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

```
cola-admin
├──cola-api                # api接口封装，entity/服务接口
├──cola-common             # 全局服务公共模块
├──cola-service            # 业务服务
|   ├─cola-service-basics  # 项目基础服务（系统服务和基础服务）
|   ├─cola-service-common  # 业务服务公共模块
|   └─cola-service-order   # 订单服务（空服务）
├──cola-web                # web接口服务
|   ├─cola-web-auth        # 鉴权模块
|   ├─cola-web-common      # web接口服务公共模块
|   └─cola-web-domain      # web接口主服务
└──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

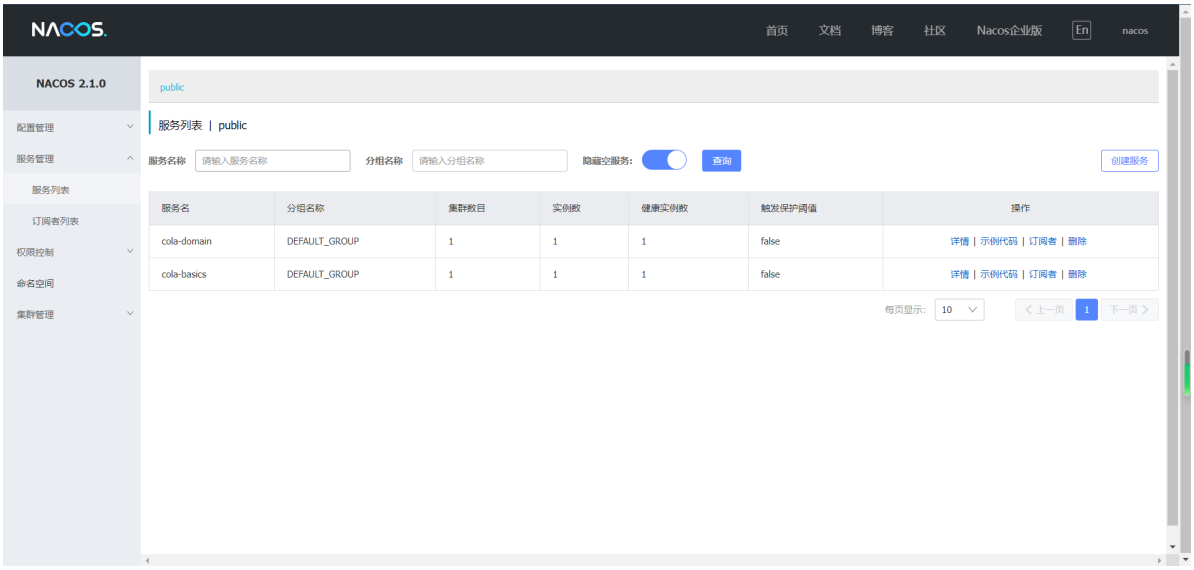
环境准备

Nacos安装

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

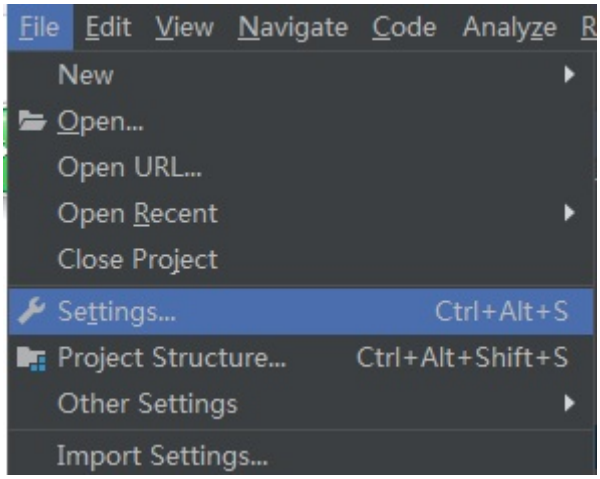
Nacos界面

默认用户名和密码都是nacos

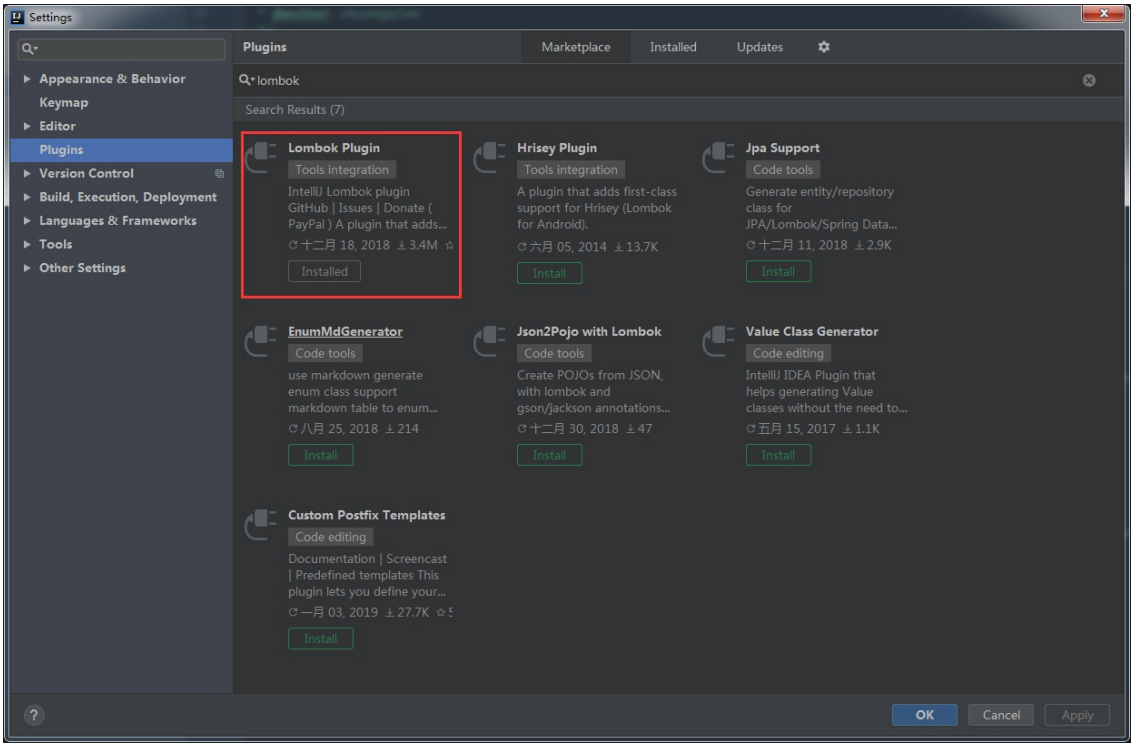


IDEA 插件安装

1. 选择 File->Settings



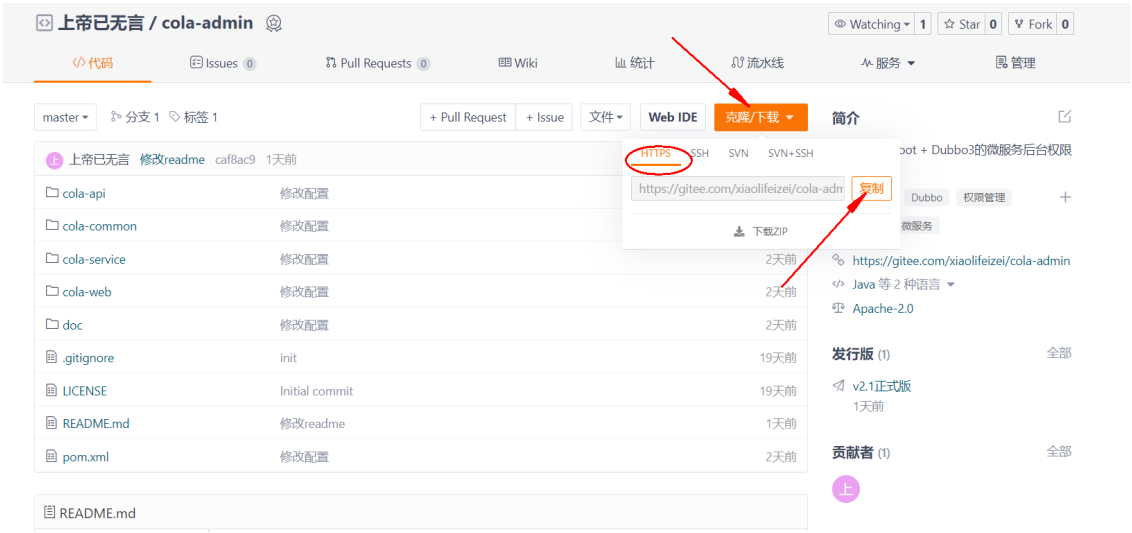
2. 选择 Plugins 并搜索 Lombok



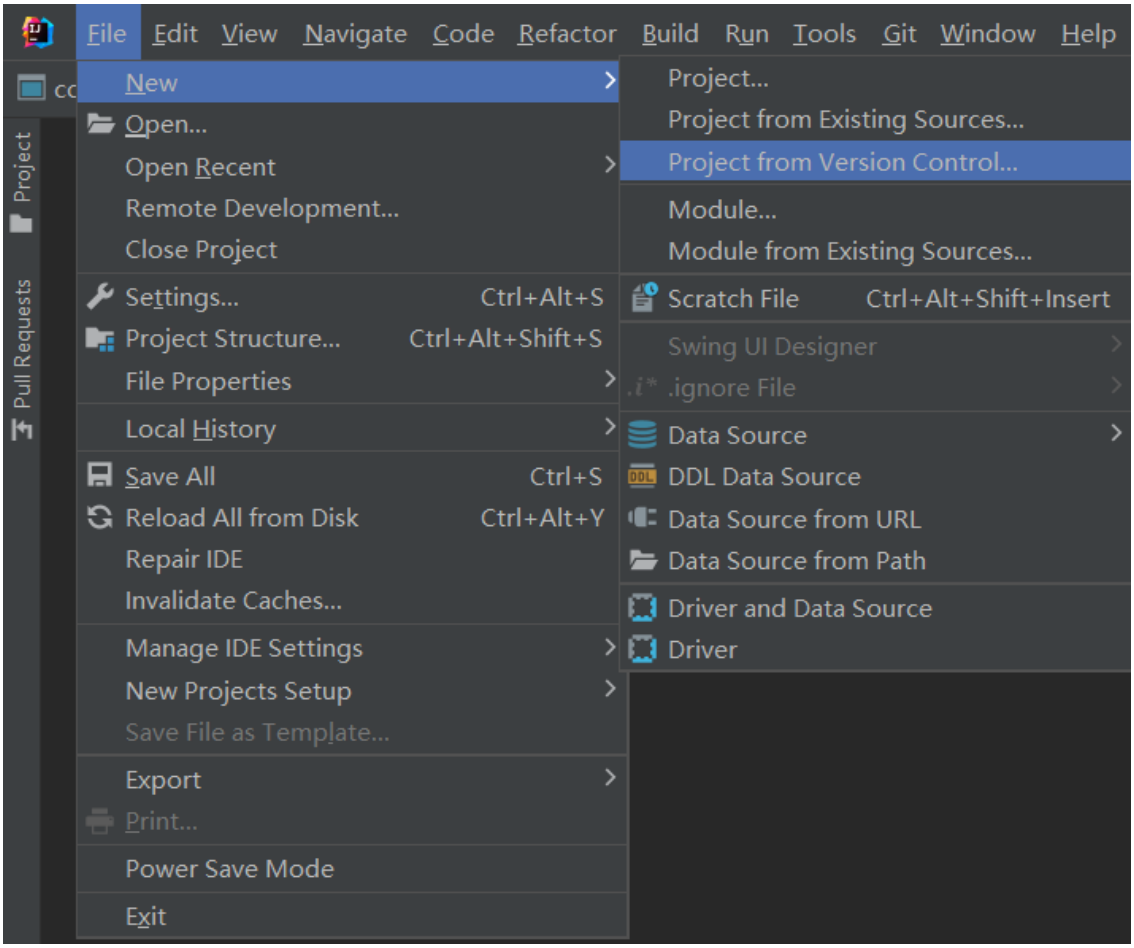
3. 点击 Install 按钮
4. 重启 idea 生效

导入工程

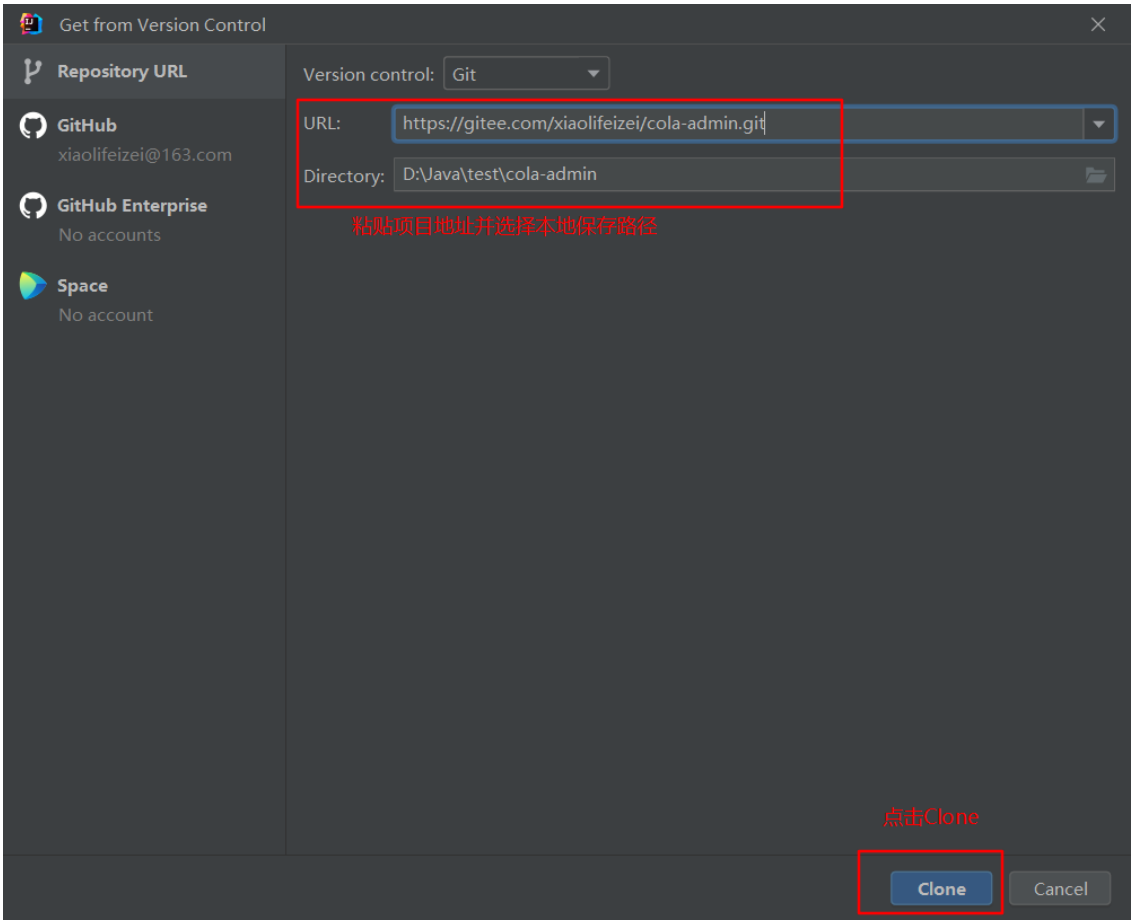
1. 进入cola-admin项目首页https://gitee.com/xiaolifeizei/cola-admin
2. 复制项目地址



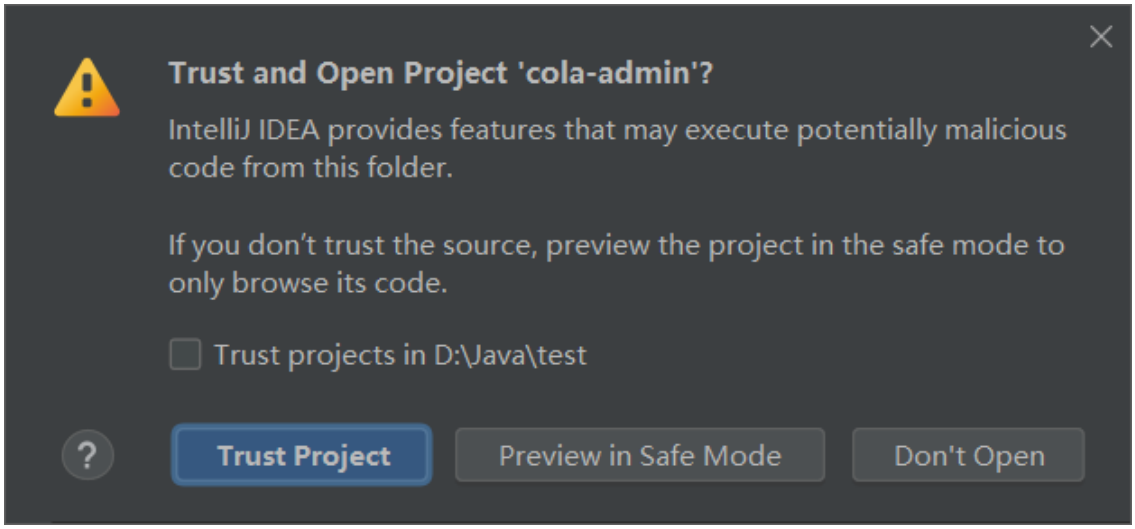
3. 打开IDEA，依次选择： File->New->Project from Version Control



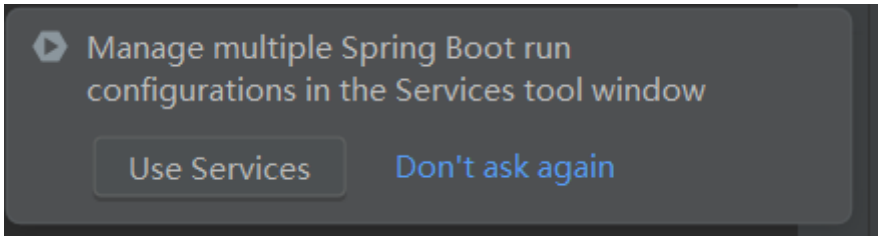
4. 在弹出的对话框中粘贴复制的项目地址



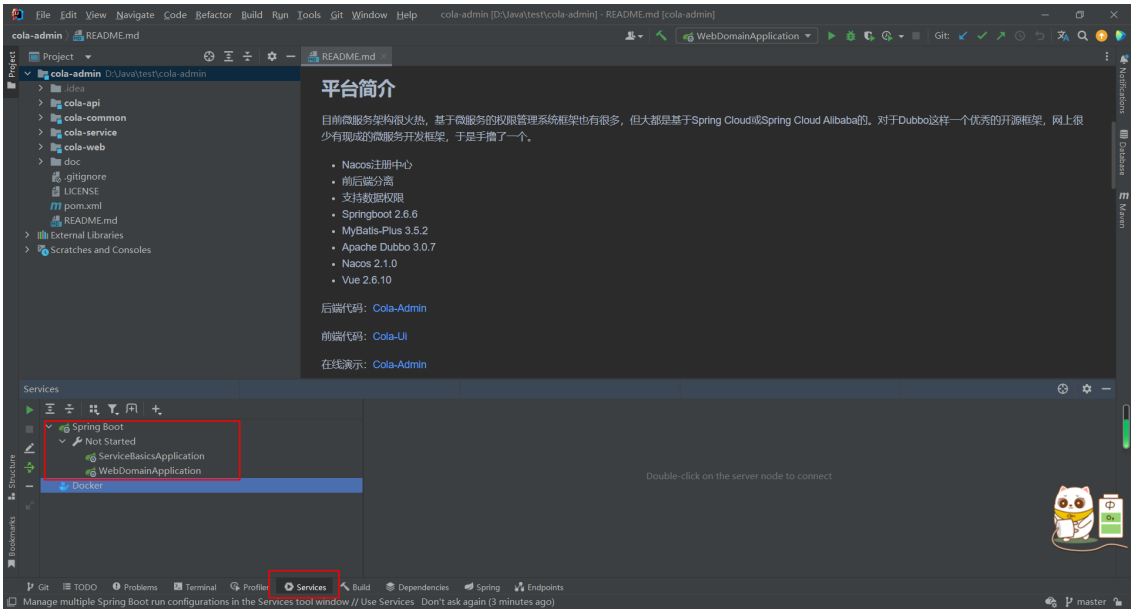
5. IDEA可能会弹出对话框提示，点击“Trust Project”



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

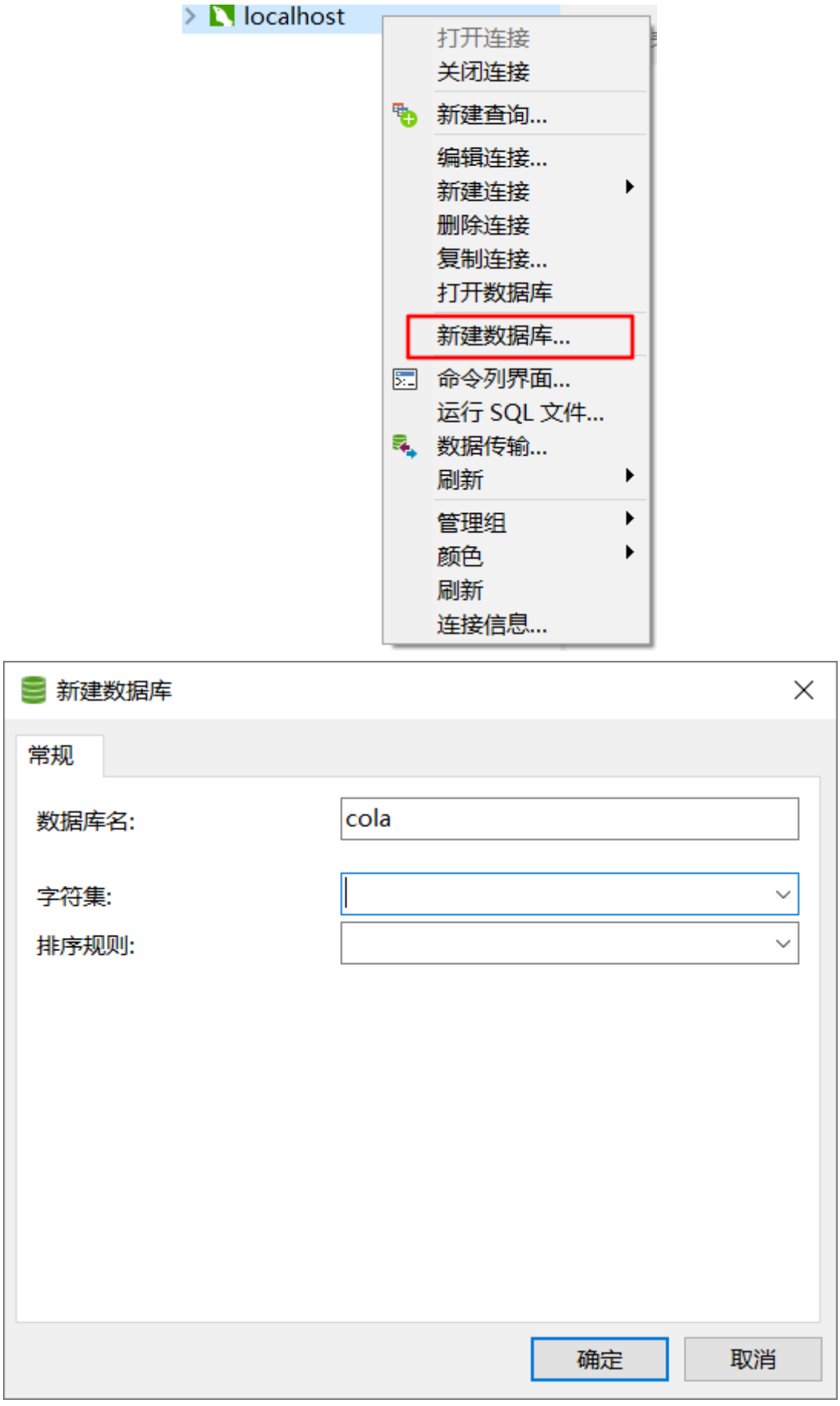


8. 点击Service面板可以看到下图的启动项则说明导入成功

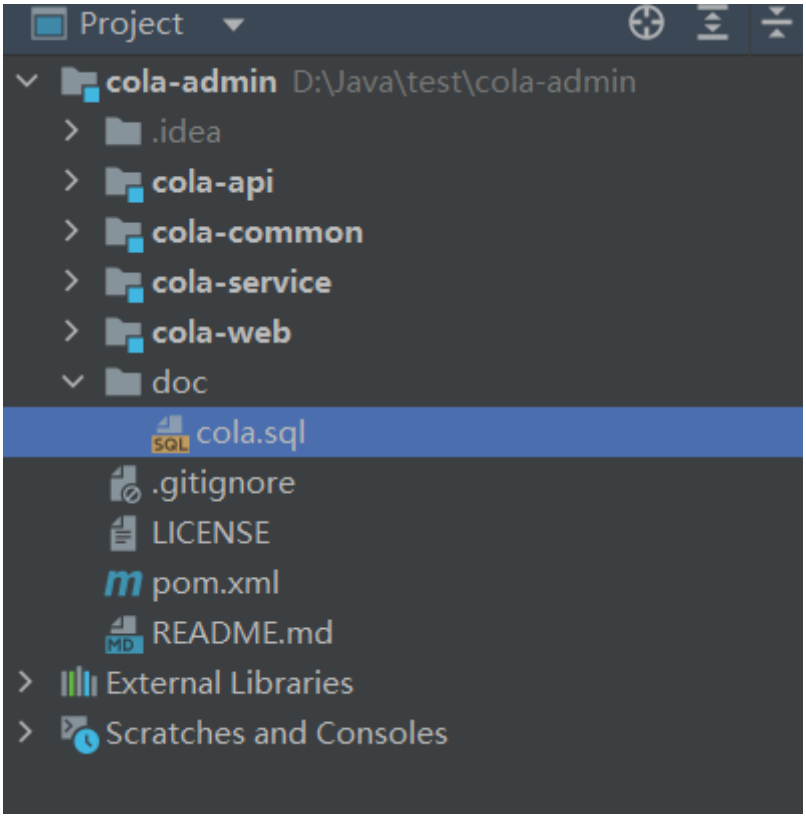


创建数据库

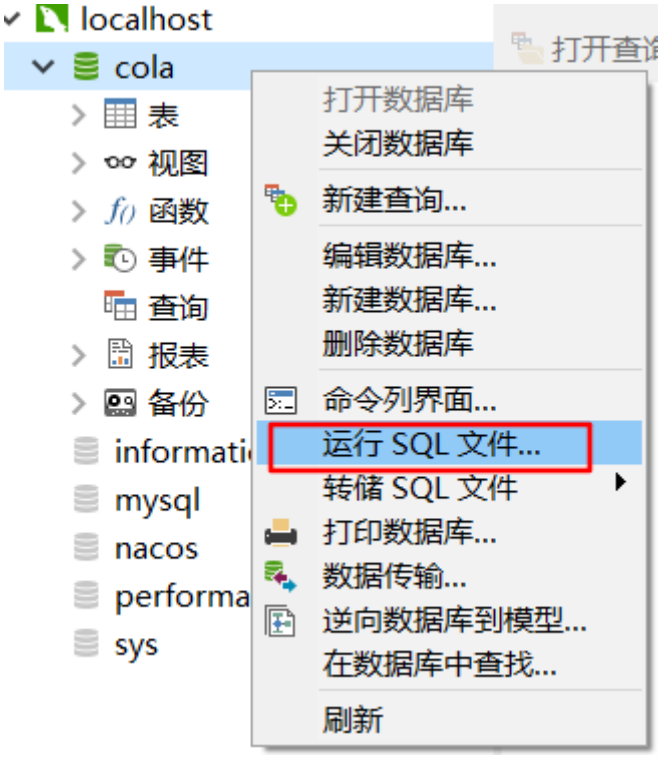
1. 打开Navicat（此处可以选择其他的客户端），新建一个数据库cola

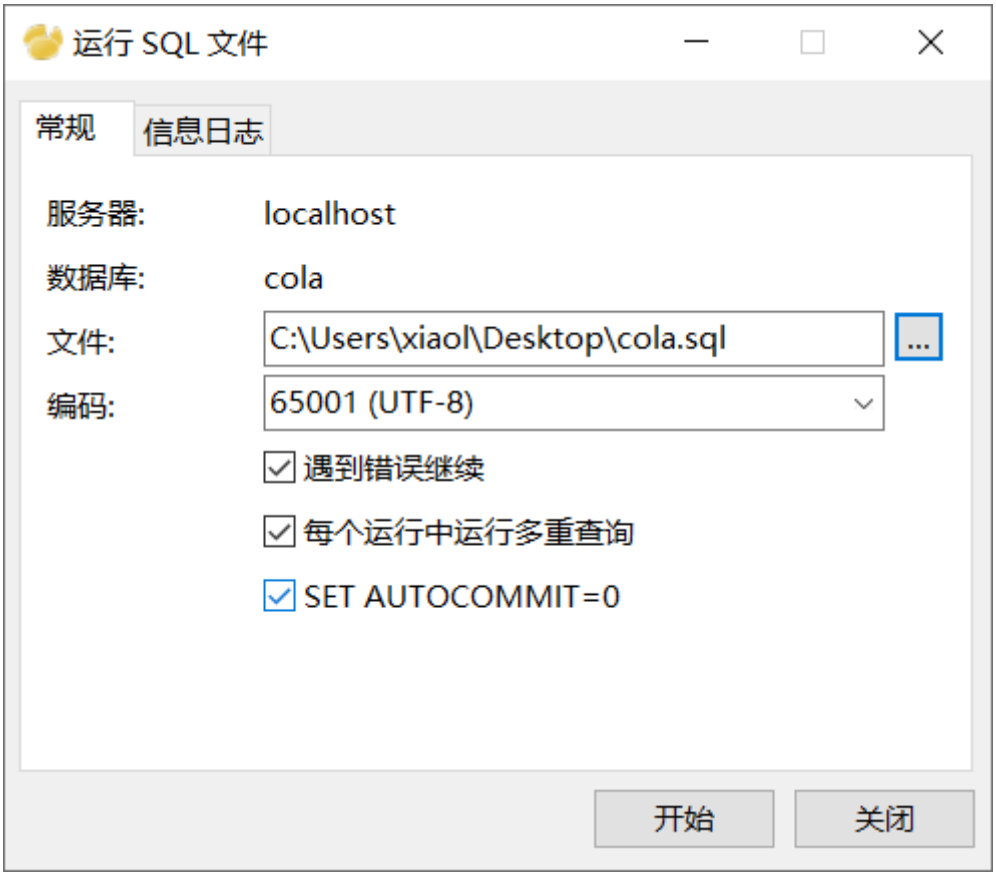


2. 找到cola-admin工程->doc->cola.sql

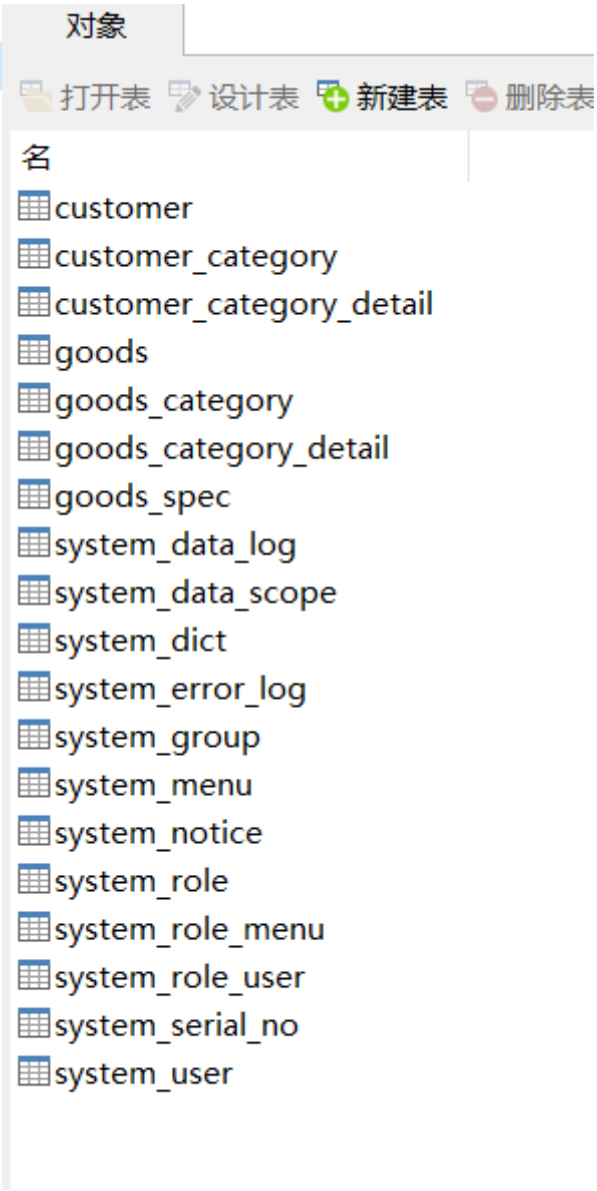

Project
cola-admin D:\Java\test\cola-admin
.idea
cola-api
cola-common
cola-service
cola-web
doc
cola.sql
.gitignore
LICENSE
pom.xml
README.md
External Libraries
Scratches and Consoles

3. 执行sql脚本


localhost
cola
表
视图
函数
事件
查询
报表
备份
mysql
nacos
sys
打开数据库
关闭数据库
新建查询...
编辑数据库...
新建数据库...
删除数据库
命令列界面...
运行 SQL 文件...
转储 SQL 文件
打印数据库...
数据传输...
逆向数据库到模型...
在数据库中查找...
刷新


运行 SQL 文件
常规
信息日志
服务器:
localhost
数据库:
cola
文件:
C:\Users\xiaol\Desktop\cola.sql
...
编码:
65001 (UTF-8)
遇到错误继续
每个运行中运行多重查询
SET AUTOCOMMIT=0
开始
关闭

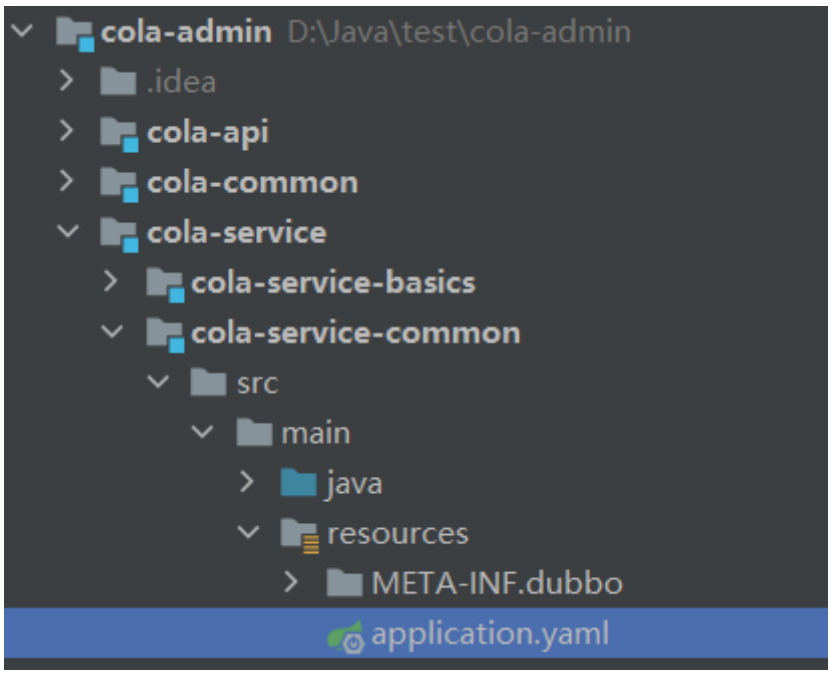
4. 最终效果



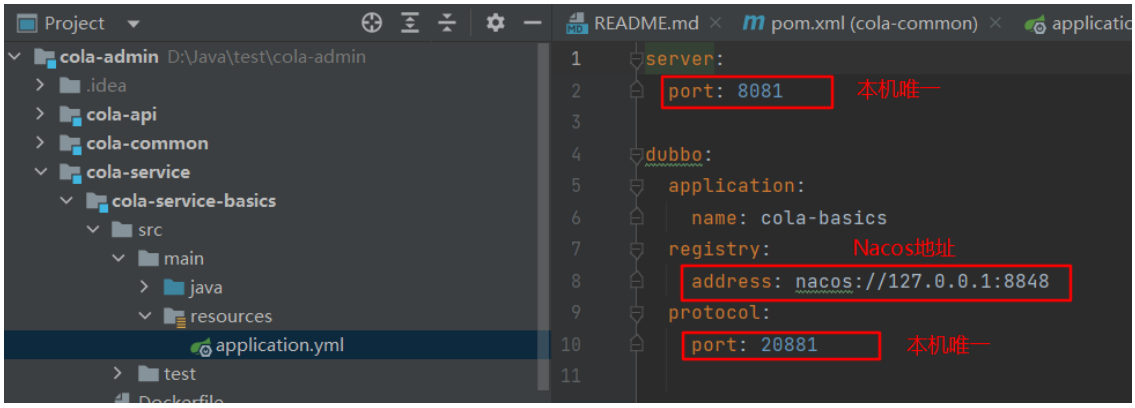
运行工程

1. 修改配置文件

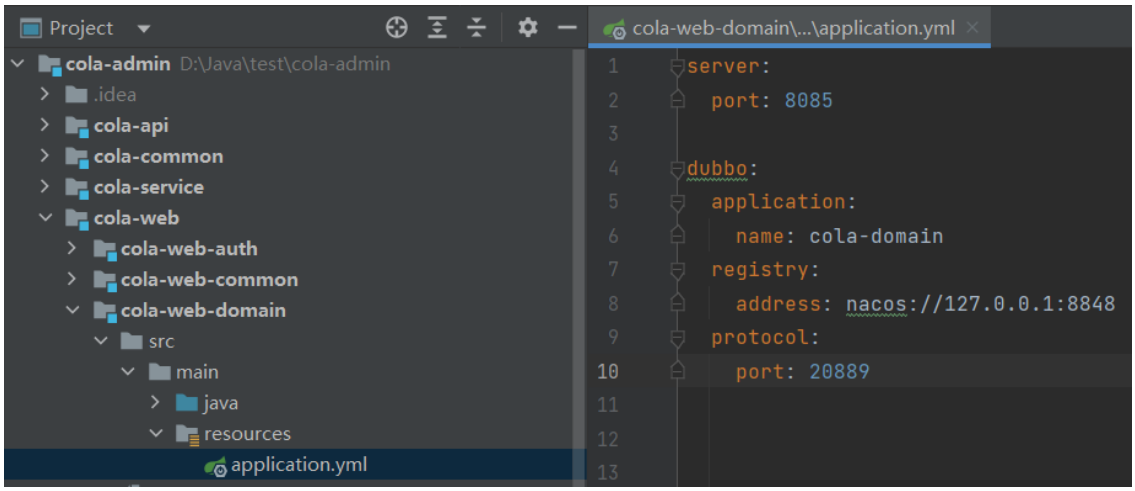
   修改数据库配置：cola-service-common/src/resources/application.yaml



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

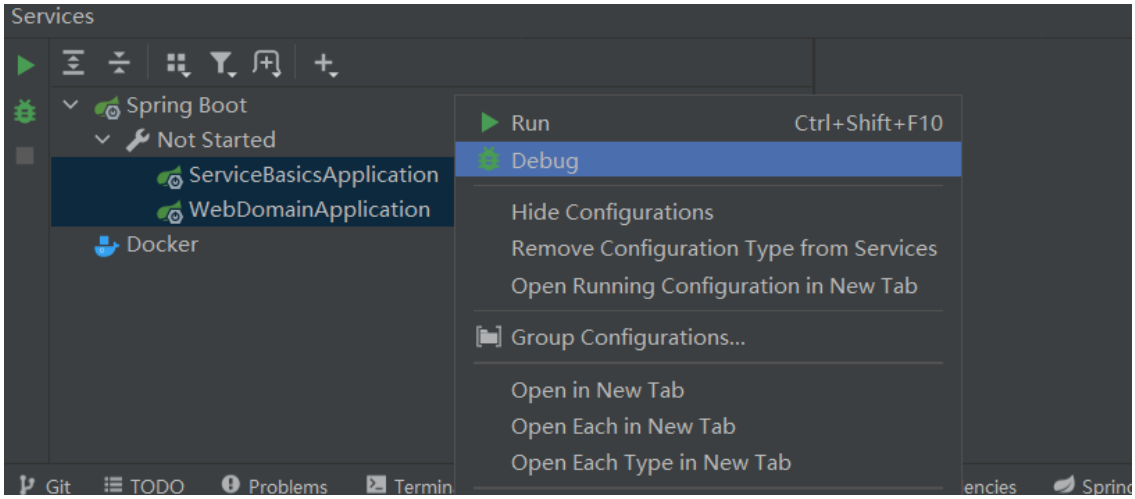


修改WebDomainApplication配置：cola-web-domain/src/resources/application.yml

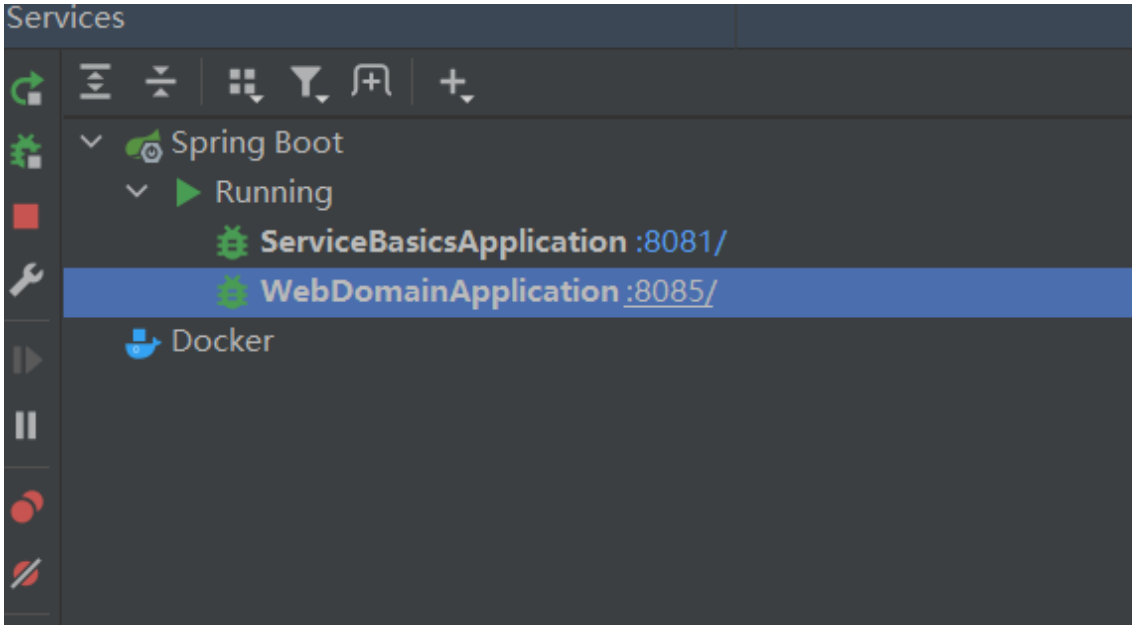


2. 启动项目

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

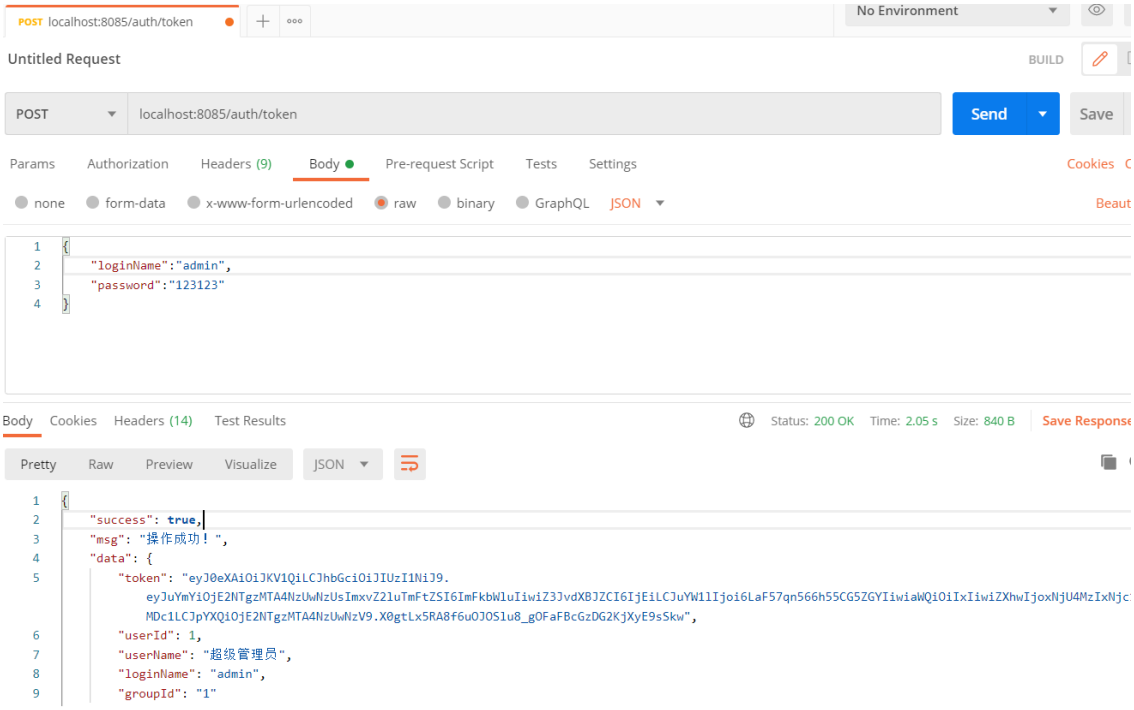
   两个服务已经成功注册上了

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

