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

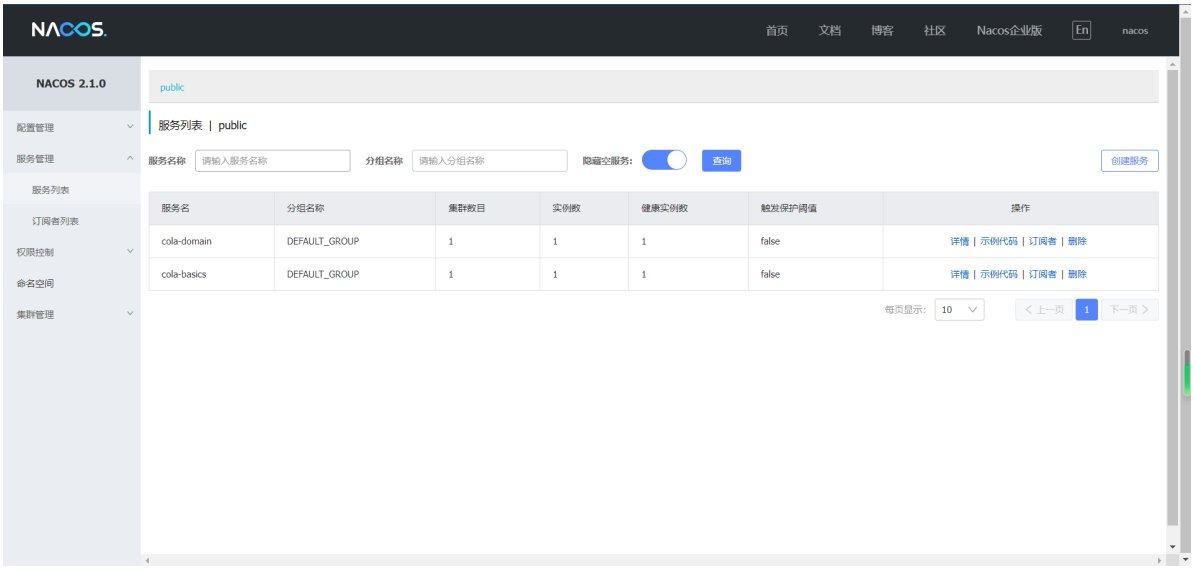
环境准备

Nacos安装

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

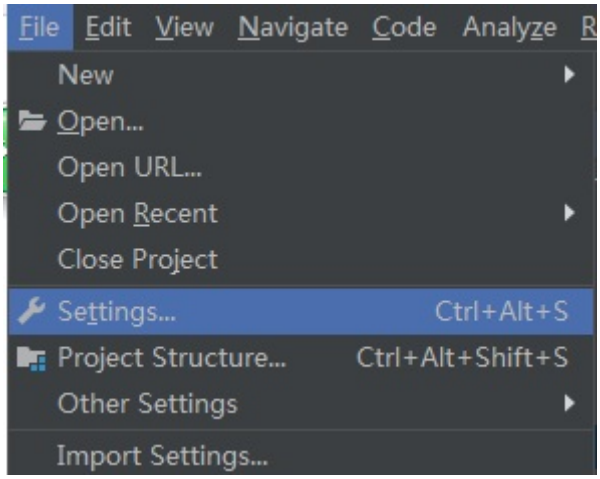
Nacos界面

默认用户名和密码都是nacos

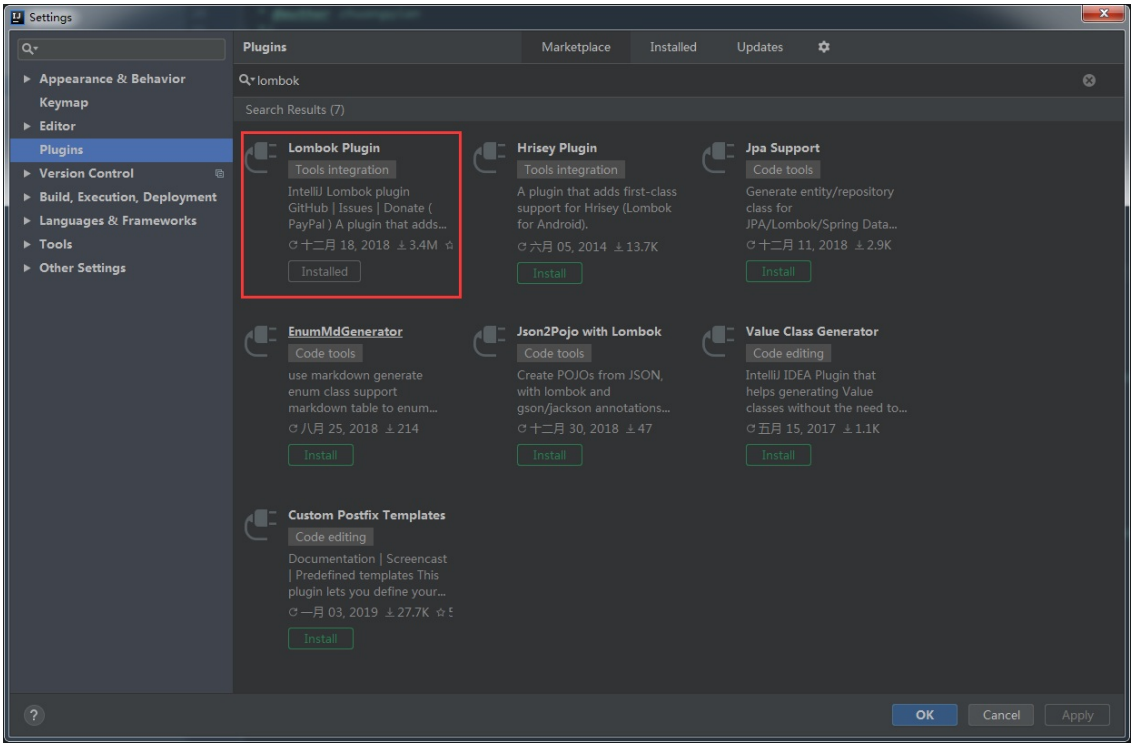


IDEA 插件安装

1. 选择 File->Settings



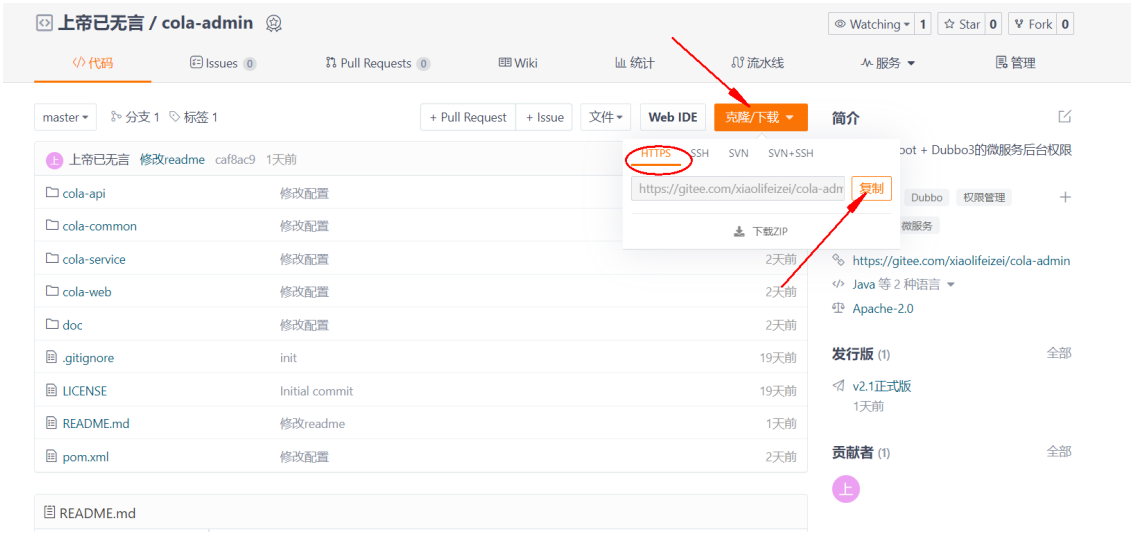
2. 选择 Plugins 并搜索 Lombok



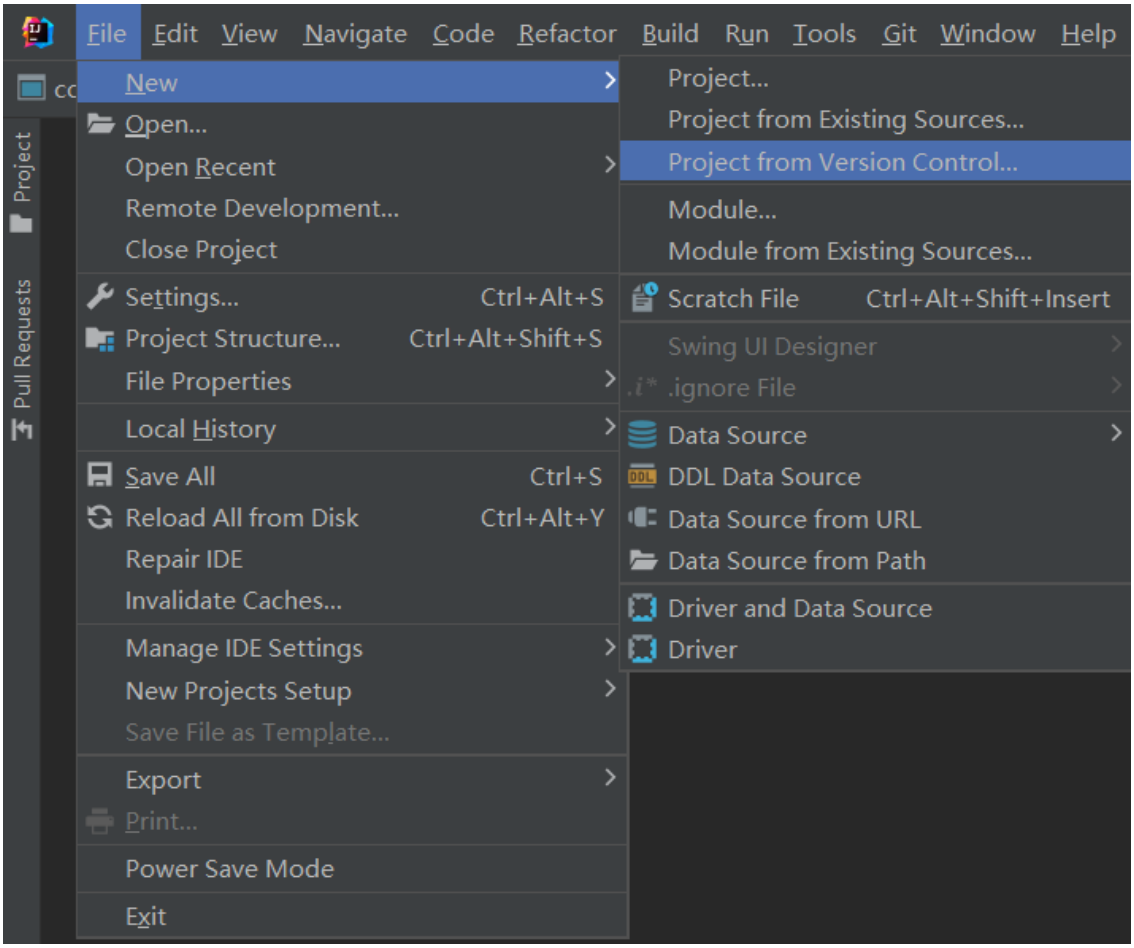
3. 点击 Install 按钮
4. 重启 idea 生效

导入工程

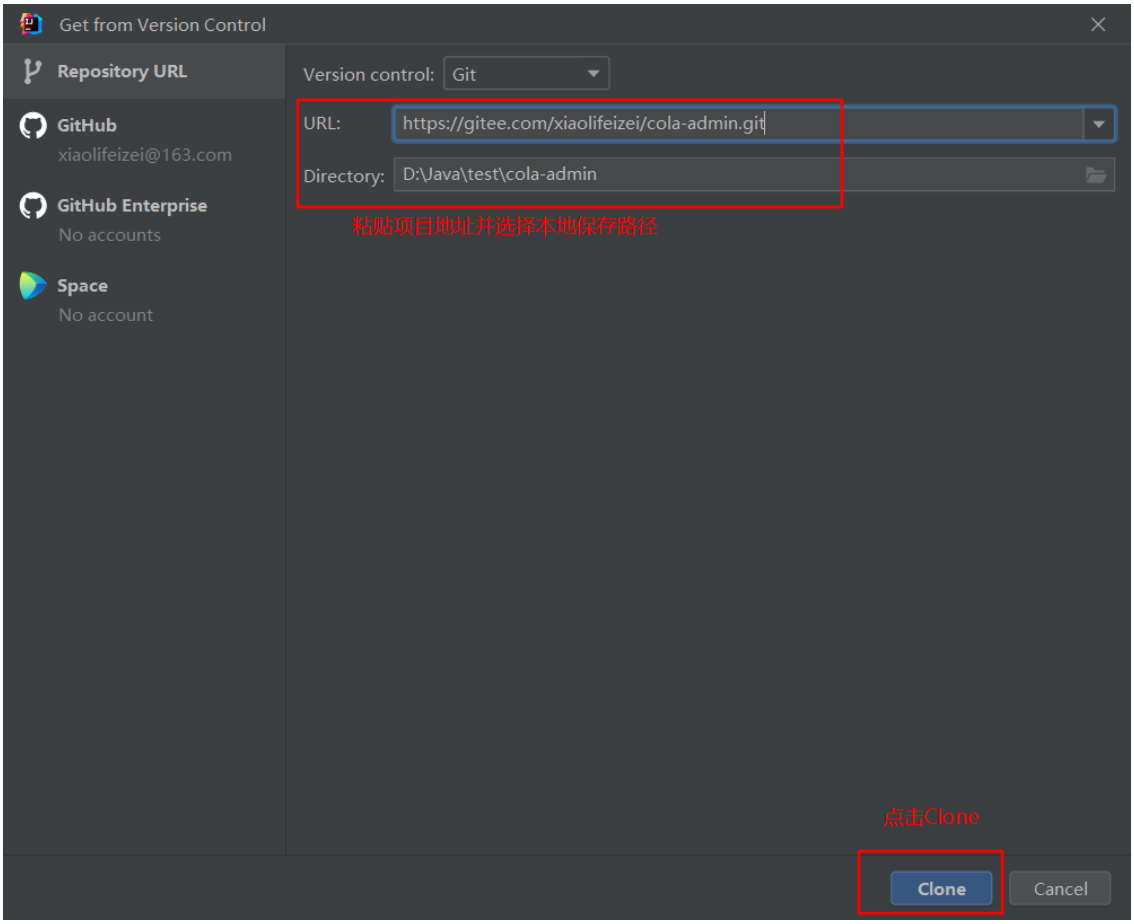
1. 进入cola-admin项目首页https://gitee.com/xiaolifeizei/cola-admin
2. 复制项目地址



