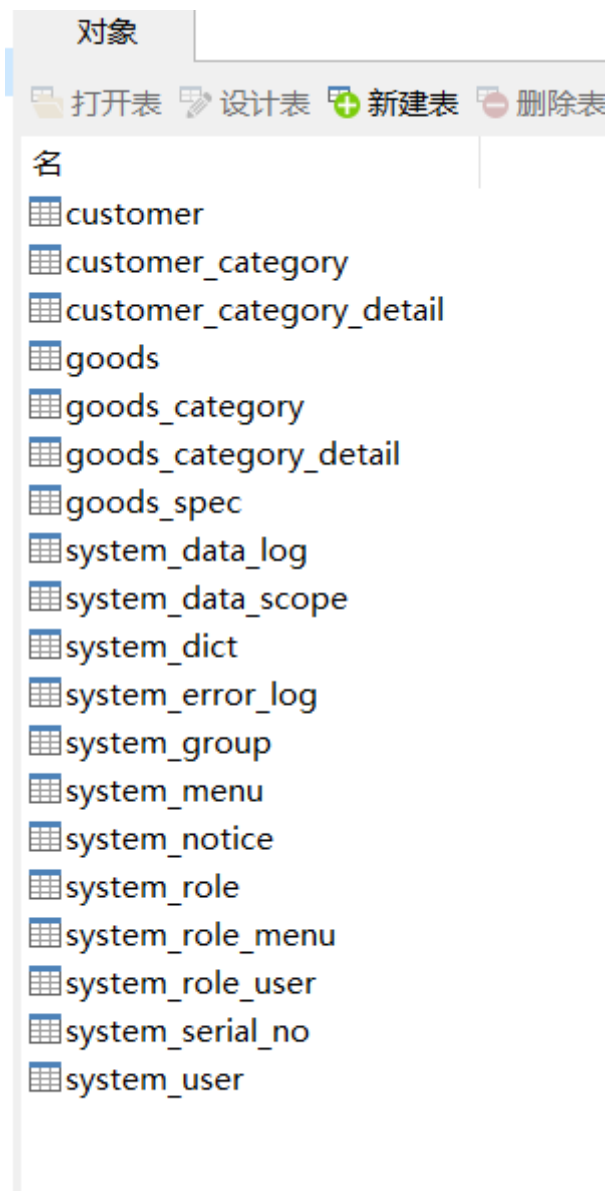
### 3. 执行sql脚本





#### 4. 最终效果

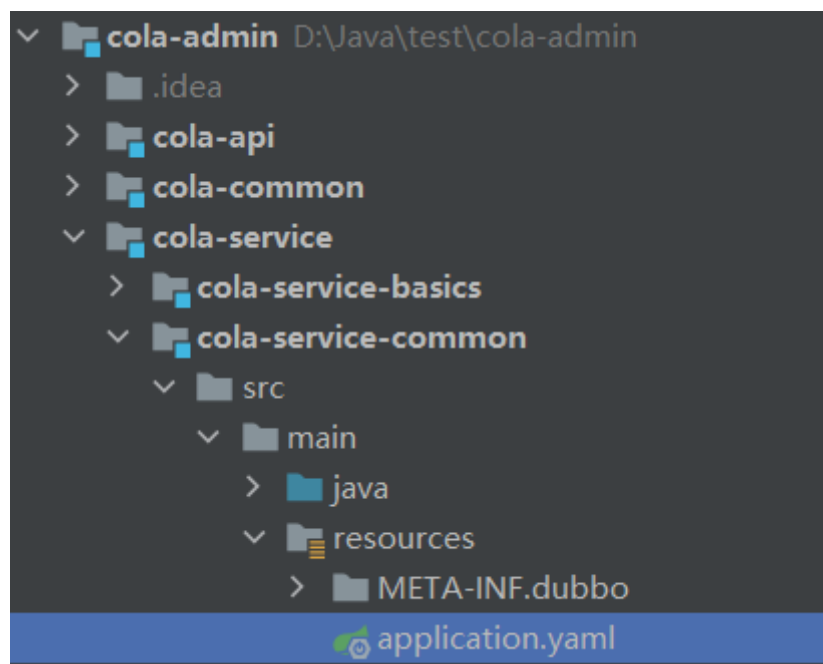




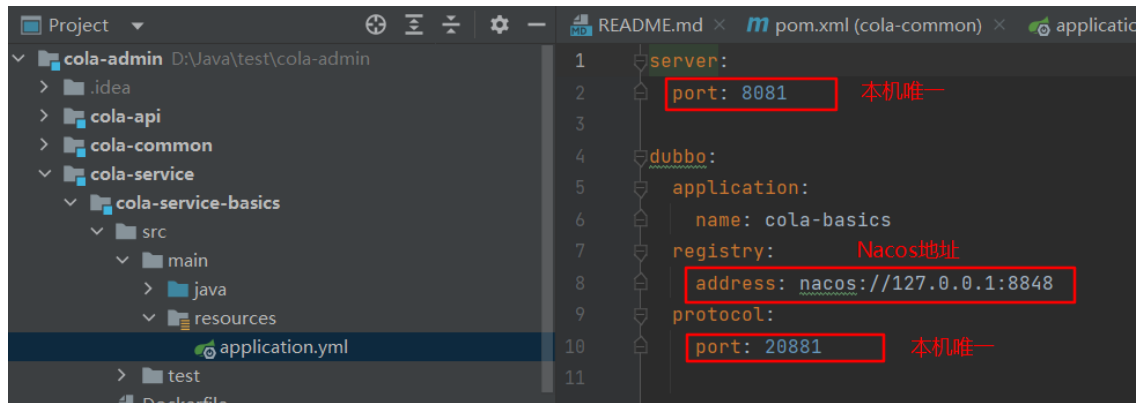
## 运行工程

### 1. 修改配置文件

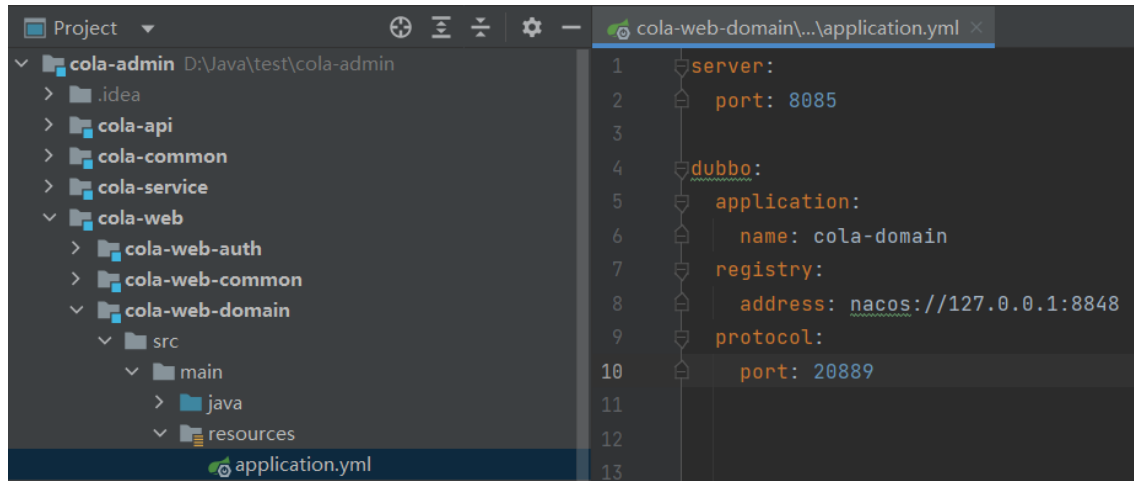
修改数据库配置: cola-service-common/src/resources/application.yaml



修改ServiceBasicsApplication配置: cola-service-basics/src/resources/application.yml

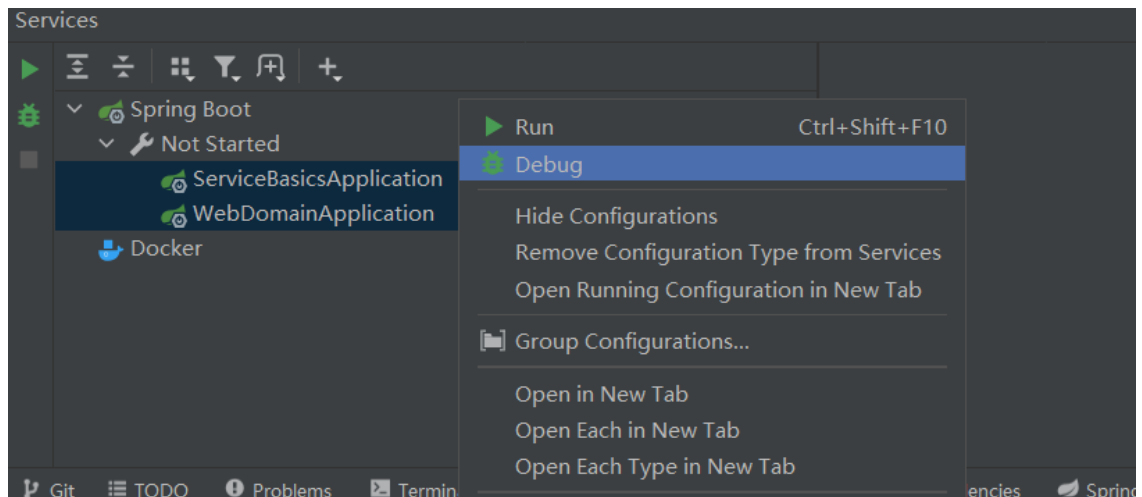


修改WebDomainApplication配置: cola-web-domain/src/resources/application.yml

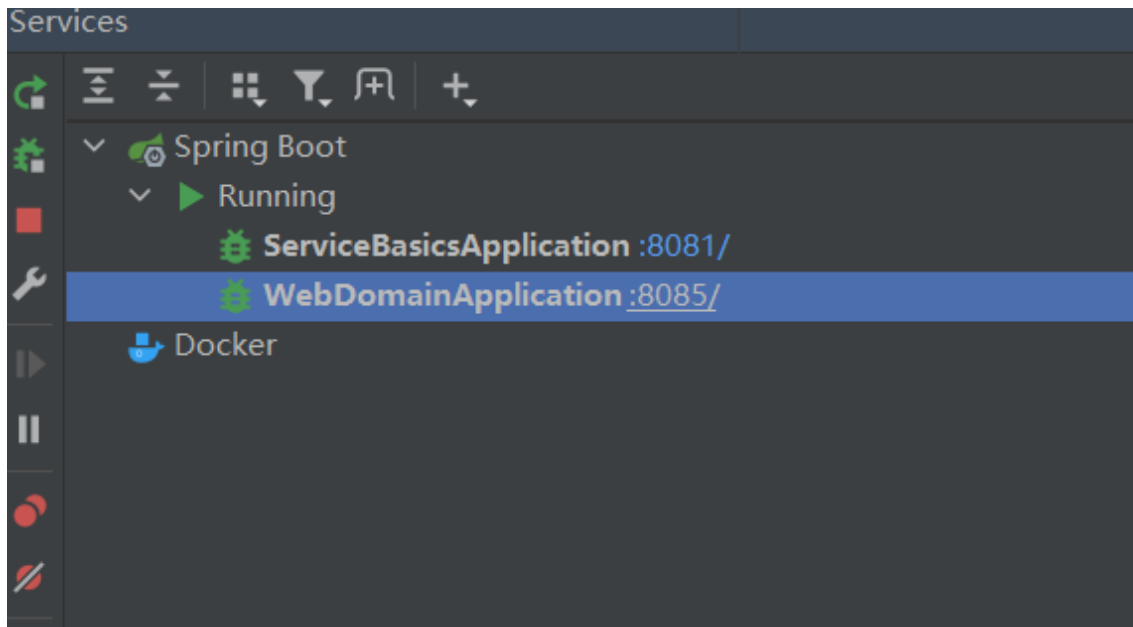


## 2. 启动项目

在Service面板中选中两个服务 (ServiceBasicsApplication和WebDomainApplication) 后点击右键并点击“Debug”



看到服务名后面的端口号时标识项目已经成功运行



### 3. 查看Nacos

打开浏览器登陆Nacos控制台并登陆，查看服务是否成功注册



两个服务已经成功注册上了

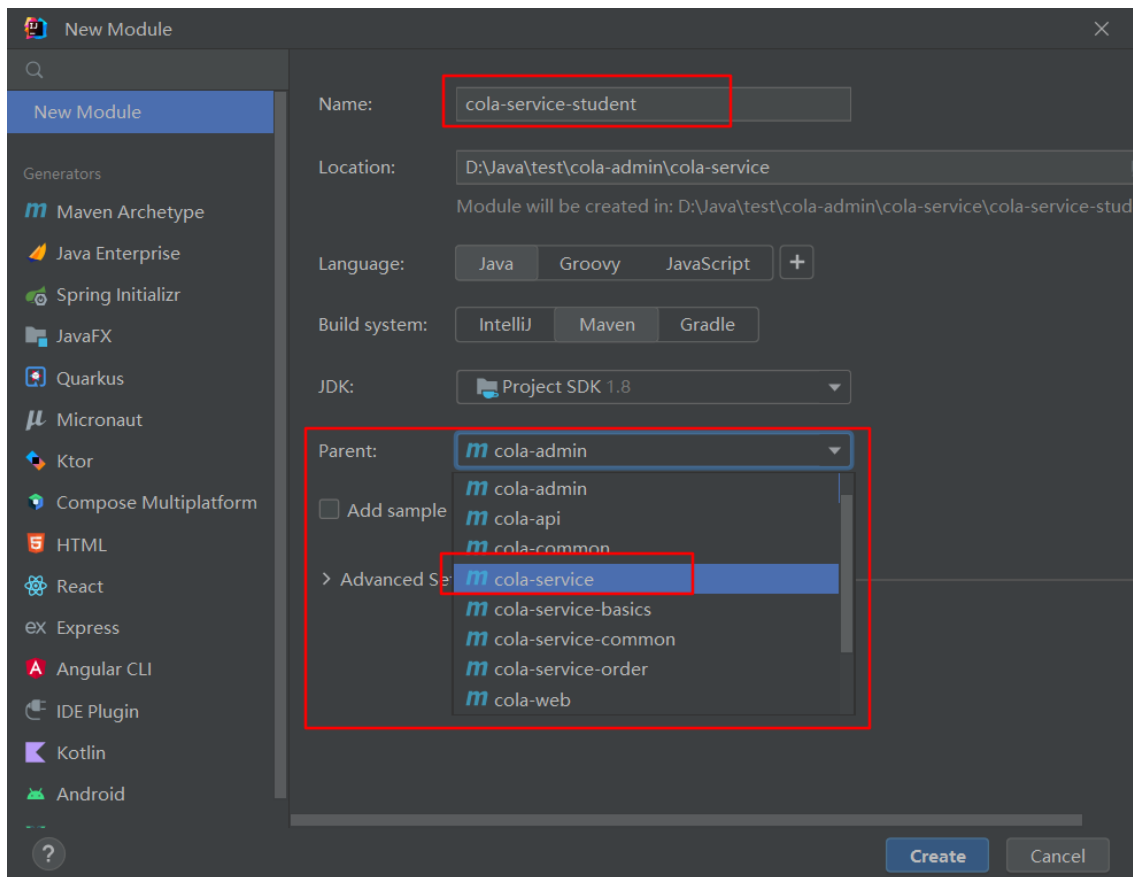
### 4. 验证

打开Postman输入地址：<http://localhost:8085/auth/token>，并选择**Post**方式提交，参数选择**Body**并选择**JSON**，输入以下内容