```
{
    "loginName":"admin",
    "password":"123123"
}
```

   点击发送



可以看到登陆成功并返回了token

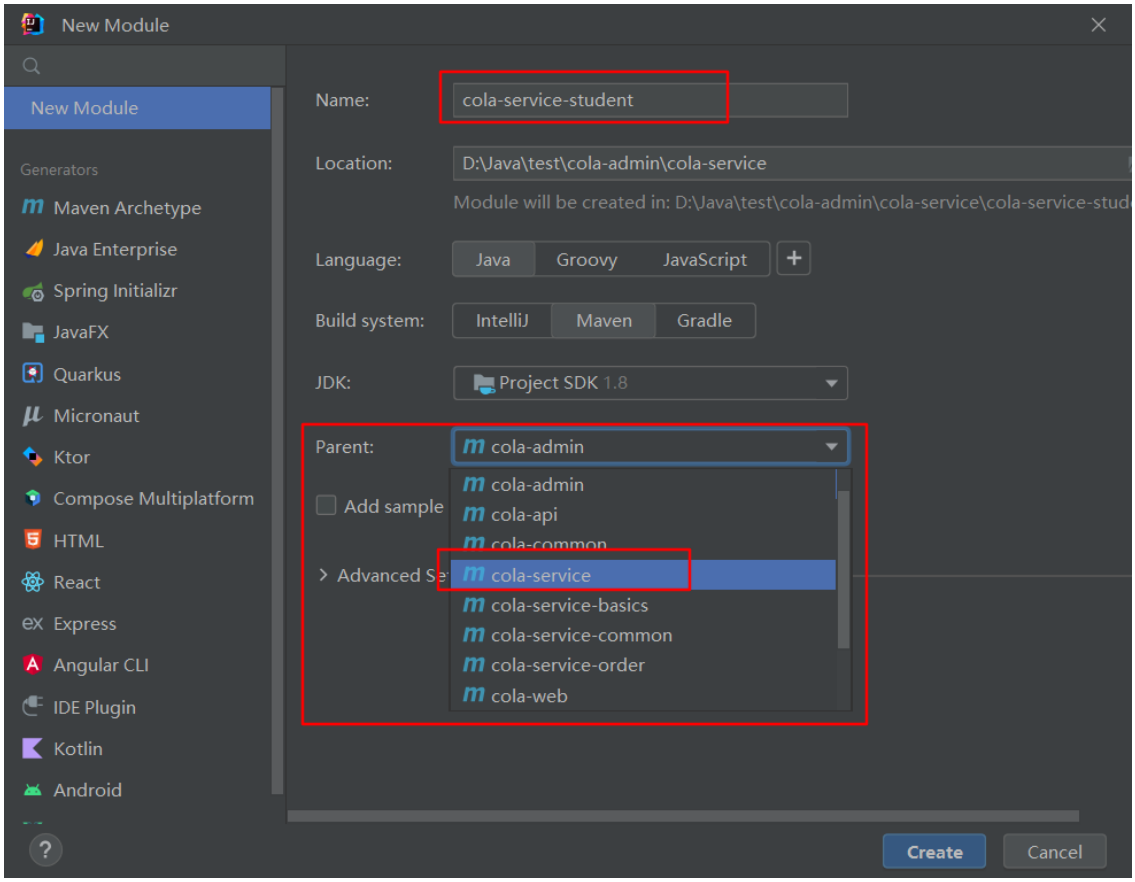
开发初探

新建微服务工程

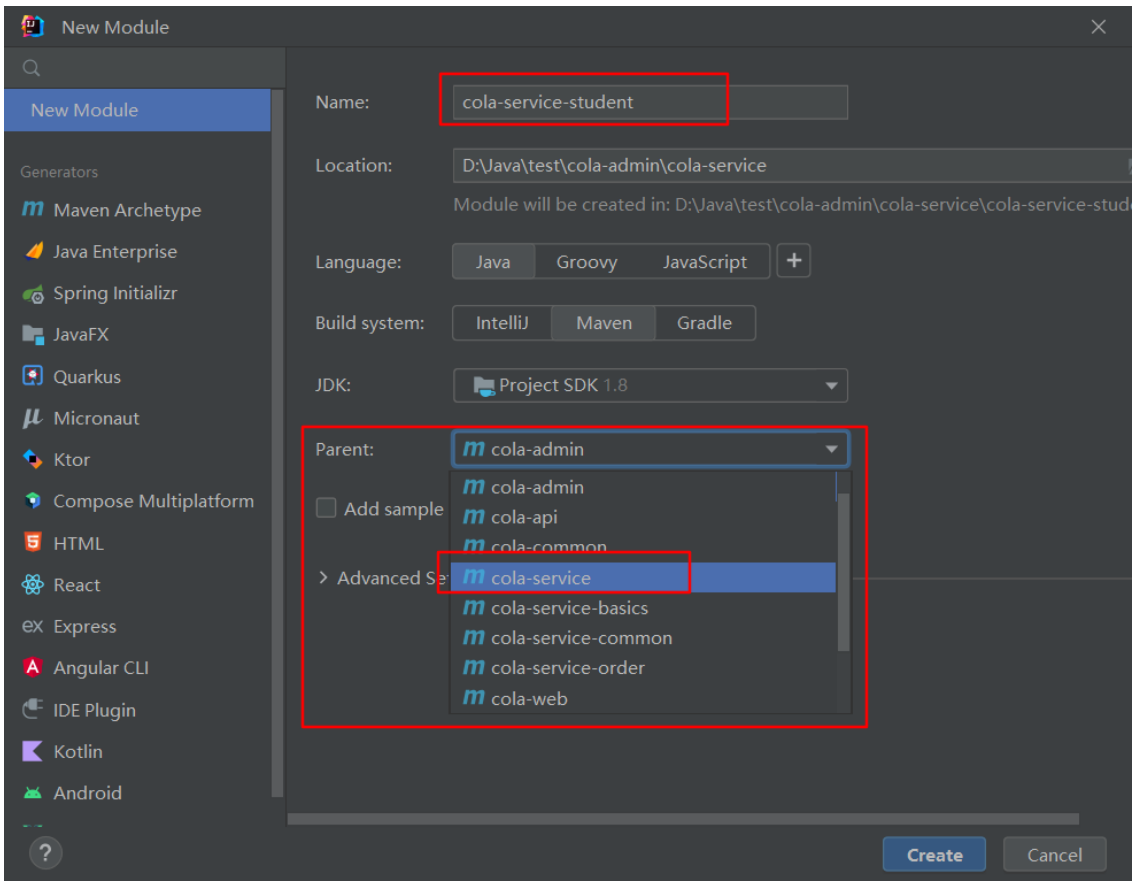
项目中已经有一个订单的空工程，可以根据需要改名后使用，下面介绍新建一个微服务工程的操作流程。

新建

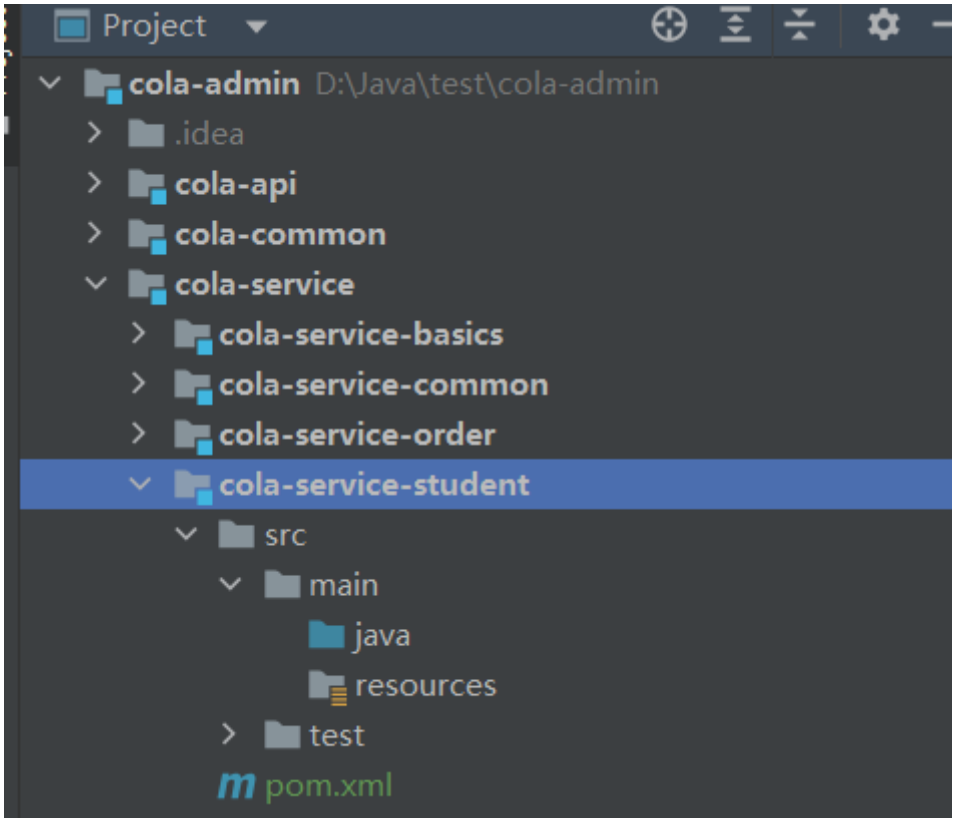
1. 在cola-service模块上右键，选择new-->module，这里以新建学生管理为例



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

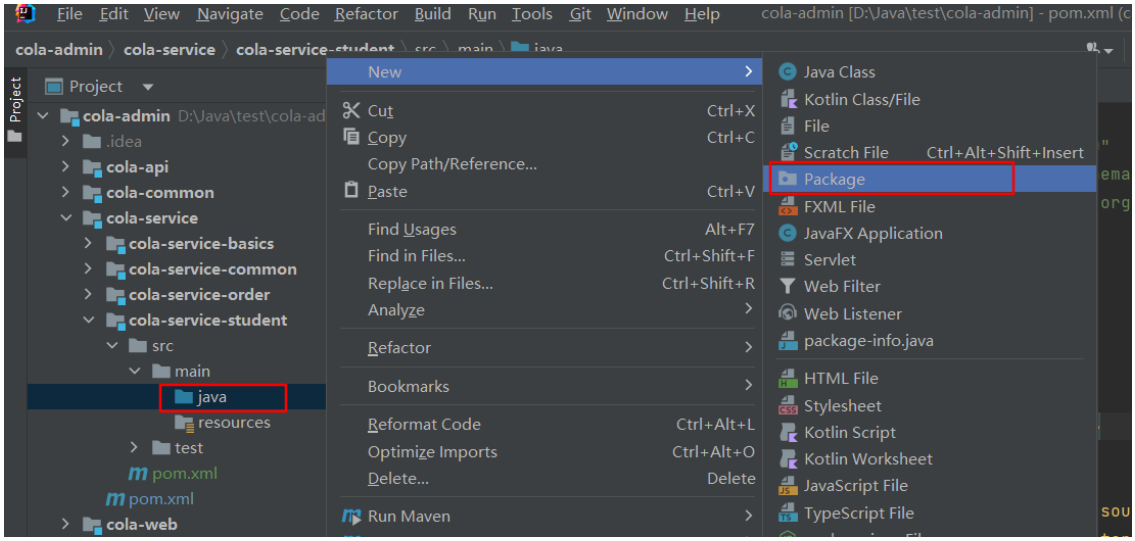


3. 创建成功

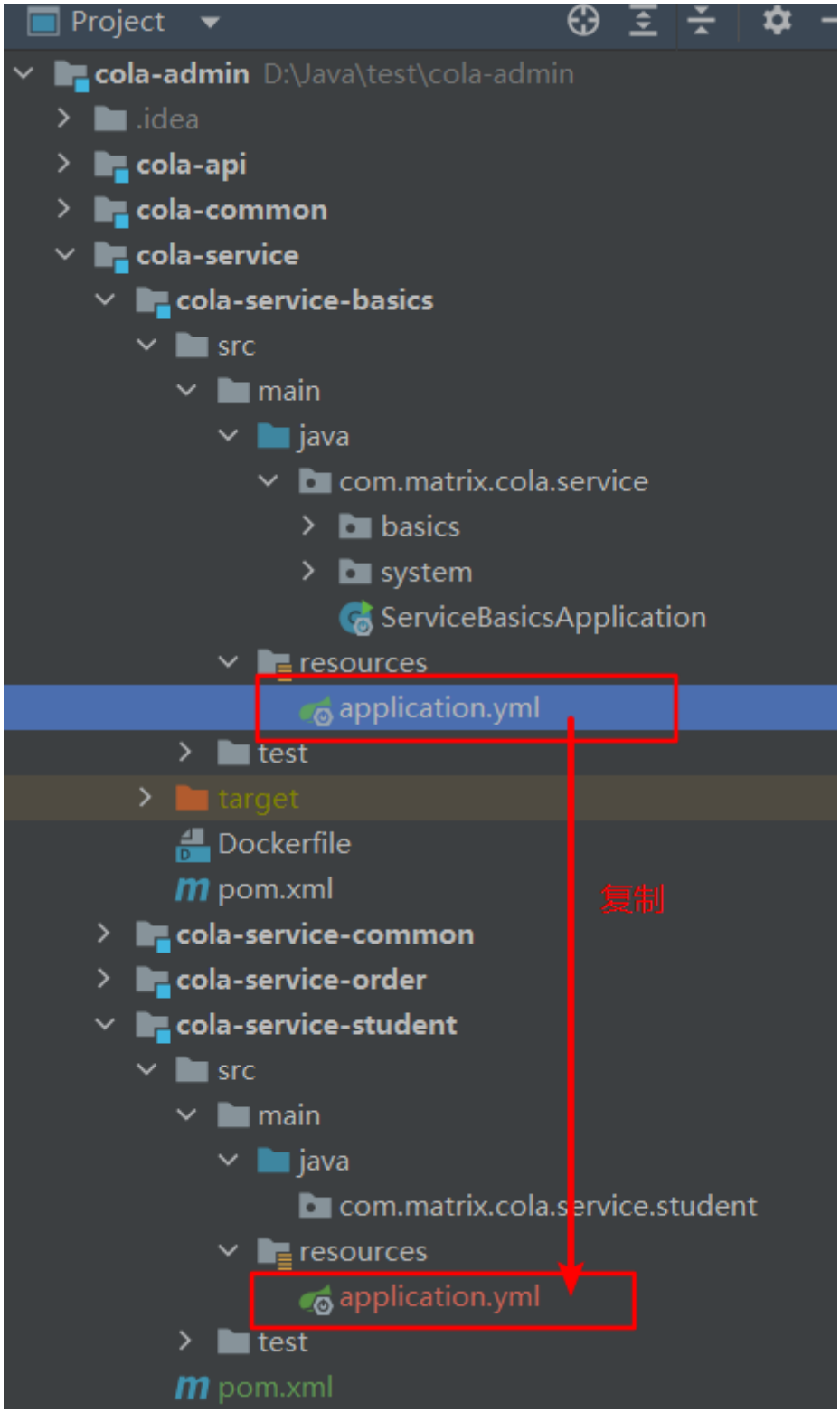


4. 建包、添加配置文件

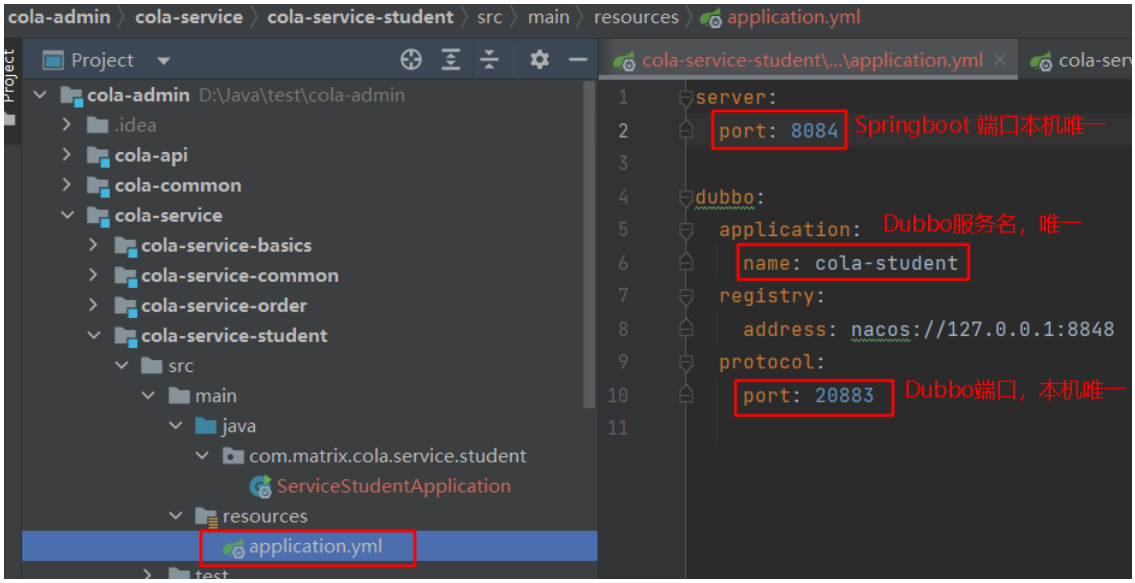
   新建包名com.matrix.cola.service.student



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

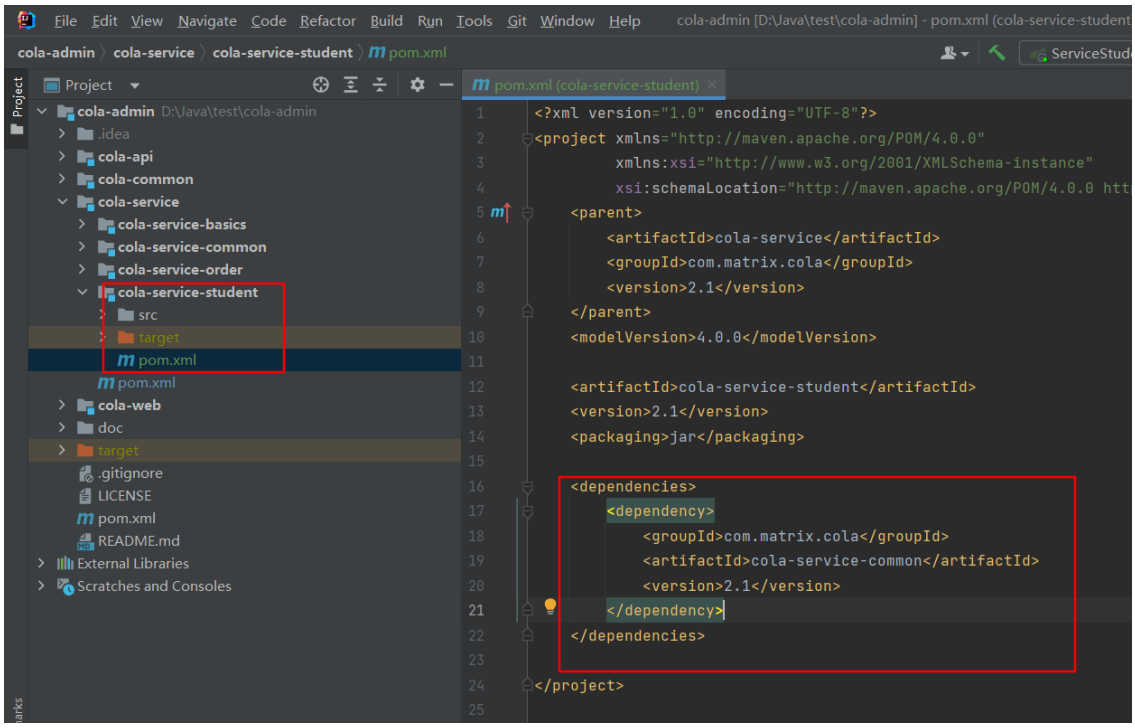

Project
cola-admin D:\Java\test\cola-admin
.idea
cola-api
cola-common
cola-service
cola-service-basics
src
main
java
com.matrix.cola.service
basics
system
ServiceBasicsApplication
resources
application.yml
test
target
Dockerfile
pom.xml
复制
cola-service-common
cola-service-order
cola-service-student
src
main
java
com.matrix.cola.service.student
resources
application.yml
test
pom.xml

修改配置文件



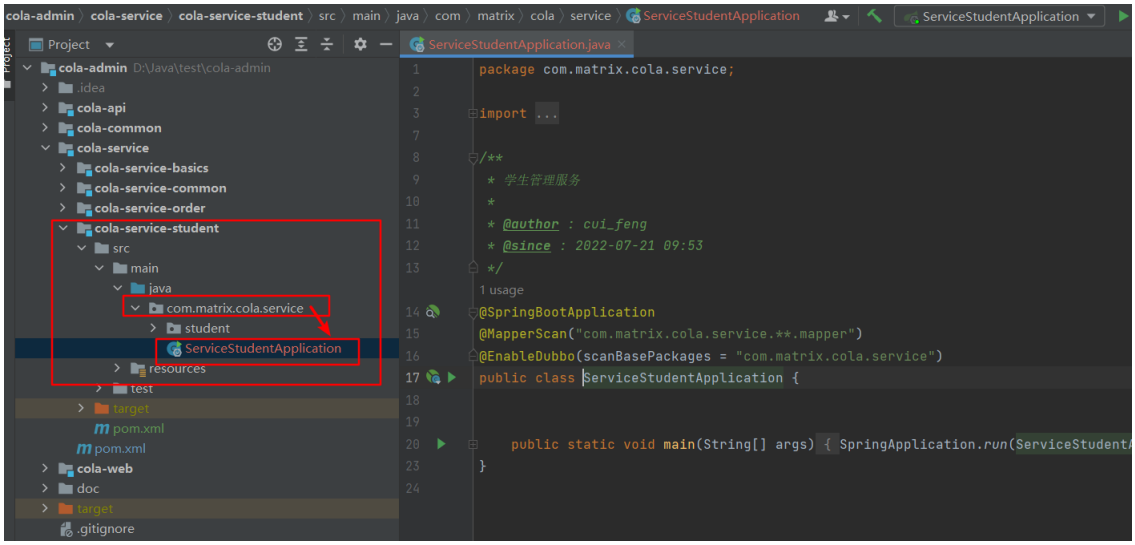
5. 引入依赖

   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



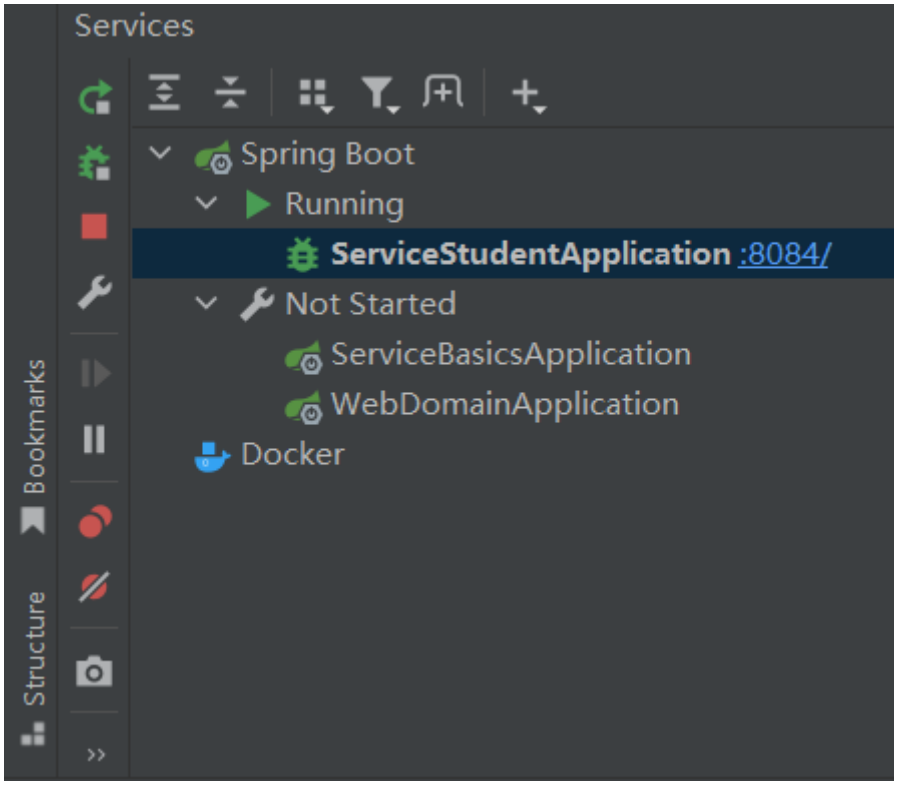
启动

1. 创建启动类



> 注意：cola-admin推荐将启动类放到com.matrix.cola.service包下，否则需要添加@ComponentScan("com.matrix.cola.service")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

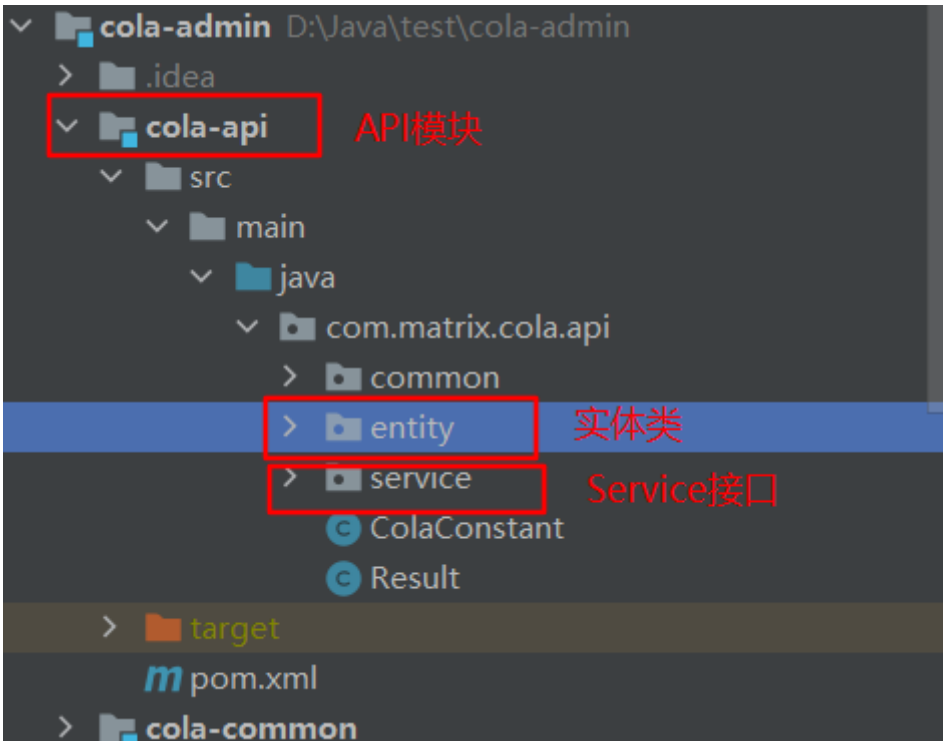
以学生管理为例，先创建一个学生表student。

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

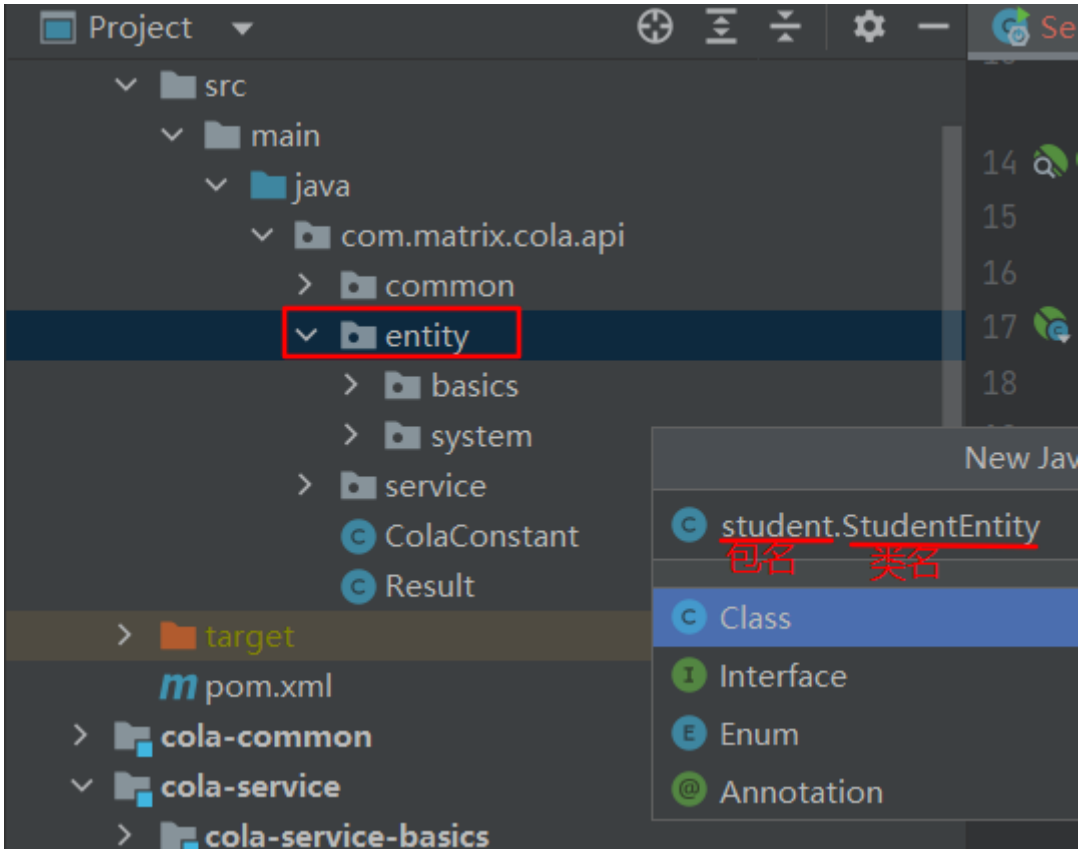
表中的“creator”、“create_time”、“reviser”、“revise_time”、“deleted”、“group_id”这六个字段是固定的业务字段，建议每个业务表都加上。表的字段名单词之间用下划线连接，这样就可以在实体类中直接使用驼峰方式命名，如字段“create_time”在实体类中的属性名就是“createTime”。

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。