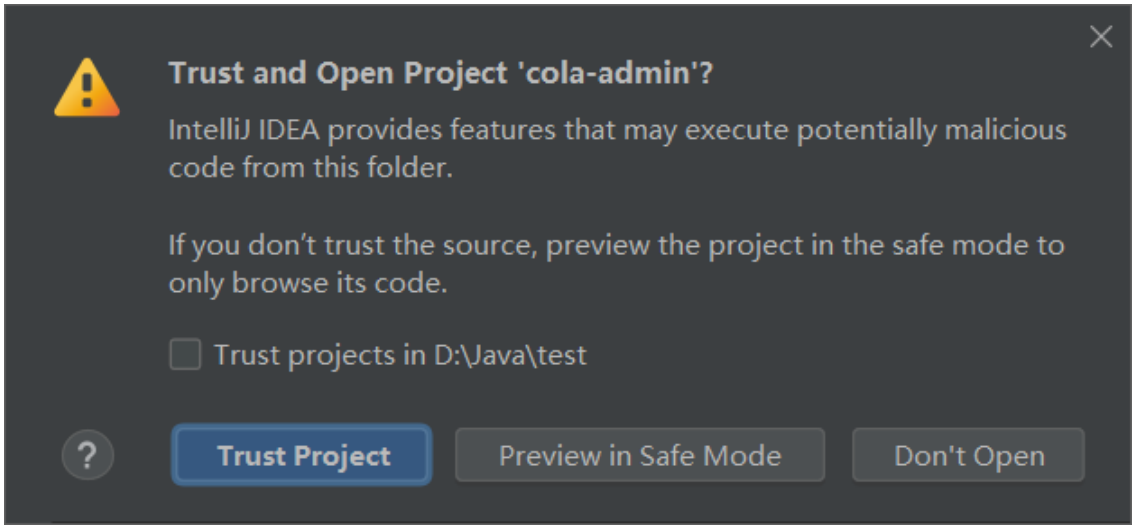
3. 打开IDEA，依次选择： File->New->Project from Version Control



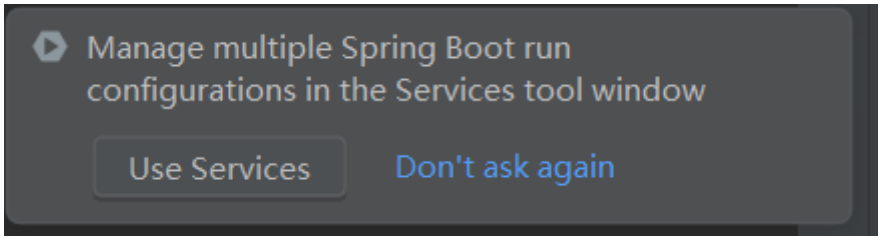
4. 在弹出的对话框中粘贴复制的项目地址



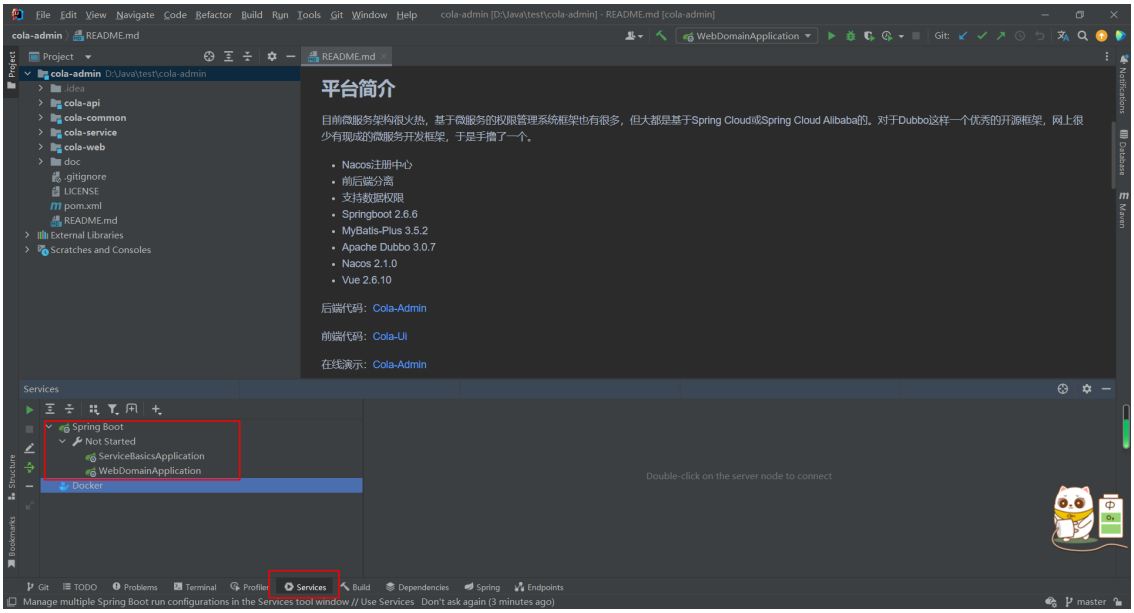
5. IDEA可能会弹出对话框提示，点击“Trust Project”