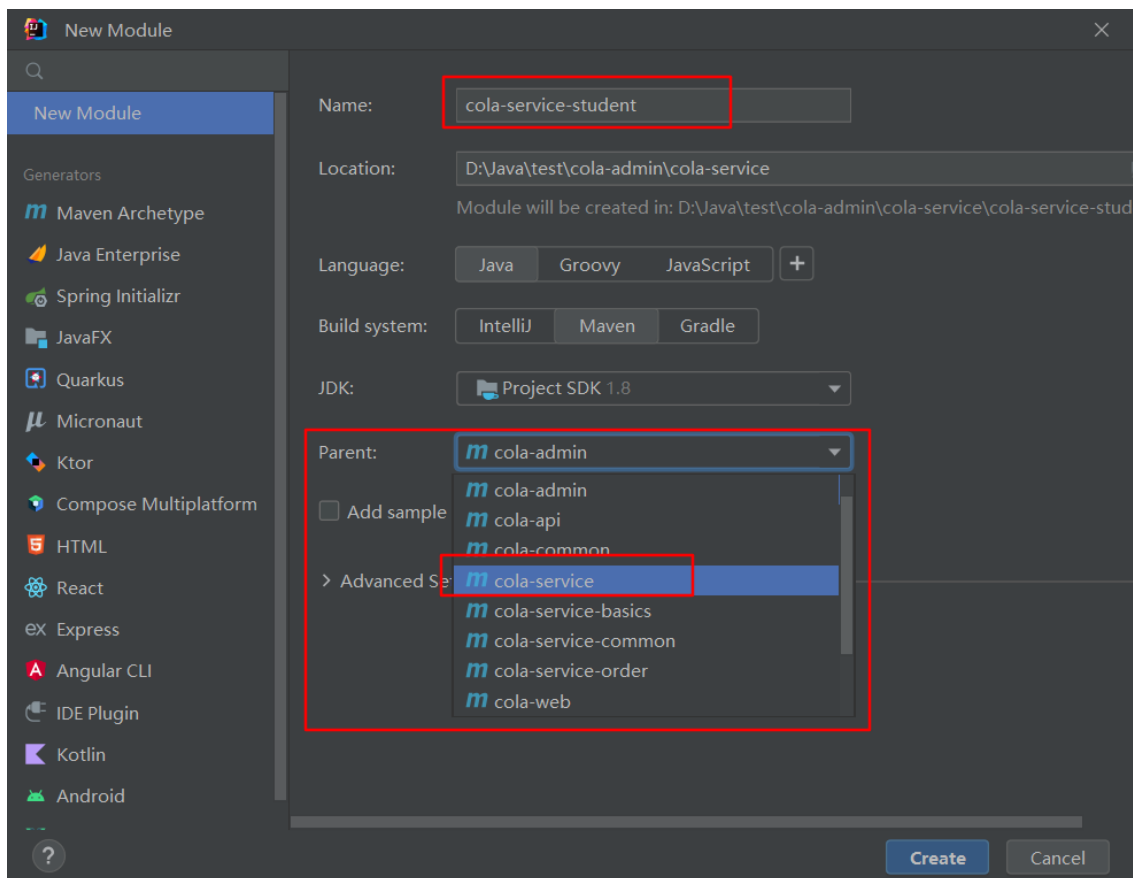
```
1 {  
2   "loginName": "admin",  
3   "password": "123123"  
4 }
```

点击发送

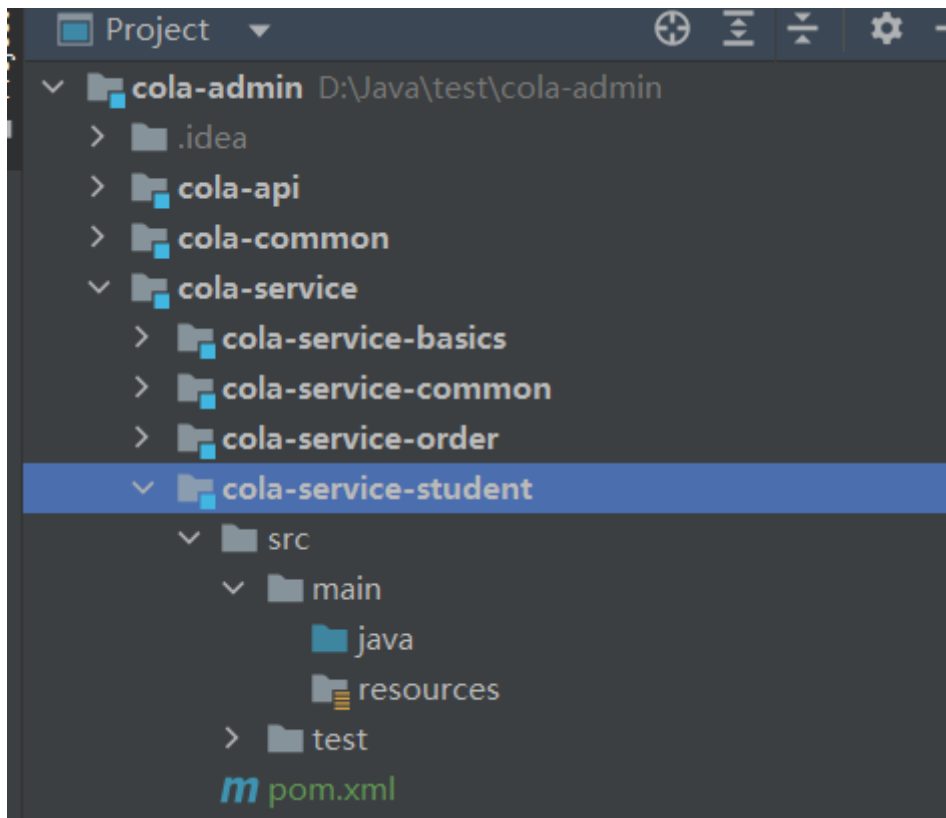




2. 在弹出的对话框中填写模块名称，并选择父模块为cola-service，点击create

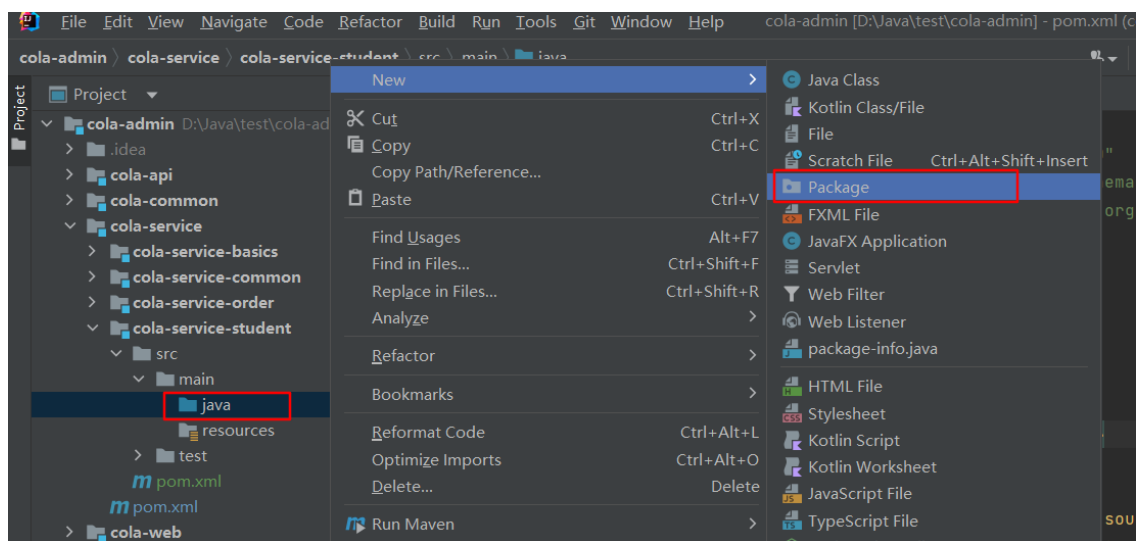


3. 创建成功

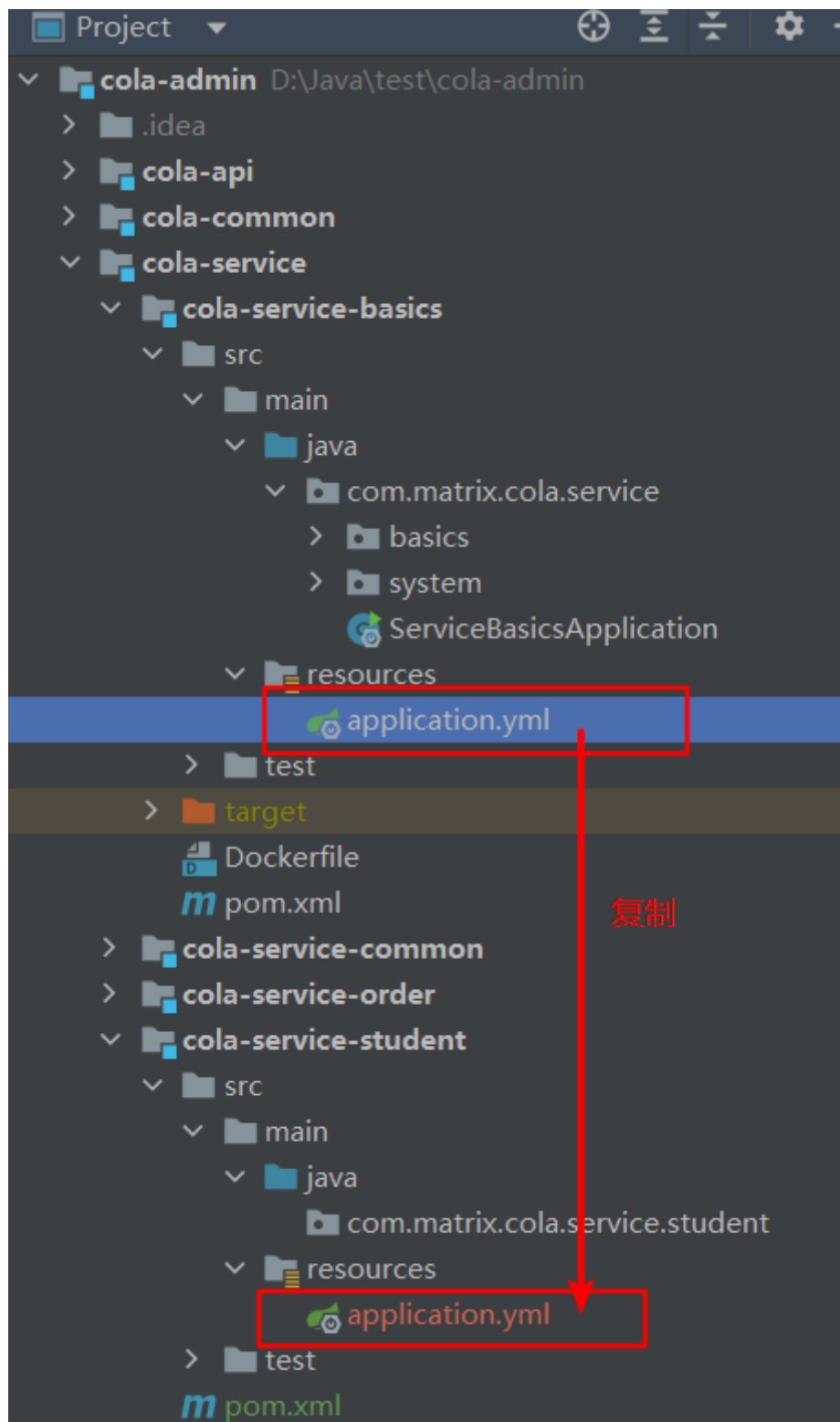


#### 4. 建包、添加配置文件

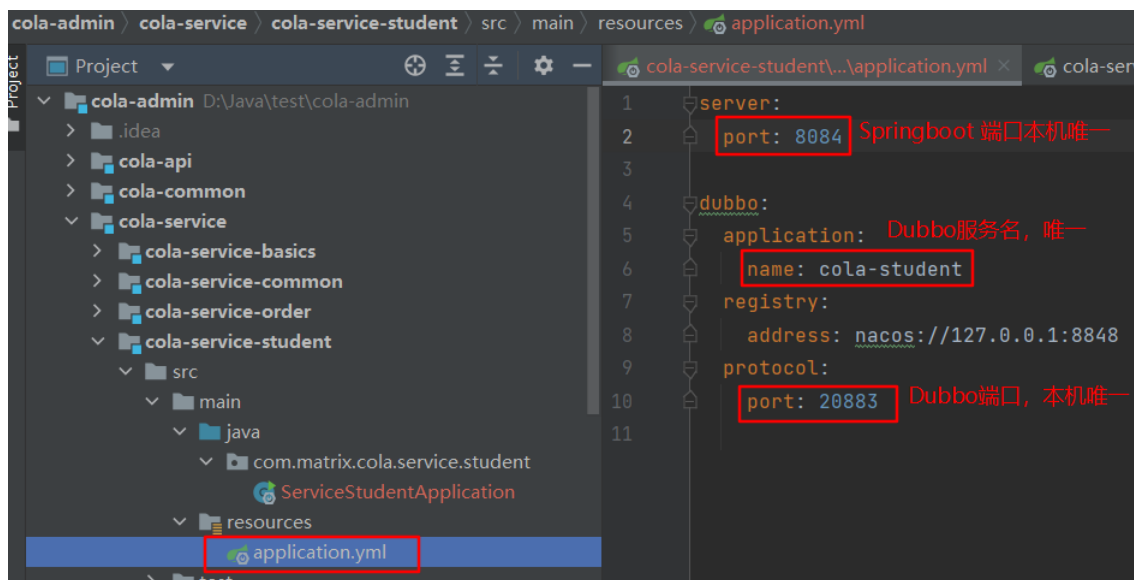
新建包名com.matrix.colaservice.student



拷贝cola-service-basics服务下的application.yml文件到cola-service-student服务的resources下

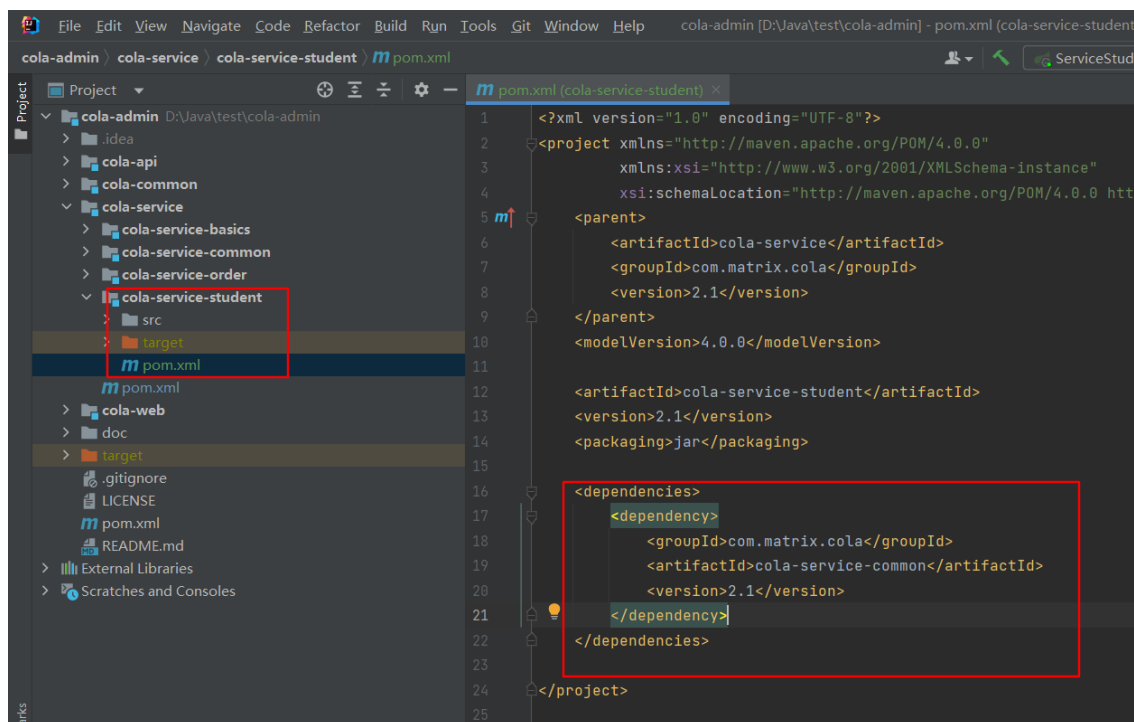


修改配置文件



## 5. 引入依赖

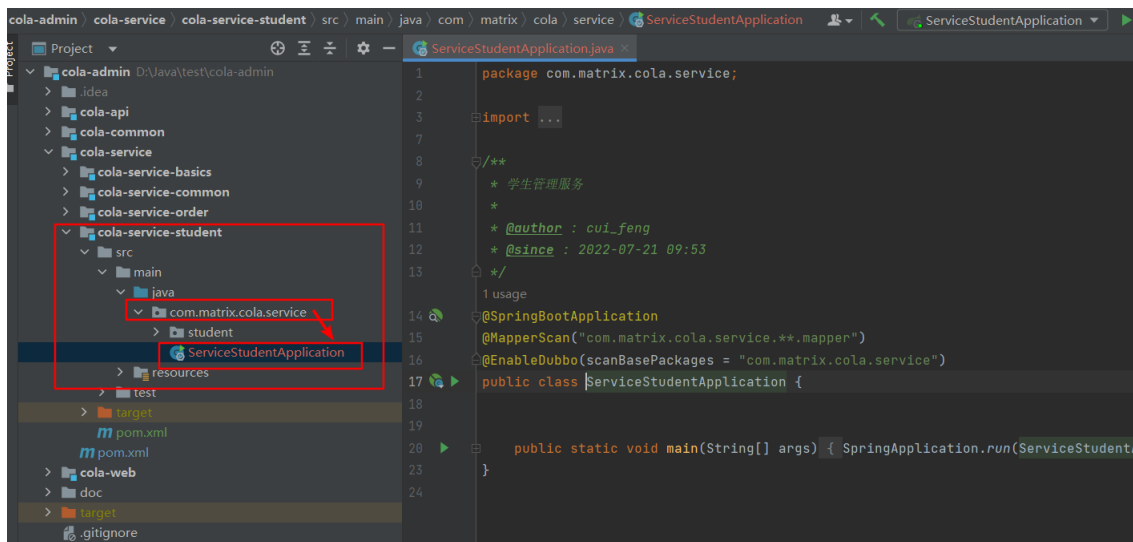
在cola-service-student的pom文件中引入cola-service-common，该模块中有数据库配置，所以需要单独引入