```java
/**
 * 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

```java
@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

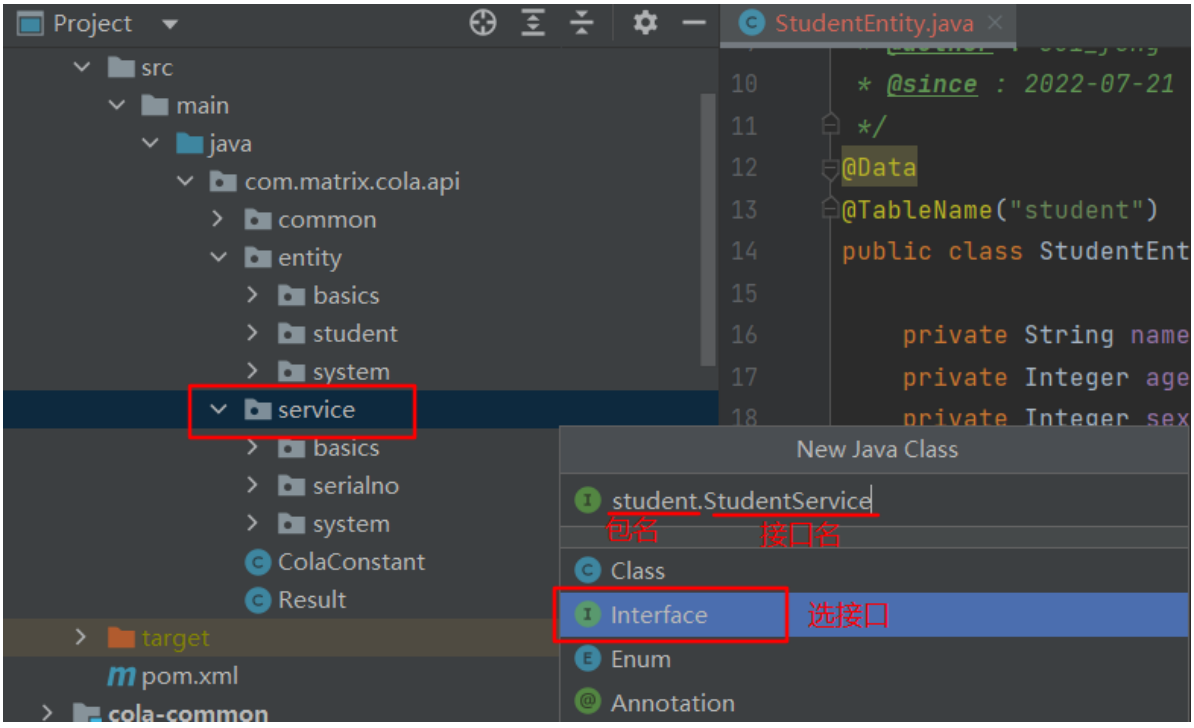
```java
@Data
@TableName("student")
public class StudentEntity extends BaseColaEntity {