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

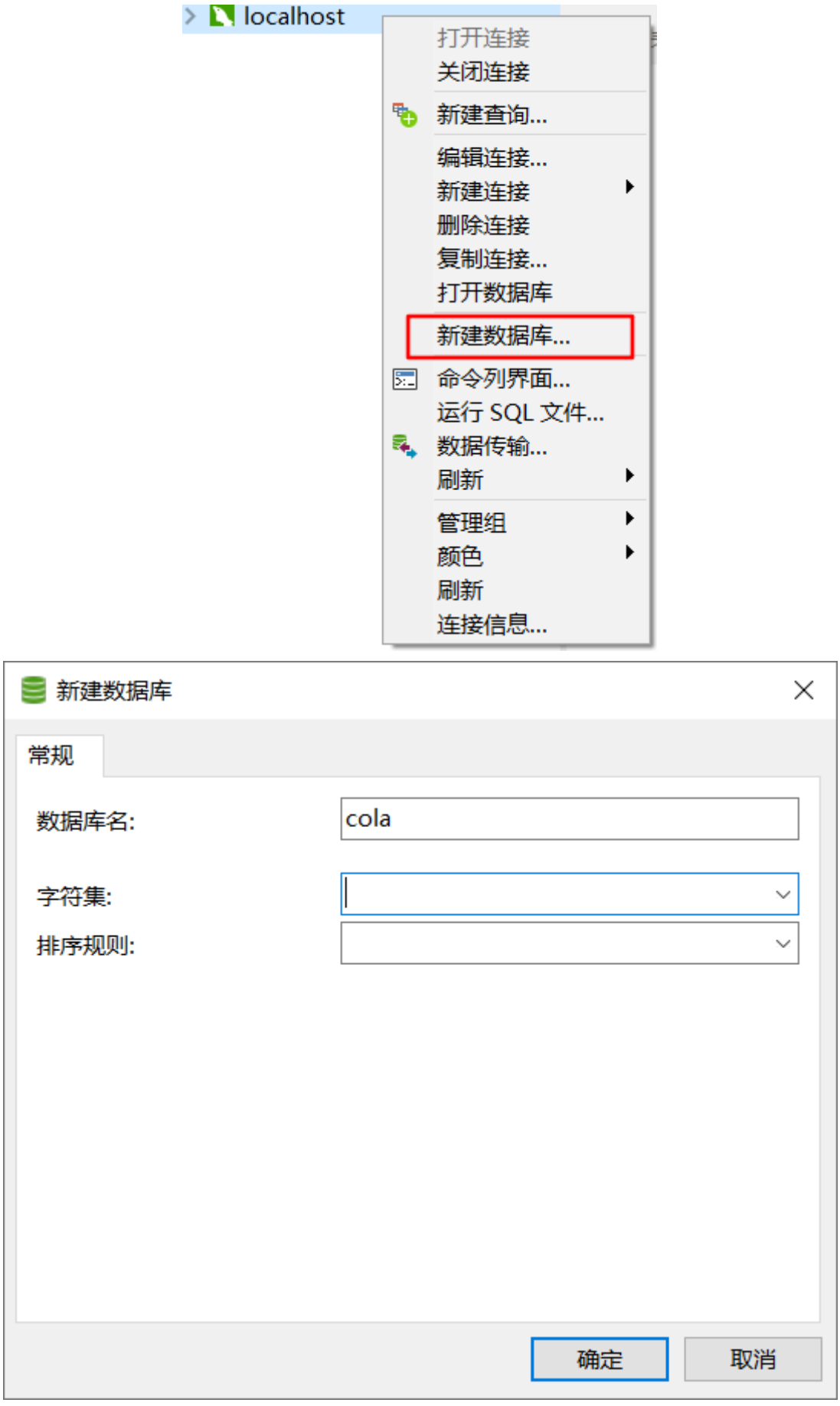


8. 点击Service面板可以看到下图的启动项则说明导入成功

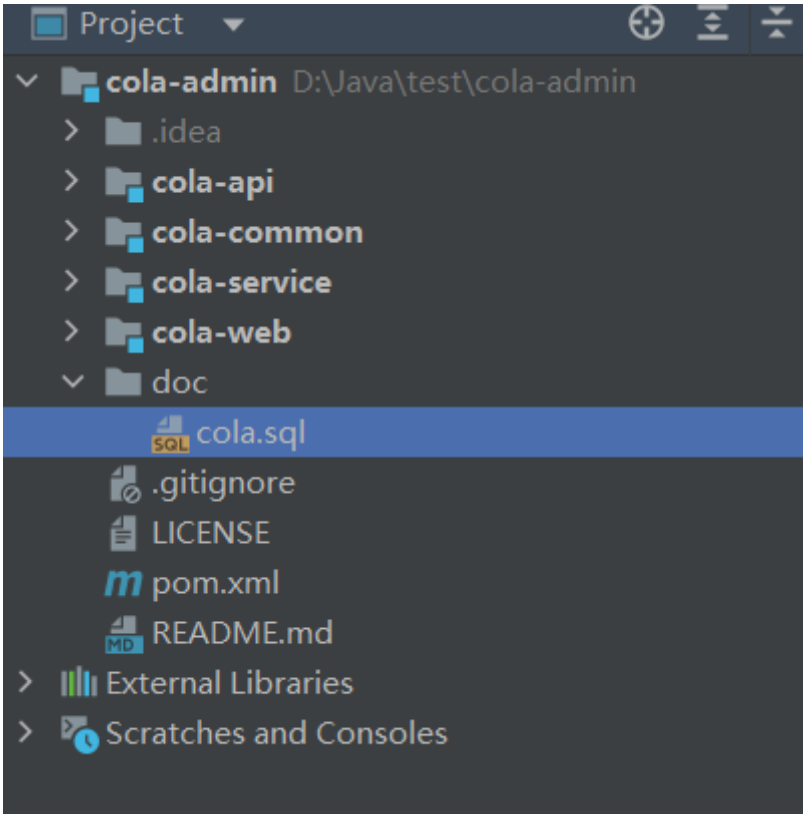


创建数据库

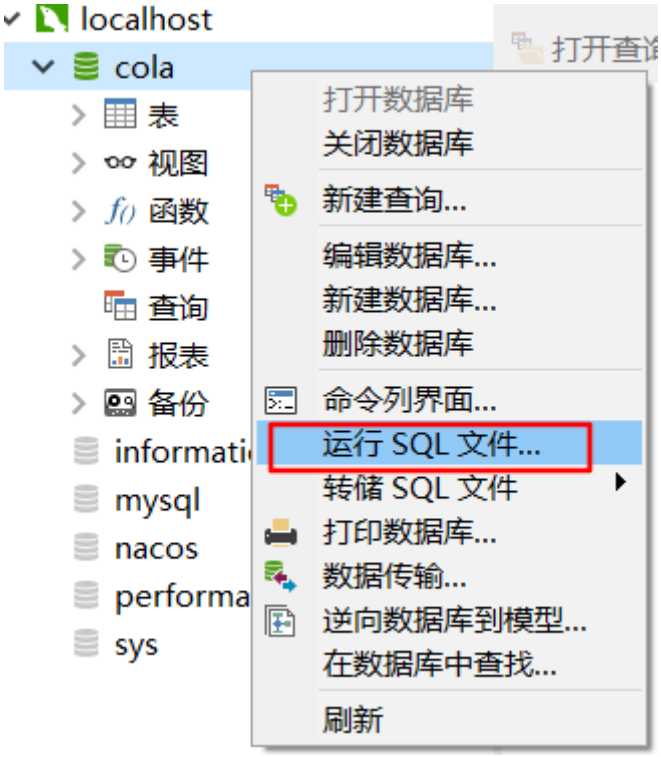
1. 打开Navicat（此处可以选择其他的客户端），新建一个数据库cola

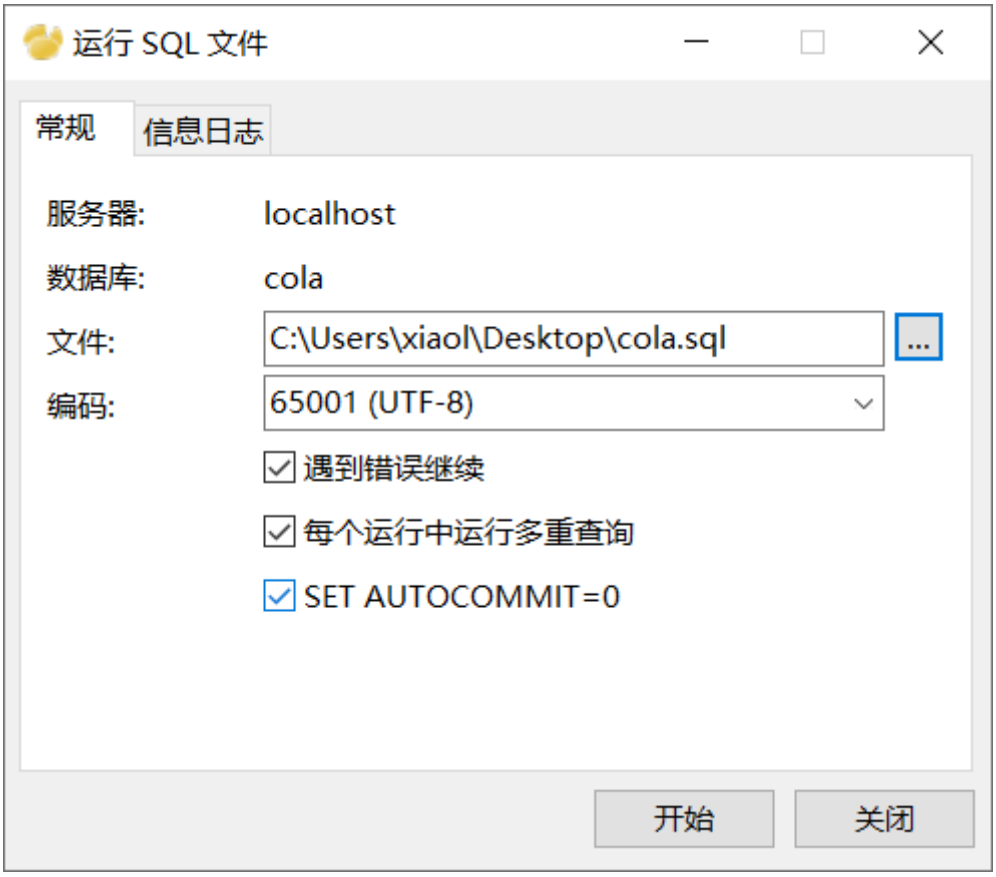


2. 找到cola-admin工程->doc->cola.sql

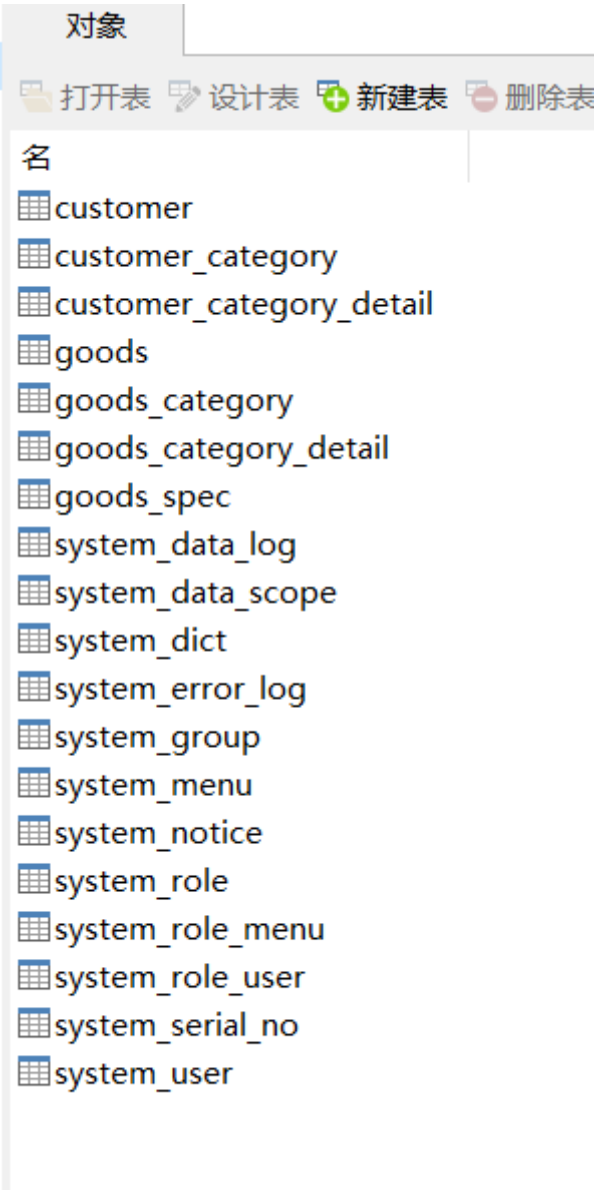

Project
cola-admin D:\Java\test\cola-admin
.idea
cola-api
cola-common
cola-service
cola-web
doc
cola.sql
.gitignore
LICENSE
pom.xml
README.md
External Libraries
Scratches and Consoles

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

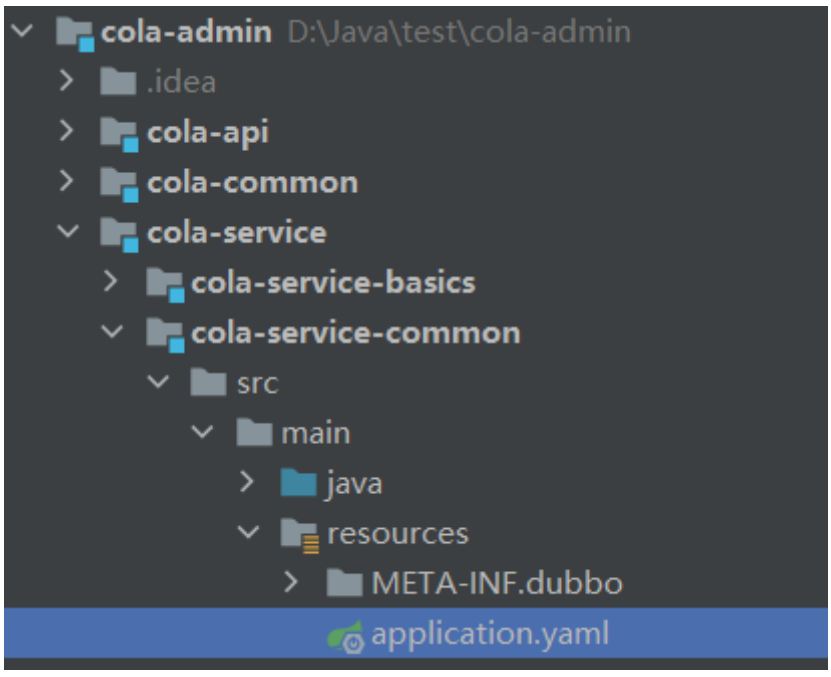
4. 最终效果



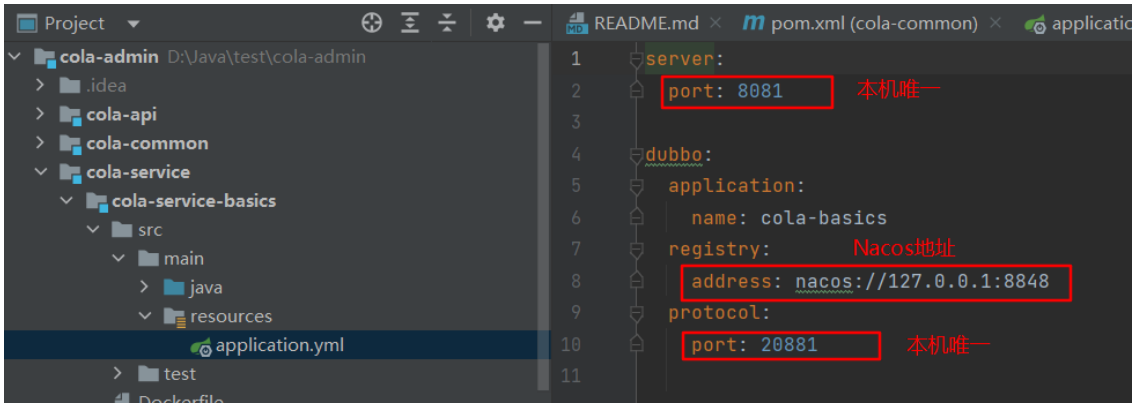
运行工程

1. 修改配置文件

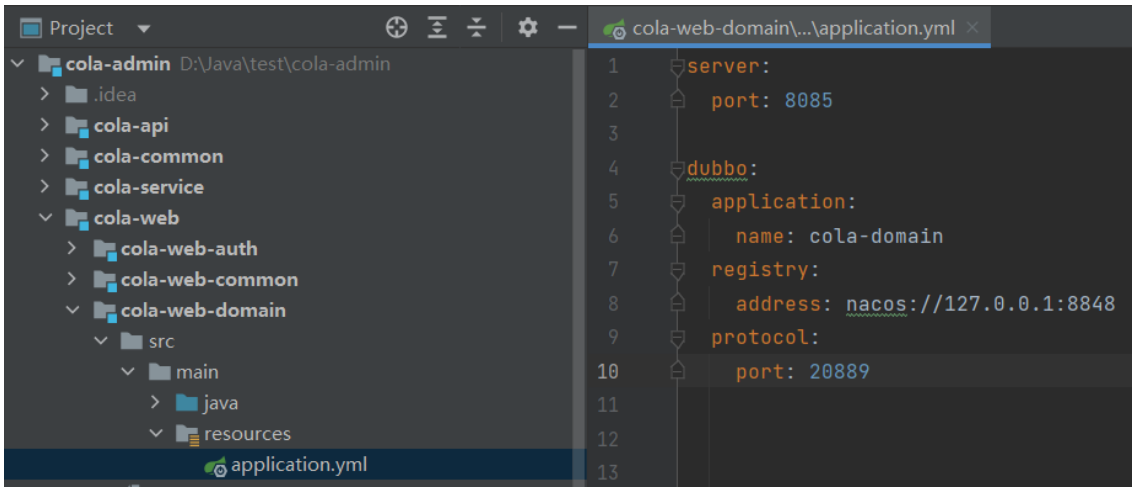
   修改数据库配置：cola-service-common/src/resources/application.yaml