## 启动

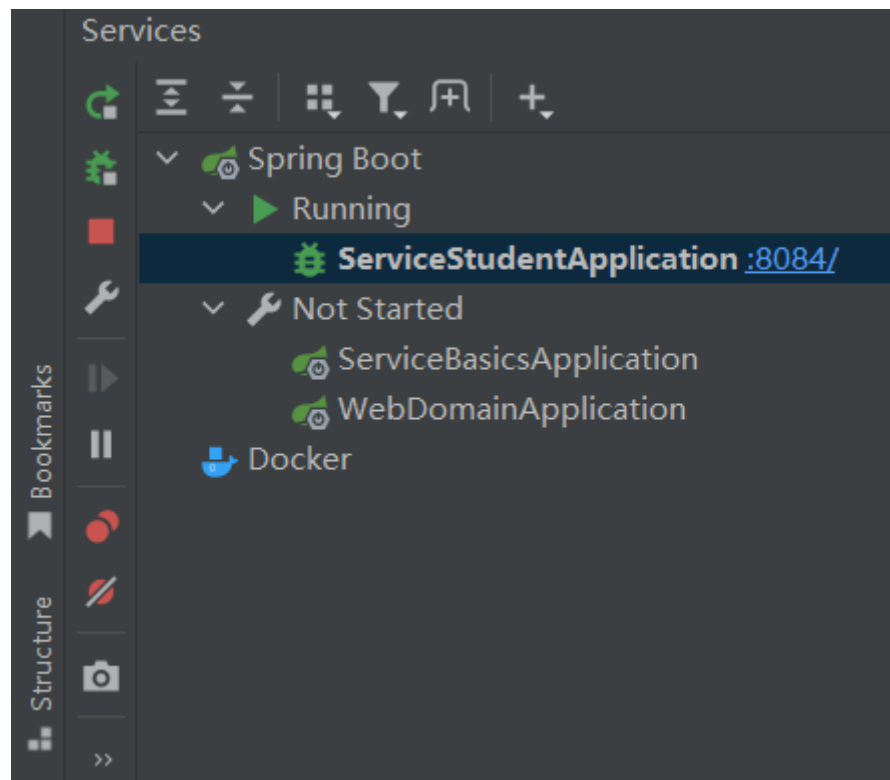
### 1. 创建启动类





注意：cola-admin推荐将启动类放到com.matrix.colaservice包下，否则需要添加@ComponentScan("com.matrix.colaservice")注解。因为很多配置是放到cola-service-common中的，如果不加则不能自动加载，如分页插件、数据库连接池配置等。

### 1. 启动成功



查看nacos中服务是否注册成功



## 第一个CURD

通过上面的学习，已经可以成功地添加一个微服务也就是Dubbo的服务提供者，下面用一个增删改查来学习一下在cola-admin中是如何实现CURD的。

### 建表

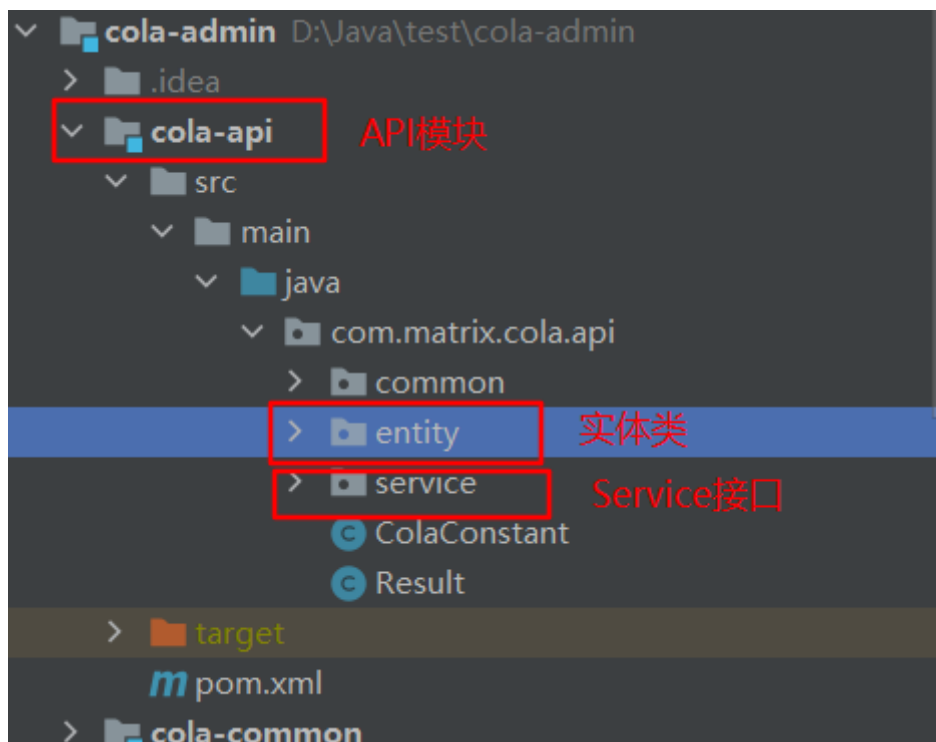
以学生管理为例，先创建一个学生表student。

```
1 CREATE TABLE `student` (  
2   `id` bigint(64) NOT NULL AUTO_INCREMENT ,  
3   `name` varchar(20) NOT NULL COMMENT '姓名' ,  
4   `age` int(2) NOT NULL COMMENT '年龄' ,  
5   `sex` int(2) NOT NULL COMMENT '性别' ,  
6   `address` varchar(100) NULL COMMENT '住址' ,  
7   `creator` bigint(64) NULL COMMENT '创建人' ,  
8   `create_time` datetime NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '创建时间' ,  
9   `reviser` bigint(64) NULL COMMENT '修改人' ,  
10  `revise_time` datetime NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '修改时间' ,  
11  `deleted` int(2) NULL DEFAULT 0 COMMENT '是否删除: 0=未删除, 1=已删除' ,  
12  `group_id` varchar(64) NULL ,  
13  PRIMARY KEY (`id`)  
14 );
```

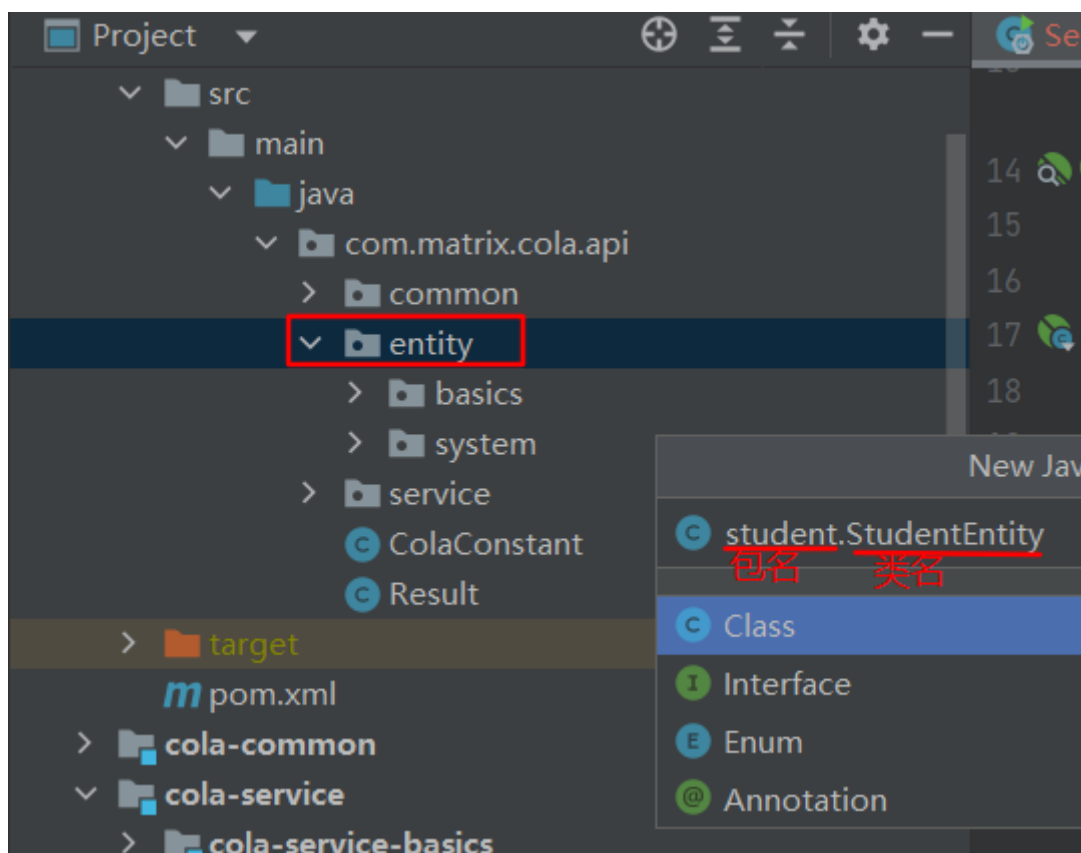
表中的“creator”、“create\_time”、“reviser”、“revise\_time”、“deleted”、“group\_id”这六个字段是固定的业务字段，建议每个业务表都加上。表的字段名单词之间用下划线连接，这样就可以在实体类中直接使用驼峰方式命名，如字段“create\_time”在实体类中的属性名就是“createTime”。

## 创建实体类

实体类和Service接口需要添加到cola-api中，这个包在Dubbo的服务提供者端和消费者端都需要引入，在cola-service中的服务只要引用了cola-service-common包就会默认引入cola-api。



展开entity包，添加包名和实体类



## 继承实体类

cola-admin中有两个实体类的抽象父类，业务实体抽象类（带那六个业务字段）BaseColaEntity和普通实体抽象类（不带业务字段）BaseEntity。我们的StudentEntity是业务实体类，所以需要继承BaseColaEntity。继承该抽象类后，StudentEntity类将自动带有那六个业务字段。

```
/**
 * 学生实体类
 * @author : cui_feng
 * @since : 2022-07-21 13:00
 */
public class StudentEntity extends BaseColaEntity {
}
```

## 主键策略

cola-admin的业务实体类默认主键策略是数据库自增长，定义在了BaseColaEntity中，可以根据需要修改

```
@Data
public abstract class BaseColaEntity extends BaseEntity {

    /**
     * id号
     * 默认数据库自增
     */
    @TableId(type = IdType.AUTO)
    private Long id;
```

## 添加属性并映射表名

由于父类中已经定义了主键id，这里可以省略

```

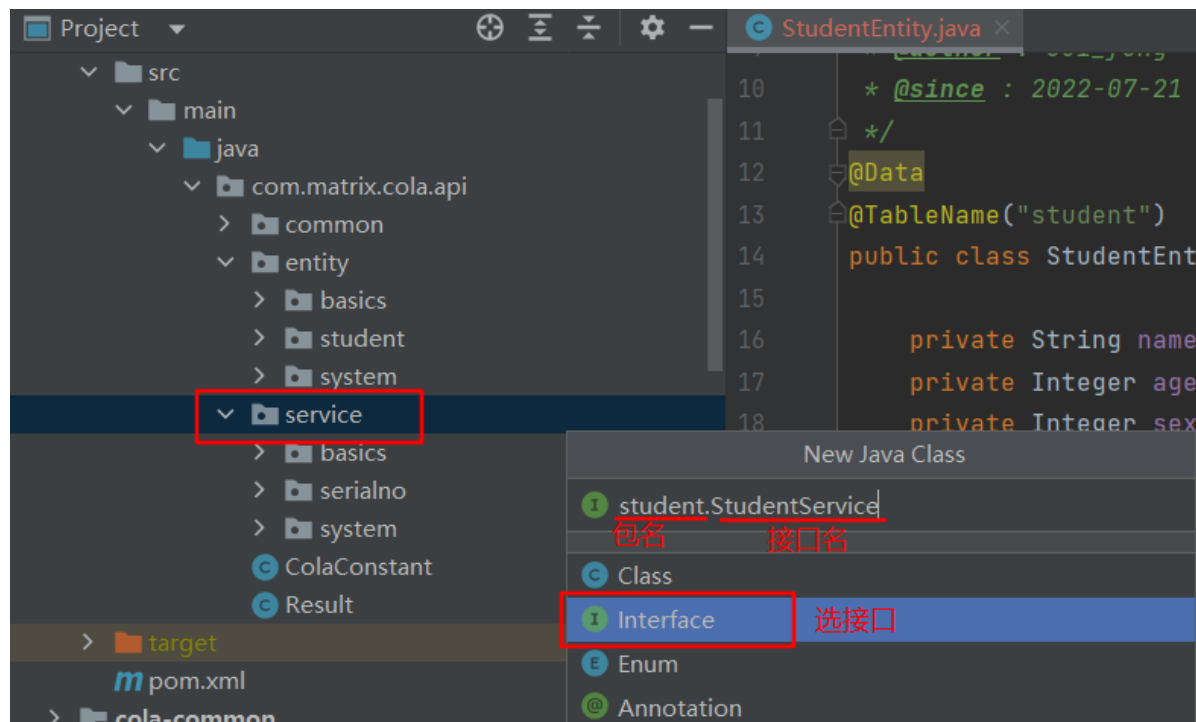
@Data Lombok注解
@TableName("student") 数据库表名
public class StudentEntity extends BaseColaEntity {

    private String name;
    private Integer age;
    private Integer sex;
    private String address;
}

```

## 创建Service接口

在service包下新建接口StudentService



## 继承Service接口

cola-admin中有两个Service接口可以继承，与实体类一样，分为业务实体类Service父接口BaseColaEntityService和普通实体类Service父接口BaseEntityService。这两个父接口不可以混合使用，如果实体类继承了业务实体抽象类，则Service必须继承BaseColaEntityService，否则需要继承BaseEntityService。

```

/**
 * 学生管理接口
 * @author : cui_feng
 * @since : 2022-07-21 13:14
 */
public interface StudentService extends BaseColaEntityService<StudentEntity> {
}

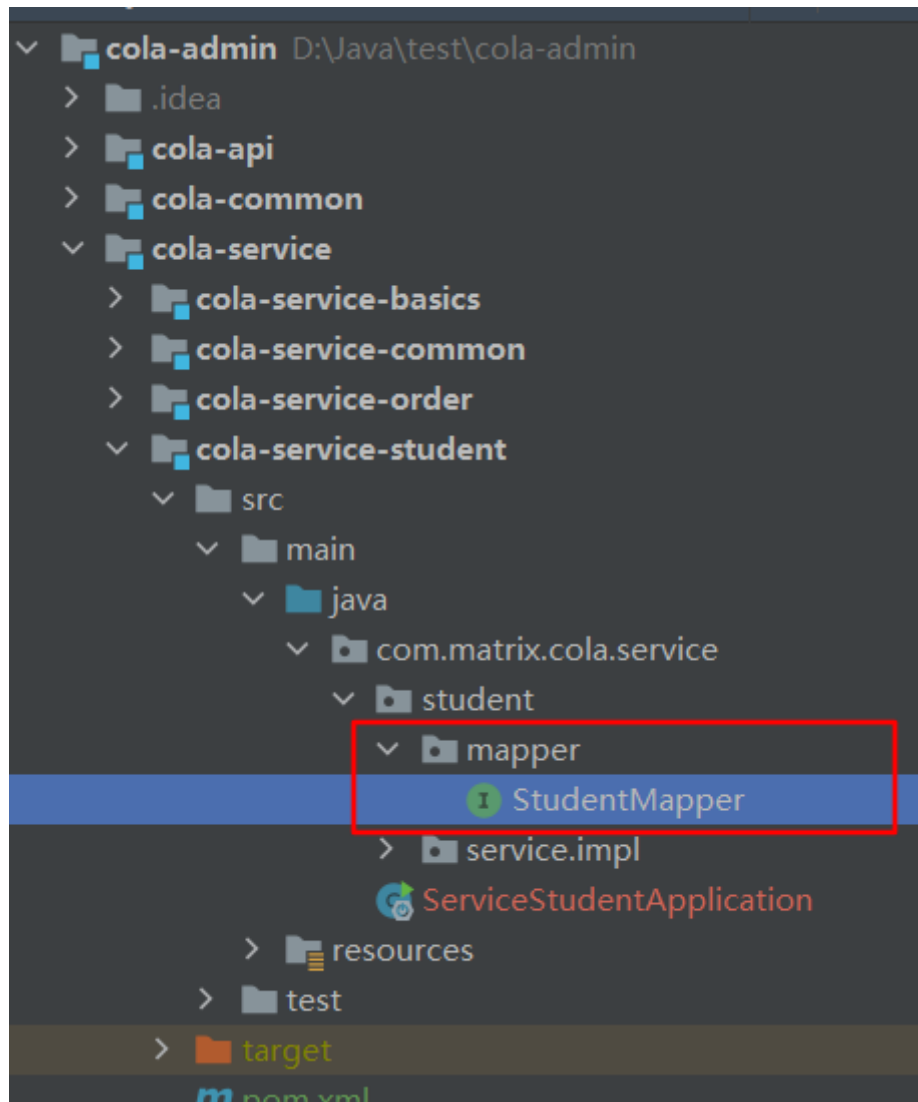
```