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

Lombok注解

数据库表名

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。

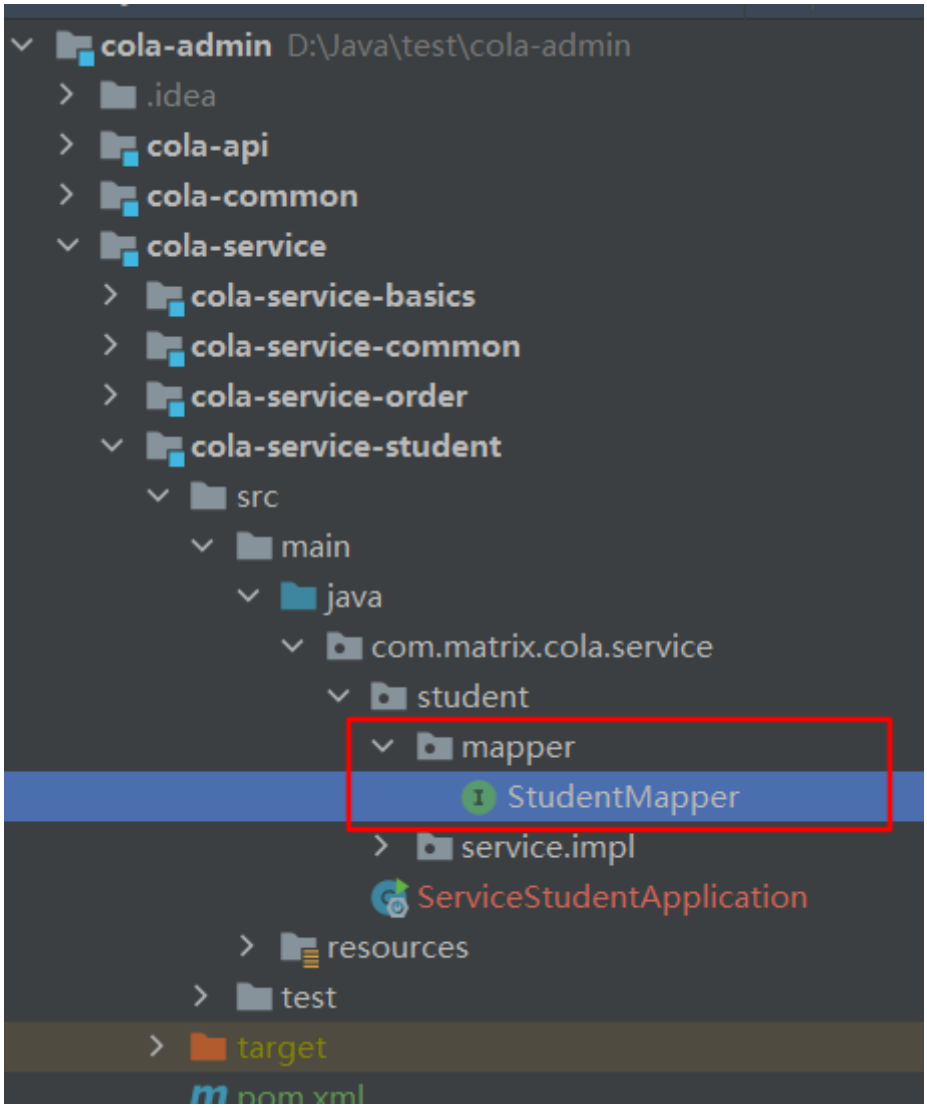
```java
/**
 * 学生管理接口
 * @author : cui_feng
 * @since : 2022-07-21 13:14
 */
public interface StudentService extends BaseColaEntityService<StudentEntity> {
}
```

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

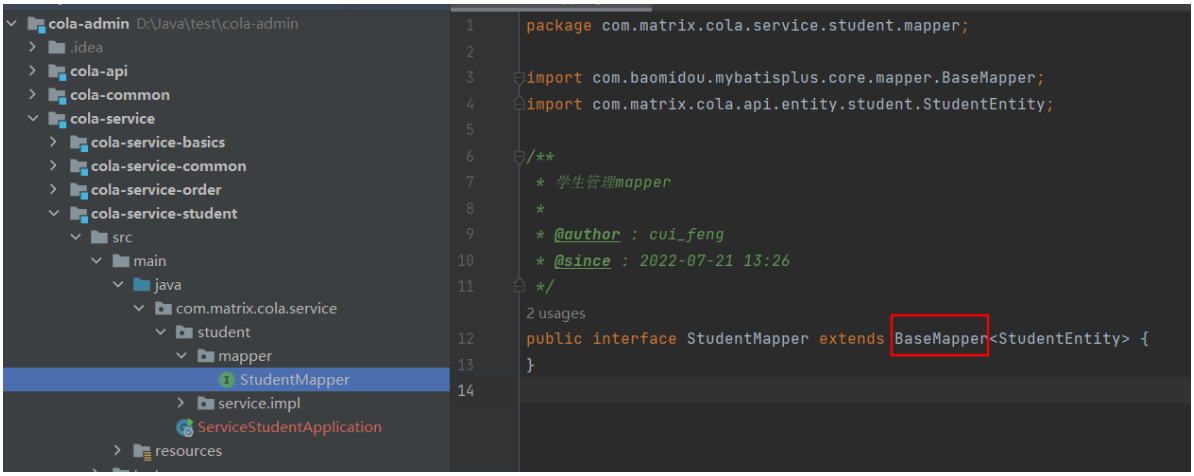
回到cola-service-student模块中，新建mapper接口StudentMapper



> mapper接口必须在mapper包下，并以xxxxMapper命名

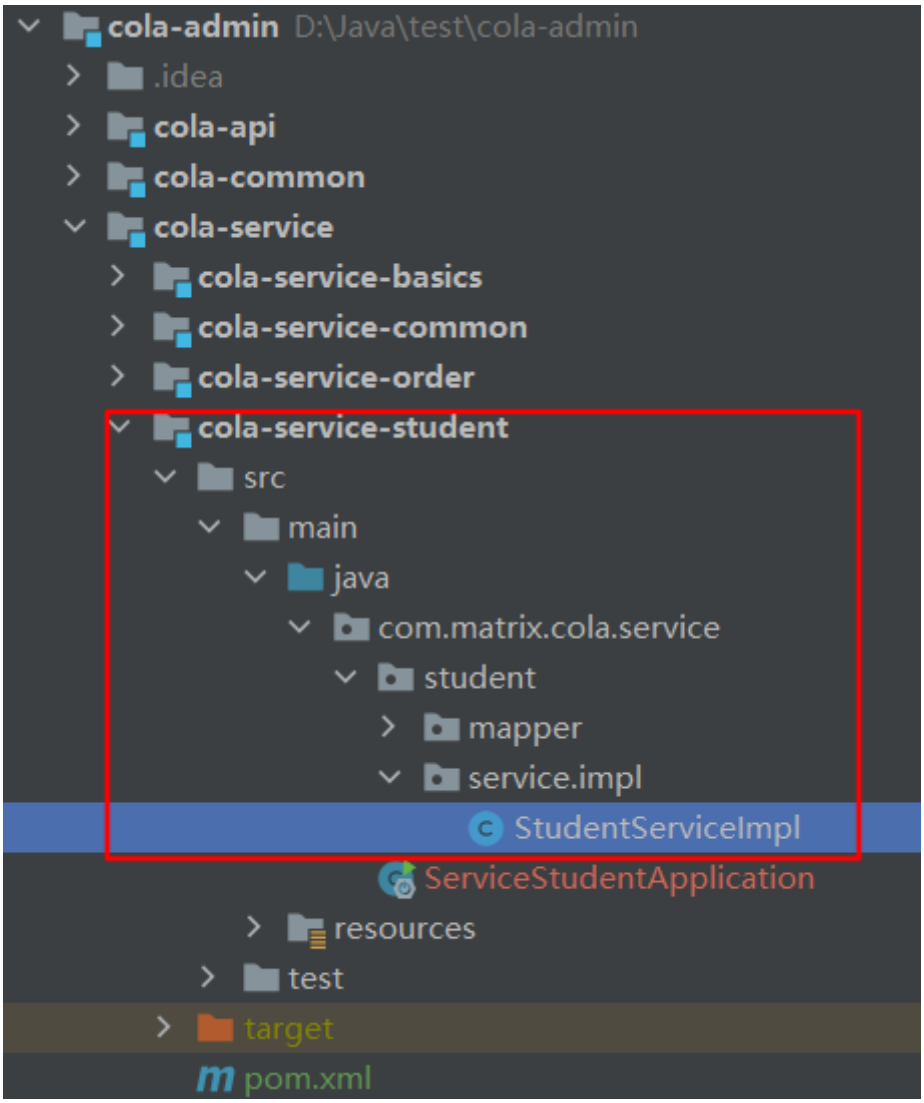
继承BaseMapper

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



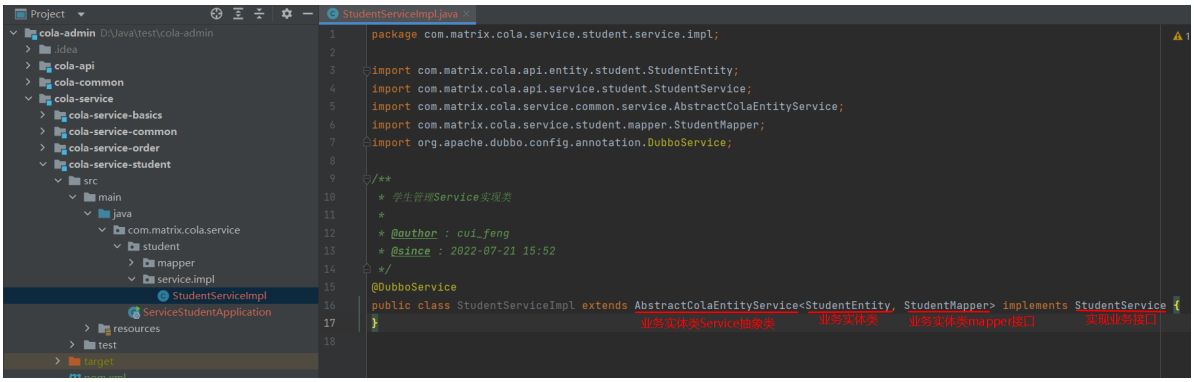
创建Service实现类

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

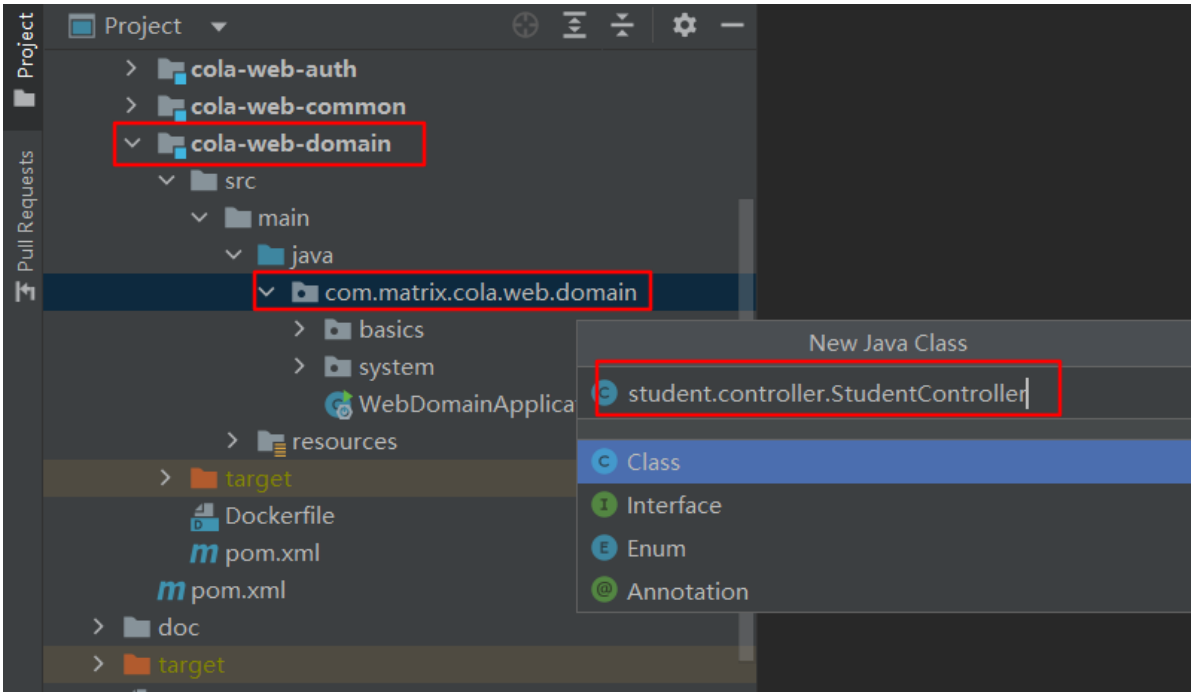
在StudentServiceImpl类上添加@DubboService注解

```
/**
 * 学生管理Service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 {
}
```