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

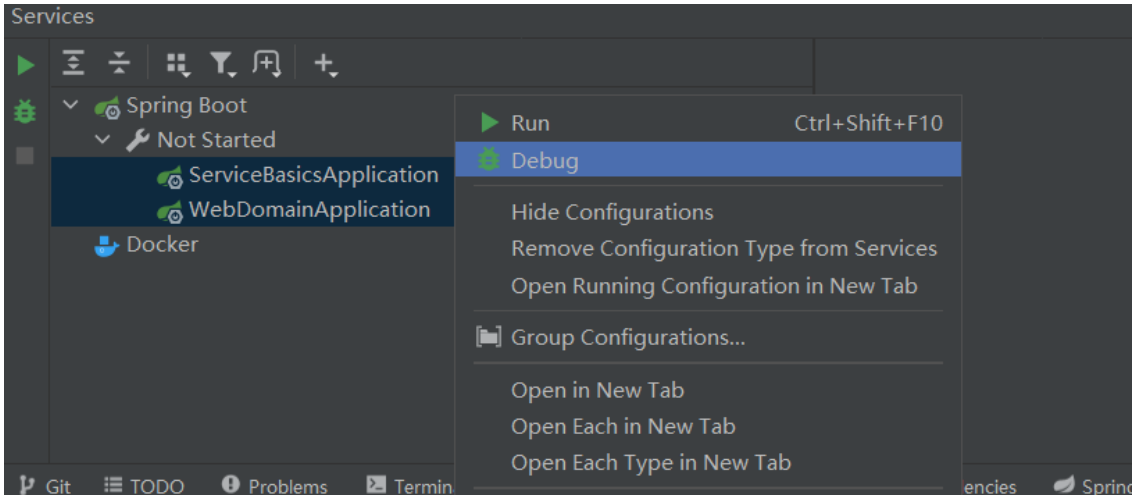


修改WebDomainApplication配置：cola-web-domain/src/resources/application.yml

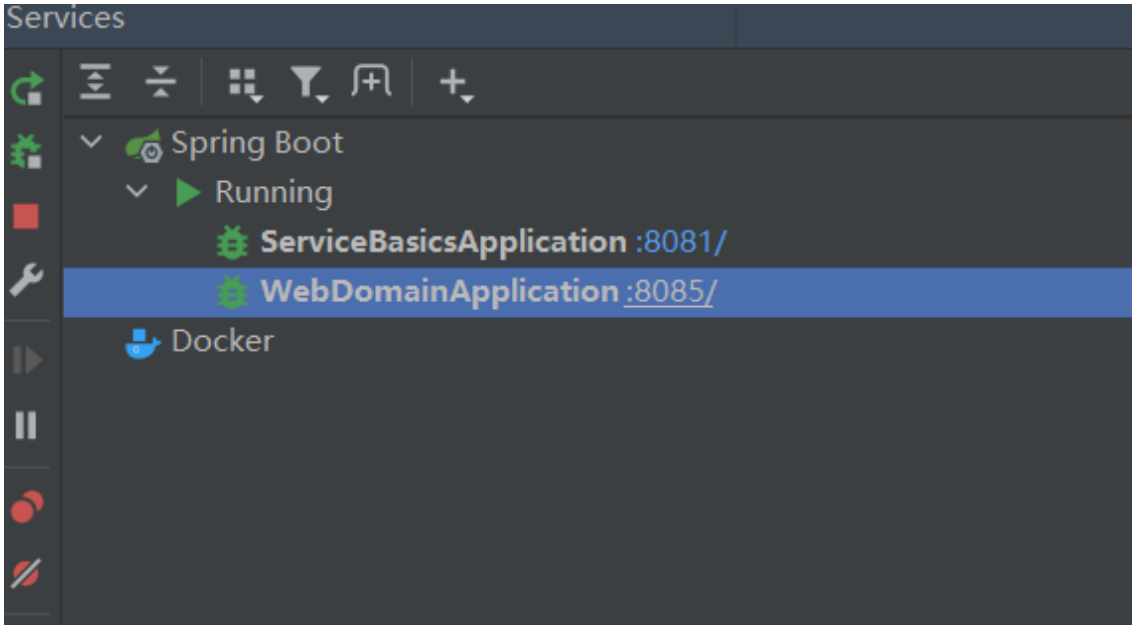


2. 启动项目

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

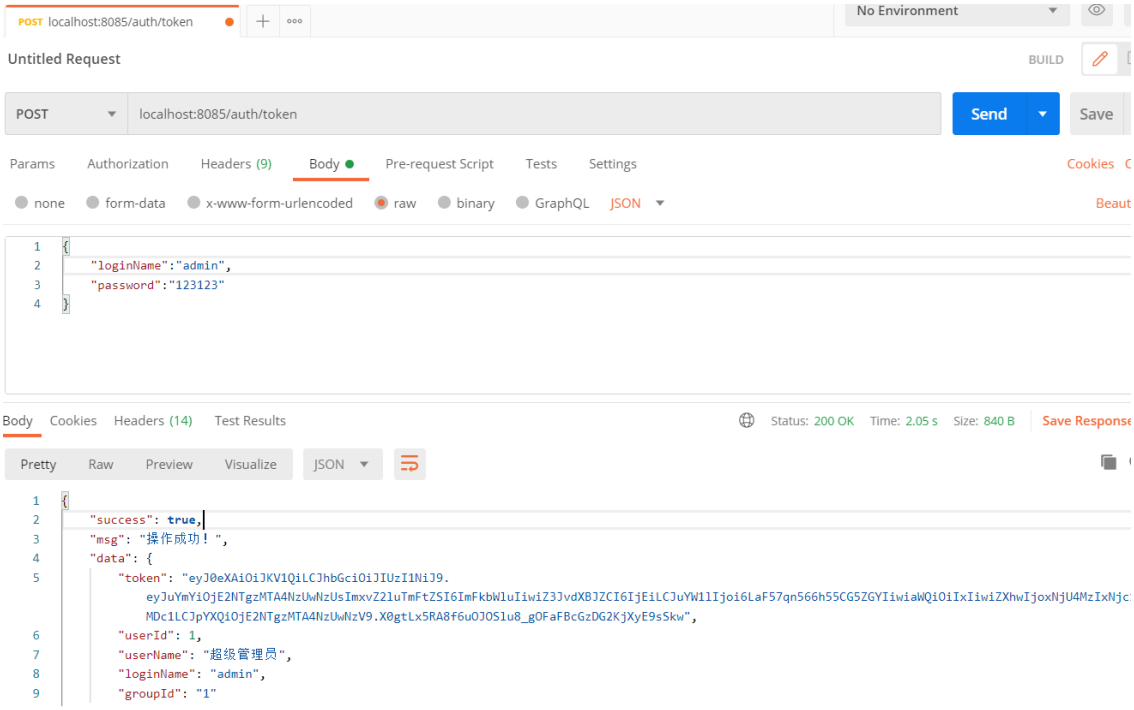
   两个服务已经成功注册上了

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

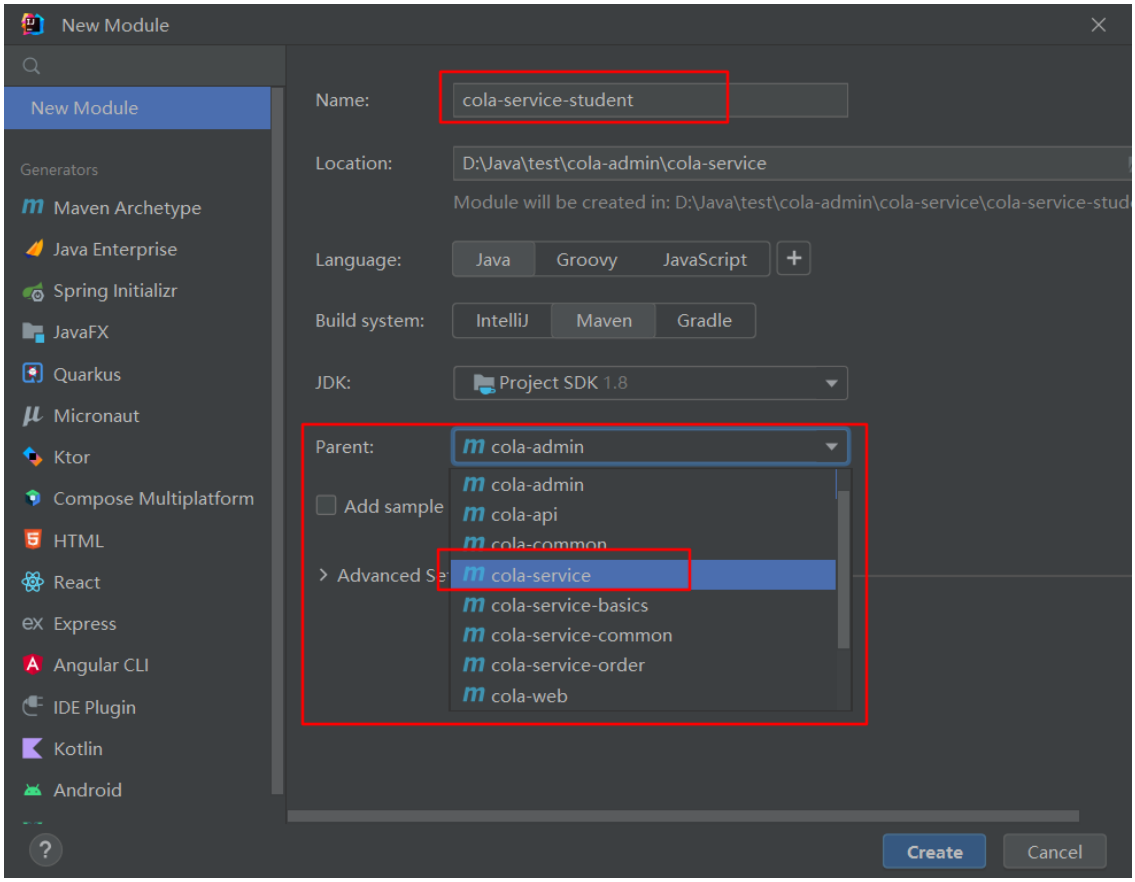
开发初探

新建微服务工程

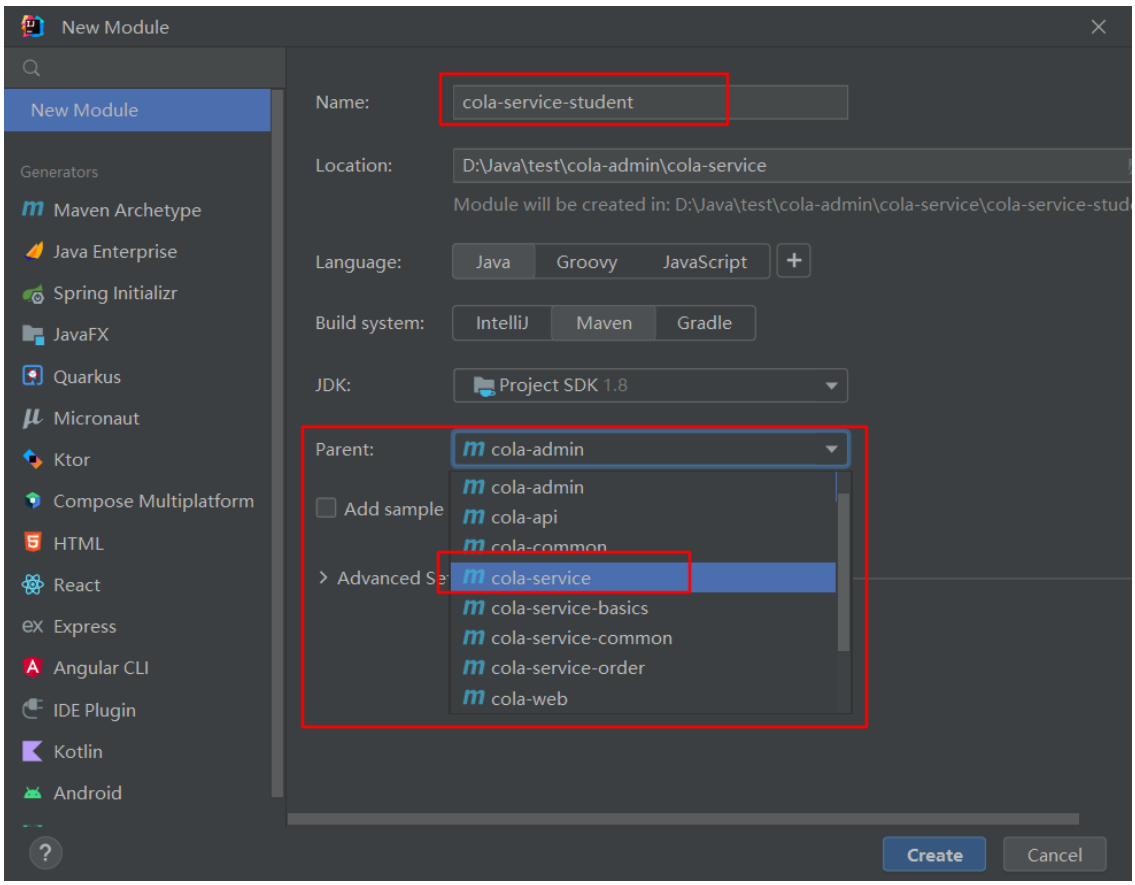
项目中已经有一个订单的空工程，可以根据需要改名后使用，下面介绍新建一个微服务工程的操作流程。

新建

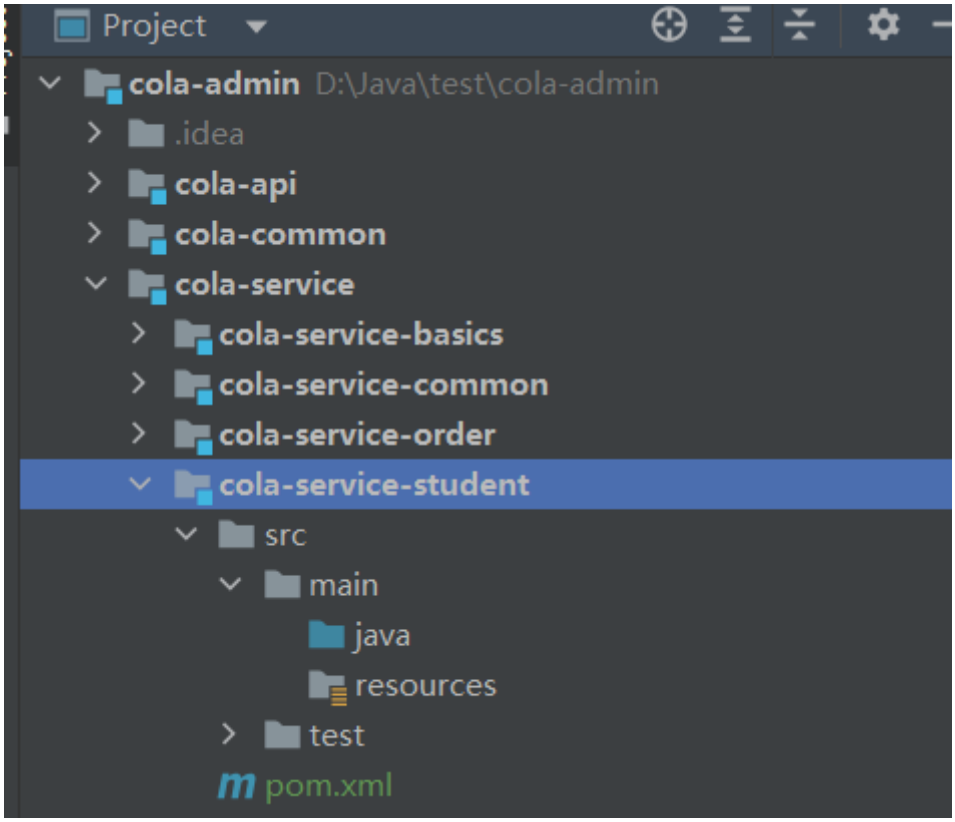
1. 在cola-service模块上右键，选择new-->module，这里以新建学生管理为例