指定实体类

继承父接口时需要指定实体类的泛型

## 创建mapper接口

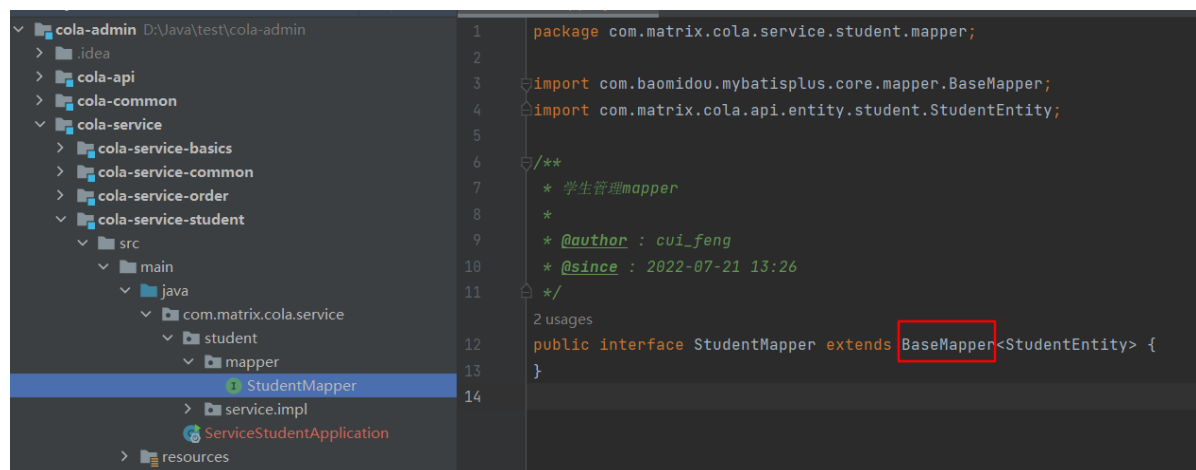
回到cola-service-student模块中，新建mapper接口StudentMapper



mapper接口必须在mapper包下，并以xxxxMapper命名

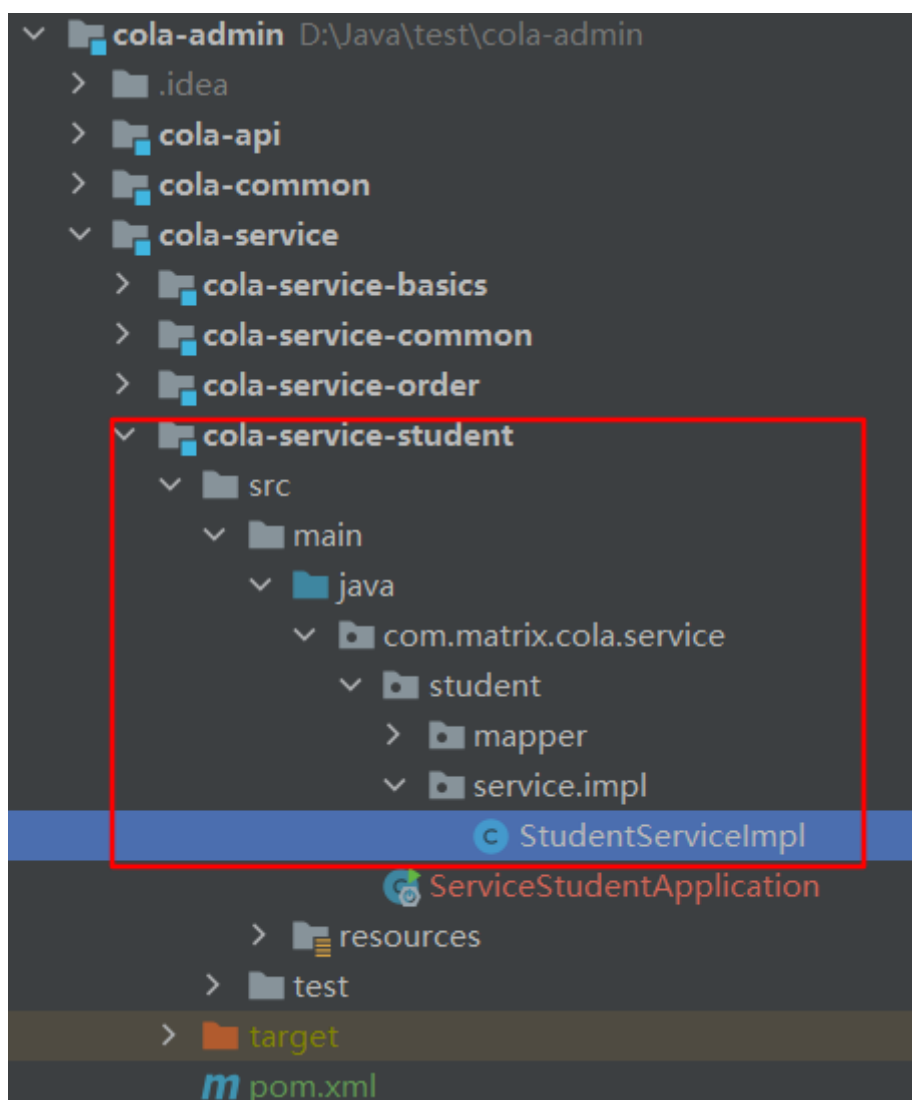
## 继承BaseMapper

建好mapper接口后需要继承BaseMapper接口，同时指定泛型为实体类



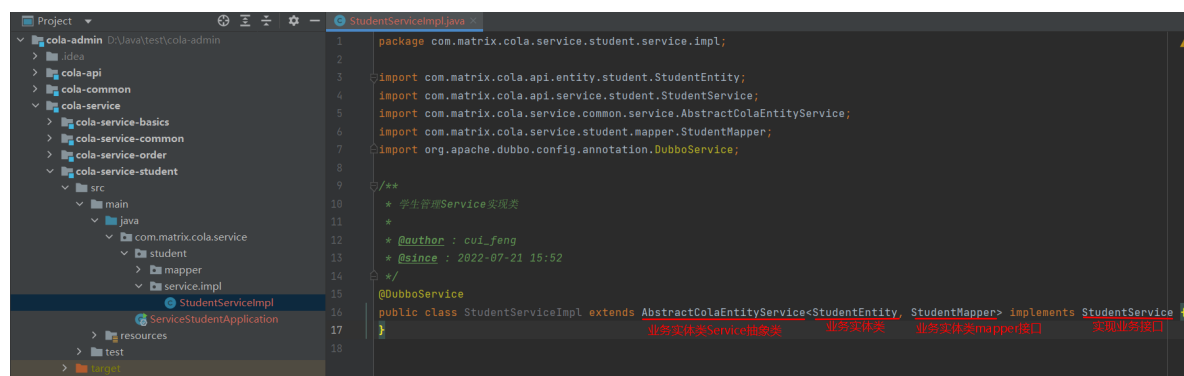
## 创建Service实现类

在cola-service-student中创建StudentService接口的实现类StudentServiceImpl



### 继承抽象类

对于Service的实现类，cola-admin也提供了两个抽象类可以用来直接继承。抽象类中实现了增、删、查、改等各种常用的实体类操作方法，无需单独实现。对于业务实体类的Service实现类需要继承AbstractColaEntityService抽象类，对于普通实体类需要继承AbstractEntityService抽象类，同时添加上实体类和Mapper接口的泛型。



通过上面的例子可以看到，业务对象相关的抽象类、接口都带有Cola标识，如BaseColaEntity、BaseColaEntityService、AbstractColaEntityService。普通的实体类相关的都没有Cola标识，如BaseEntity、BaseEntityService、AbstractEntityService。这样可以方便记忆。

## 添加Dubbo注解

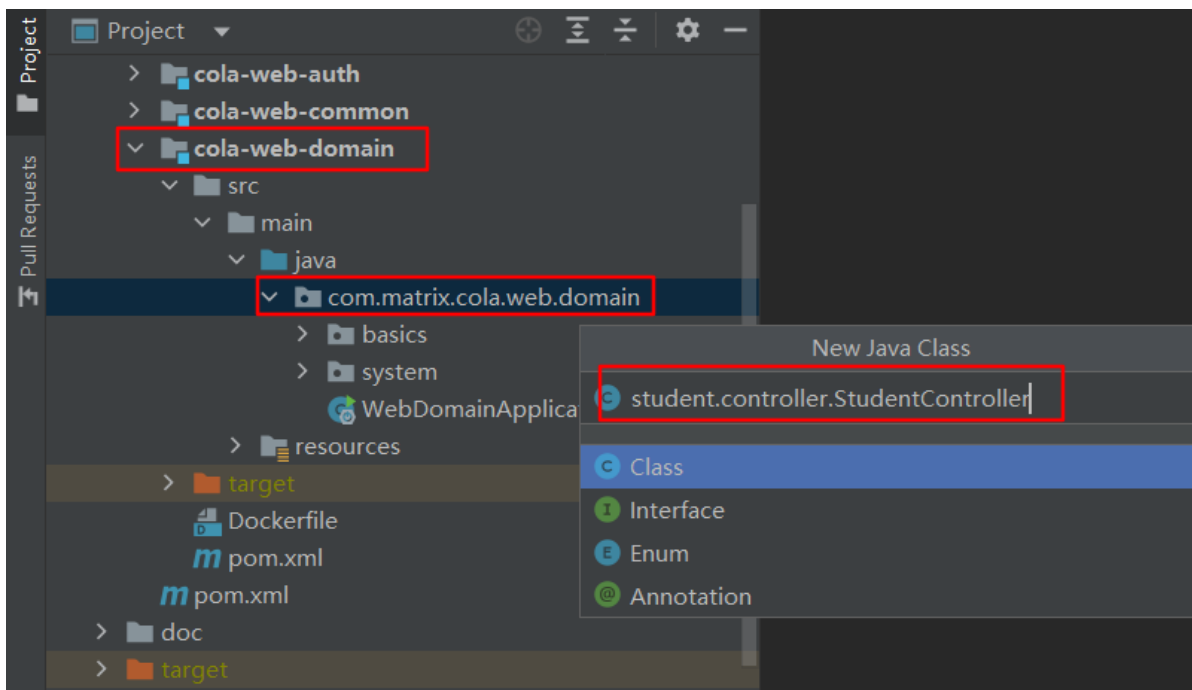
在StudentServiceImpl类上添加@DubboService注解

```
/**
 * 学生管理服务实现类
 *
 * @author : cui_feng
 * @since : 2022-07-21 15:52
 */
@DubboService  // 声明为Dubbo的服务提供者
public class StudentServiceImpl extends AbstractColaEntityService<StudentEntity, StudentMapper> implements StudentService {
}
```

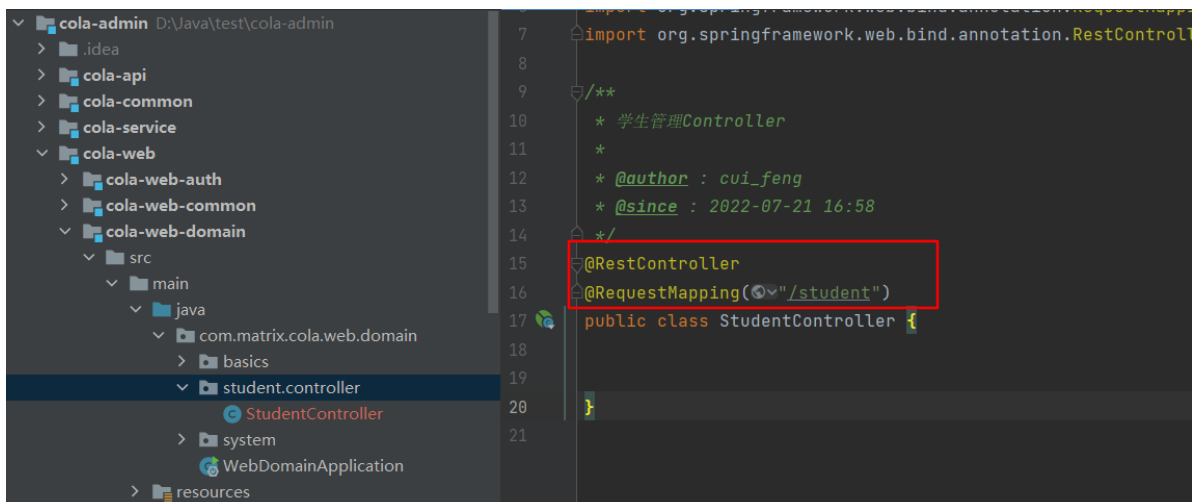
这样就完成了学生管理服务的全部功能（包括增、删、查、改、分页等功能），是不是很简单。

## 创建web接口

在cola-web-domain中添加controller



添加@RestController和@RequestMapping注解，因为是前后端分离项目所以需要使用@RestController。





## 添加CURD接口

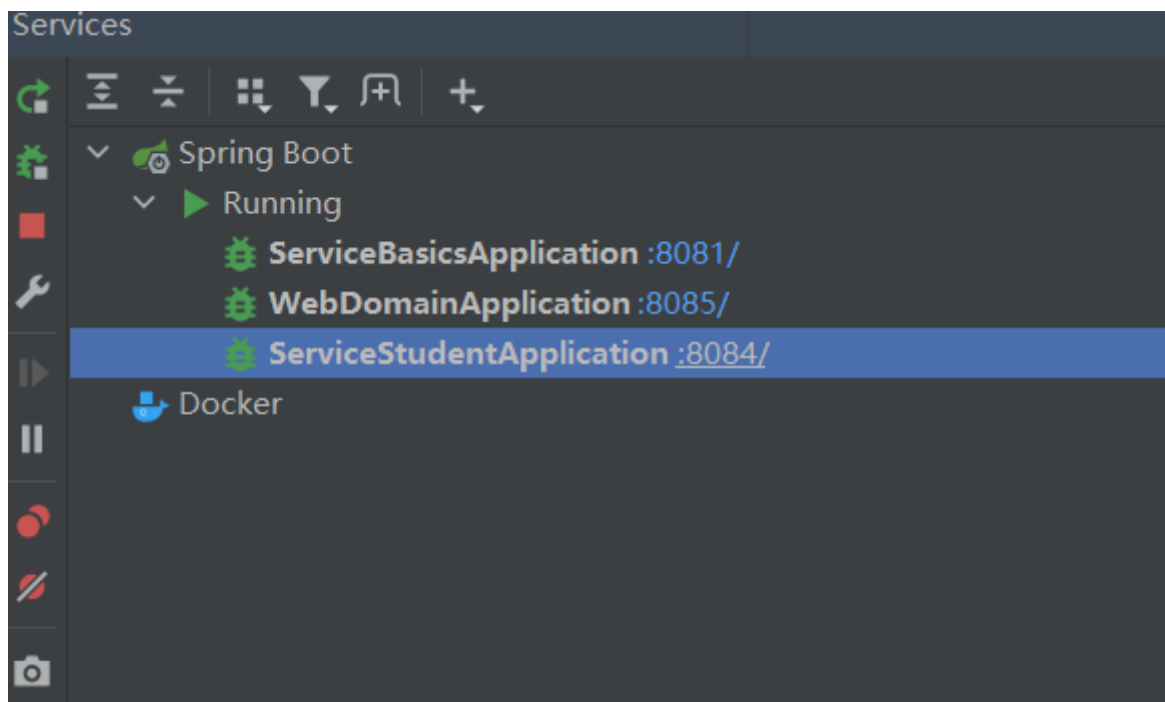
```
1 package com.matrix.cola.web.domain.student.controller;
2
3 import com.matrix.cola.api.Result;
4 import com.matrix.cola.api.common.entity.Query;
5 import com.matrix.cola.api.entity.student.StudentEntity;
6 import com.matrix.cola.api.service.student.StudentService;
7 import org.apache.dubbo.config.annotation.DubboReference;
8 import org.springframework.stereotype.Controller;
9 import org.springframework.web.bind.annotation.PostMapping;
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13
14 /**
15  * 学生管理Controller
16  *
17  * @author : cui_feng
18  * @since : 2022-07-21 16:58
19  */
20 @RestController
21 @RequestMapping("/student")
22 public class StudentController {
23
24     @DubboReference
25     StudentService studentService;
26
27     @PostMapping("/getStudentPage")
28     public Result getStudentPage(@RequestBody Query<StudentEntity> query) {
29         return Result.page(studentService.getPage(query));
30     }
31
32     @PostMapping("/addStudent")
33     public Result addStudent(@RequestBody StudentEntity student) {
34         return studentService.insert(student);
35     }
36
37     @PostMapping("/editStudent")
38     public Result editStudent(@RequestBody StudentEntity student) {
39         return studentService.modify(student);
40     }
41
42     @PostMapping("/deleteStudent")
43     public Result deleteStudent(@RequestBody StudentEntity student) {
44         return studentService.remove(student);
45     }
46 }
47
```