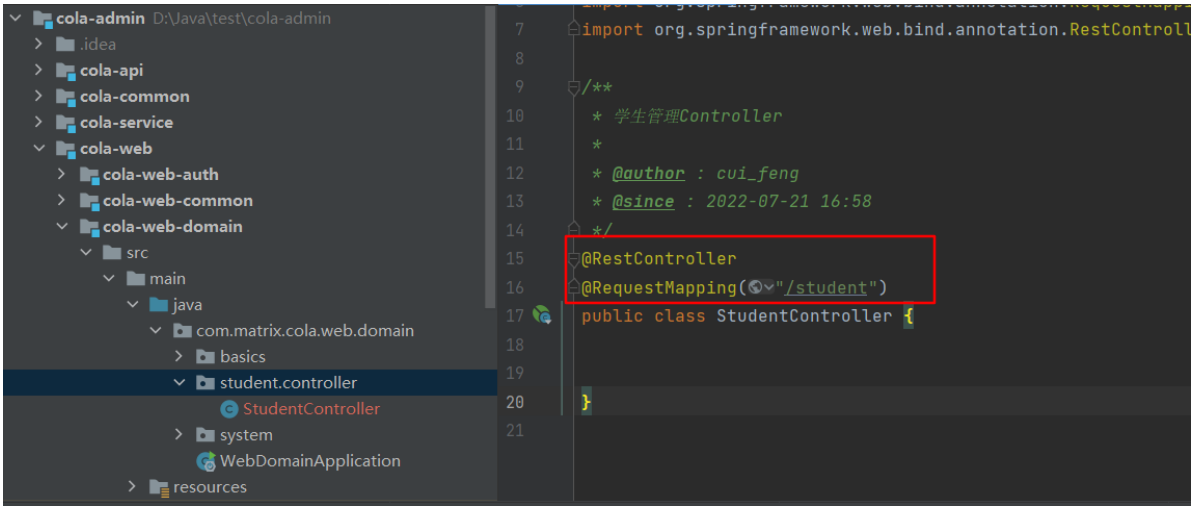
这样就完成了学生管理Service的全部功能（包括增、删、查、改、分页等功能），是不是很简单。

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

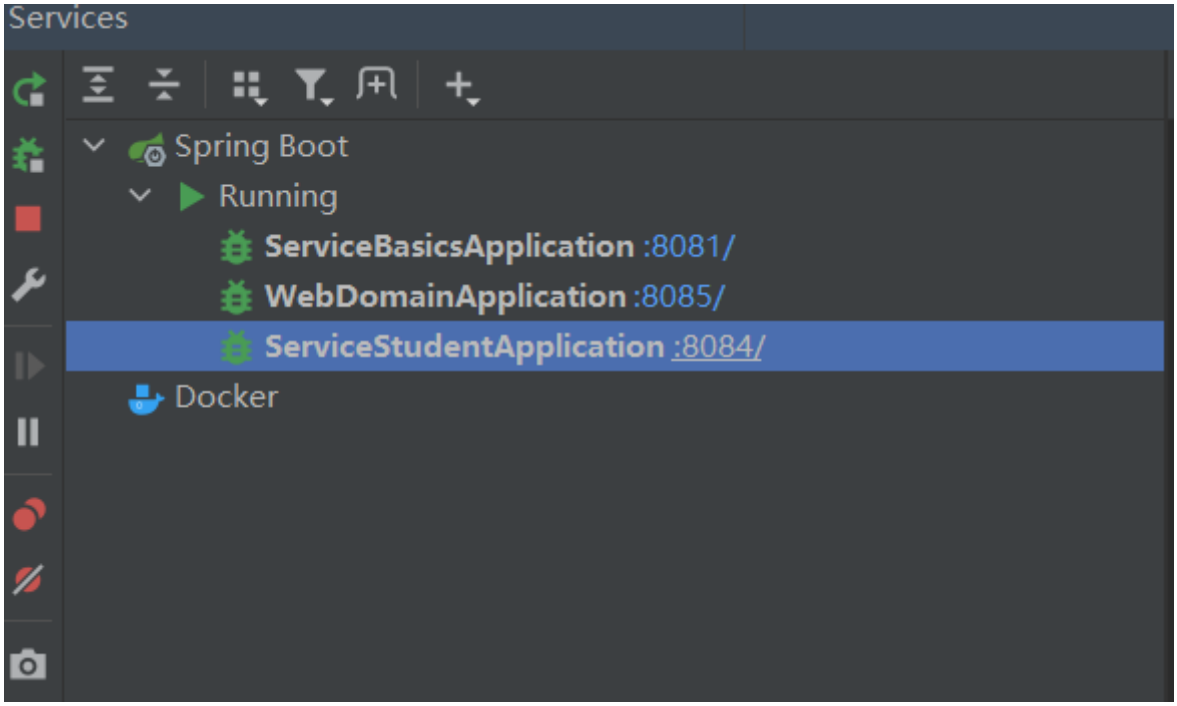
```java
package com.matrix.cola.web.domain.student.controller;

import com.matrix.cola.api.Result;
import com.matrix.cola.api.common.entity.Query;
import com.matrix.cola.api.entity.student.StudentEntity;
import com.matrix.cola.api.service.student.StudentService;
import org.apache.dubbo.config.annotation.DubboReference;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * 学生管理Controller
 *
 * @author : cui_feng
 * @since : 2022-07-21 16:58
 */
@RestController
@RequestMapping("/student")
public class StudentController {

    @DubboReference
    StudentService studentService;

    @PostMapping("/getStudentPage")
    public Result getStudentPage(@RequestBody Query<StudentEntity> query) {
        return Result.page(studentService.getPage(query));
    }

    @PostMapping("/addStudent")
    public Result addStudent(@RequestBody StudentEntity student) {
        return studentService.insert(student);
    }

    @PostMapping("/editStudent")
    public Result editStudent(@RequestBody StudentEntity student) {
        return studentService.modify(student);
    }

    @PostMapping("/deleteStudent")
    public Result deleteStudent(@RequestBody StudentEntity student) {
        return studentService.remove(student);
    }
}

```

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。

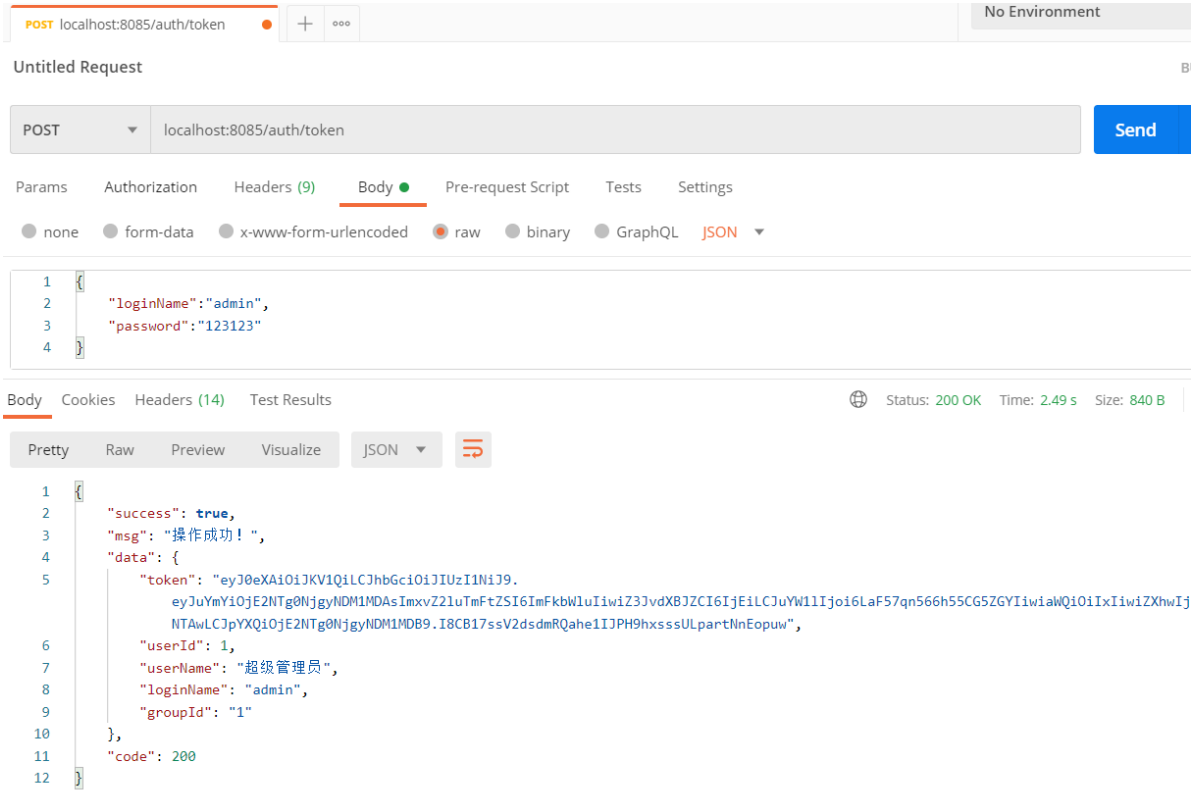
接口测试

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

```
{
    "loginName":"admin",
    "password":"123123"
}
```

点击send，看到如下内容则表示获取token成功



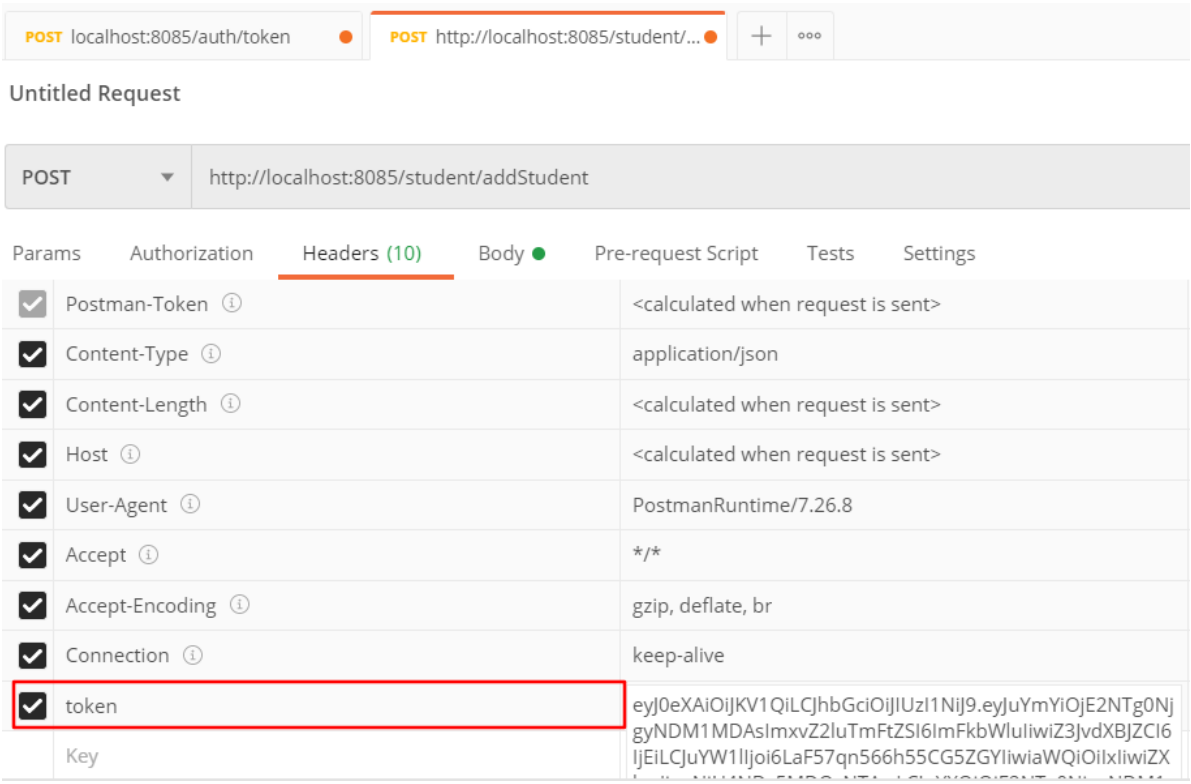
复制获取到的token值，后面的请求都需要用到。

新增学生

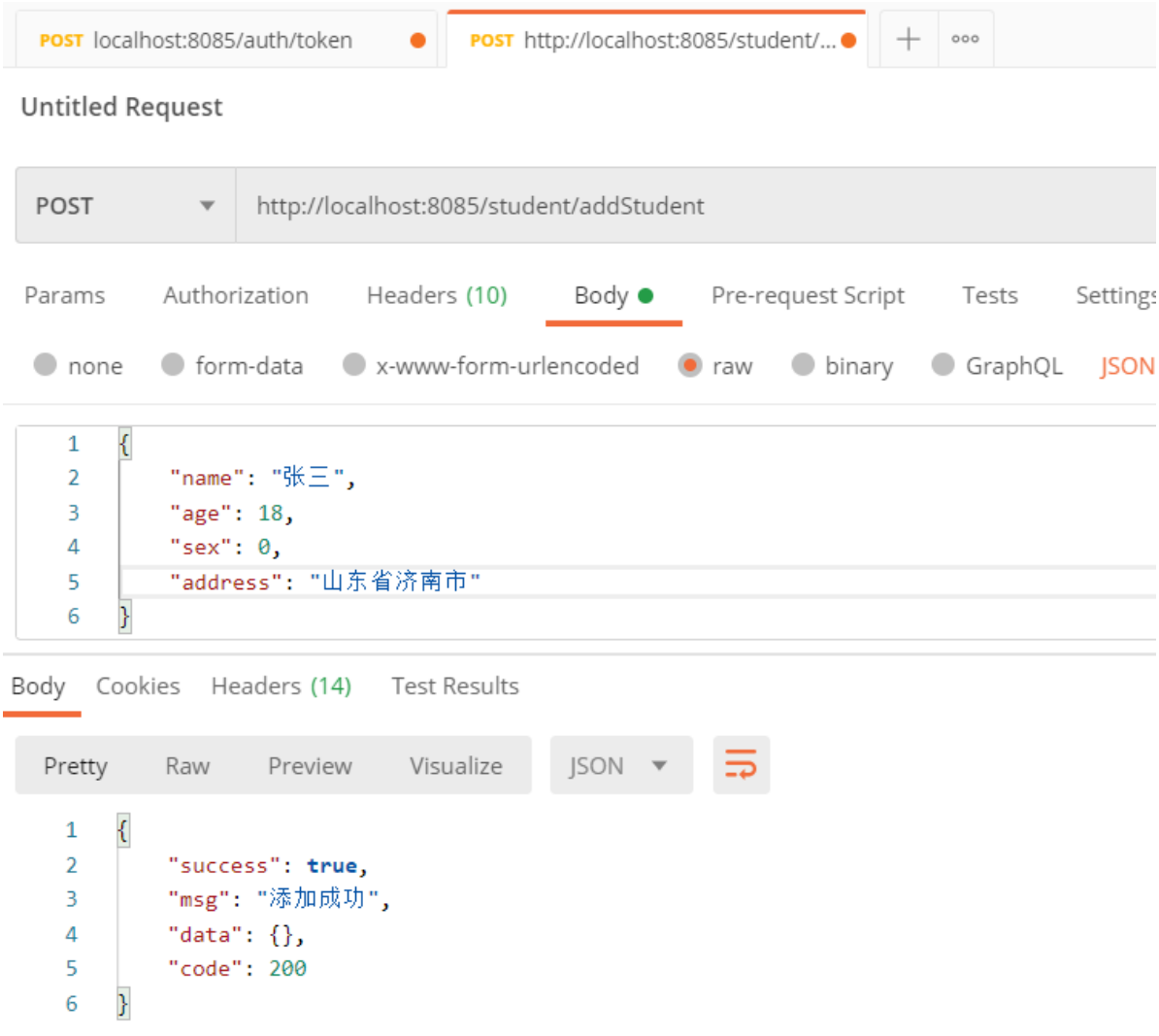
在Postman中新建一个标签，输入http://localhost:8085/student/addStudent ，选择Post方式提交，输入下面的参数：

```
{
    "name": "张三",
    "age": 18,
    "sex": 0,
    "address": "山东省济南市"
}
```

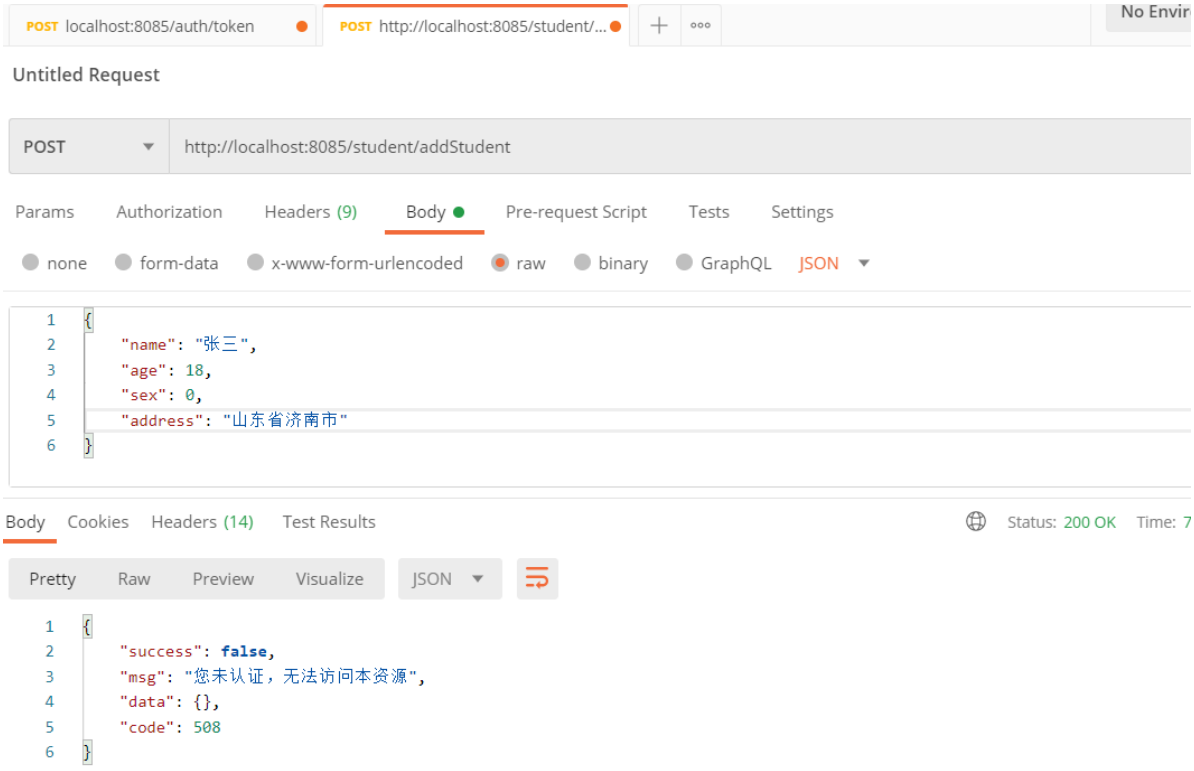
在Headers中添加参数token并粘贴上刚才复制的token值，如下图



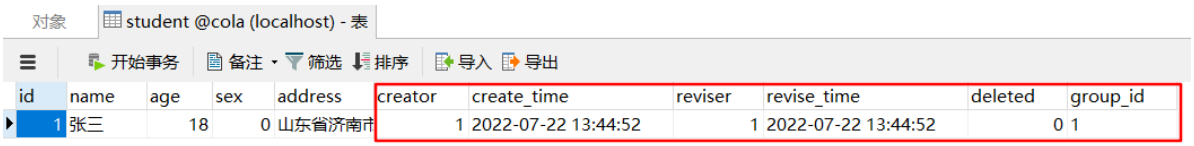
点击send提交，提示保存成功



如果没有在Header中添加token则会出现下面的错误



我们看一下数据库中的记录



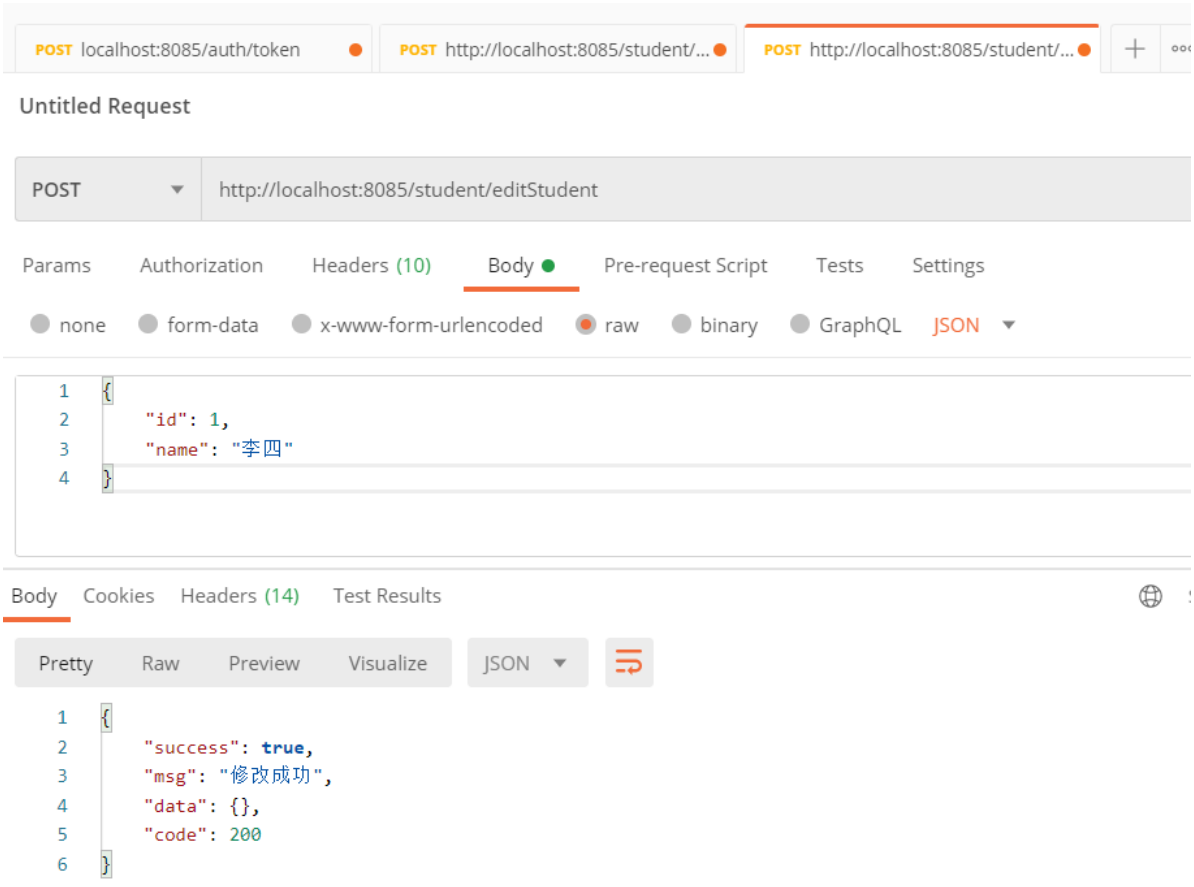
我们可以看到，最后的六个业务字段已经被填充了数据，这就是在AbstractColaEntityService中实现的。