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

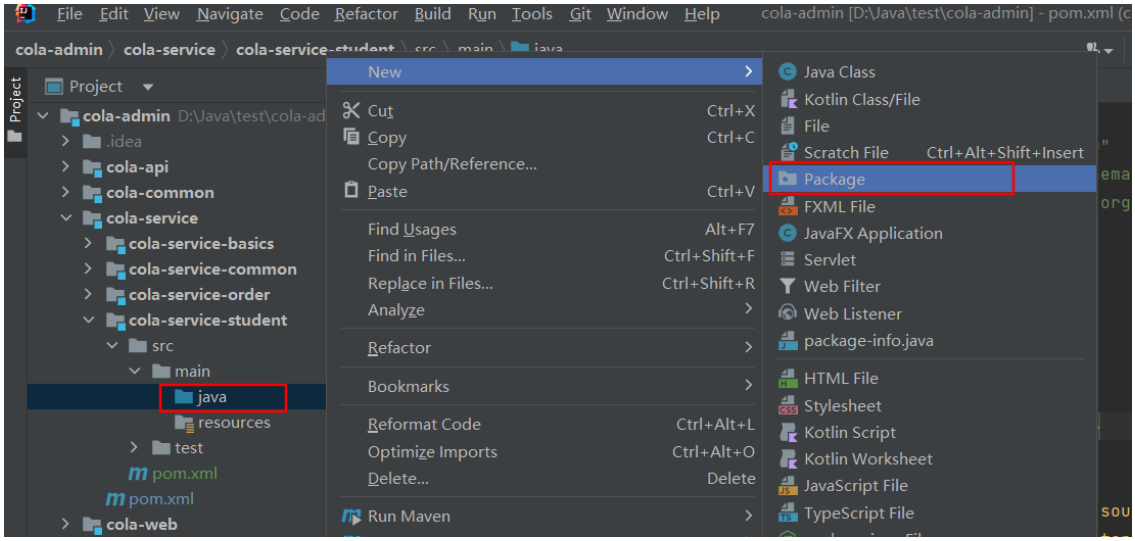


3. 创建成功

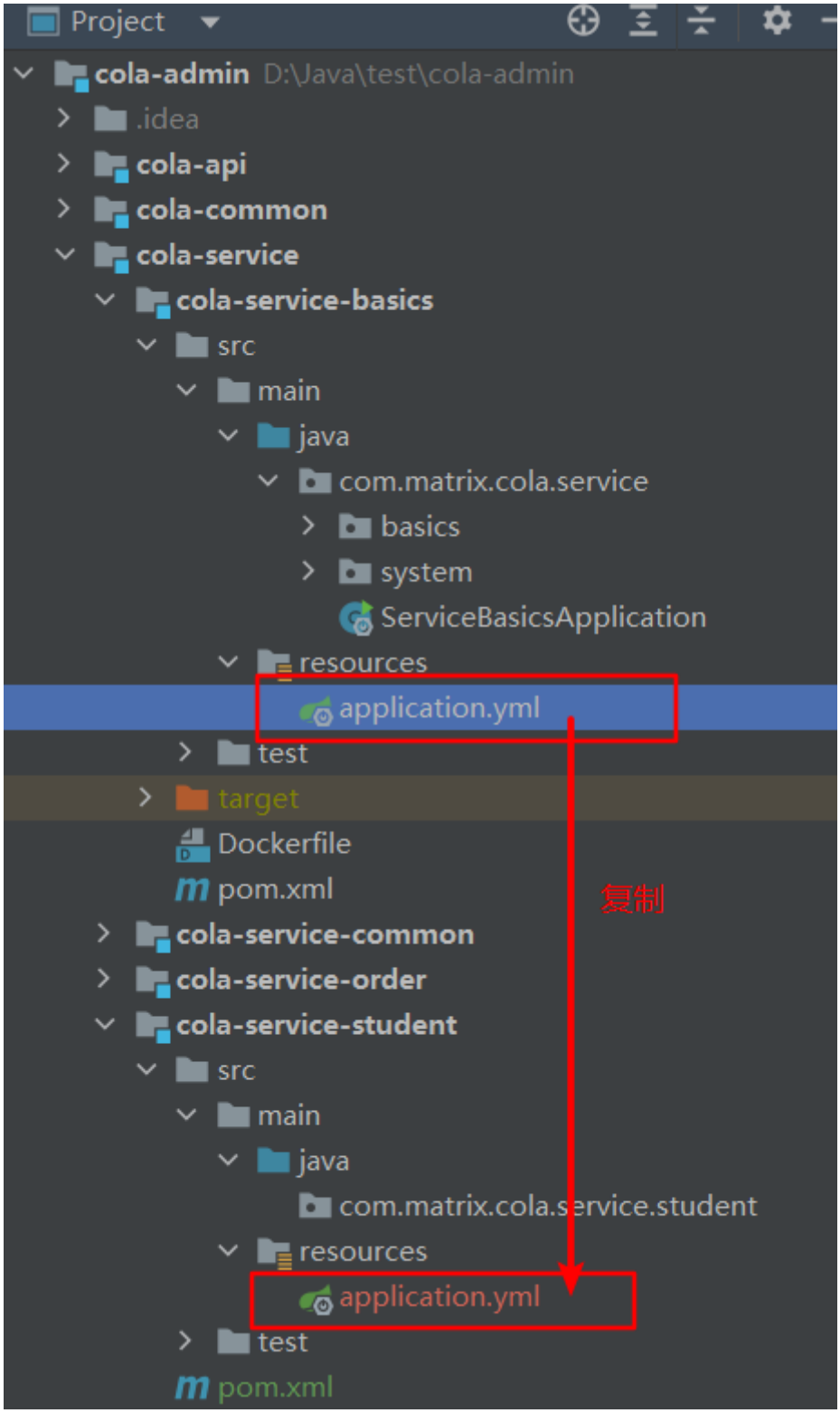


4. 建包、添加配置文件

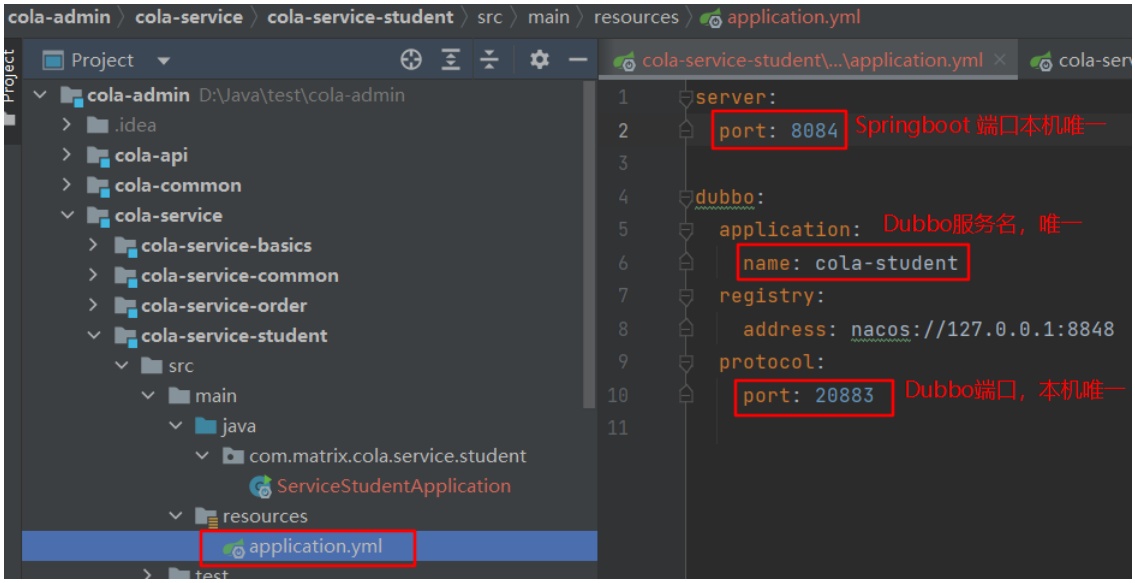
   新建包名com.matrix.cola.service.student



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

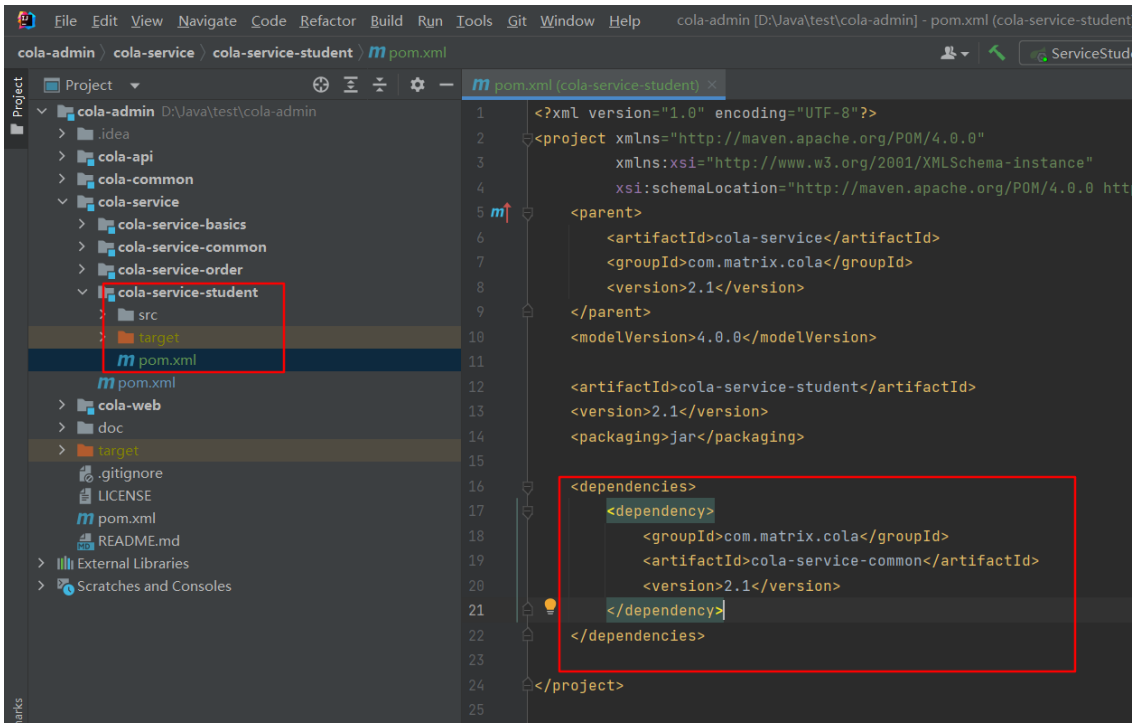

Project
cola-admin D:\Java\test\cola-admin
.idea
cola-api
cola-common
cola-service
cola-service-basics
src
main
java
com.matrix.cola.service
basics
system
ServiceBasicsApplication
resources
application.yml
test
target
Dockerfile
pom.xml
复制
cola-service-common
cola-service-order
cola-service-student
src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

修改配置文件



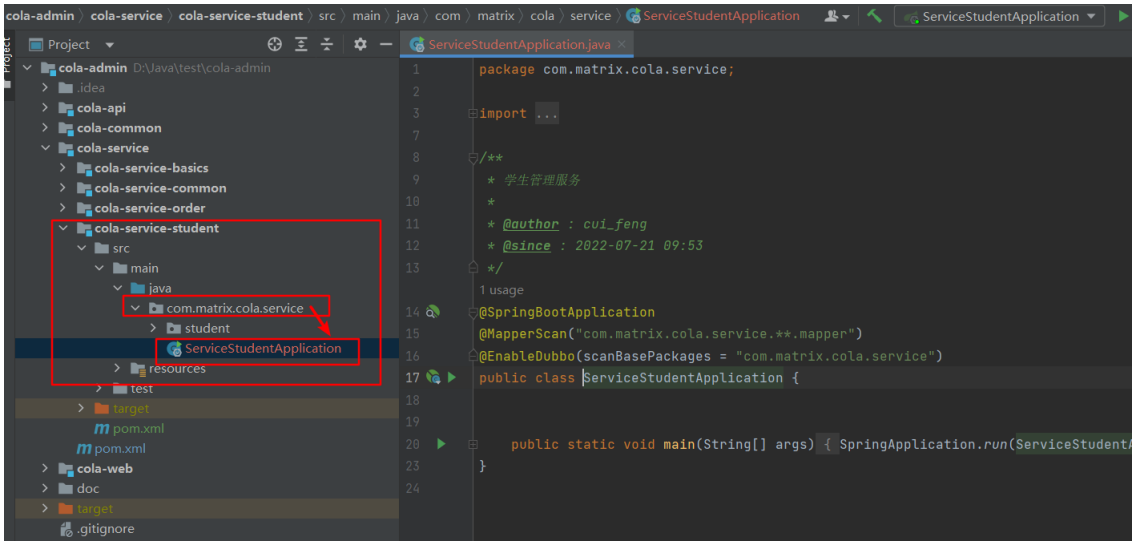
5. 引入依赖

   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



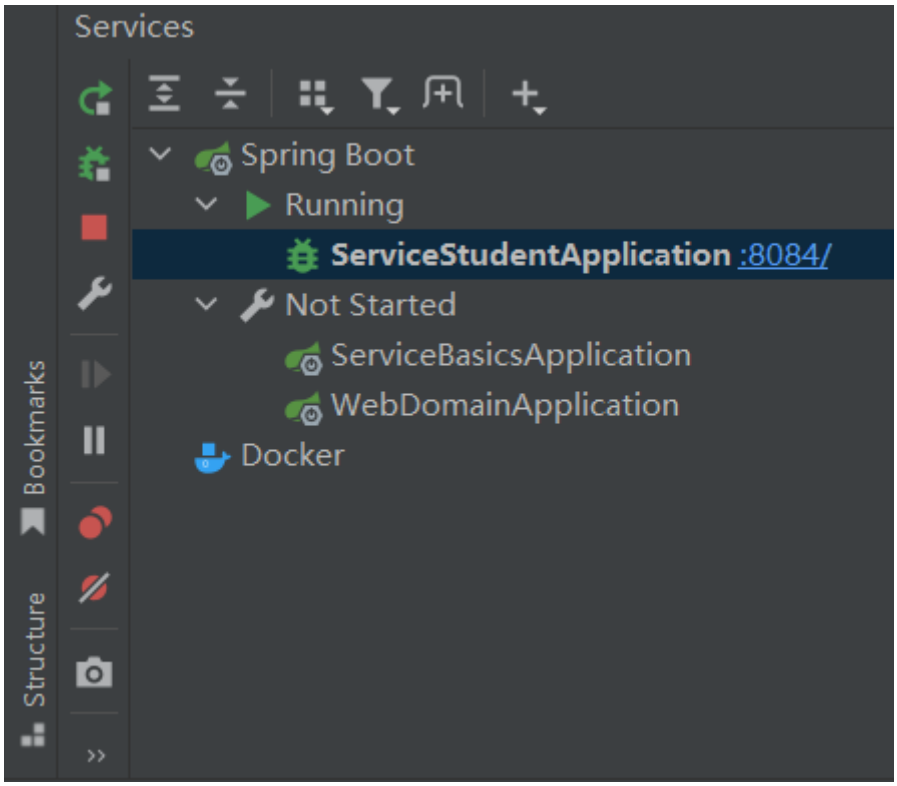
启动

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

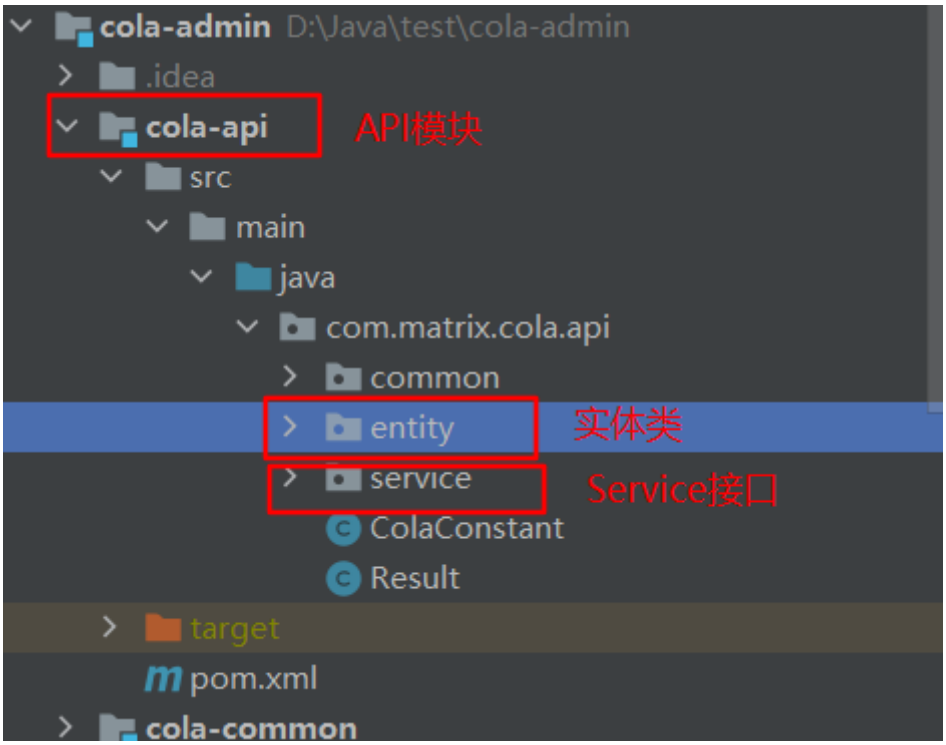
以学生管理为例，先创建一个学生表student。

```
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`)
);
```

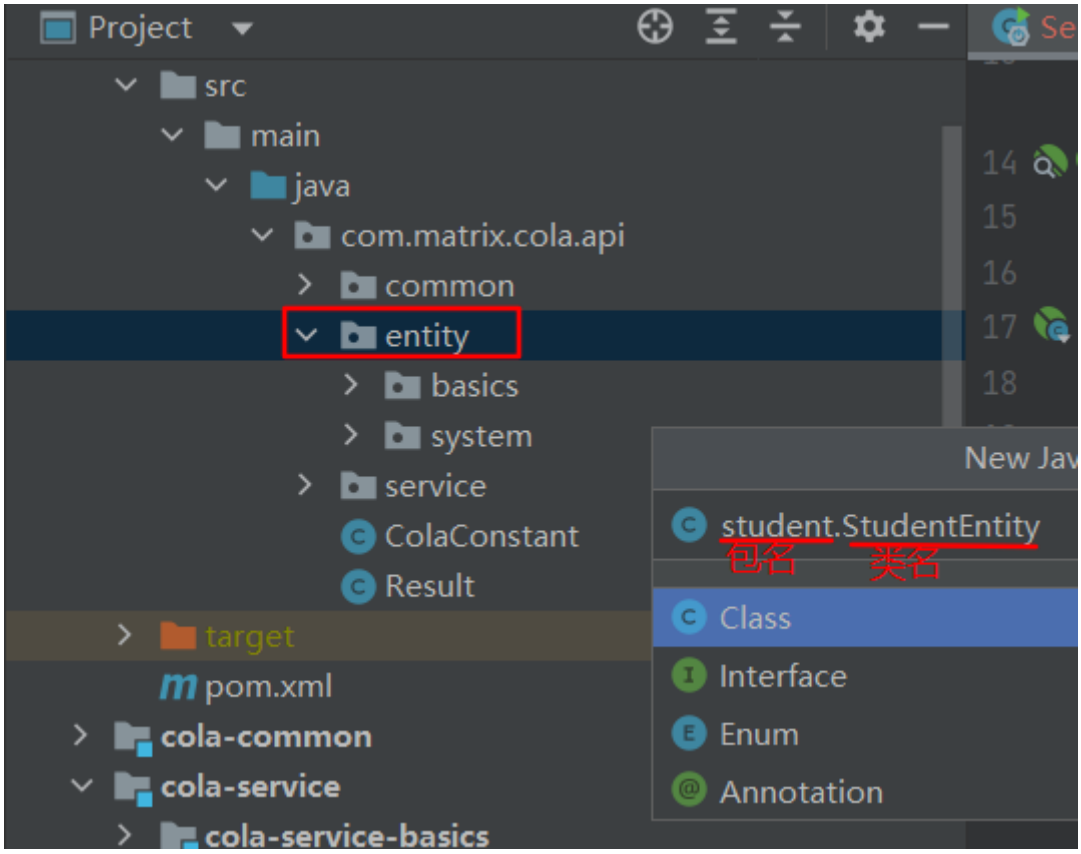
表中的“creator”、“create_time”、“reviser”、“revise_time”、“deleted”、“group_id”这六个字段是固定的业务字段，建议每个业务表都加上。表的字段名单词之间用下划线连接，这样就可以在实体类中直接使用驼峰方式命名，如字段“create_time”在实体类中的属性名就是“createTime”。