这样就大功告成了。

cola-admin中Web层接口就是Dubbo的服务消费者，Service层就是服务提供者。这样在物理上就将Controller和Service进行了分离。cola-admin不建议在controller中处理业务逻辑，所有的业务处理都应该放到服务提供者也就是service中进行实现。

## 启动服务

打开services面板，启动ServiceBasicsApplication、ServiceStudentApplication、WebDomainApplication三个服务。



由于系统管理的业务都放到了ServiceBasicsApplication服务中，所以该服务必须要启动。cola-admin默认关闭了Dubbo的服务检查，所以并不要求服务的启动顺序。

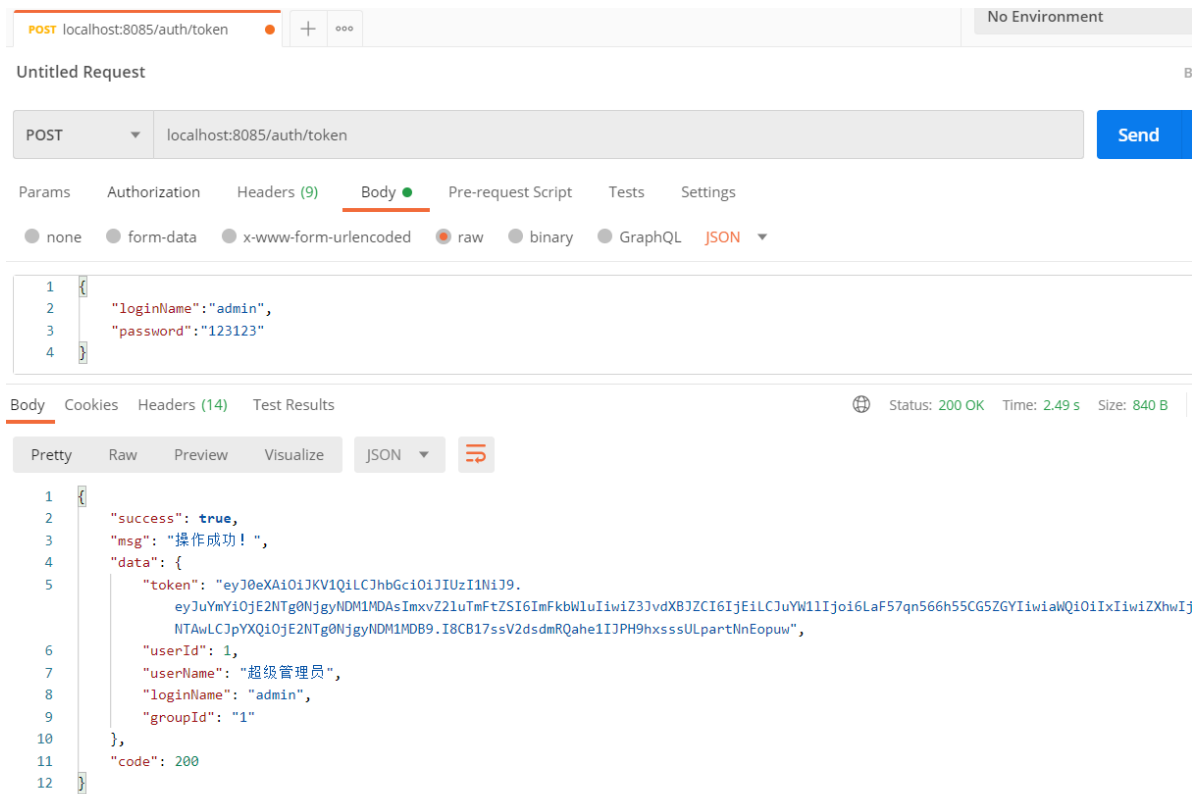
## 接口测试

### 获取token

打开Postman，输入<http://localhost:8085/auth/token>，选择Post方式提交，参数选择Body并选择JSON，输入以下内容

```
1 {  
2   "loginName": "admin",  
3   "password": "123123"  
4 }
```

点击send，看到如下内容则表示获取token成功



复制获取到的token值，后面的请求都需要用到。

## 新增学生

在Postman中新建一个标签，输入<http://localhost:8085/student/addStudent>，选择Post方式提交，输入下面的参数：

```
1 {  
2     "name": "张三",  
3     "age": 18,  
4     "sex": 0,  
5     "address": "山东省济南市"  
6 }
```

在Headers中添加参数token并粘贴上刚才复制的token值，如下图

如果没有在Header中添加token则会出现下面的错误

POST localhost:8085/auth/token    POST http://localhost:8085/student/...    +    ...    No Envir

### Untitled Request

POST    http://localhost:8085/student/addStudent

Params    Authorization    Headers (9)    **Body**    Pre-request Script    Tests    Settings

● none    ● form-data    ● x-www-form-urlencoded    ● **raw**    ● binary    ● GraphQL    **JSON** ▼

```
1 {
2   "name": "张三",
3   "age": 18,
4   "sex": 0,
5   "address": "山东省济南市"
6 }
```

Body    Cookies    Headers (14)    Test Results    Status: 200 OK    Time: 7

Pretty    Raw    Preview    Visualize    JSON ▼    ↻

```
1 {
2   "success": false,
3   "msg": "您未认证, 无法访问本资源",
4   "data": {},
5   "code": 508
6 }
```

我们看一下数据库中的记录

对象    student @cola (localhost) - 表

id	name	age	sex	address	creator	create_time	reviser	revise_time	deleted	group_id
1	张三	18	0	山东省济南市	1	2022-07-22 13:44:52	1	2022-07-22 13:44:52	0	1

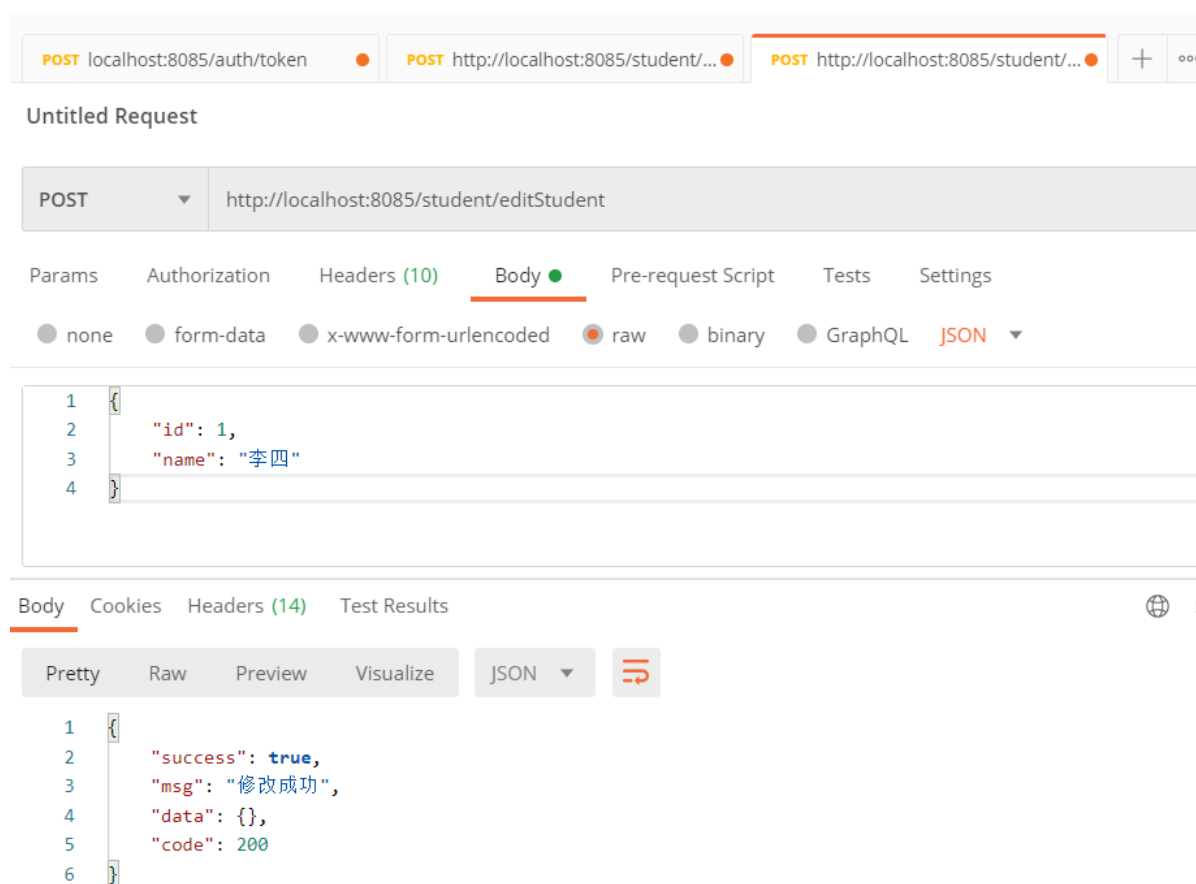
我们可以看到，最后的六个业务字段已经被填充了数据，这就是在AbstractColaEntityService中实现的。

## 修改学生

在Postman中再新建一个标签，输入<http://localhost:8085/student/editStudent>，选择Post方式提交，输入下面的参数：

```
1 {
2   "id": 1,
3   "name": "李四"
4 }
```

在Header中添加token，点击send



数据修改成功

对象 student @cola (localhost) - 表						
开始事务 备注 筛选 排序						
id	name	age	sex	address	creator	
1	李四	18	0	山东省济南市	1	

## 删除学生

在Postman中新建一个标签页面，输入<http://localhost:8085/student/deleteStudent>，选择Post方式提交，输入下面的参数：

1

POST localhost:8085/auth/t...

POST http://localhost:8085/...

POST http://localhost:8085/...

POST http://localhost:8085/...

### Untitled Request

POST

http://localhost:8085/student/deleteStudent

ParamsAuthorizationHeaders (10)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
1 {
2   "id": 1
3 }
```

BodyCookiesHeaders (14)Test Results

PrettyRawPreviewVisualizeJSON

```
1 {
2   "success": true,
3   "msg": "删除成功",
4   "data": {},
5   "code": 200
6 }
```

查看数据库可以看到deleted已为1，逻辑删除成功。

对象 student @cola (localhost) - 表										
开始事务 备注 筛选 排序 导入 导出										
id	name	age	sex	address	creator	create_time	reviser	revise_time	deleted	group_id
1	李四	18	0	山东省济南市	1	2022-07-22 15:15:58	1	2022-07-22 15:15:58	1	1

## 分页查询

先填充几条数据

```
1 INSERT INTO `student` VALUES ('2', '张三', '16', '1', null, '1', '2022-07-22 15:38:15', null, '2022-07-22 15:38:15', '0', '1');
2 INSERT INTO `student` VALUES ('3', '王五', '19', '0', null, '1', '2022-07-22 15:38:15', null, '2022-07-22 15:38:15', '0', '1');
3 INSERT INTO `student` VALUES ('4', '赵六', '20', '1', null, '1', '2022-07-22 15:38:16', null, '2022-07-22 15:38:16', '0', '1');
4 INSERT INTO `student` VALUES ('5', '张生', '19', '0', null, '1', '2022-07-22 15:38:18', null, '2022-07-22 15:38:18', '0', '1');
```

在Post中新建一个标签页，输入<http://localhost:8085/student/getStudentPage>，选择Post方式提交，输入下面的参数：

```
1 {
2   "pageSize": 2
3 }
```

这样的查询就是每页2条数据

```

1  {
2      "success": true,
3      "msg": "操作成功!",
4      "data": {
5          "page": {
6              "records": [
7                  {
8                      "id": 2,
9                      "creator": 1,
10                     "createTime": "2022-07-22 15:38:15",
11                     "reviser": null,
12                     "reviseTime": "2022-07-22 15:38:15",
13                     "deleted": 0,
14                     "groupId": "1",
15                     "name": "张三",
16                     "age": 16,
17                     "sex": 1,
18                     "address": null
19                 },
20                 {
21                     "id": 3,
22                     "creator": 1,
23                     "createTime": "2022-07-22 15:38:15",
24                     "reviser": null,
25                     "reviseTime": "2022-07-22 15:38:15",
26                     "deleted": 0,
27                     "groupId": "1",
28                     "name": "王五",
29                     "age": 19,
30                     "sex": 0,
31                     "address": null
32                 }
33             ],
34             "total": 4,
35             "size": 2,
36             "current": 1,
37             "orders": [],
38             "optimizeCountSql": true,
39             "searchCount": true,
40             "countId": null,
41             "maxLimit": null,
42             "pages": 2
43         }
44     },
45     "code": 200
46 }

```

## 带条件的查询

参数部分输入以下内容，查询姓名为张三的学生



```
1 {
2   "data": {
3     "name": "张三"
4   },
5   "pageSize": 2
6 }
```

返回结果为

```
1 {
2   "success": true,
3   "msg": "操作成功! ",
4   "data": {
5     "page": {
6       "records": [
7         {
8           "id": 2,
9           "creator": 1,
10          "createTime": "2022-07-22 15:38:15",
11          "reviser": null,
12          "reviseTime": "2022-07-22 15:38:15",
13          "deleted": 0,
14          "groupId": "1",
15          "name": "张三",
16          "age": 16,
17          "sex": 1,
18          "address": null
19        }
20      ],
21      "total": 1,
22      "size": 2,
23      "current": 1,
24      "orders": [],
25      "optimizeCountSql": true,
26      "searchCount": true,
27      "countId": null,
28      "maxLimit": null,
29      "pages": 1
30    }
31  },
32  "code": 200
33 }
```

POST http://localhost:8085/student/getStudentPage Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "data": {
3     "name": "张三"
4   },
5   "pageSize": 2
6 }
```

Body Cookies Headers (14) Test Results

Status: 200 OK Time: 107 ms Size: 823 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": true,
3   "msg": "操作成功!",
4   "data": {
5     "page": {
6       "records": [
7         {
8           "id": 2,
9           "creator": 1,
```

下面的name查询是=号查询，在定义在data中的数据默认都是=号查询，下面演示like查询，参数部分改成如下内容

```
1 {
2   "conditions": [
3     {
4       "name": "name",
5       "keyword": "like"
6     },
7     {
8       "name": "age",
9       "keyword": "between",
10      "value1": 10,
11      "value2": 20
12    }
13  ],
14  "data": {
15    "name": "张"
16  },
17  "pageSize": 2
18 }
```

上面的查询条件会转化为以下SQL:

```
1 select * from student where name like '%张%' and age between 10 and 20 and
   deleted=0
```

查询结果如下:

```
1 {
2   "success": true,
3   "msg": "操作成功!",
4   "data": {
5     "page": {
6       "records": [
7         {
8           "id": 2,
```

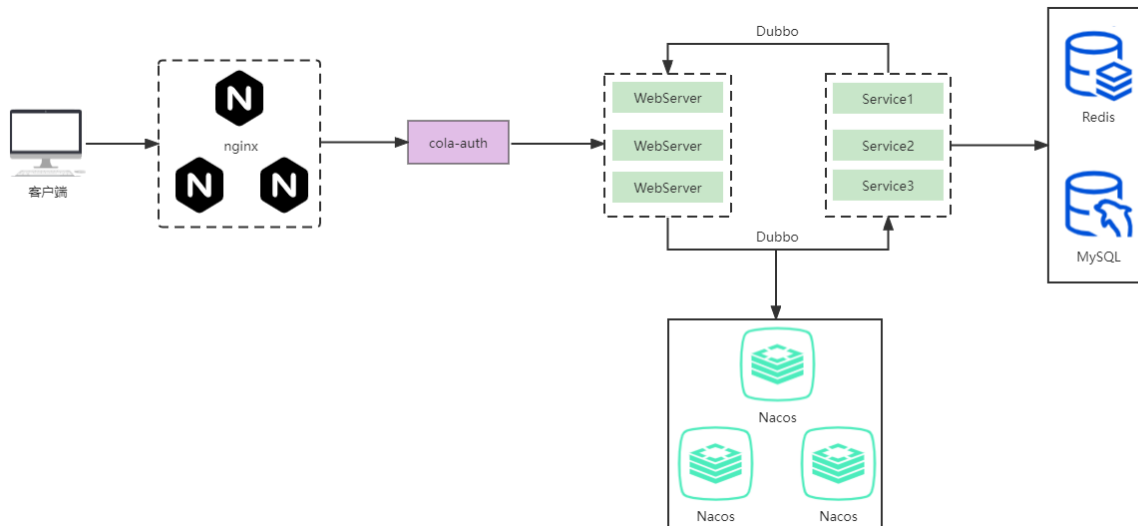
```
9         "creator": 1,
10        "createTime": "2022-07-22 15:38:15",
11        "reviser": null,
12        "reviseTime": "2022-07-22 15:38:15",
13        "deleted": 0,
14        "groupId": "1",
15        "name": "张三",
16        "age": 16,
17        "sex": 1,
18        "address": null
19    },
20    {
21        "id": 5,
22        "creator": 1,
23        "createTime": "2022-07-22 17:31:11",
24        "reviser": null,
25        "reviseTime": "2022-07-22 17:31:11",
26        "deleted": 0,
27        "groupId": "1",
28        "name": "张生",
29        "age": 19,
30        "sex": 0,
31        "address": null
32    }
33 ],
34 "total": 2,
35 "size": 2,
36 "current": 1,
37 "orders": [],
38 "optimizeCountSql": true,
39 "searchCount": true,
40 "countId": null,
41 "maxLimit": null,
42 "pages": 1
43 }
44 },
45 "code": 200
46 }
```

可以看到，查询出了张三和张生两条学生记录，更多查询用法请参考开发进阶中的Query对象。

## 开发进阶

---

### 系统架构图



## @Autowired和@DubboReference

@Autowired是注入本服务内的Bean，而@DubboReference是注入Dubbo的Service。引用一个本服务内的Service推荐使用@Autowired，原因是Mybatis-Plus的QueryWrapper不支持Dubbo的Rpc调用传输。

## Query对象

为了解决Mybatis-Plus的QueryWrapper不支持Dubbo调用的问题，cola-admin提供了Query类，再配合QueryUtil工具类实现了对QueryWrapper的封装，使其不但支持了Dubbo的调用，还支持前端直接传参查询的功能，且支持查询条件的嵌套。下面举例说明

### 一个例子

在上面的StudentController中新建一个测试方法

```
1  @PostMapping("/test")
2  public Result test() {
3      QueryWrapper<StudentEntity> querywrapper = new Querywrapper<>();
4      querywrapper.like("name", "张");
5      List<StudentEntity> list1 = studentService.getList(querywrapper);
6      System.out.println("测试Mybatis-Plus的Querywrapper");
7      list1.forEach(System.out::println);
8
9      Query<StudentEntity> query = new Query<>();
10     query.like("name", "张");
11     List<StudentEntity> list2 = studentService.getList(query);
12     System.out.println("测试Query");
13     list2.forEach(System.out::println);
14
15     return Result.ok();
16 }
```

测试一下看输出结果，报错了

```

1  com.alibaba.com.caucho.hessian.io.HessianFieldException:
   com.baomidou.mybatisplus.core.conditions.segments.MergeSegments.normal:
   'java.lang.invoke.SerializedLambda' could not be instantiated
2      at
   com.alibaba.com.caucho.hessian.io.JavaDeserializer.logDeserializeError(JavaD
   eserializer.java:168)
3      at
   com.alibaba.com.caucho.hessian.io.JavaDeserializer$ObjectFieldDeserializer.d
   eserialize(JavaDeserializer.java:415)
4      at
   com.alibaba.com.caucho.hessian.io.JavaDeserializer.readObject(JavaDeserializ
   er.java:277)
5      at
   com.alibaba.com.caucho.hessian.io.JavaDeserializer.readObject(JavaDeserializ
   er.java:204)
6      at
   com.alibaba.com.caucho.hessian.io.Hessian2Input.readObjectInstance(Hessian2I
   nput.java:2848)
7      at
   com.alibaba.com.caucho.hessian.io.Hessian2Input.readObject(Hessian2Input.jav
   a:2175)
8      at
   com.alibaba.com.caucho.hessian.io.Hessian2Input.readObject(Hessian2Input.jav
   a:2104)
9      at
   com.alibaba.com.caucho.hessian.io.Hessian2Input.readObject(Hessian2Input.jav
   a:2148)
10     at
   com.alibaba.com.caucho.hessian.io.Hessian2Input.readObject(Hessian2Input.jav
   a:2104)

```

说明Dubbo不支持QueryWrapper，我们把QueryWrapper的代码注释掉

```

1  @PostMapping("/test")
2  public Result test() {
3      // QueryWrapper<StudentEntity> queryWrapper = new QueryWrapper<>();
4      // queryWrapper.like("name", "张");
5      // List<StudentEntity> list1 = studentService.getList(queryWrapper);
6      // System.out.println("测试Mybatis-Plus的QueryWrapper");
7      // list1.forEach(System.out::println);
8
9      Query<StudentEntity> query = new Query<>();
10     query.like("name", "张");
11     List<StudentEntity> list2 = studentService.getList(query);
12     System.out.println("测试Query");
13     list2.forEach(System.out::println);
14
15     return Result.ok();
16 }

```

重启一下WebDomainApplication再看结果

```

===== 请求开始 =====
请求地址: POST: /student/test
请求参数: 无
请求头: token : eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
      .eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
      U00DIwLCJpYXQiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.NiEYBMu1yGm3H9DvM6logKAXz70NDiLCCNqugNIxz-A
请求头: content-type : application/json
请求头: user-agent : PostmanRuntime/7.26.8
请求头: accept : */*
请求头: cache-control : no-cache
请求头: postman-token : 0db970fa-b97b-45dc-9fe3-92aacc97edb7
请求头: host : localhost:8085
请求头: accept-encoding : gzip, deflate, br
请求头: connection : keep-alive
请求头: content-length : 319

测试Query
StudentEntity(name=张三, age=16, sex=1, address=null)
StudentEntity(name=张生, age=19, sex=0, address=null)
2022-07-25 11:42:31.851 INFO 12360 --- [nio-8085-exec-3] c.m.c.w.c.aspect.ControllerLogAspect :

返回结果:
{"success":true,"msg":"操作成功! ","data":{},"code":200}
响应地址: POST: /student/test
执行时间: 41 ms
===== 请求结束 =====

```

可以看到查询成功

## 查询方法

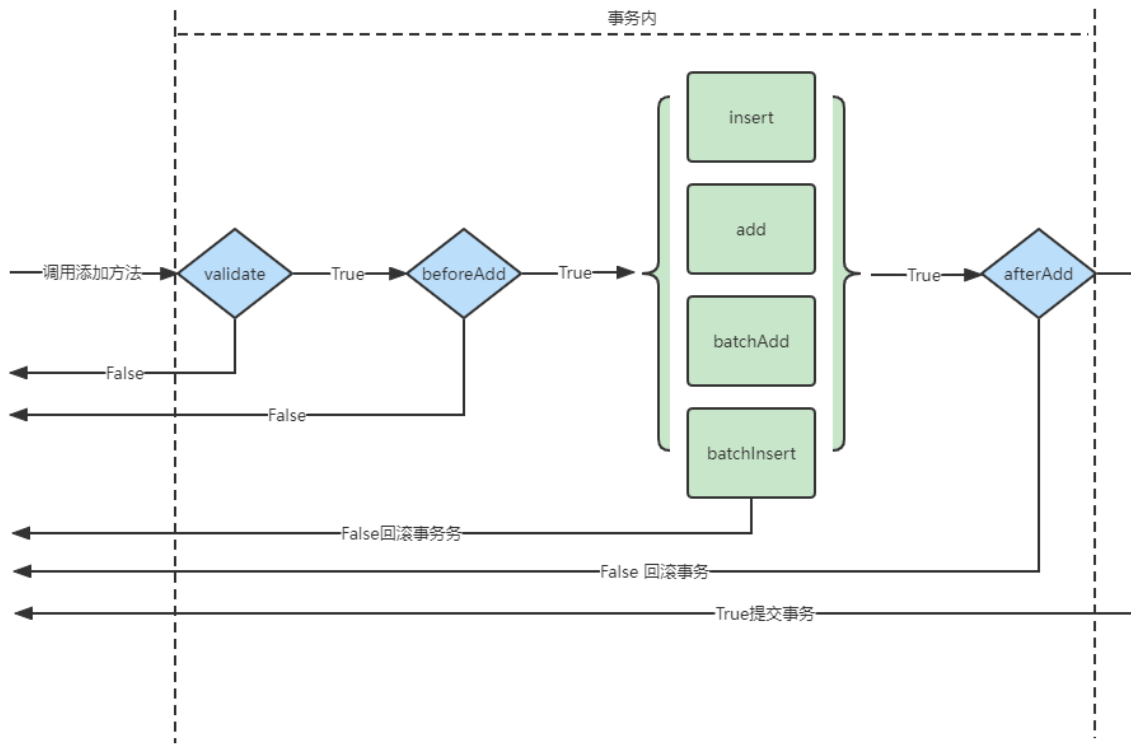
与QueryWrapper一样，Query类提供了大部分QueryWrapper一样的方法，如下表

函数名	说明	例子
eq	等于=	eq("name","老王") -> name='老王'
ne	不等于<>	ne("name","老王") -> name<>'老王'
gt	大于>	gt("age",20) -> age>20
ge	大于等于>=	ge("age",20) -> age>=20
lt	小于<	lt("age",20) -> age<20
le	小于等于<=	le("age",20) -> age<=20
between	between 值1 and 值2	between("age",10,20) -> age between 10 and 20
notBetween	not between 值1 and 值2	notBetween("age",10,20) -> age not between 10 and 20
like	like '%值%'	like("name","王") -> name like '%王%'
notLike	not like '%值%'	notLike("name","王") -> name not like '%王%'
likeLeft	like '%值'	likeLeft("name","王") -> name like '%王'
likeRight	like '值%'	likeRight("name","王") -> name like '王%'
isNull	字段 is null	isNull("name") -> name is null
isNotNull	字段 is not null	isNotNull("name") -> name is not null
in	字段 in (值1, 值2, 值3)	in("id","1,2,3") -> id in ('1','2','3') 或 in("id",{1,2,3}) -> id in (1,2,3)
notIn	字段 not in (值1, 值2, 值3)	notIn("id","1,2,3") -> id not in ('1','2','3') 或 notIn("id",{1,2,3}) -> id not in (1,2,3)
orderBy	order by 字段	orderBy("id") -> order by id 或 orderBy("id",false) order by id desc

## CRUD生命周期

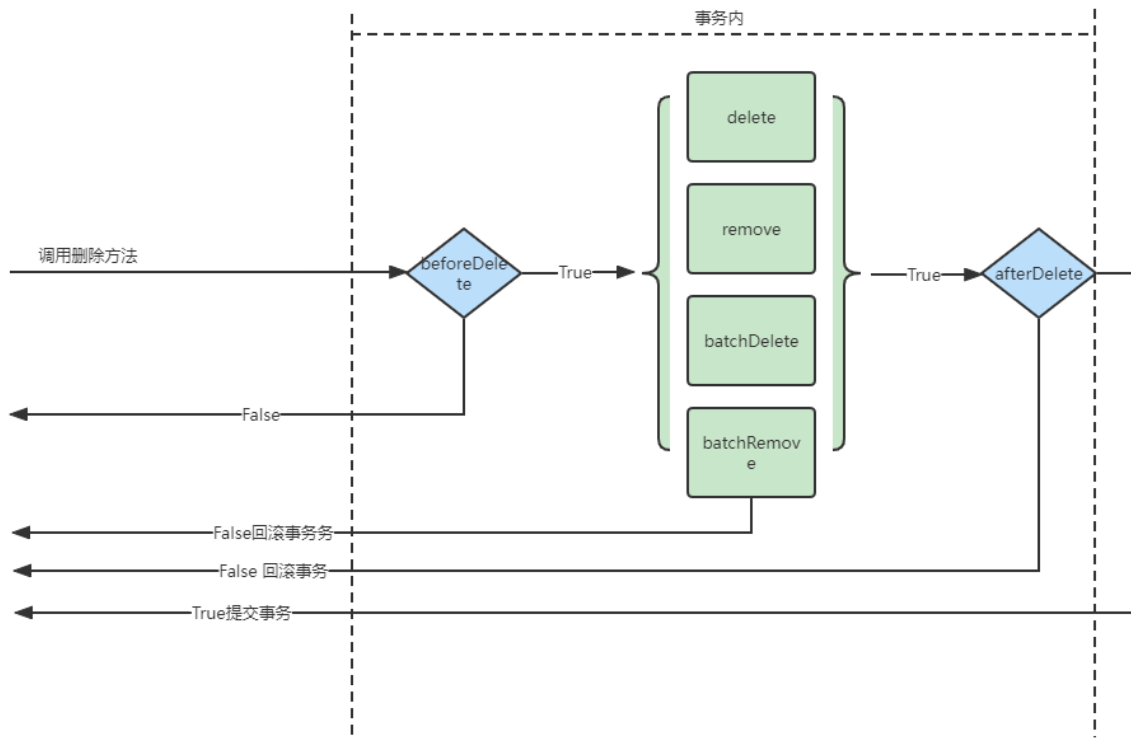
cola-admin对于CRUD操作进行了封装，并可以根据需要进行扩展

### 添加过程生命周期



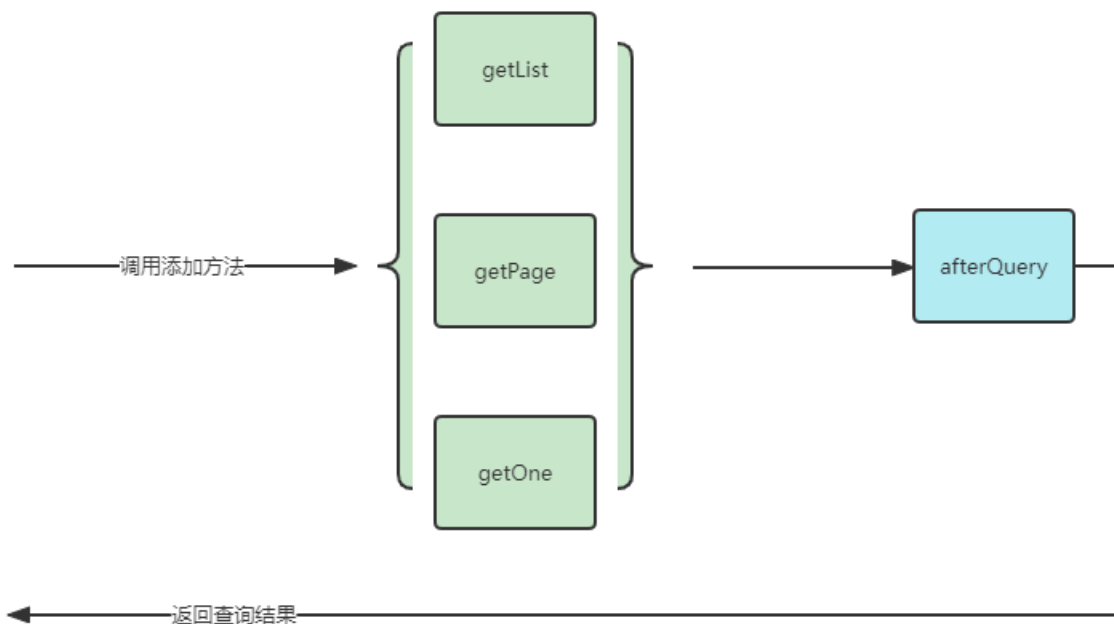
## 修改过程生命周期

## 删除过程生命周期





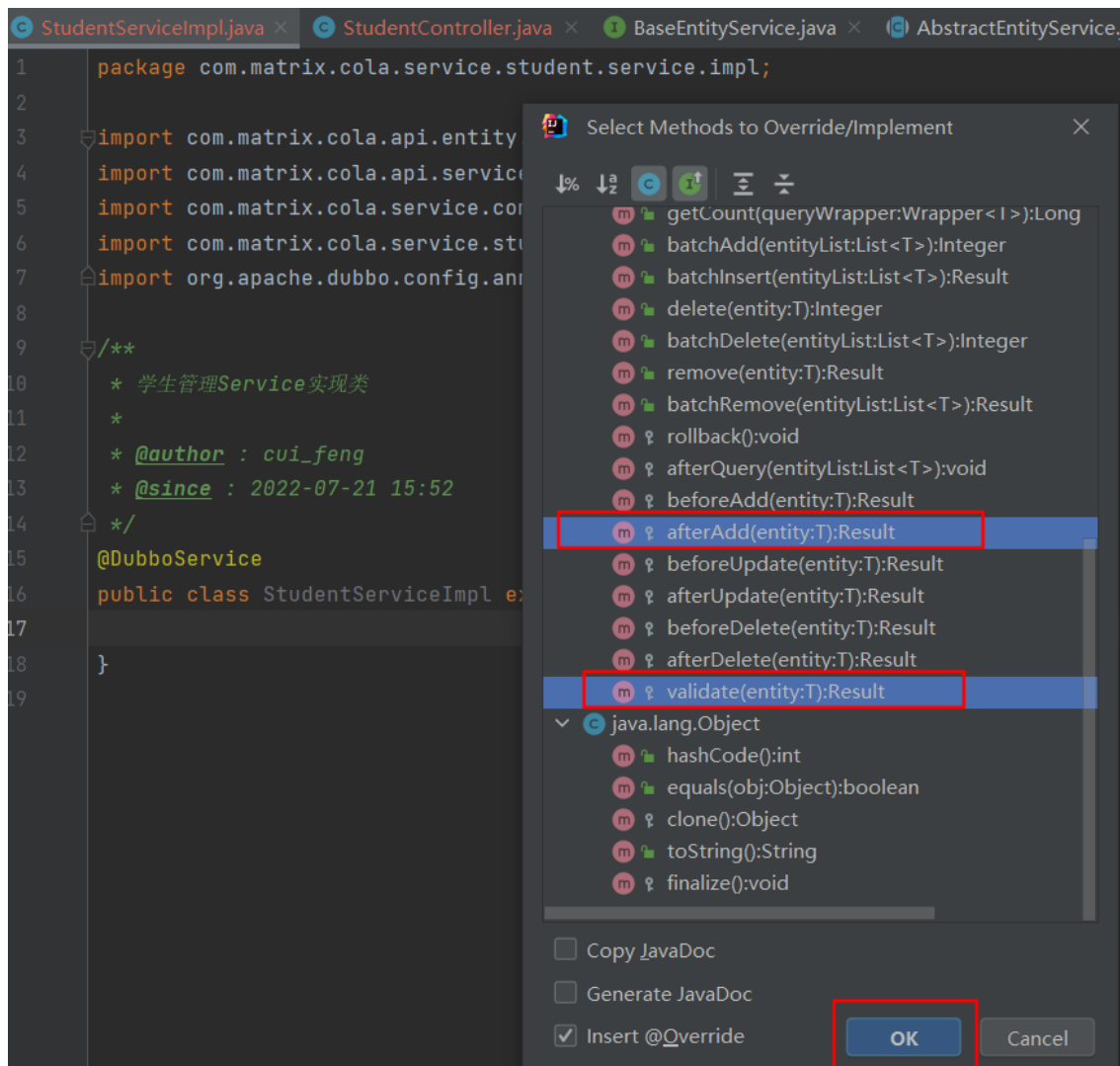
## 查询过程生命周期



## 生命周期怎么用

以学生管理为例，现在我们要求学生的姓名为必填项，长度最大为4，添加学生时，同时生成一条学生证的信息。

1. 进入StudentServiceImpl类，按下Ctrl+O快捷键



2. 选择validate和afterAdd方法，点击OK