修改学生

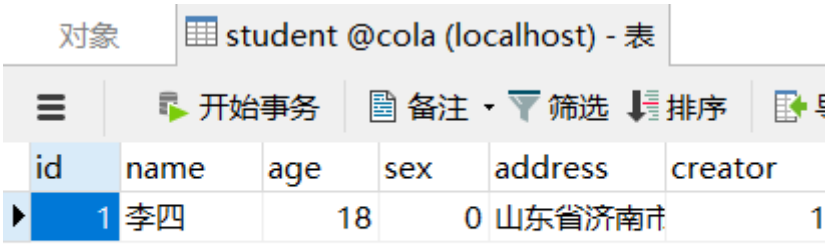
在Postman中再新建一个标签，输入http://localhost:8085/student/editStudent，选择Post方式提交，输入下面的参数：

```
{
    "id": 1,
    "name": "李四"
}
```

在Header中添加token，点击send



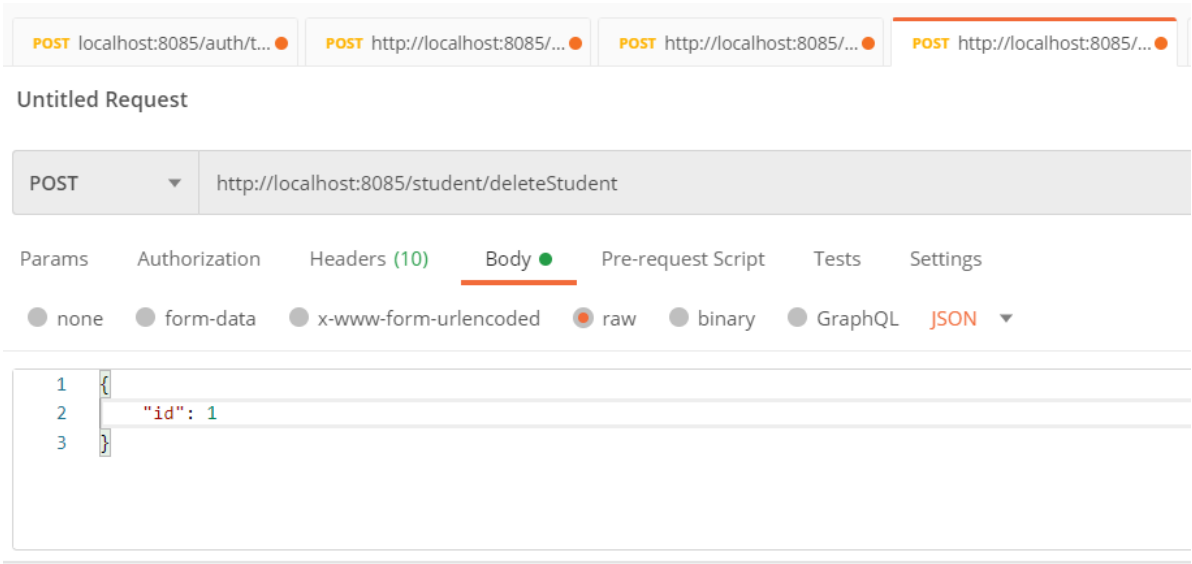
数据修改成功

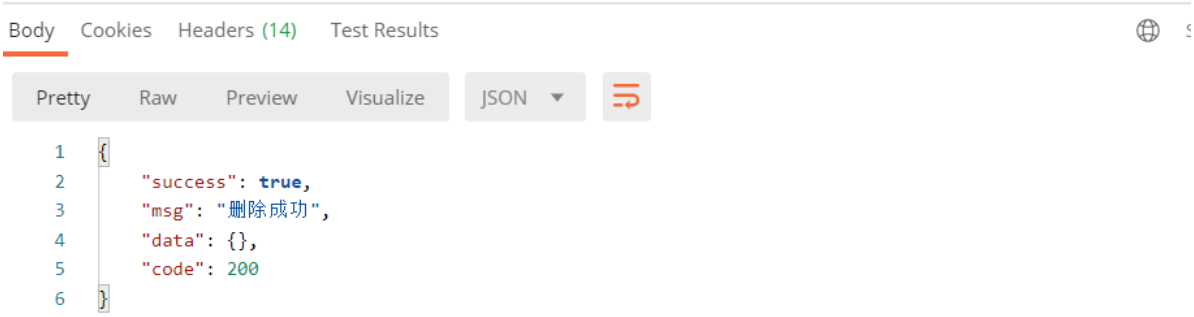


删除学生

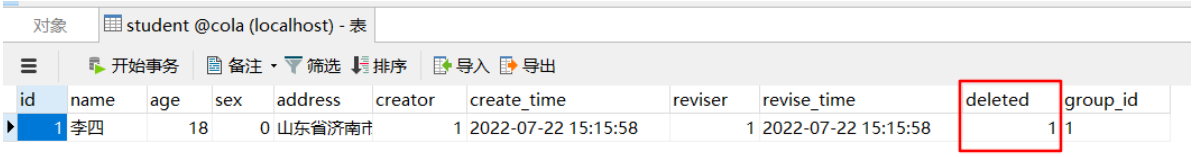
在Postman中新建一个标签页面，输入http://localhost:8085/student/deleteStudent，选择Post方式提交，输入下面的参数：

```
1
```





查看数据库可以看到deleted已为1，逻辑删除成功。



分页查询

先填充几条数据

```
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');
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');
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');
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：

```
{
    "pageSize": 2
}
```

这样的查询就是每页2条数据

```
{
    "success": true,
    "msg": "操作成功！",
    "data": {
        "page": {
            "records": [
                {
                    "id": 2,
                    "creator": 1,
                    "createTime": "2022-07-22 15:38:15",
                    "reviser": null,
                    "reviseTime": "2022-07-22 15:38:15",
                    "deleted": 0,
                    "groupId": "1",
                    "name": "张三",
                    "age": 16,
                    "sex": 1,
                    "address": null
                },
                {
                    "id": 3,
                    "creator": 1,
                    "createTime": "2022-07-22 15:38:15",
                    "reviser": null,
                    "reviseTime": "2022-07-22 15:38:15",
                    "deleted": 0,
                    "groupId": "1",
                    "name": "王五",
                    "age": 19,
                    "sex": 0,
                    "address": null
                }
            ],
            "total": 4,
            "size": 2,
            "current": 1,
            "orders": [],
            "optimizeCountSql": true,
            "searchCount": true,
            "countId": null,
            "maxLimit": null,
            "pages": 2
        }
    },
    "code": 200
}
```

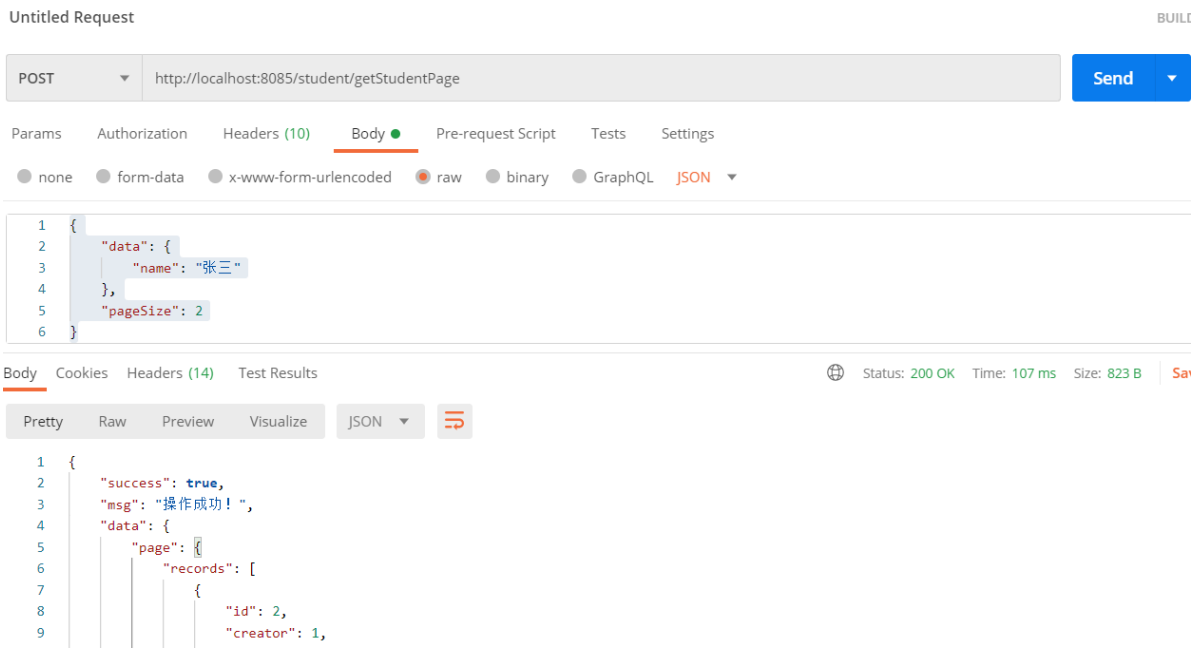

带条件的查询

参数部分输入以下内容，查询姓名为张三的学生

```
{
    "data": {
        "name": "张三"
    },
    "pageSize": 2
}
```

返回结果为

```
{
    "success": true,
    "msg": "操作成功！",
    "data": {
        "page": {
            "records": [
                {
                    "id": 2,
                    "creator": 1,
                    "createTime": "2022-07-22 15:38:15",
                    "reviser": null,
                    "reviseTime": "2022-07-22 15:38:15",
                    "deleted": 0,
                    "groupId": "1",
                    "name": "张三",
                    "age": 16,
                    "sex": 1,
                    "address": null
                }
            ],
            "total": 1,
            "size": 2,
            "current": 1,
            "orders": [],
            "optimizeCountSql": true,
            "searchCount": true,
            "countId": null,
            "maxLimit": null,
            "pages": 1
        }
    },
    "code": 200
}
```



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

```
{
    "conditions": [
        {
            "name": "name",
            "keyword": "like"
        },
        {
            "name": "age",
            "keyword": "between",
            "value1": 10,
            "value2": 20
        }
    ],
    "data": {
        "name": "张"
    },
    "pageSize": 2
}
```

上面的查询条件会转化为以下SQL：

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

查询结果如下：

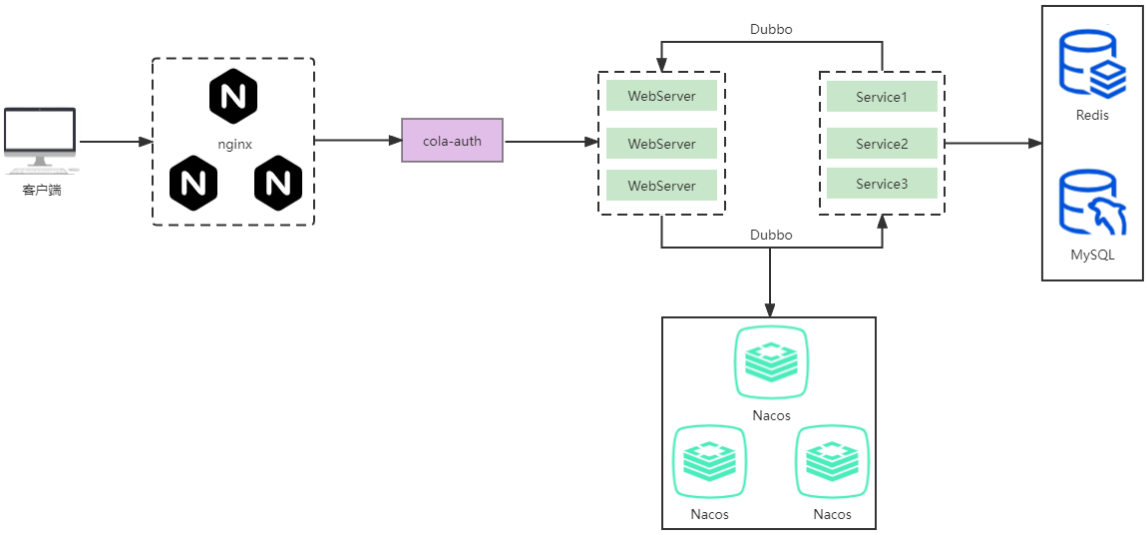
```
{
    "success": true,
    "msg": "操作成功！",
    "data": {
        "page": {
            "records": [
                {
                    "id": 2,
```

```
                    "creator": 1,
                    "createTime": "2022-07-22 15:38:15",
                    "reviser": null,
                    "reviseTime": "2022-07-22 15:38:15",
                    "deleted": 0,
                    "groupId": "1",
                    "name": "张三",
                    "age": 16,
                    "sex": 1,
                    "address": null
                },
                {
                    "id": 5,
                    "creator": 1,
                    "createTime": "2022-07-22 17:31:11",
                    "reviser": null,
                    "reviseTime": "2022-07-22 17:31:11",
                    "deleted": 0,
                    "groupId": "1",
                    "name": "张生",
                    "age": 19,
                    "sex": 0,
                    "address": null
                }
            ],
            "total": 2,
            "size": 2,
            "current": 1,
            "orders": [],
            "optimizeCountSql": true,
            "searchCount": true,
            "countId": null,
            "maxLimit": null,
            "pages": 1
        }
    },
    "code": 200
}
```


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

```
@PostMapping("/test")
public Result test() {
    QueryWrapper<StudentEntity> queryWrapper = new QueryWrapper<>();
    queryWrapper.like("name", "张");
    List<StudentEntity> list1 = studentService.getList(queryWrapper);
    System.out.println("测试Mybatis-Plus的QueryWrapper");
    list1.forEach(System.out::println);

    Query<StudentEntity> query = new Query<>();
    query.like("name","张");
    List<StudentEntity> list2 = studentService.getList(query);
    System.out.println("测试Query");
    list2.forEach(System.out::println);

    return Result.ok();
}
```

测试一下看输出结果，报错了

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

```java
@PostMapping("/test")
public Result test() {
    // QueryWrapper<StudentEntity> queryWrapper = new QueryWrapper<>();
    // queryWrapper.like("name", "张");
    // List<StudentEntity> list1 = studentService.getList(queryWrapper);
    // System.out.println("测试Mybatis-Plus的QueryWrapper");
    // list1.forEach(System.out::println);

    Query<StudentEntity> query = new Query<>();
    query.like("name","张");
    List<StudentEntity> list2 = studentService.getList(query);
    System.out.println("测试Query");
    list2.forEach(System.out::println);