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

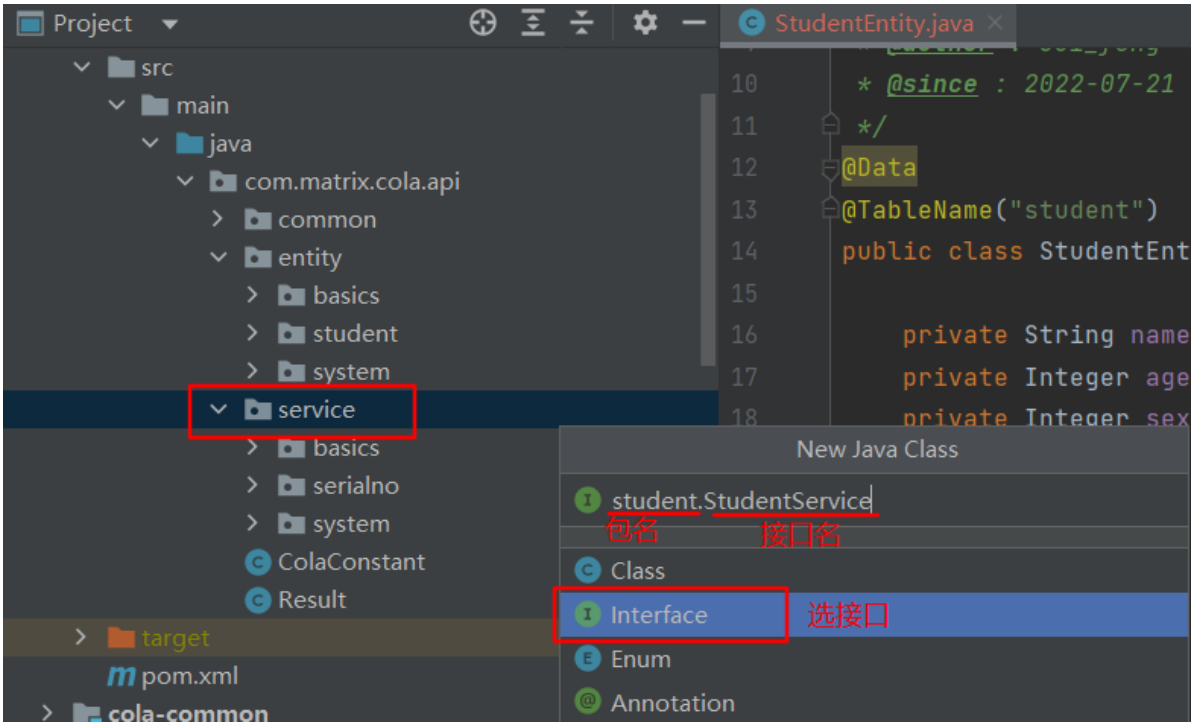
```java
@Data
@TableName("student")
public class StudentEntity extends BaseColaEntity {

    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。

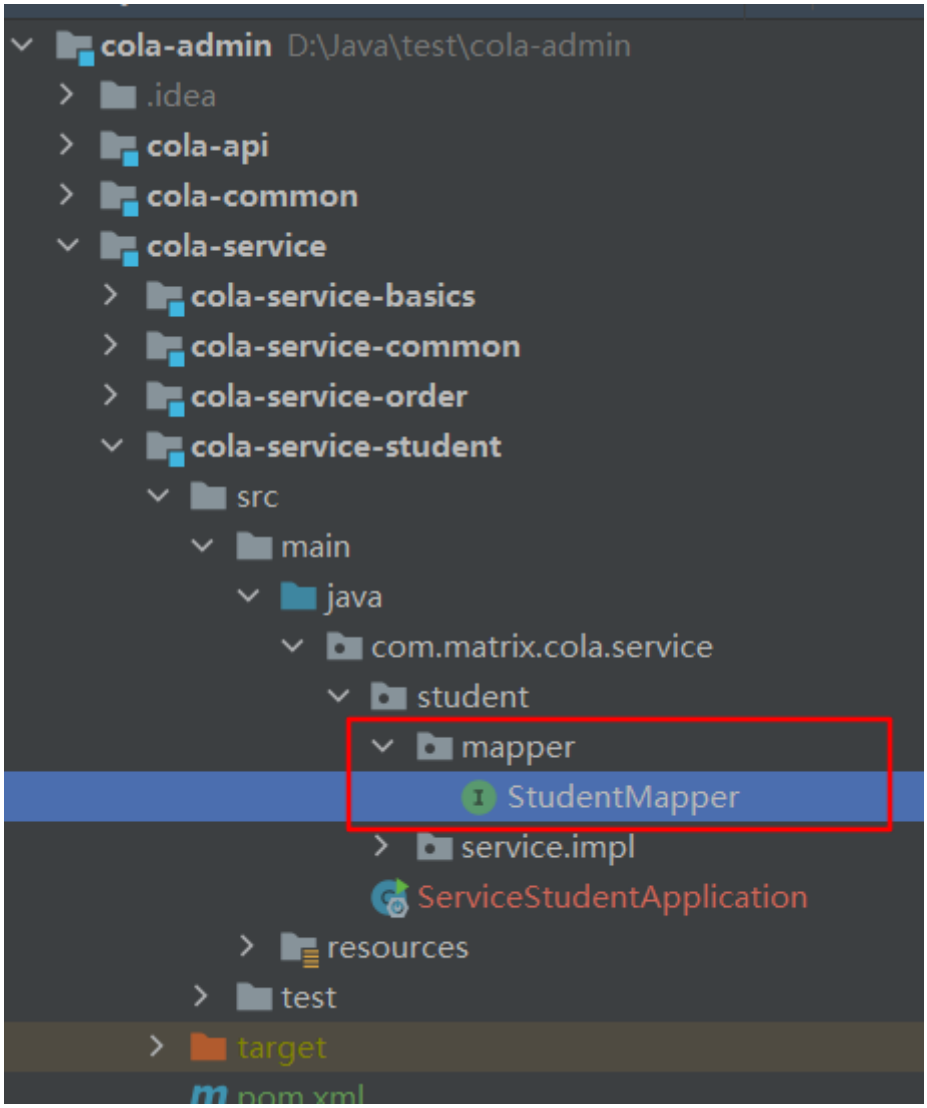
```java
/**
 * 学生管理接口
 * @author : cui_feng
 * @since : 2022-07-21 13:14
 */
public interface StudentService extends BaseColaEntityService<StudentEntity> {
}
```

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

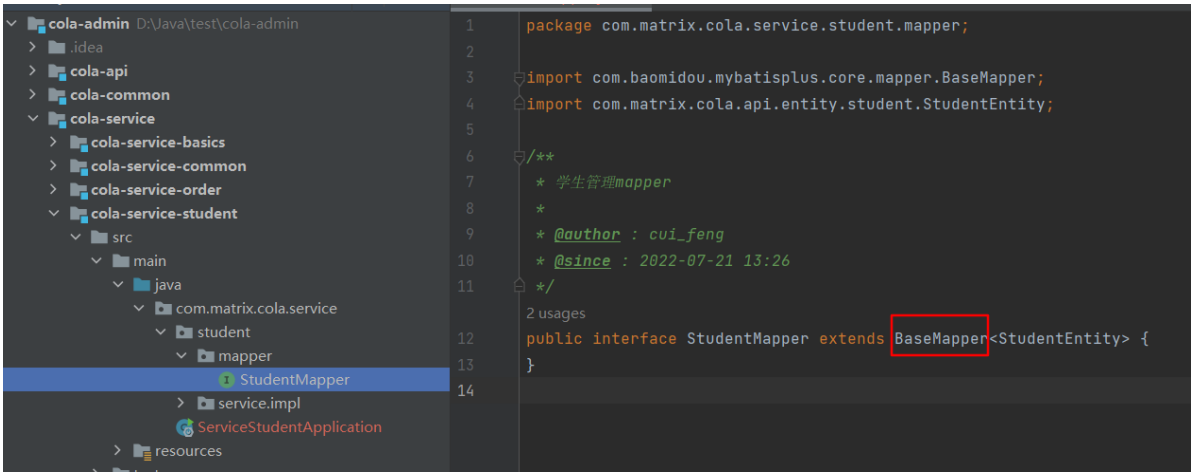
回到cola-service-student模块中，新建mapper接口StudentMapper



> mapper接口必须在mapper包下，并以xxxxMapper命名

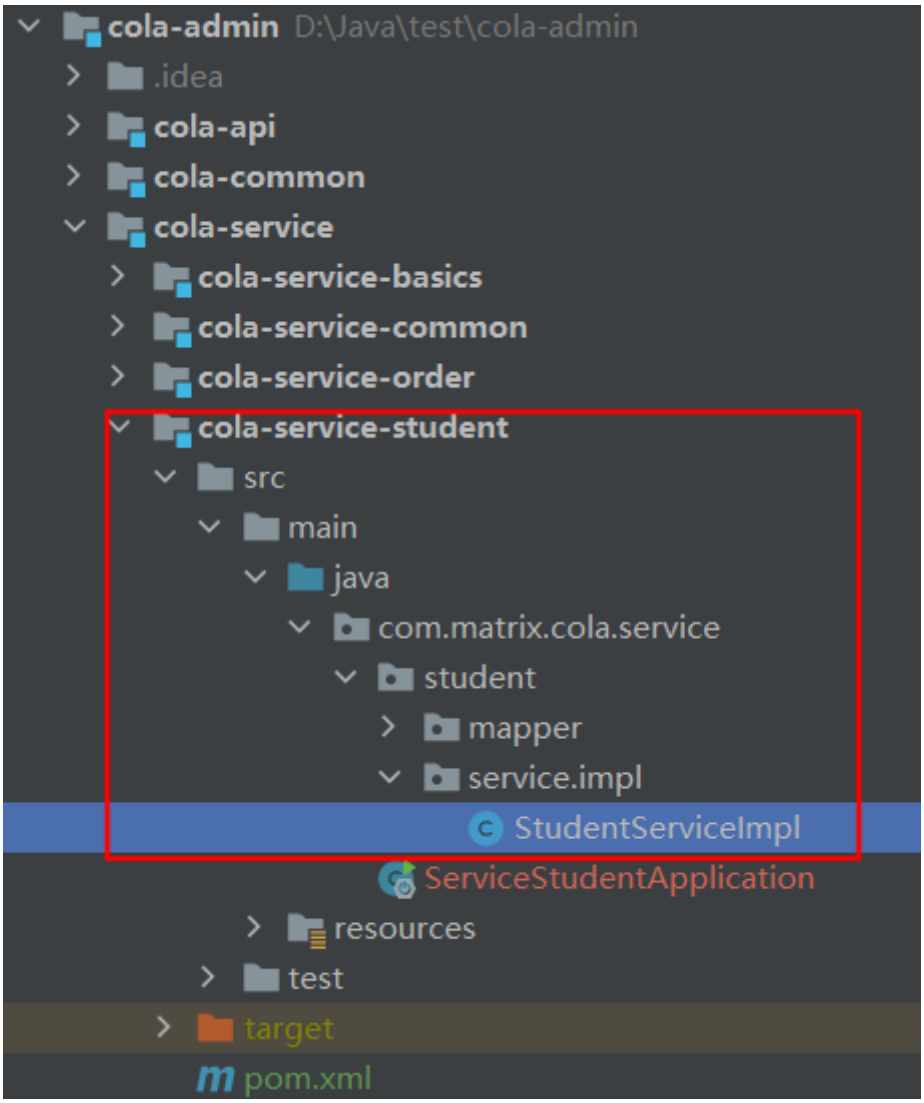
继承BaseMapper

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



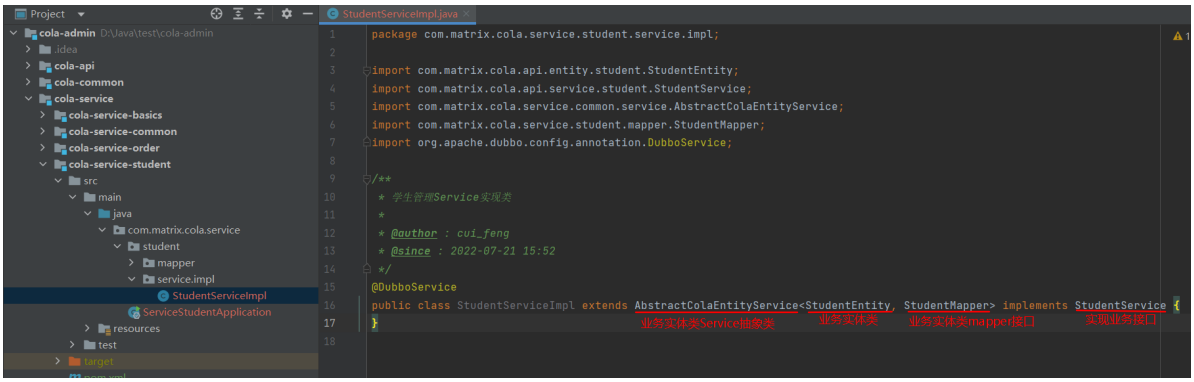
创建Service实现类

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

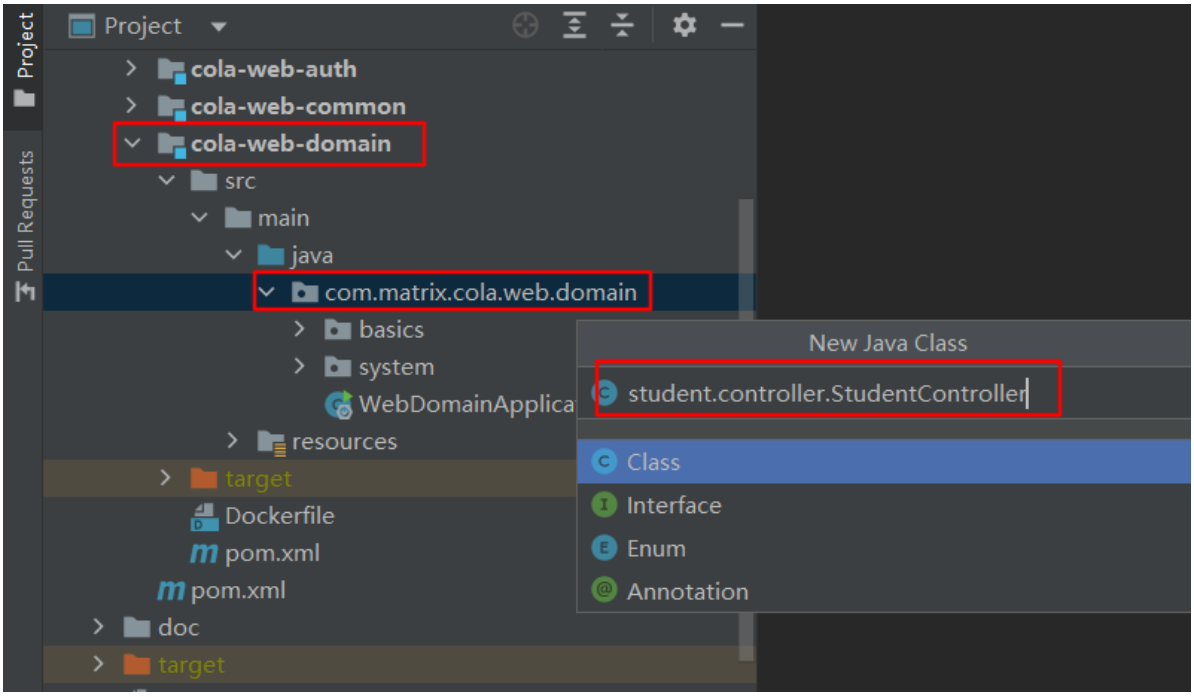
在StudentServiceImpl类上添加@DubboService注解