```
@DubboService
public class StudentServiceImpl extends AbstractColaEntityService<StudentEntity> {
    4 usages
    @Override
    protected Result afterAdd(StudentEntity entity) {
        return super.afterAdd(entity);
    }

    @Override
    protected Result validate(StudentEntity entity) {
        return super.validate(entity);
    }
}
```

3. 实现validate

```
1  @Override
2  protected Result validate(StudentEntity entity) {
3      if (ObjectUtil.isNull(entity)) {
4          return Result.err("学生信息不能为空");
5      }
6      if (StrUtil.isEmpty(entity.getName())) {
7          return Result.err("姓名不能为空");
8      }
9      if (entity.getName().length() > 4) {
10         return Result.err("姓名不能超过四个字");
11     }
12     return super.validate(entity);
13 }
```

4. 实现afterAdd

```
1  @Override
2  protected Result afterAdd(StudentEntity entity) {
3      // 添加学生证
4      Result result = studentIdCardService.addIdCard(entity);
5      if (!result.isSuccess()) {
6          // 回滚事务，学生信息和学生证信息同时回滚
7          rollback();
8          return result;
9      }
10     return super.afterAdd(entity);
11 }
```

5. 最终效果

POST localhost:8085/aut...

POST http://localhost:808...

POST http://localhost:808...

POST http://lc

Intitled Request

POST

http://localhost:8085/student/addStudent

ParamsAuthorizationHeaders (10)BodyPre-request ScriptTestsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

1{  
2  "name": "",  
3  "age": 18,  
4  "sex": 0,  
5  "address": "山东省济南市"  
6}

bodyCookiesHeaders (14)Test Results

PrettyRawPreviewVisualizeJSON

1{  
2  "success": false,  
3  "msg": "姓名不能为空",  
4  "data": {},  
5  "code": 500  
6}

Service中自定义的方法不在生命周期中，如果需要请自行调用。

## EntityWrapper

EntityWrapper用于解决前端页面显示问题，一般我们在一个表中会存储另一个表的id进行关联，对应的，Entity中也只存了另一个对象的id字段，可是有时候前端查询的时候会要求显示出另一个表中的其他信息，这时候就用到了EntityWrapper。

### 举个例子

我们再新建一个选课表

```
1 CREATE TABLE `student_course` (  
2   `id` bigint(64) NOT NULL ,  
3   `student_id` bigint(64) NULL COMMENT '学生' ,  
4   `course_name` varchar(100) NULL COMMENT '课程名称' ,  
5   `creator` bigint(64) NULL COMMENT '创建人' ,  
6   `create_time` datetime NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '创建时间' ,  
7   `reviser` bigint(64) NULL COMMENT '修改人' ,  
8   `revise_time` datetime NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '修改是时间' ,  
9   `deleted` int(2) NULL DEFAULT 0 COMMENT '是否删除, 0=未删除, 1=已删除' ,  
10  `group_id` varchar(64) NULL COMMENT '所属机构' ,  
11  PRIMARY KEY (`id`)  
12 );
```

创建选课Entity

```

1  @Data
2  @TableName("student_course")
3  public class StudentCourseEntity extends BaseEntity {
4
5      private Long studentId;
6      private String courseName;
7  }

```

创建选课Service

```

1  public interface StudentCourseService extends
    BaseEntityService<StudentCourseEntity> {
2  }

```

创建选课Mapper

```

1  public interface StudentCourseMapper extends BaseMapper<StudentCourseEntity>
    {
2  }

```

创建选课Service实现类

```

1  @DubboService
2  public class StudentCourseServiceImpl extends
    AbstractColaEntityService<StudentCourseEntity,
                                StudentCourseMapper> implements
    StudentCourseService {
3
4  }

```

创建选课Controller，为方便测试，只添加查询方法

```

1  @RestController
2  @RequestMapping("/studentCourse")
3  public class StudentCourseController {
4
5      @DubboReference
6      StudentCourseService studentCourseService;
7
8      @PostMapping("getList")
9      public Result getList() {
10         return Result.list(studentCourseService.getList(new Query<>()));
11     }
12 }

```

添加几条数据

```

1  INSERT INTO `student_course` VALUES ('1', '2', '语文', '1', '2022-07-25
    15:30:18', '1', '2022-07-25 15:30:18', '0', '1');
2  INSERT INTO `student_course` VALUES ('2', '3', '数学', '1', '2022-07-25
    15:30:18', '1', '2022-07-25 15:30:18', '0', '1');
3  INSERT INTO `student_course` VALUES ('3', '4', '地理', '1', '2022-07-25
    15:30:26', '1', '2022-07-25 15:30:26', '0', '1');

```

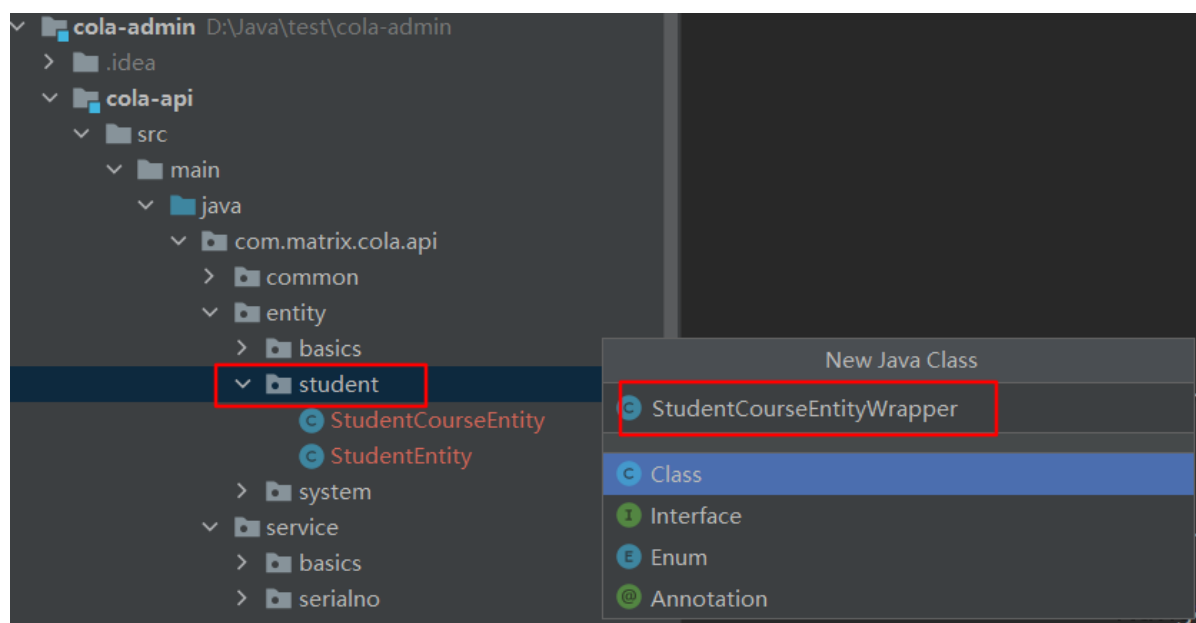
查看查询结果

```
1  {
2      "success": true,
3      "msg": "操作成功! ",
4      "data": {
5          "list": [
6              {
7                  "id": 1,
8                  "creator": 1,
9                  "createTime": "2022-07-25 15:30:18",
10                 "reviser": 1,
11                 "reviseTime": "2022-07-25 15:30:18",
12                 "deleted": 0,
13                 "groupId": "1",
14                 "studentId": 2,
15                 "courseName": "语文"
16             },
17             {
18                 "id": 2,
19                 "creator": 1,
20                 "createTime": "2022-07-25 15:30:18",
21                 "reviser": 1,
22                 "reviseTime": "2022-07-25 15:30:18",
23                 "deleted": 0,
24                 "groupId": "1",
25                 "studentId": 3,
26                 "courseName": "数学"
27             },
28             {
29                 "id": 3,
30                 "creator": 1,
31                 "createTime": "2022-07-25 15:30:26",
32                 "reviser": 1,
33                 "reviseTime": "2022-07-25 15:30:26",
34                 "deleted": 0,
35                 "groupId": "1",
36                 "studentId": 4,
37                 "courseName": "地理"
38             }
39         ]
40     },
41     "code": 200
42 }
```

可以看到查到了三条数据，但是我们想除了查询学生的id外还要查询学生的姓名，这时候就需要EntityWrapper了

## 创建EntityWrapper

1. 在StudentCourseEntity类的包下创建StudentCourseEntityWrapper



2. 添加Lombok的@Data注解并复制StudentCourseEntity中的全部字段到StudentCourseEntityWrapper中

```
@Data
public class StudentCourseEntityWrapper {

    private Long studentId;
    private String courseName;
}
```

3. 添加需要包装的字段，这里添加studentName

```
@Data
public class StudentCourseEntityWrapper {

    private Long studentId;
    private String courseName;

    // 需要包装的字段
    private String studentName;
}
```

4. 继承抽象类

同Entity一样，EntityWrapper也提供了两个抽象类可以继承，一个是带业务字段的BaseColaEntityWrapper，该抽象类中添加了创建人、修改人、组织机构的包装字段，直接可以使用；另一个是BaseEntityWrapper，该抽象类中没有业务字段。

```

@Data
public class StudentCourseEntityWrapper extends BaseColaEntityWrapper {

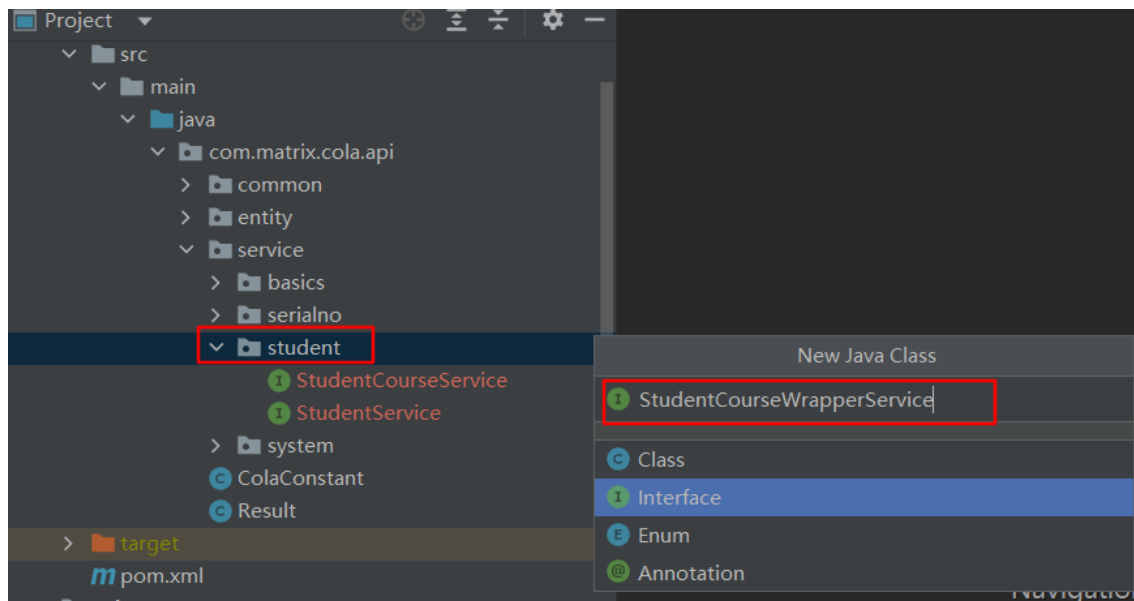
    private Long studentId;
    private String courseName;

    // 需要包装的字段
    private String studentName;
}

```

## 创建WrapperService

### 1. 创建StudentCourseWrapperService接口



### 2. 继承父接口

同EntityService一样，WrapperService也提供了两个父接口，一个是BaseColaEntityWrapperService，另一个是BaseEntityWrapperService。如果EntityWrapper继承了BaseColaEntityWrapper，那么接口就需要继承BaseColaEntityWrapperService；如果继承了BaseEntityWrapper，那么接口就需要继承BaseEntityWrapperService。

这里我们继承BaseColaEntityWrapperService

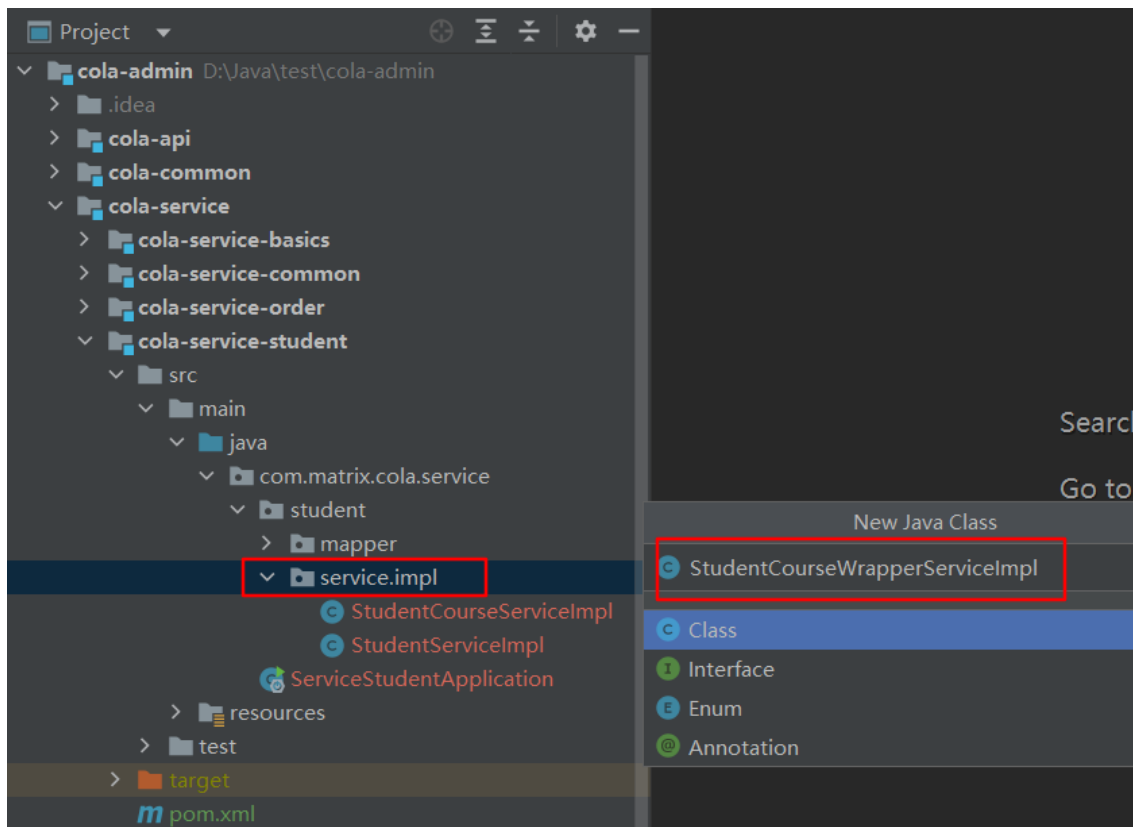
```