    return Result.ok();
}
```

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
 .eyJuYmYiOjE2NTg3MjAxNTQ4MjAsImxvZ2luTmFtZSI6ImFkbWluIiwiZ3JvdXBJZCI6IjEiLCJuYW1lIjoi6LaF57qn566h55C
 U0ODIwLCJpYXQiOjE2NTg3MjAxNTQ4MjB9.NiEYBMu1yGm3H9DvMG1ogKAXz70NDiLCCNqugNIxz-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

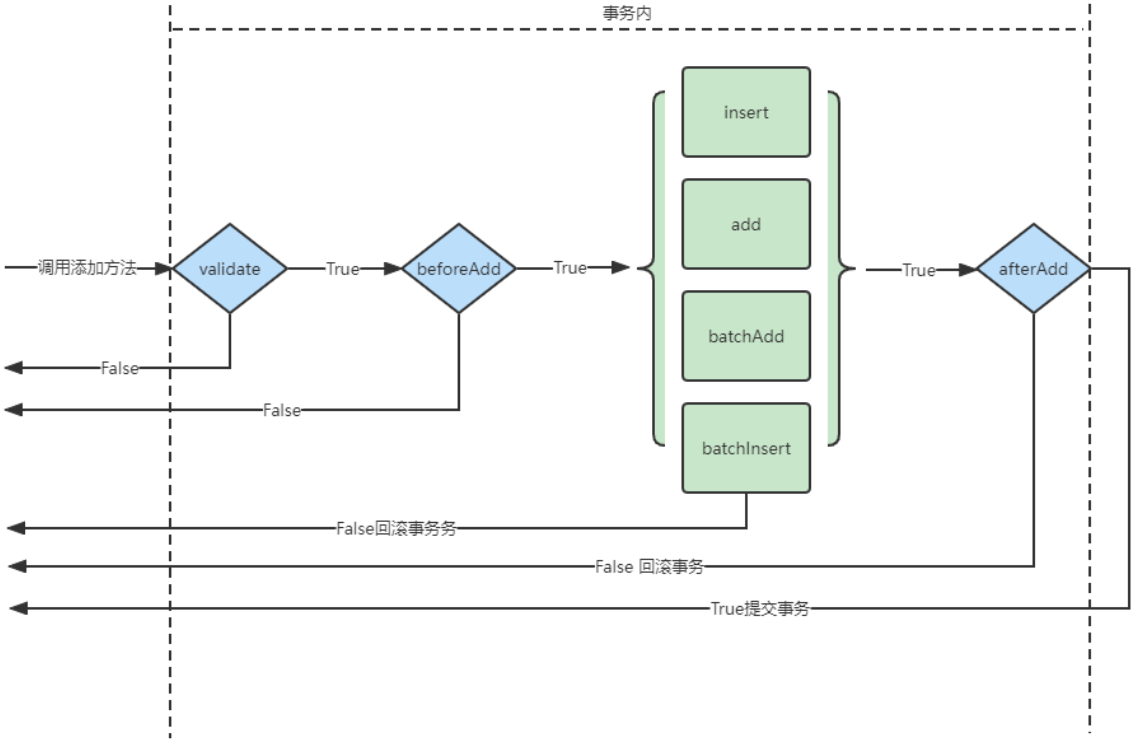
与QueryWrapper一样，Query类提供了大部分QueryWrapper一样的方法，如下表

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
notBetwwen	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

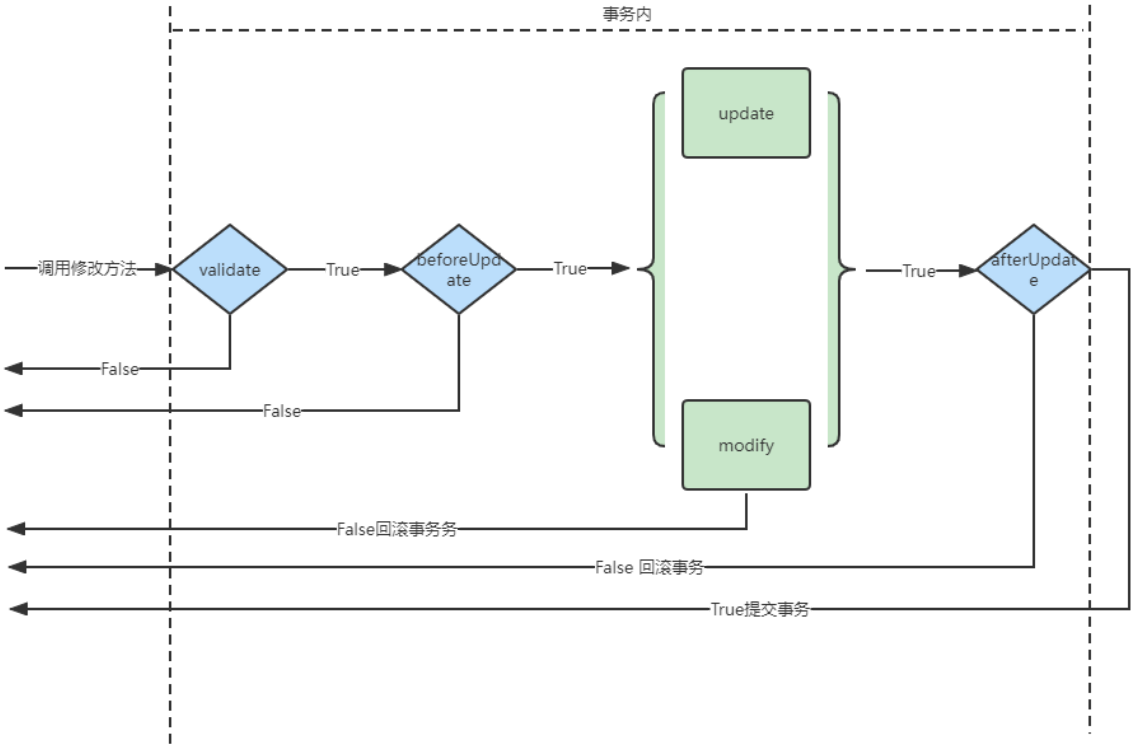
CRUD生命周期

cola-admin对于CRUD操作进行了封装，并可以根据需要进行扩展

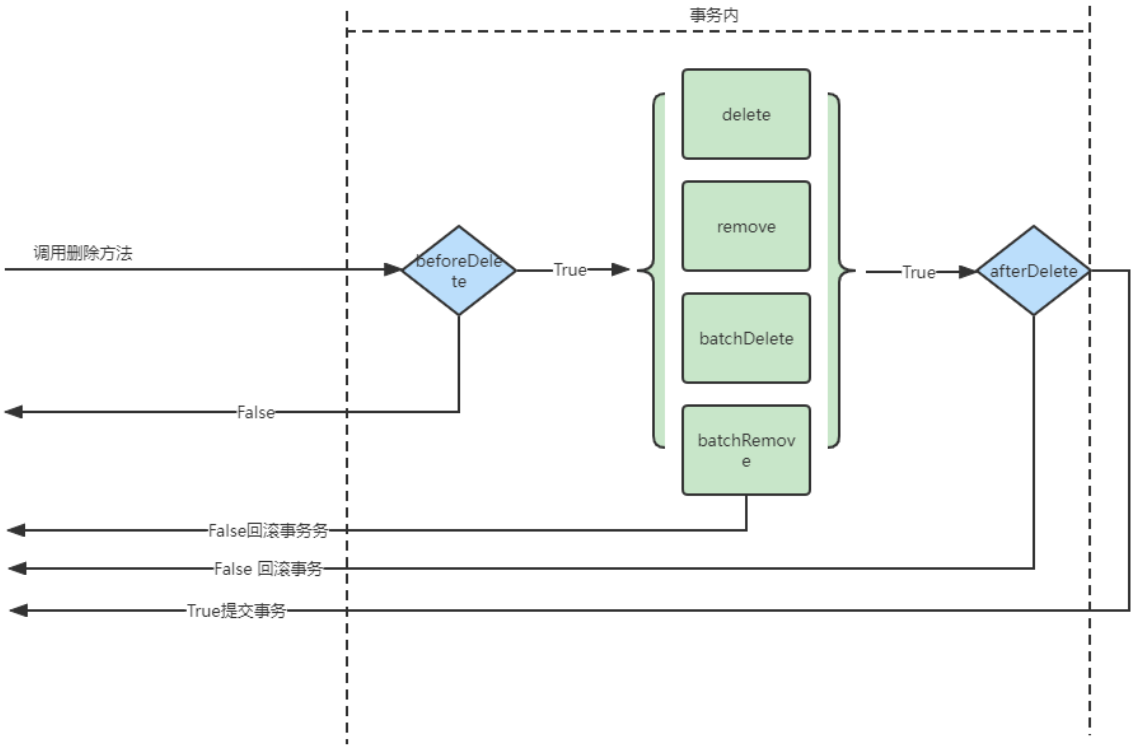
添加过程生命周期



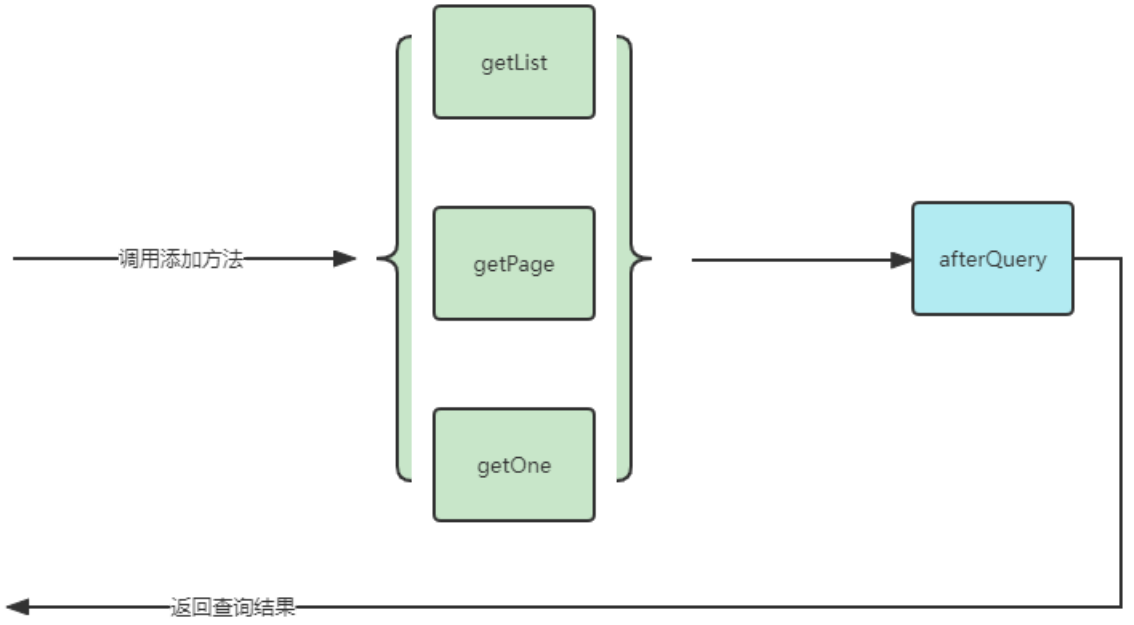
修改过程生命周期



删除过程生命周期



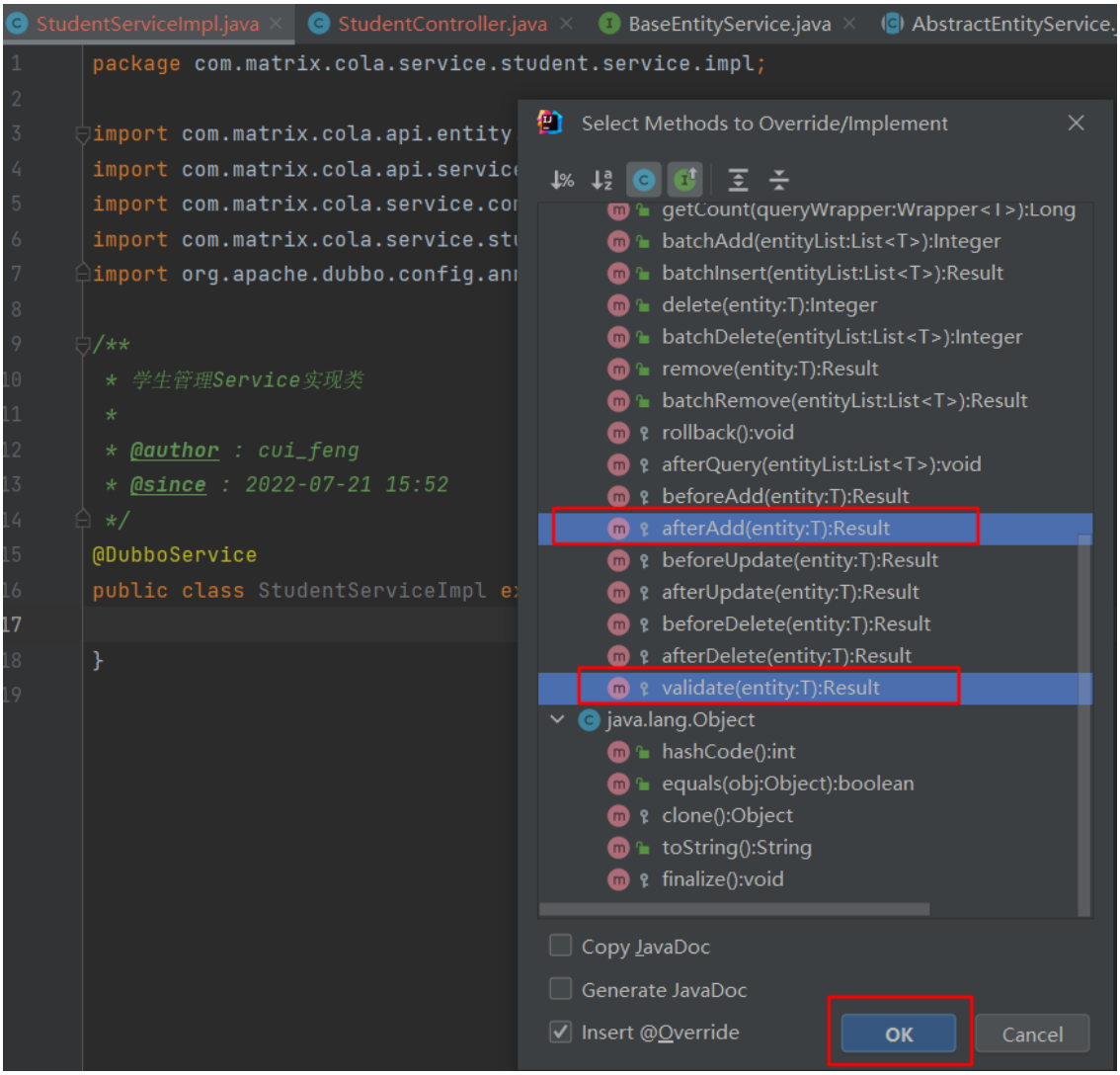
查询过程生命周期



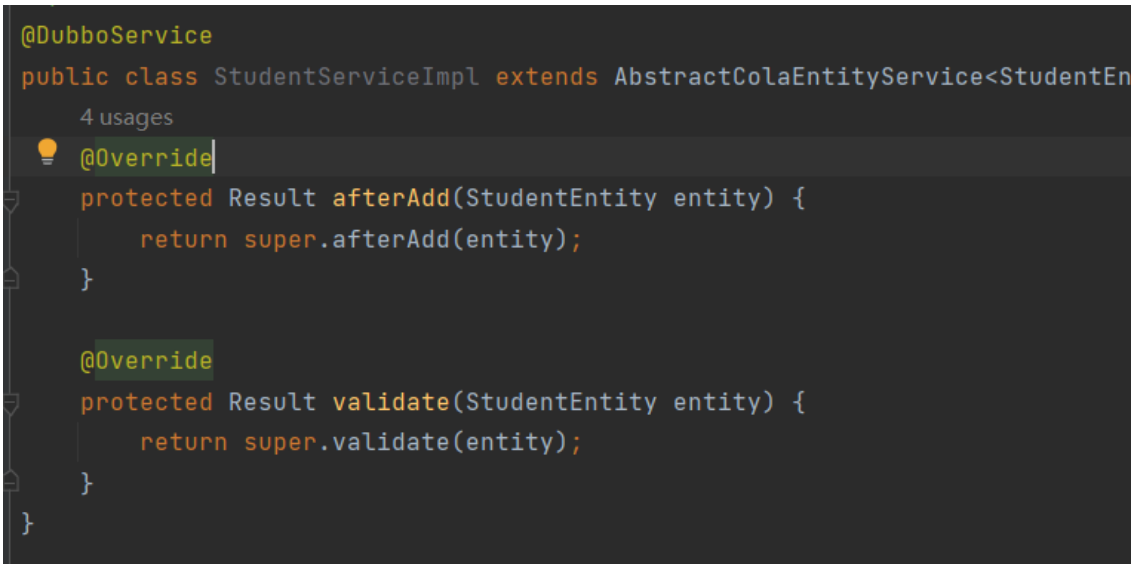
生命周期怎么用

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



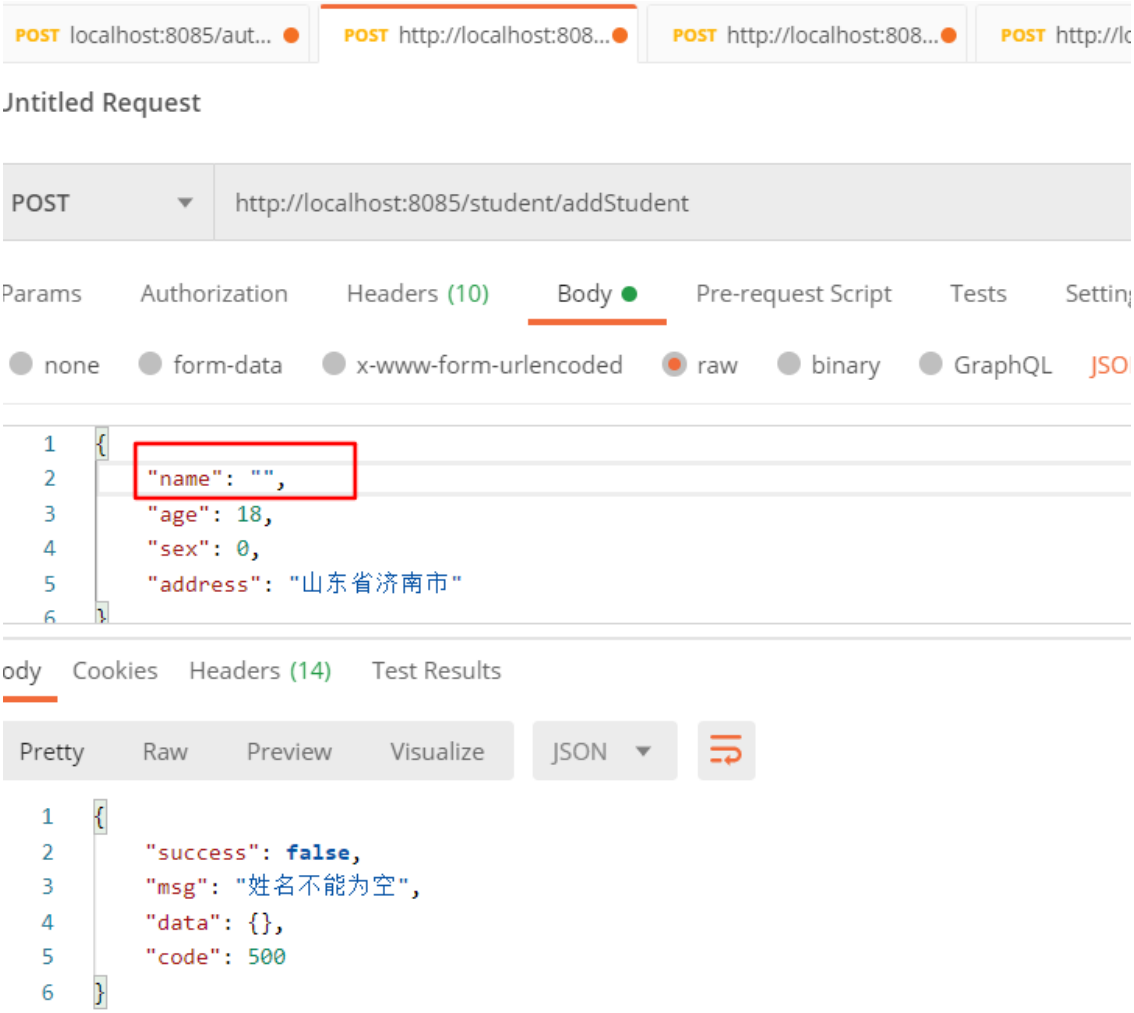
3. 实现validate

```java
@Override
protected Result validate(StudentEntity entity) {
    if (ObjectUtil.isNull(entity)) {
        return Result.err("学生信息不能为空");
    }
    if (StrUtil.isEmpty(entity.getName())) {
        return Result.err("姓名不能为空");
    }
    if (entity.getName().length()>4) {
        return Result.err("姓名不能超过四个字");
    }
    return super.validate(entity);
}
```

4. 实现afterAdd

```java
@Override
protected Result afterAdd(StudentEntity entity) {
    // 添加学生证
    Result result = studentIdCardService.addIdCard(entity);
    if (!result.isSuccess()) {
        // 回滚事务，学生信息和学生证信息同时回滚
        rollback();
        return result;
    }
    return super.afterAdd(entity);
}
```

5. 最终效果



> 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

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

创建选课Entity

```java
@Data
@TableName("student_course")
public class StudentCourseEntity extends BaseColaEntity {

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

创建选课Service

```java
public interface StudentCourseService extends
BaseColaEntityService<StudentCourseEntity> {
}
```

创建选课Mapper

```java
public interface StudentCourseMapper extends BaseMapper<StudentCourseEntity>
{
}
```

创建选课Service实现类

```java
@DubboService
public class StudentCourseServiceImpl extends
AbstractColaEntityService<StudentCourseEntity,
                                StudentCourseMapper> implements
StudentCourseService {

}
```

创建选课Controller，为方便测试，只添加查询方法

```java
@RestController
@RequestMapping("/studentCourse")
public class StudentCourseController {

    @DubboReference
    StudentCourseService studentCourseService;

    @PostMapping("getList")
    public Result getList() {
        return Result.list(studentCourseService.getList(new Query<>()));
    }
}
```

添加几条数据

```sql
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');
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');
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');
```

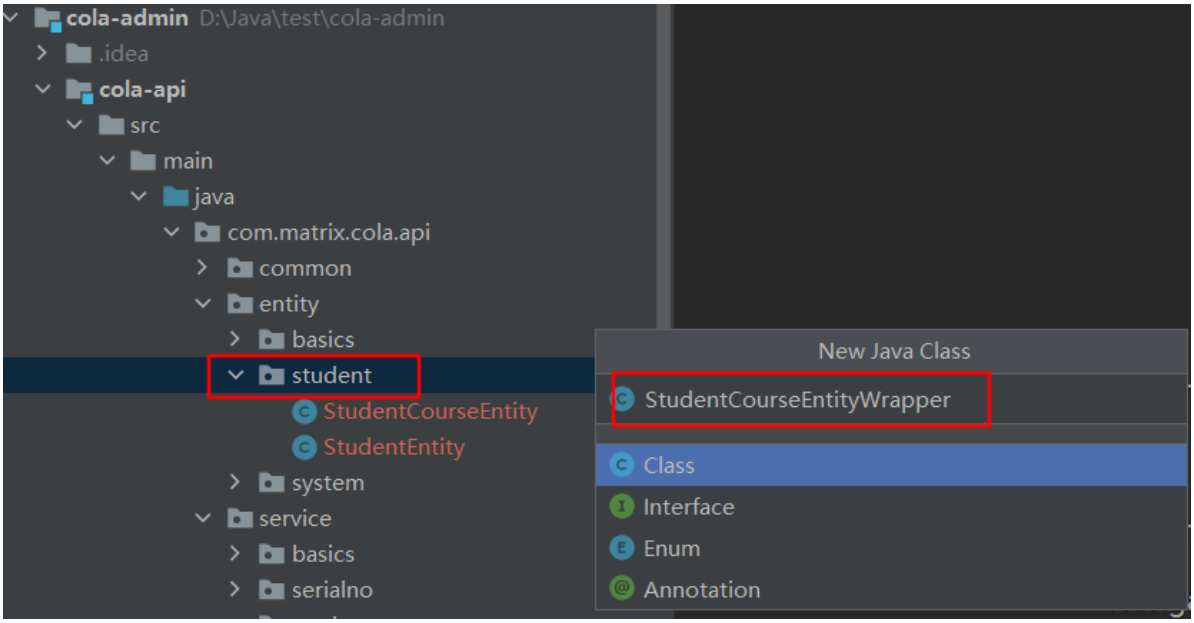
查看查询结果


```
{
    "success": true,
    "msg": "操作成功！",
    "data": {
        "list": [
            {
                "id": 1,
                "creator": 1,
                "createTime": "2022-07-25 15:30:18",
                "reviser": 1,
                "reviseTime": "2022-07-25 15:30:18",
                "deleted": 0,
                "groupId": "1",
                "studentId": 2,
                "courseName": "语文"
            },
            {
                "id": 2,
                "creator": 1,
                "createTime": "2022-07-25 15:30:18",
                "reviser": 1,
                "reviseTime": "2022-07-25 15:30:18",
                "deleted": 0,
                "groupId": "1",
                "studentId": 3,
                "courseName": "数学"
            },
            {
                "id": 3,
                "creator": 1,
                "createTime": "2022-07-25 15:30:26",
                "reviser": 1,
                "reviseTime": "2022-07-25 15:30:26",
                "deleted": 0,
                "groupId": "1",
                "studentId": 4,
                "courseName": "地理"
            }
        ]
    },
    "code": 200
}
```


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

```java
@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

```java
@Data
public class StudentCourseEntityWrapper {

    private Long studentId;
    private String courseName;

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

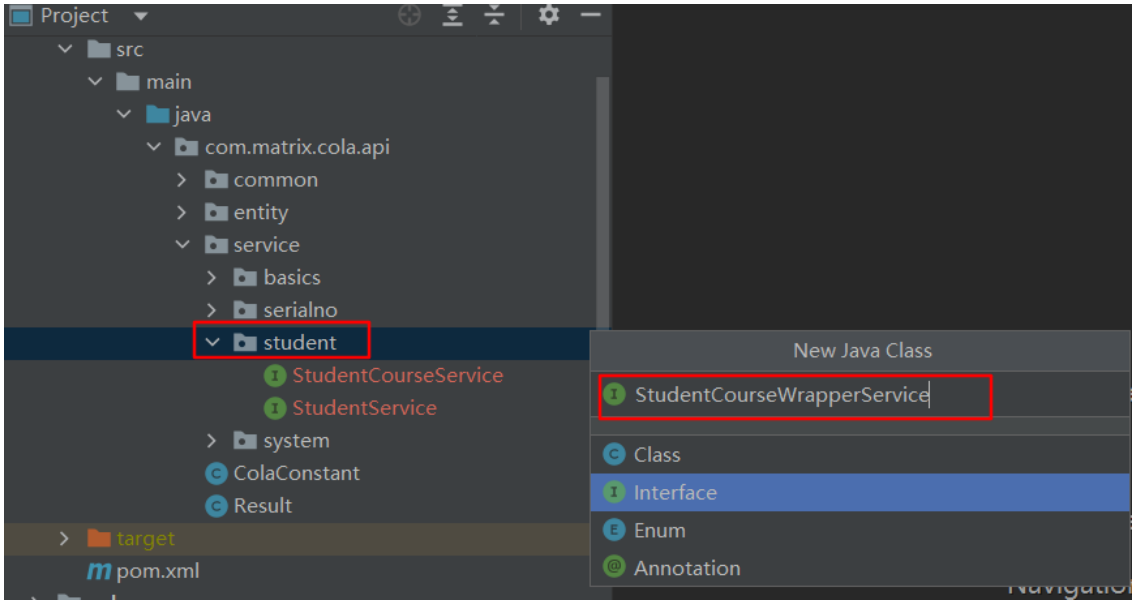
4. 继承抽象类

   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。

```java
@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

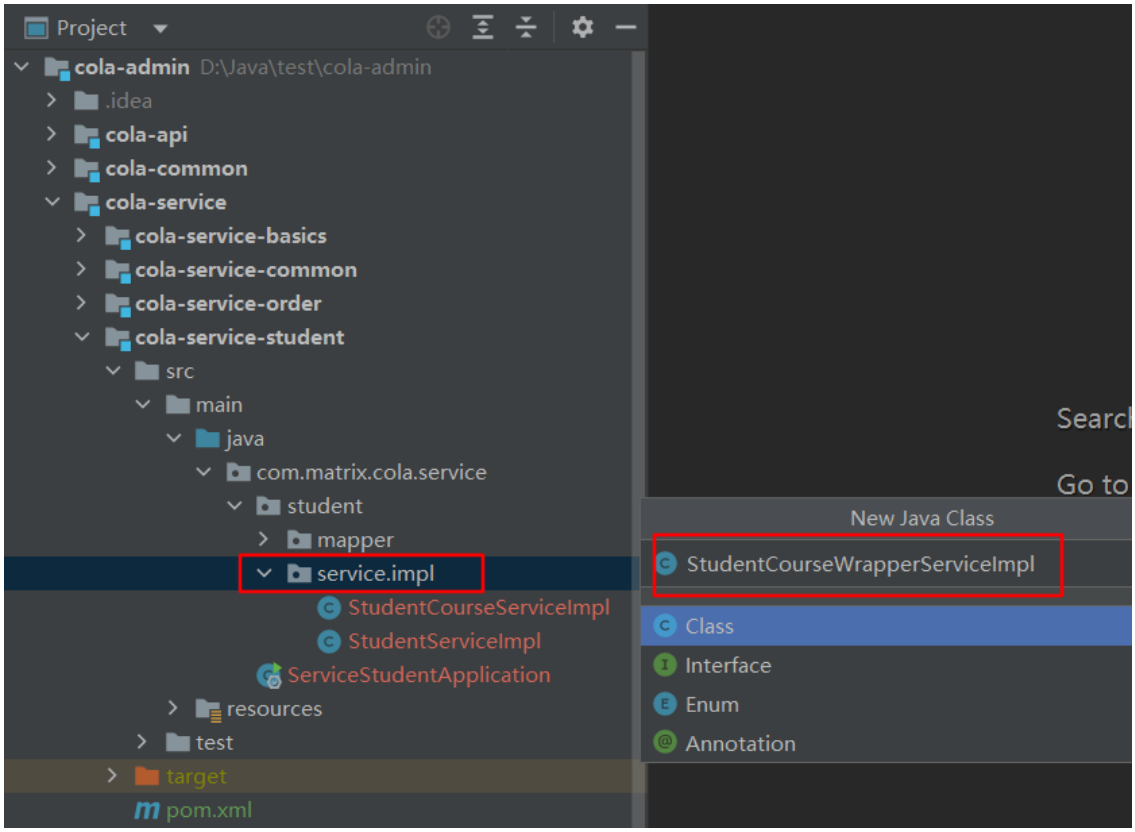
   同EntityService一样，WrapperService也提供了两个父接口，一个是BaseColaEntityWrapperService，另一个是BaseEntityWrapperService。如果EntityWrapper继承了BaseColaEntityWrapper，那么接口就需要继承BaseColaEntityWrapperService；如果继承了BaseEntityWrapper，那么接口就需要继承BaseEntityWrapperService。

   这 里我们继承BaseColaEntityWrapperService

```java
public interface StudentCourseWrapperService extends
BaseColaEntityWrapperService<StudentCourseEntity,
StudentCourseEntityWrapper> {
}
```

> 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。

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