```
/**
 * 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 {
}
```

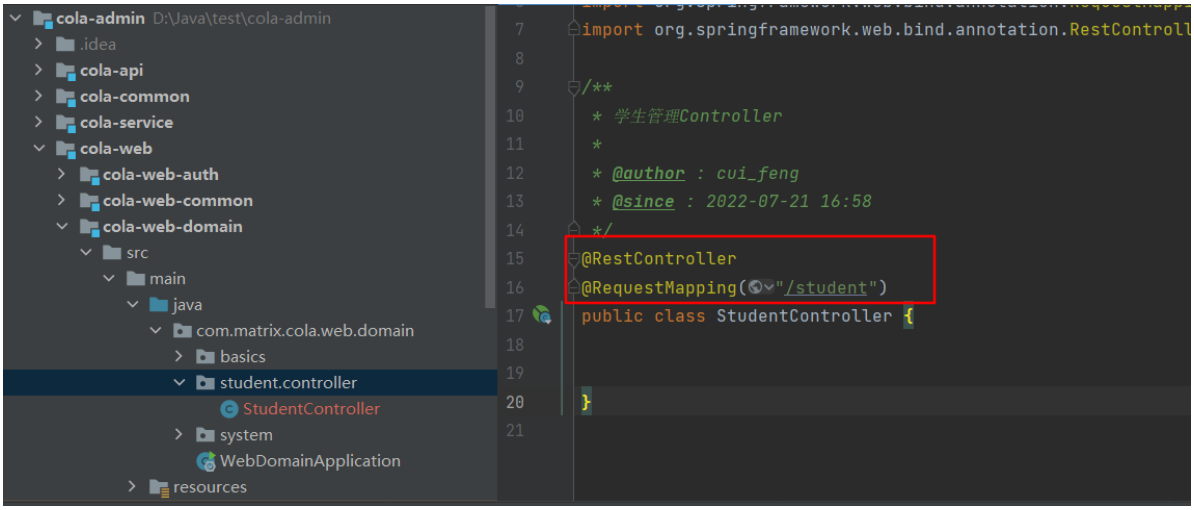
这样就完成了学生管理Service的全部功能（包括增、删、查、改、分页等功能），是不是很简单。

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

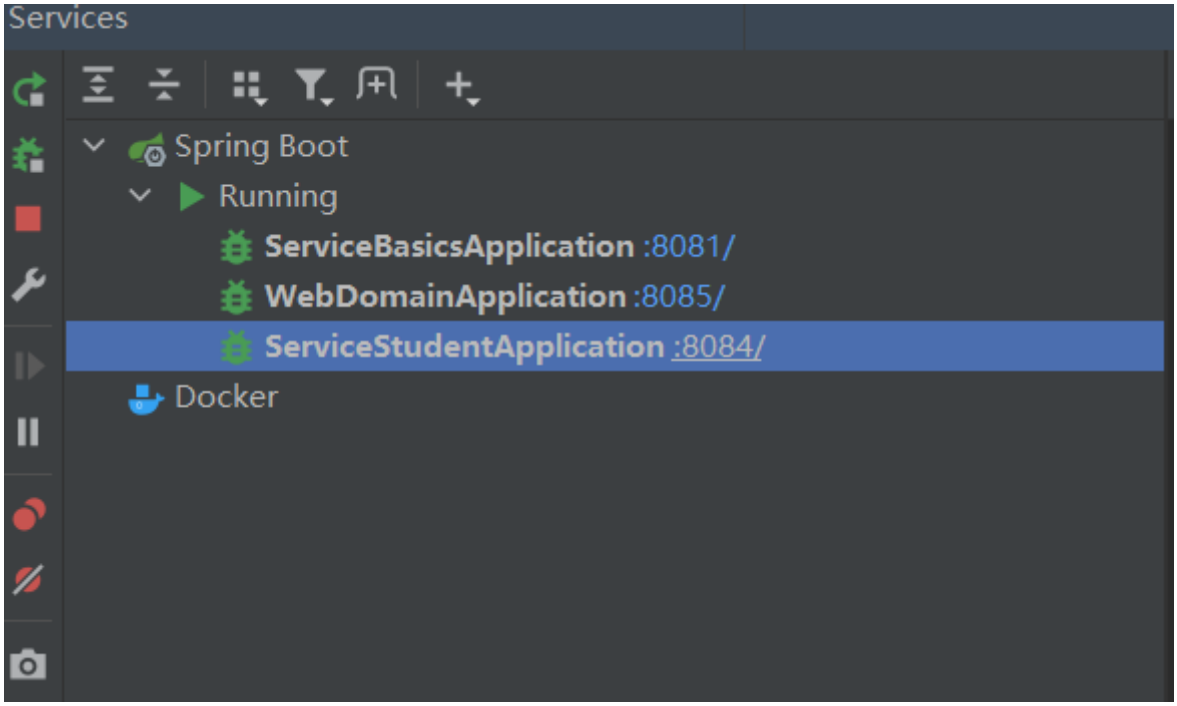
```java
package com.matrix.cola.web.domain.student.controller;

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。

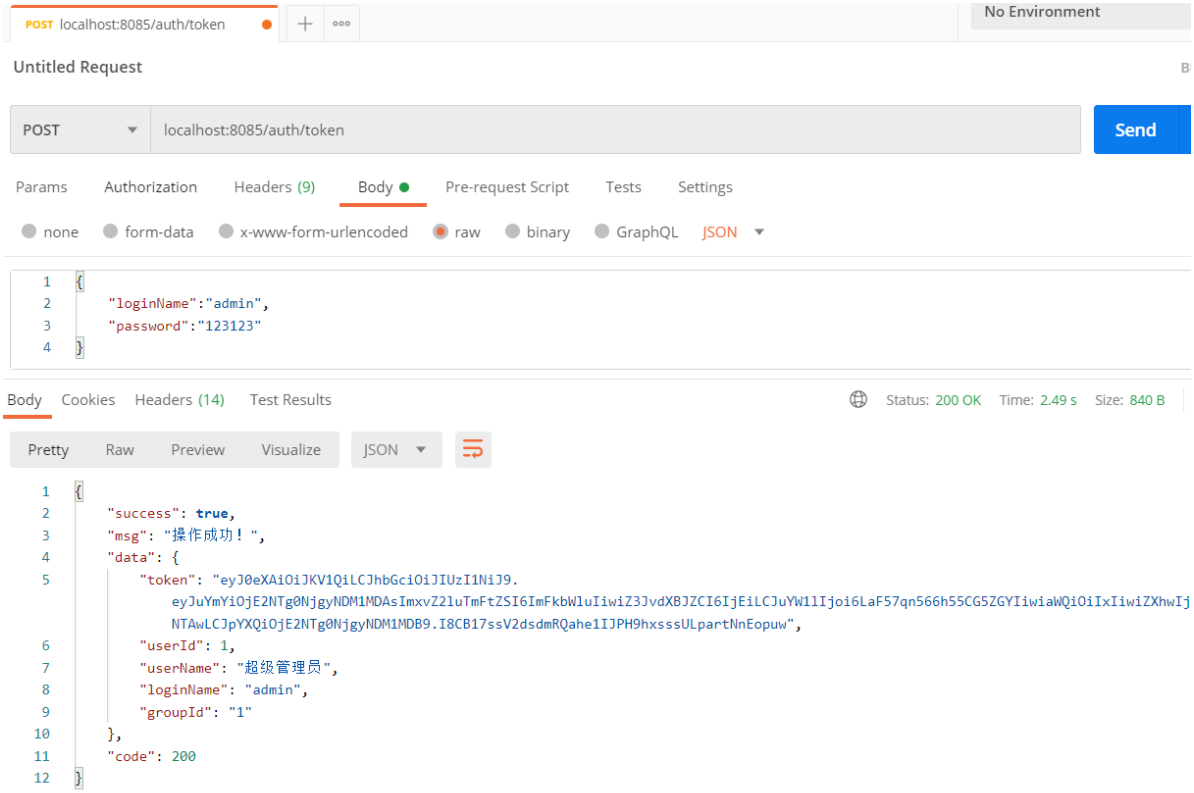
接口测试

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



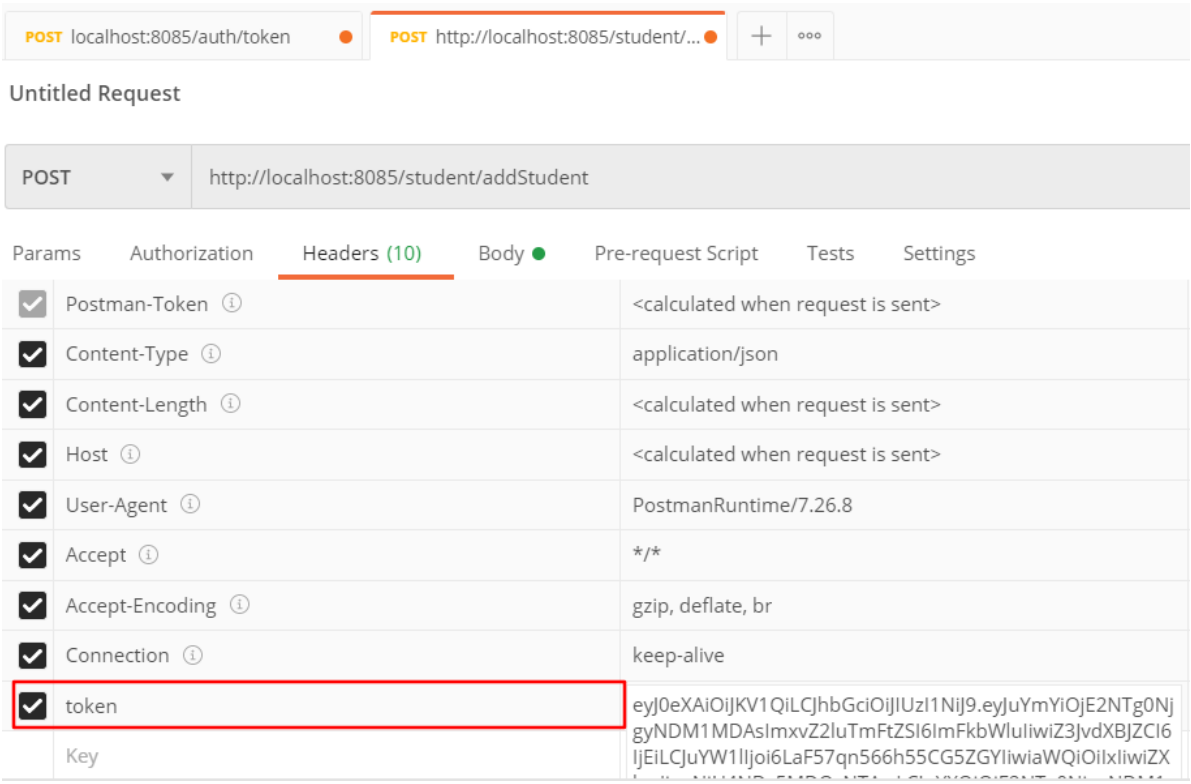
复制获取到的token值，后面的请求都需要用到。

新增学生

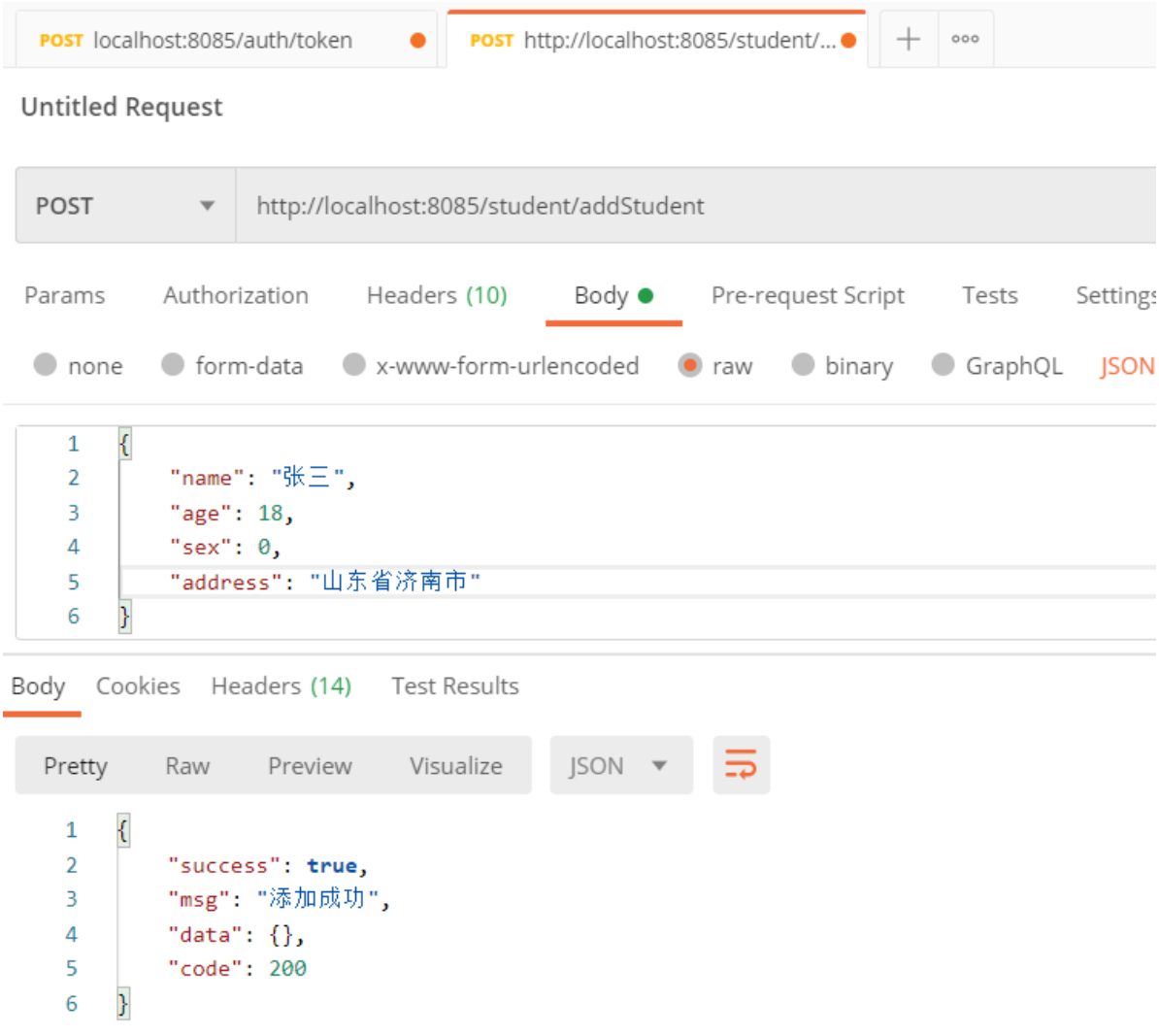
在Postman中新建一个标签，输入http://localhost:8085/student/addStudent ，选择Post方式提交，输入下面的参数：