1 public interface StudentCourseWrapperService extends
  BaseColaEntityWrapperService<StudentCourseEntity,
    StudentCourseEntityWrapper> {
2 }

```

这里BaseColaEntityWrapperService需要两个泛型，一个是BaseColaEntity的子类，一个是BaseColaEntityWrapper的子类

### 3. 创建EntityWrapperService的实现类



#### 4. 继承抽象类

同Service一样，WrapperService有两个抽象类可以继承，一个是AbstractColaEntityWrapperService，另一个是AbstractEntityWrapperService。规则同Service的继承规则一样。这里继承AbstractColaEntityWrapperService。

```
1 public class StudentCourseWrapperServiceImpl extends
  AbstractColaEntityWrapperService<StudentCourseEntity,
  StudentCourseEntityWrapper, StudentCourseService> implements
  StudentCourseWrapperService {
2 }
3
```

AbstractColaEntityWrapperService需要三个泛型，第一个是BaseColaEntity的子类，第二个是BaseColaEntityWrapper的子类，第三个是BaseColaEntityService接口的实现类。

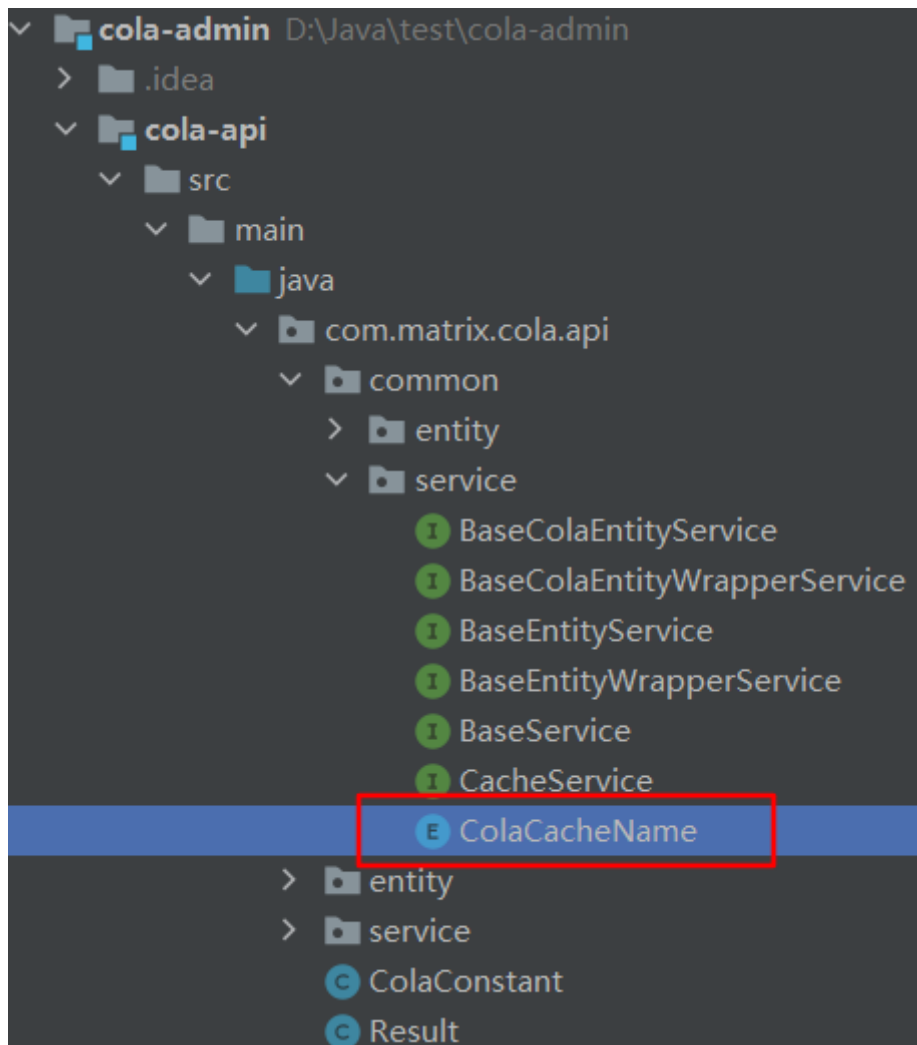
添加@DubboService注解

```
@DubboService
public class StudentCourseWrapperServiceImpl extends AbstractColaEntityWrapperService
```

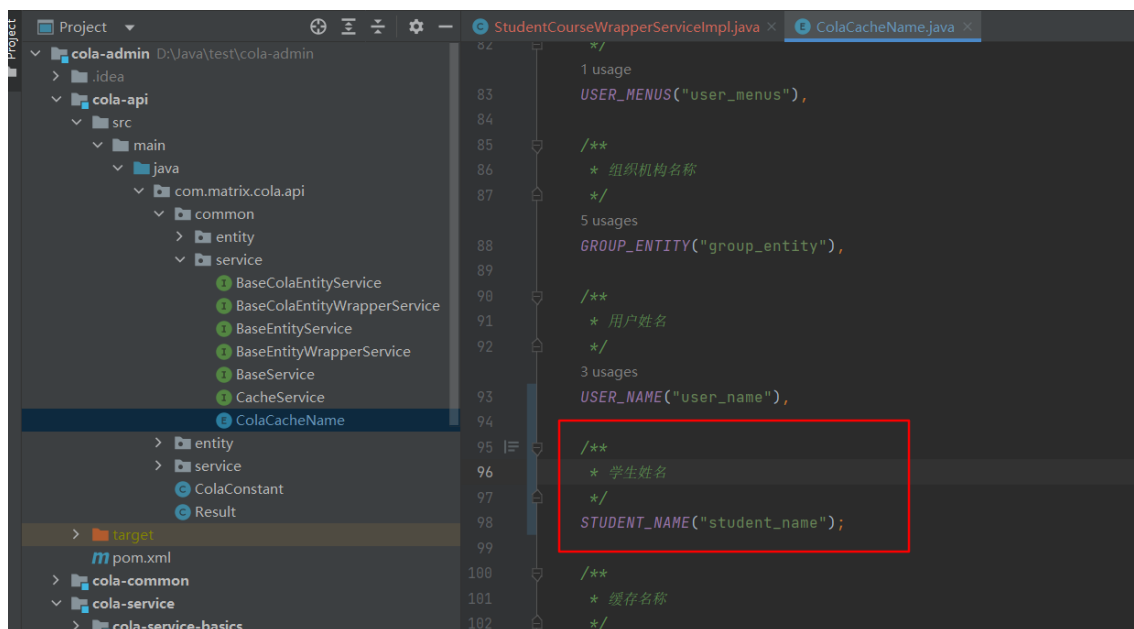
#### 5. 重写包装方法

打开ColaCacheName类

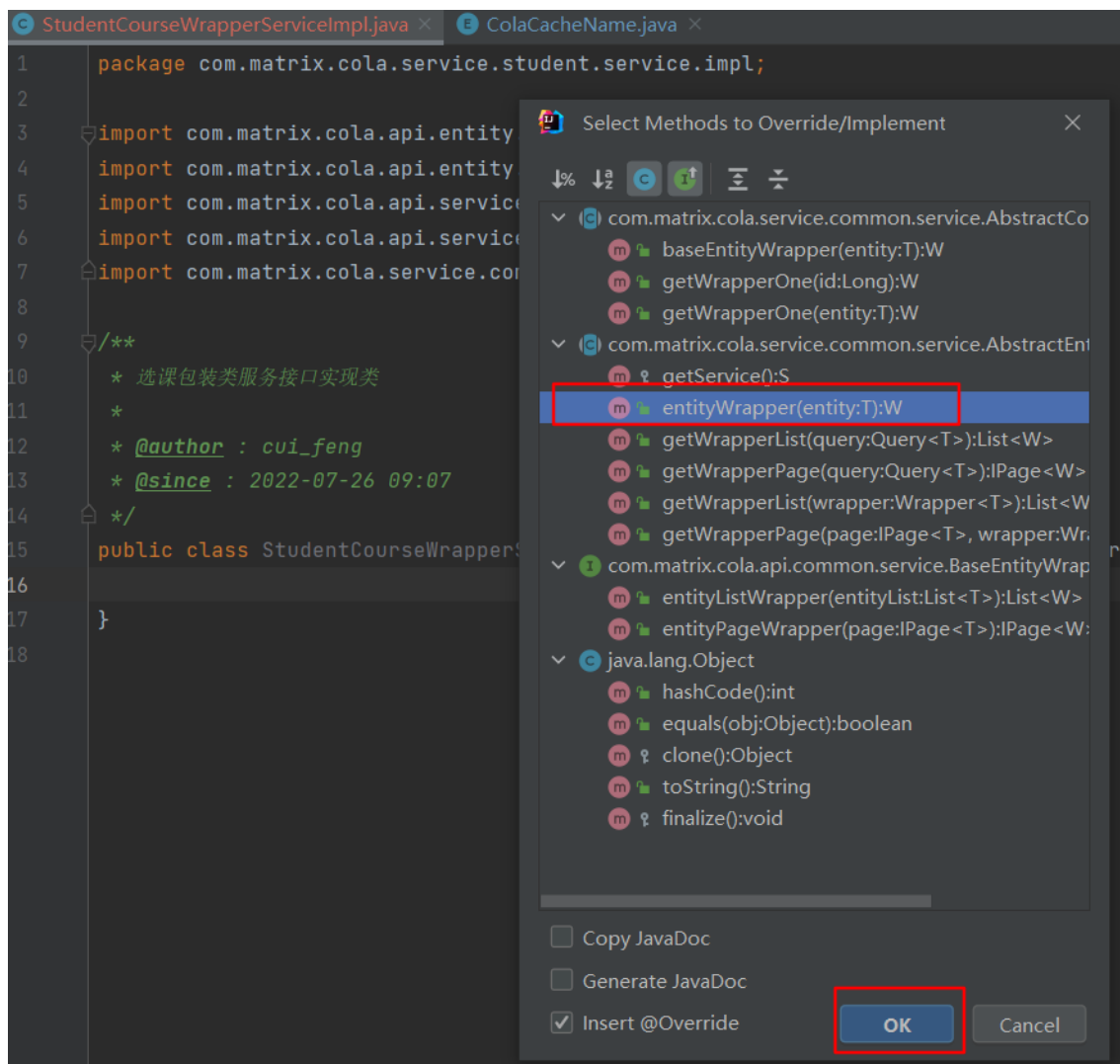




添加缓存名称



在StudentCourseWrapperServiceImpl中按下Ctrl+O键，选择entityWrapper方法，点OK



重写entityWrapper方法

```
1 @DubboService
2 public class StudentCourseWrapperServiceImpl extends
AbstractColaEntityWrapperService<StudentCourseEntity,
StudentCourseEntityWrapper, StudentCourseService> implements
StudentCourseWrapperService {
3
4     // 由于StudentService的实现类与当前类在同一服务中，故使用@Autowired注解，如
果使用@DubboReference注解，studentService将无法使用QueryWrapper，所以推荐服务
内的相互引用使用@Autowired注解
5     @Autowired
6     StudentService studentService;
7
8     @Override
9     public StudentCourseEntityWrapper entityWrapper(StudentCourseEntity
entity) {
10         StudentCourseEntityWrapper studentCourseEntityWrapper = new
StudentCourseEntityWrapper();
11         if (ObjectUtil.isNotNull(entity.getStudentId())) {
12             String studentName =
cacheProxy.getObjectFromLoader(ColaCacheName.STUDENT_NAME,
entity.getStudentId().toString(), () -> {
13                 StudentEntity student =
studentService.getOne(entity.getStudentId());
14                 if (ObjectUtil.isNotNull(student)) {
15                     return null;
16                 }
17             })
18         }
19     }
20 }
```

```
16     }
17     return student.getName();
18 });
19     studentCourseEntityWrapper.setStudentName(studentName);
20 }
21     return studentCourseEntityWrapper;
22 }
23 }
```

## 6. 使用EntityWrapperService

## 修改StudentCourseController

```
1 @RestController
2 @RequestMapping("/studentCourse")
3 public class StudentCourseController {
4
5     @DubboReference
6     StudentCourseWrapperService studentCourseWrapperService;
7
8     @PostMapping("getList")
9     public Result getList() {
10         return
11         Result.list(studentCourseWrapperService.getWrapperList(new Query<>()));
12     }
13 }
```

## 7. 查看查询结果

```
1  {
2    "success": true,
3    "msg": "操作成功!",
4    "data": {
5      "list": [
6        {
7          "id": 1,
8          "creator": 1,
9          "creatorName": "超级管理员",
10         "createTime": "2022-07-25 15:30:18",
11         "startTime": null,
12         "endTime": null,
13         "reviser": 1,
14         "reviserName": "超级管理员",
15         "reviseTime": "2022-07-25 15:30:18",
16         "deleted": 0,
17         "showDeleted": null,
18         "groupId": "1",
19         "groupName": "集团公司",
20         "studentId": 2,
21         "courseName": "语文",
22         "studentName": "张三"
23       },
24       {
25         "id": 2,
26         "creator": 1,
27         "creatorName": "超级管理员",
28         "createTime": "2022-07-25 15:30:18",
```

```

29         "startTime": null,
30         "endTime": null,
31         "reviser": 1,
32         "reviserName": "超级管理员",
33         "reviseTime": "2022-07-25 15:30:18",
34         "deleted": 0,
35         "showDeleted": null,
36         "groupId": "1",
37         "groupName": "集团公司",
38         "studentId": 3,
39         "courseName": "数学",
40         "studentName": "王五"
41     },
42     {
43         "id": 3,
44         "creator": 1,
45         "creatorName": "超级管理员",
46         "createTime": "2022-07-25 15:30:26",
47         "startTime": null,
48         "endTime": null,
49         "reviser": 1,
50         "reviserName": "超级管理员",
51         "reviseTime": "2022-07-25 15:30:26",
52         "deleted": 0,
53         "showDeleted": null,
54         "groupId": "1",
55         "groupName": "集团公司",
56         "studentId": 4,
57         "courseName": "地理",
58         "studentName": "赵六"
59     }
60 ]
61 },
62 "code": 200
63 }

```

可以看到成功显示了学生姓名，而且创建人、修改人、组织机构也都显示正确。

## 数据日志

cola-admin中提供了DataLogService来记录数据日志，接口定义如下

```

1  public interface DataLogService extends BaseColaEntityService<DataLogEntity>
2  {
3      /**
4       * 记录修改日志
5       * @param tableName 表名
6       * @param before 更新前的数据
7       * @param after 更新后的数据
8       */
9       void addUpdateLog(String tableName, BaseColaEntity before,
10                        BaseColaEntity after);
11
12     /**
13      * 记录删除日志
14      * @param tableName 表名

```

```

14      * @param before 删除前的记录
15      */
16      void addDeleteLog(String tableName, BaseColaEntity before);
17
18      /**
19       * 物理删除数据日志
20       * @param dataLogEntity 数据日志实体类
21       * @return 统一结果
22       */
23      Result deleteDataLog(DataLogEntity dataLogEntity);
24
25      /**
26       * 删除全部数据日志
27       * @return 统一结果
28       */
29      Result clearDataLog();
30  }

```

使用方式，例如需要修改学生信息和删除学生信息时需要记录日志，

## 工具类说明

DubboUtil

WebUtil

CacheProxy

## FAQ

### 1、添加服务后启动报错

```

1      *****
2      APPLICATION FAILED TO START
3      *****
4
5      Description:
6
7      Failed to configure a DataSource: 'url' attribute is not specified and no
      embedded datasource could be configured.
8
9      Reason: Failed to determine a suitable driver class

```

请打开maven面板刷新整个工程