```

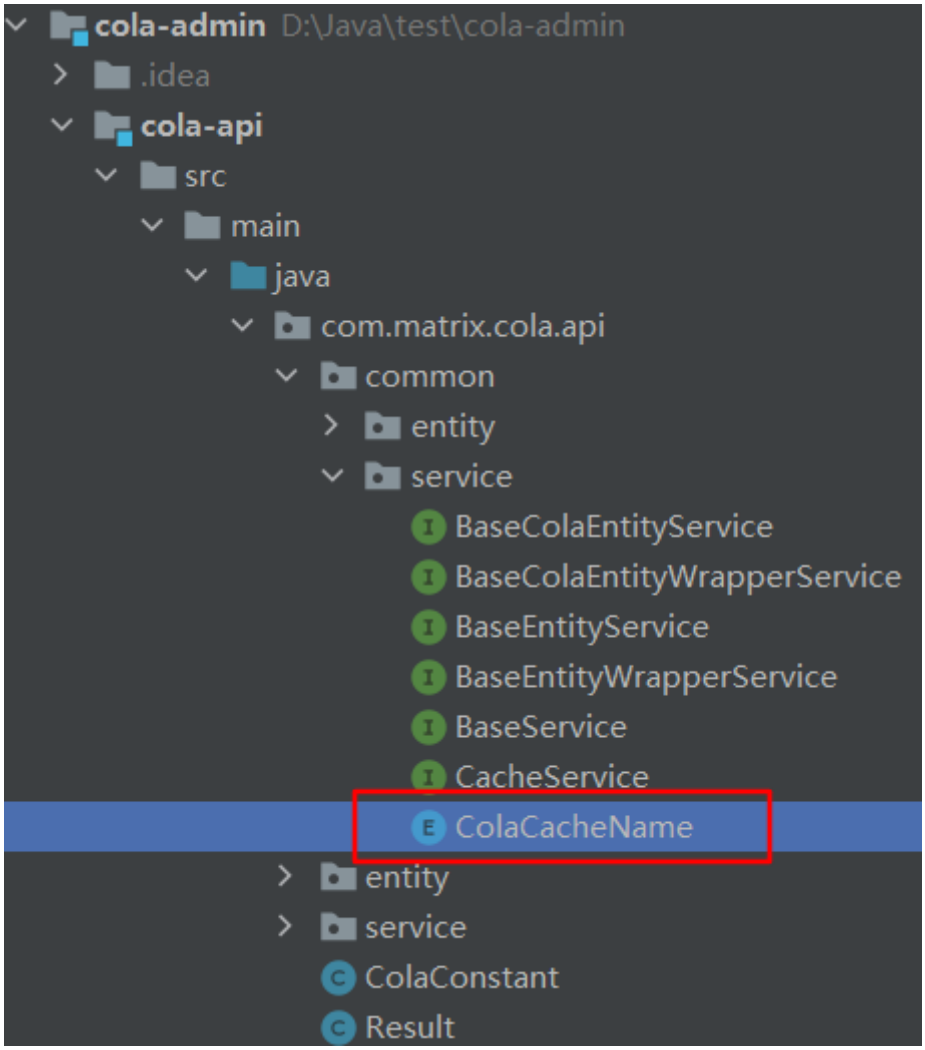
> AbstractColaEntityWrapperService需要三个泛型，第一个是BaseColaEntity的子类，第二个是BaseColaEntityWrapper的子类，第三个是BaseColaEntityService接口的实现类。

添加@DubboService注解

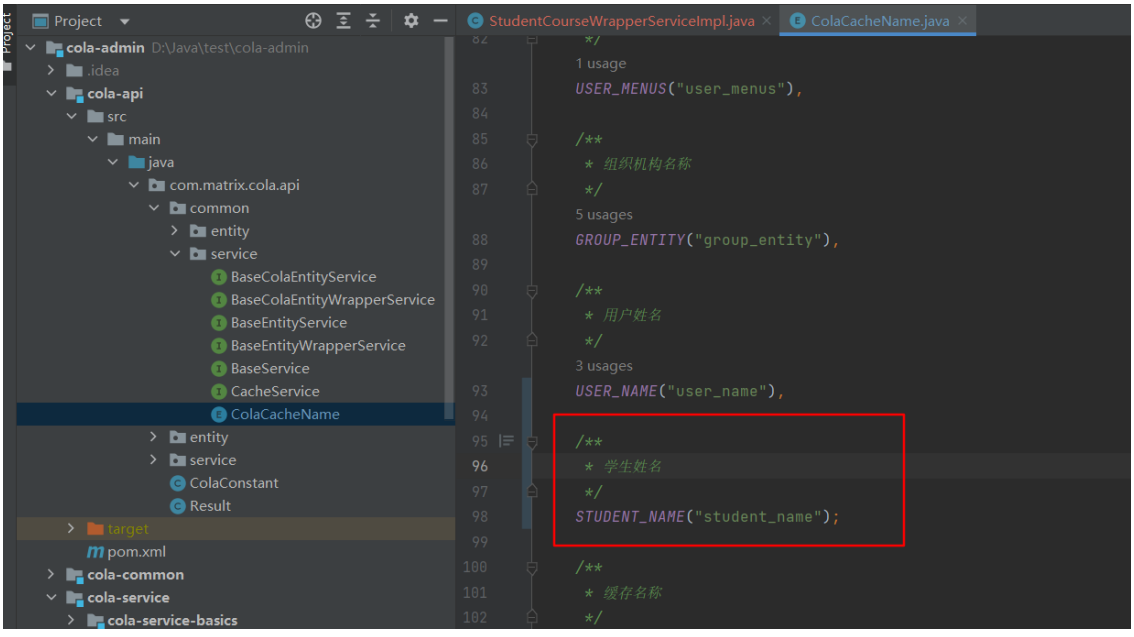
```
@DubboService
public class StudentCourseWrapperServiceImpl extends AbstractColaEntityWrapperServic
```

5. 重写包装方法

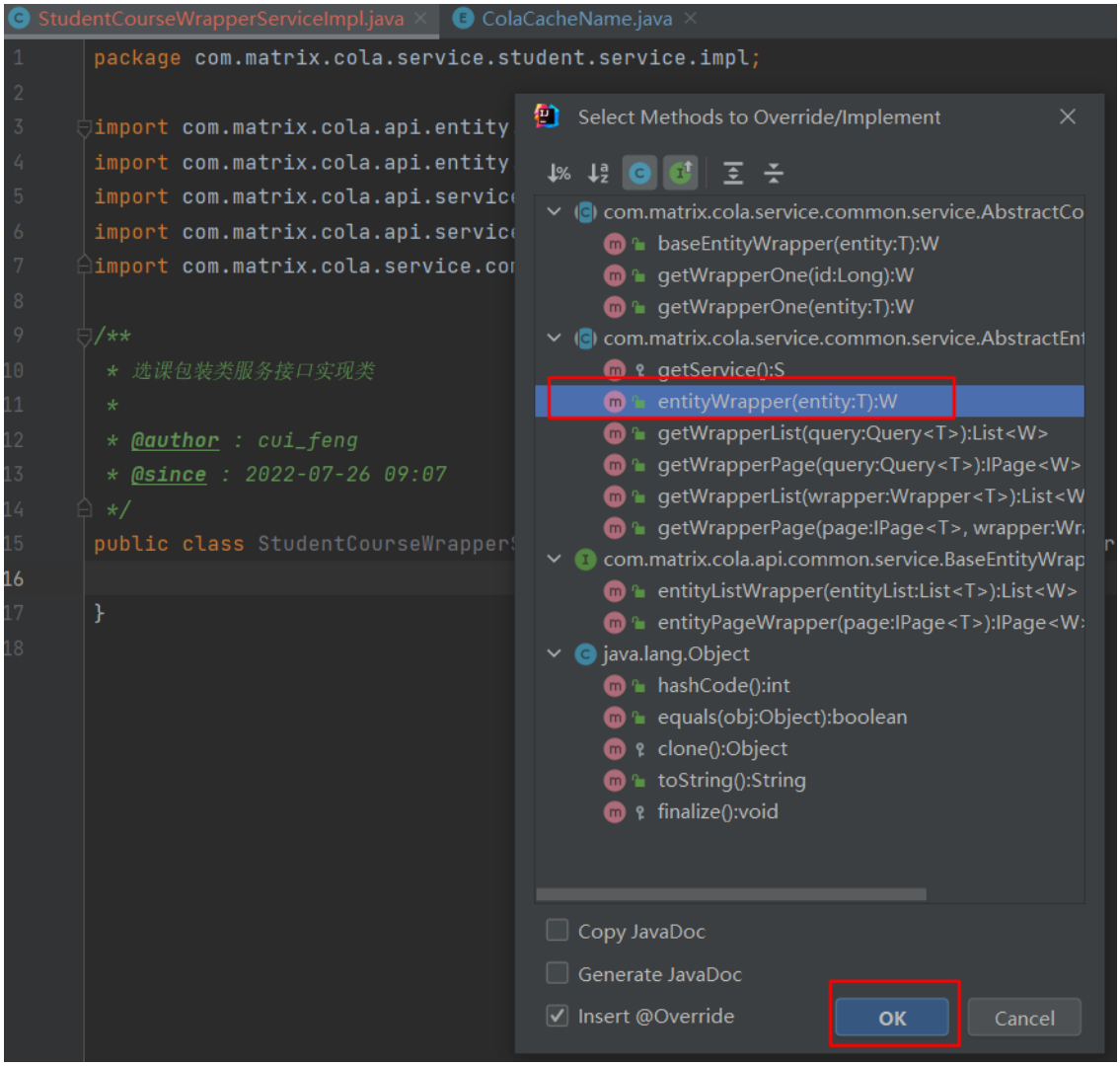
打开ColaCacheName类



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

```
@DubboService
public class StudentCourseWrapperServiceImpl extends
AbstractColaEntityWrapperService<StudentCourseEntity,
StudentCourseEntityWrapper, StudentCourseService> implements
StudentCourseWrapperService {

    // 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
    @Autowired
    StudentService studentService;

    @Override
    public StudentCourseEntityWrapper entityWrapper(StudentCourseEntity entity) {
        StudentCourseEntityWrapper studentCourseEntityWrapper = new StudentCourseEntityWrapper();
        if (ObjectUtil.isNotNull(entity.getStudentId())) {
            String studentName = cacheProxy.getObjectFromLoader(ColaCacheName.STUDENT_NAME, entity.getStudentId().toString(),() -> {
                StudentEntity student = studentService.getOne(entity.getStudentId());
                if (ObjectUtil.isNull(student)) {
                    return null;
```

```
                }
                return student.getName();
            });
            studentCourseEntityWrapper.setStudentName(studentName);
        }
        return studentCourseEntityWrapper;
    }
}
```

6. 使用EntityWrapperService

   修改StudentCourseController

```
@RestController
@RequestMapping("/studentCourse")
public class StudentCourseController {

    @DubboReference
    StudentCourseWrapperService studentCourseWrapperService;

    @PostMapping("getList")
    public Result getList() {
        return
Result.list(studentCourseWrapperService.getWrapperList(new Query<>()));
    }
}
```

7. 查看查询结果

```
{
    "success": true,
    "msg": "操作成功！",
    "data": {
        "list": [
            {
                "id": 1,
                "creator": 1,
                "creatorName": "超级管理员",
                "createTime": "2022-07-25 15:30:18",
                "startTime": null,
                "endTime": null,
                "reviser": 1,
                "reviserName": "超级管理员",
                "reviseTime": "2022-07-25 15:30:18",
                "deleted": 0,
                "showDeleted": null,
                "groupId": "1",
                "groupName": "集团公司",
                "studentId": 2,
                "courseName": "语文",
                "studentName": "张三"
            },
            {
                "id": 2,
                "creator": 1,
                "creatorName": "超级管理员",
                "createTime": "2022-07-25 15:30:18",
```

```
                "startTime": null,
                "endTime": null,
                "reviser": 1,
                "reviserName": "超级管理员",
                "reviseTime": "2022-07-25 15:30:18",
                "deleted": 0,
                "showDeleted": null,
                "groupId": "1",
                "groupName": "集团公司",
                "studentId": 3,
                "courseName": "数学",
                "studentName": "王五"
            },
            {
                "id": 3,
                "creator": 1,
                "creatorName": "超级管理员",
                "createTime": "2022-07-25 15:30:26",
                "startTime": null,
                "endTime": null,
                "reviser": 1,
                "reviserName": "超级管理员",
                "reviseTime": "2022-07-25 15:30:26",
                "deleted": 0,
                "showDeleted": null,
                "groupId": "1",
                "groupName": "集团公司",
                "studentId": 4,
                "courseName": "地理",
                "studentName": "赵六"
            }
        ]
    },
    "code": 200
}
```


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

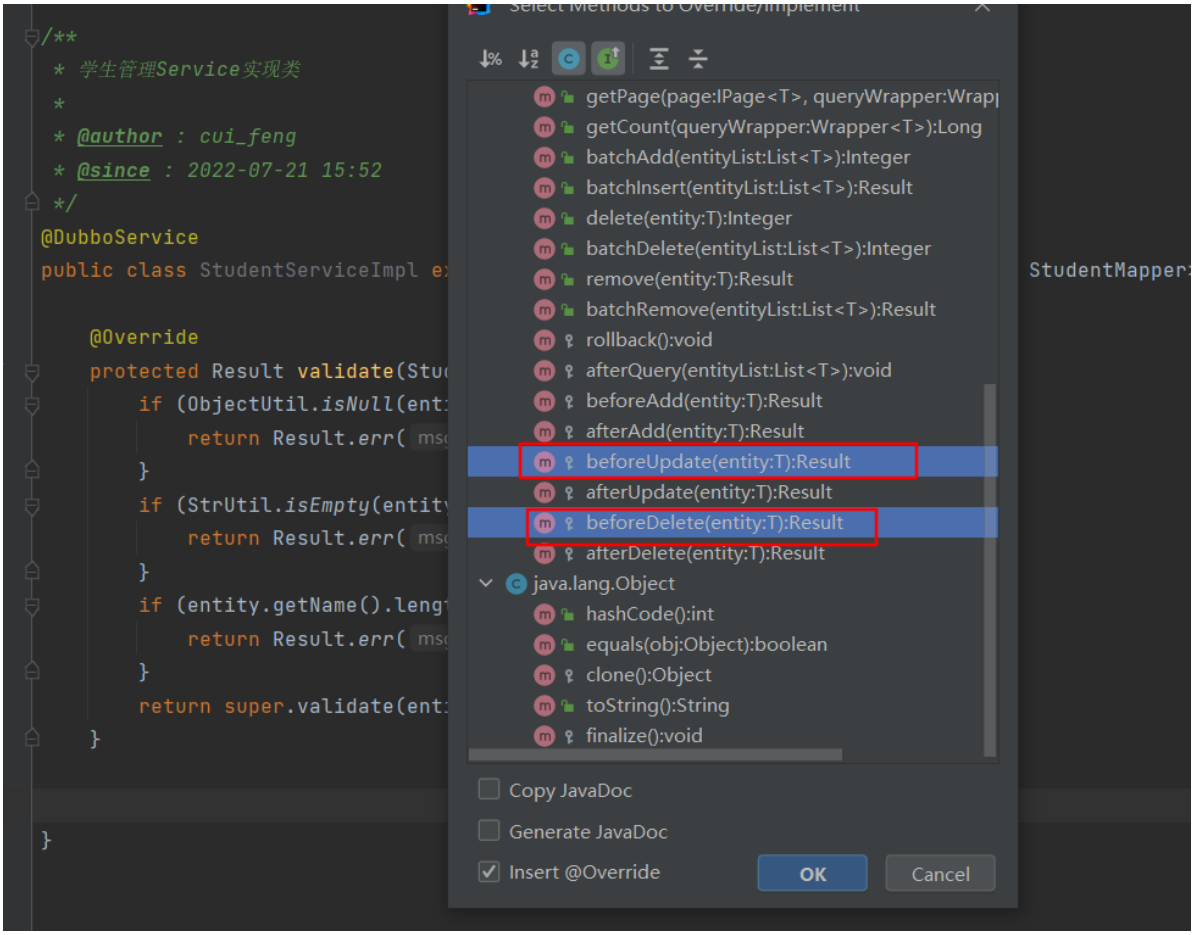
```
public interface DataLogService extends BaseColaEntityService<DataLogEntity>
{

    /**
     * 记录修改日志
     * @param tableName 表名
     * @param before 更新前的数据
     * @param after 更新后的数据
     */
    void addUpdateLog(String tableName, BaseColaEntity before,
BaseColaEntity after);

    /**
     * 记录删除日志
     * @param tableName 表名
```

```
     * @param before 删除前的记录
     */
    void addDeleteLog(String tableName, BaseColaEntity before);

    /**
     * 物理删除数据日志
     * @param dataLogEntity 数据日志实体类
     * @return 统一结果
     */
    Result deleteDataLog(DataLogEntity dataLogEntity);

    /**
     * 删除全部数据日志
     * @return 统一结果
     */
    Result clearDataLog();
}
```

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，

在StudentServiceImpl中重写beforeUpdate和beforeDelete两个方法




```
@DubboService
public class StudentServiceImpl extends
AbstractColaEntityService<StudentEntity, StudentMapper> implements
StudentService {

    @Autowired
    DataLogService dataLogService;
```

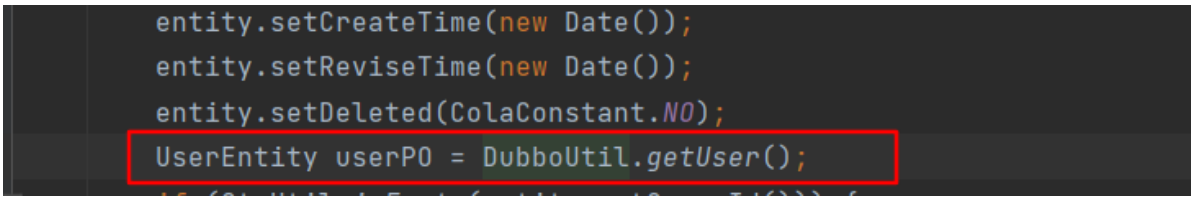
```
6
7       @Override
8       protected Result validate(StudentEntity entity) {
9           if (ObjectUtil.isNull(entity)) {
10              return Result.err("学生信息不能为空");
11          }
12          if (StrUtil.isEmpty(entity.getName())) {
13              return Result.err("姓名不能为空");
14          }
15          if (entity.getName().length()>4) {
16              return Result.err("姓名不能超过四个字");
17          }
18          return super.validate(entity);
19      }
20
21      @Override
22      protected Result beforeUpdate(StudentEntity entity) {
23          dataLogService.addUpdateLog("学生管
    理",getOne(entity.getId()),entity);
24          return super.beforeUpdate(entity);
25      }
26
27      @Override
28      protected Result beforeDelete(StudentEntity entity) {
29          dataLogService.addDeleteLog("学生管理",getOne(entity.getId()));
30          return super.beforeDelete(entity);
31      }
32  }
```

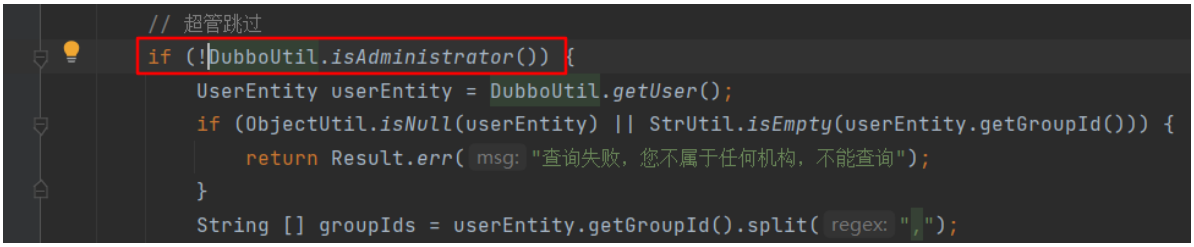
工具类说明

cola-admin提供了几个常用的工具类用来简化开发。

DubboUtil

用于cola-service中，DubboUtil中主要有两个方法，一个是getUser()，用来获取当前登陆的用户信息；另一个是isAdministrator()，该方法用于判断当前用户是否为超级管理员。





WebUtil

用于cola-web中

```
1   public class WebUtil {
2
```

```
    /**
     * 获取IP地址
     * @return ip地址
     */
    public static String getIP();

    /**
     * 获取Request对象
     * @return request对象
     */
    public static HttpServletRequest getRequest();

    /**
     * 获取当前登陆用户
     * @return 用户实体类
     */
    public static UserEntity getUser();

    /**
     * 获取当前请求的token
     */
    public static String getToken();

    /**
     * 判断token是否过期
     *
     * @param token 前端传过来的Token
     * @return 是否过期
     */
    public static boolean isTokenExp(String token);
}
```

CacheProxy

缓存代理类，用于缓存的操作，实现了CacheService

```
/**
 * 缓存接口
 *
 * @author cui_feng
 * @since : 2022-04-20 14:18
 */
public interface CacheService extends BaseService {

    /**
     * 向缓存中添加对象
     * @param cacheName 缓存名 {@link ColaCacheName}
     * @param key 缓存key值
     * @param value 缓存对象
     */
    void put(ColaCacheName cacheName, String key, Object value);

    /**
     * 从缓存中获取一个对象
     * @param cacheName 缓存名 {@link ColaCacheName}
     * @param key 缓存key值
     * @return 缓存对象
```

```
     */
    <T> T getObject(ColaCacheName cacheName, String key);


    /**
     * 从缓存中获取指定对象，如果不存在则调用valueloader回调，并将value添加到缓存中
     * 此方法主要用于兼容Redis
     *
     * @param cacheName 缓存名 {@link ColaCacheName}
     * @param key 缓存key值
     * @param valueLoader 值加载器，一个 {@link Callable}接口，值不存在时执行，执行后将对象加入到缓存中
     * @param <T> 缓存对象泛型
     * @return 缓存对象
     */
    <T> T getObjectFromLoader(ColaCacheName cacheName, String key, Callable<T> valueLoader);

    /**
     * 从缓存中获取指定的对象
     * @param cacheName 缓存名 {@link ColaCacheName}
     * @param key 缓存key值
     * @param t 缓存对象的class
     * @param <T> 缓存对象泛型
     * @return 缓存对象
     */
    <T> T getObjectFromClass(ColaCacheName cacheName, String key, Class<T> t);

    /**
     * 从指定的缓存中删除一个指定的缓存
     * @param cacheName 缓存名 {@link ColaCacheName}
     * @param key 缓存key值
     */
    void evict(ColaCacheName cacheName, String key);

    /**
     * 清空指定的缓存
     * @param cacheName 缓存名 {@link ColaCacheName}
     */
    void clear(ColaCacheName cacheName);
}
```

如果继承了AbstractColaEntityService、AbstractEntityService、AbstractColaEntityWrapperService、AbstractEntityWrapperService，无需引入可以直接使用。

FAQ

1、添加服务后启动报错

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no
embedded datasource could be configured.

Reason: Failed to determine a suitable driver class
```

请打开maven面板刷新整个工程