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

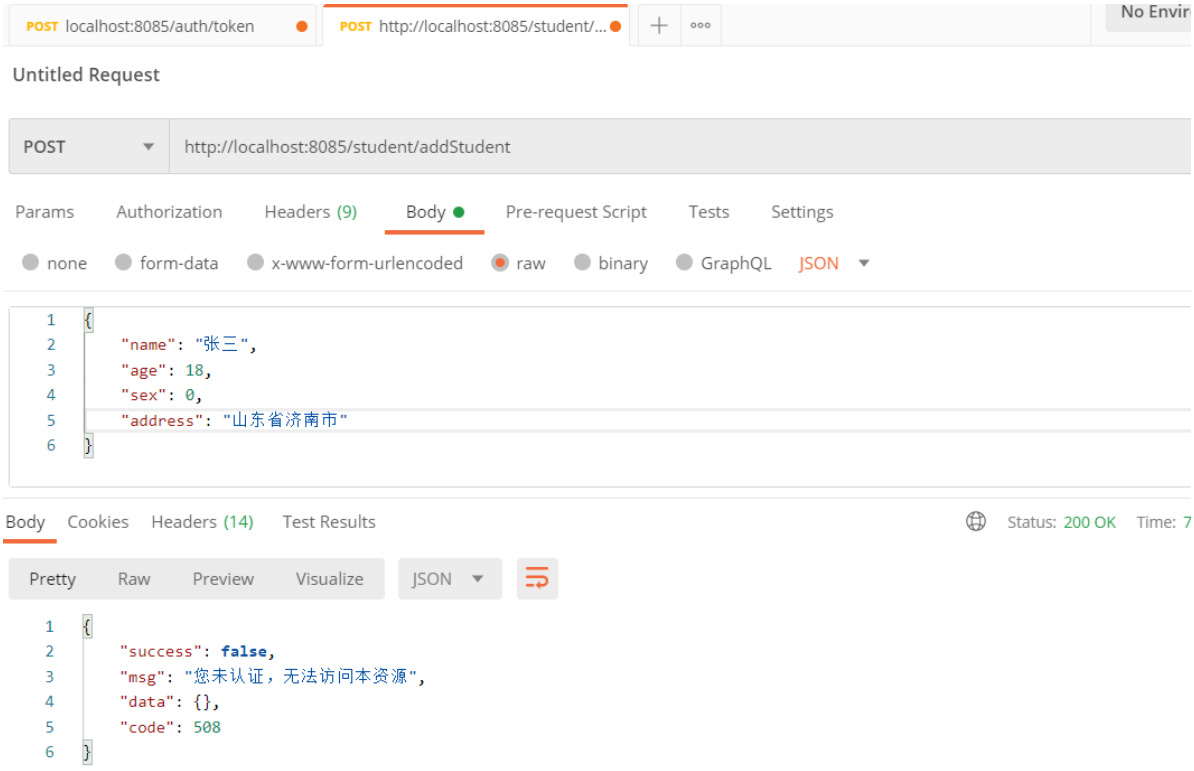
在Headers中添加参数token并粘贴上刚才复制的token值，如下图



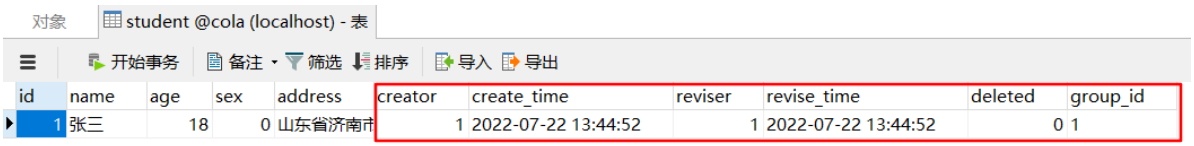
点击send提交，提示保存成功



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



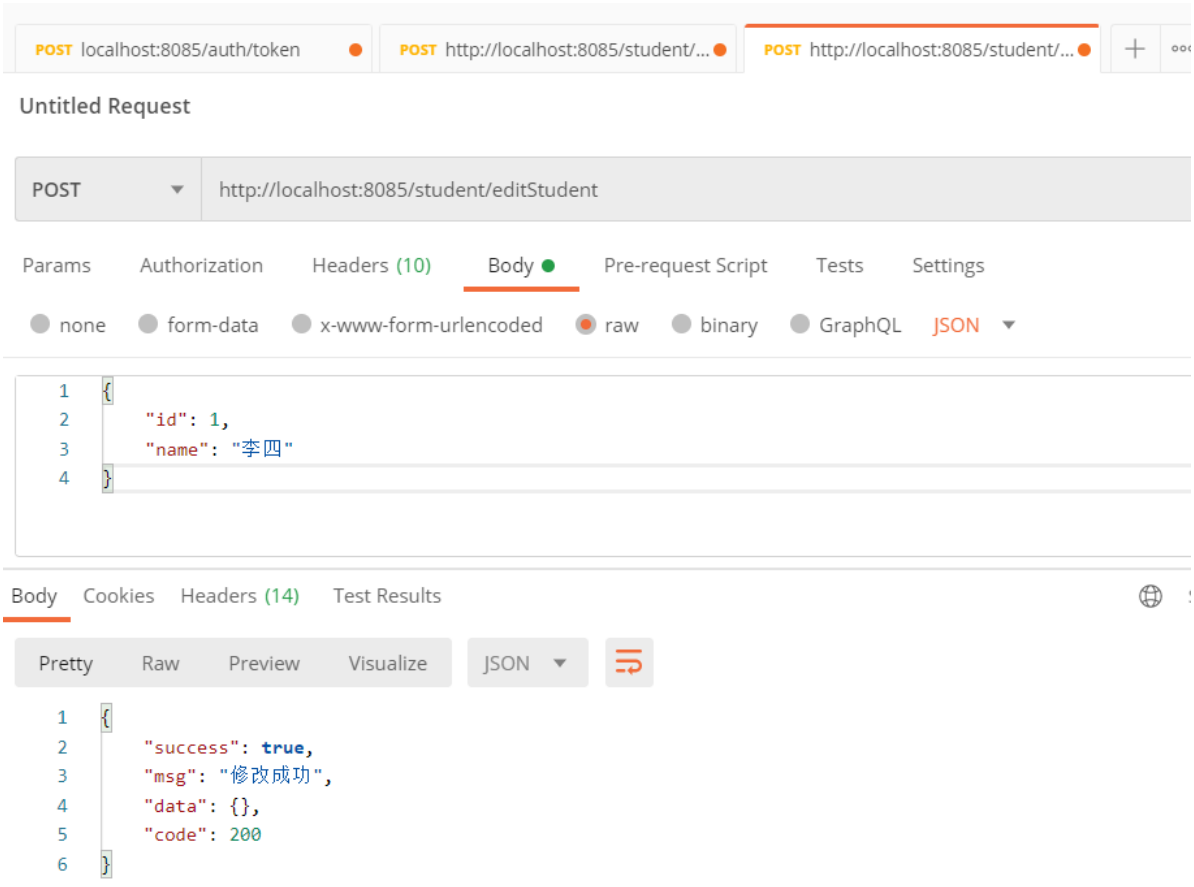
我们可以看到，最后的六个业务字段已经被填充了数据，这就是在AbstractColaEntityService中实现的。

修改学生

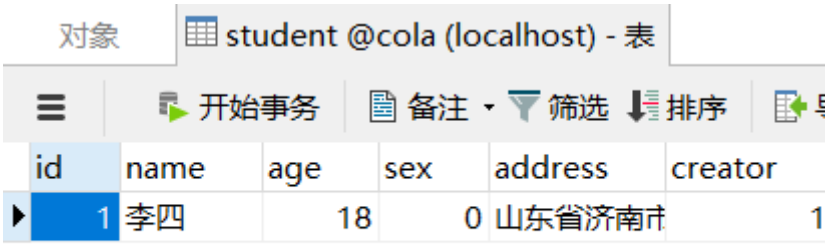
在Postman中再新建一个标签，输入http://localhost:8085/student/editStudent，选择Post方式提交，输入下面的参数：

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

在Header中添加token，点击send



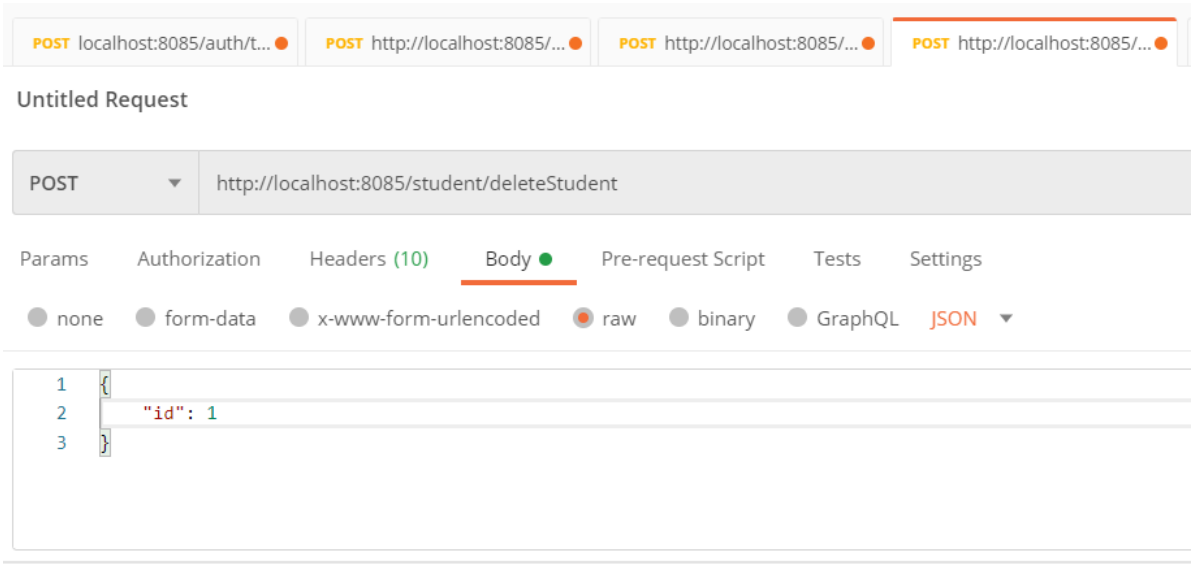
数据修改成功

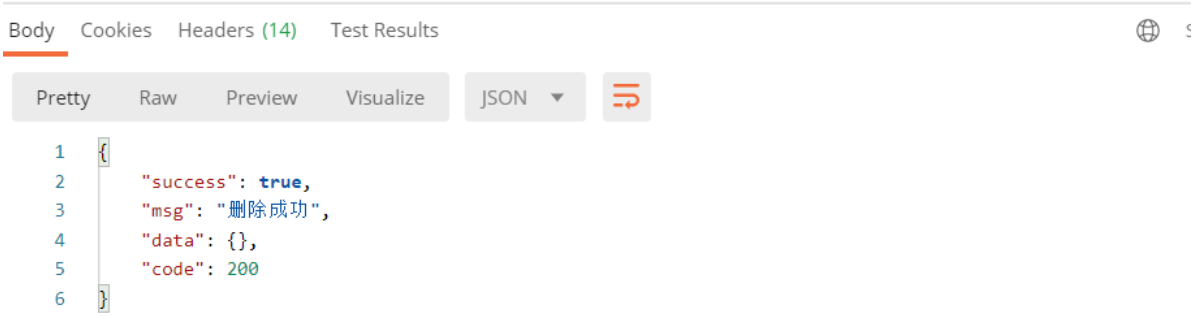


删除学生

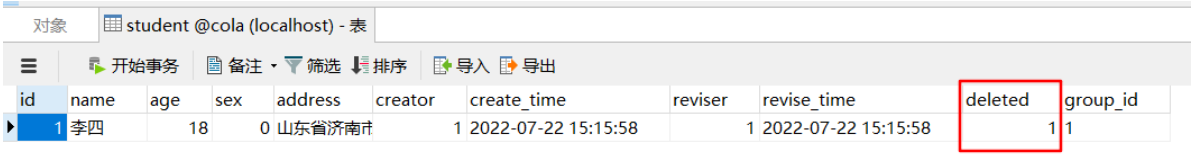
在Postman中新建一个标签页面，输入http://localhost:8085/student/deleteStudent，选择Post方式提交，输入下面的参数：

```
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
}
```

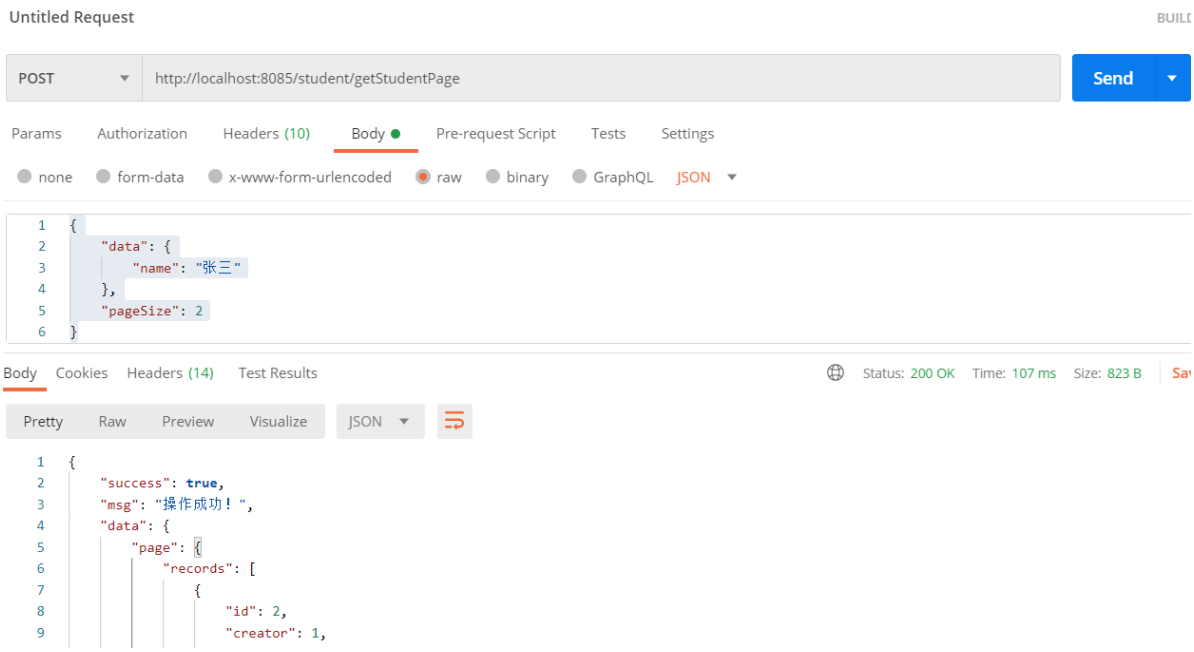

带条件的查询

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

@Autowired和@DubboReference

QueryWrapper

Query对象

CRUD生命周期

添加过程的生命周期

修改过程的生命周期

删除过程的生命周期

查询过程的生命周期

EntityWrapper

EntityWrapperService

Dubbo数据透传

数据日志

工具类说明

DubboUtil

WebUtil

CacheProxy

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