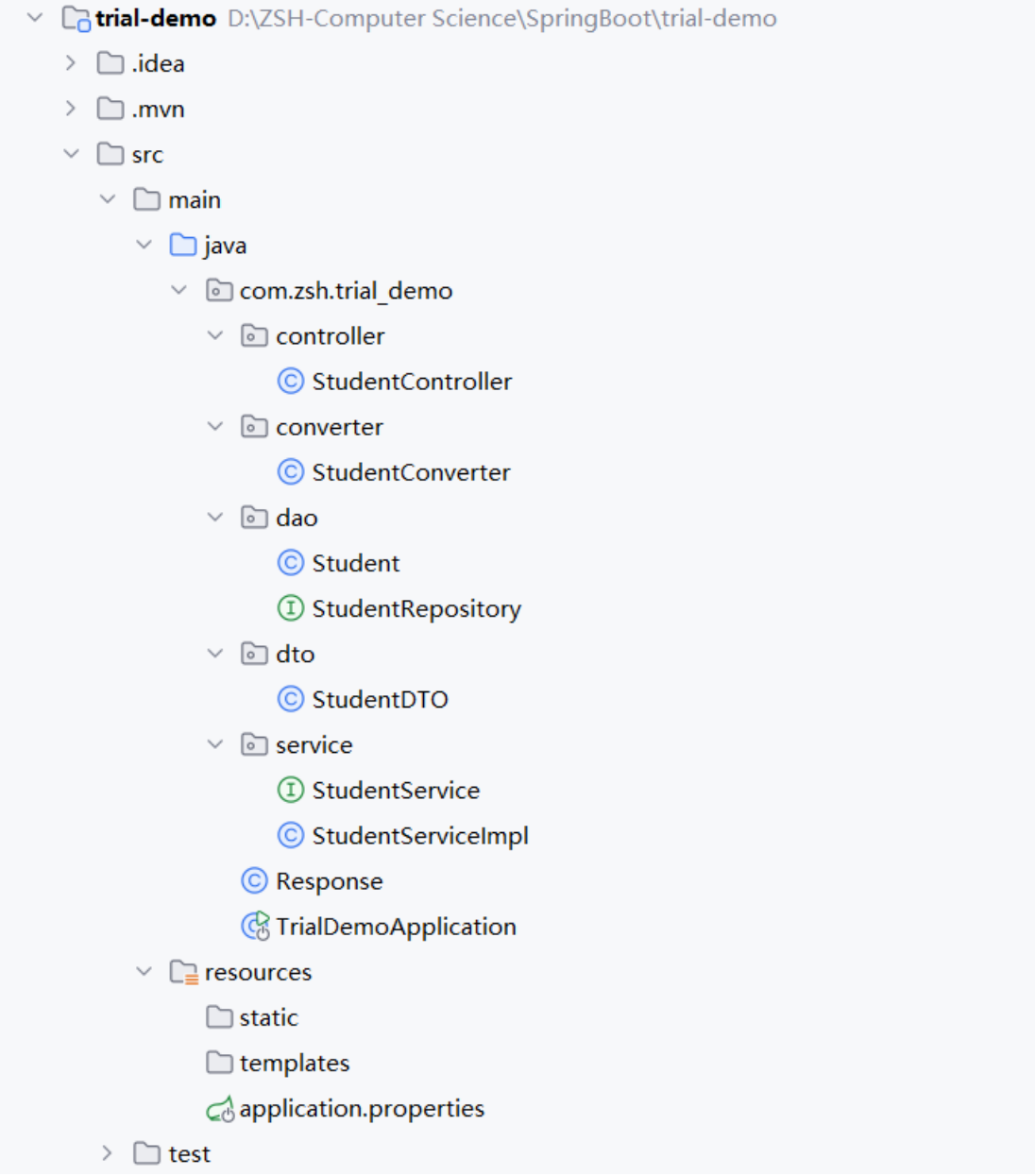
# 零基础SpringBoot小项目

> 该文章是本人观看B站up主“程序员风筑”于2024-04-23发布的“一小时带你从0到1实现一个SpringBoot项目开发”后，自行总结的笔记，视频链接如下：

https://www.bilibili.com/video/BV1gm411m7i6/?spm_id_from=333.337.search-card.all.click&vd_source=9837edfda9265147596b239f82083a53

## 项目结构分析

### 一、项目总览




1.首先新建一个SpringBoot-Maven工程，工程名称为trial-demo

2.我们重点研究src-main下的结构

3.main下面有两个包，一个是java包，用来存放我们的核心代码；另一个是resource包，用于存放我们的资源和配置文件

## 二、`resource`结构分析

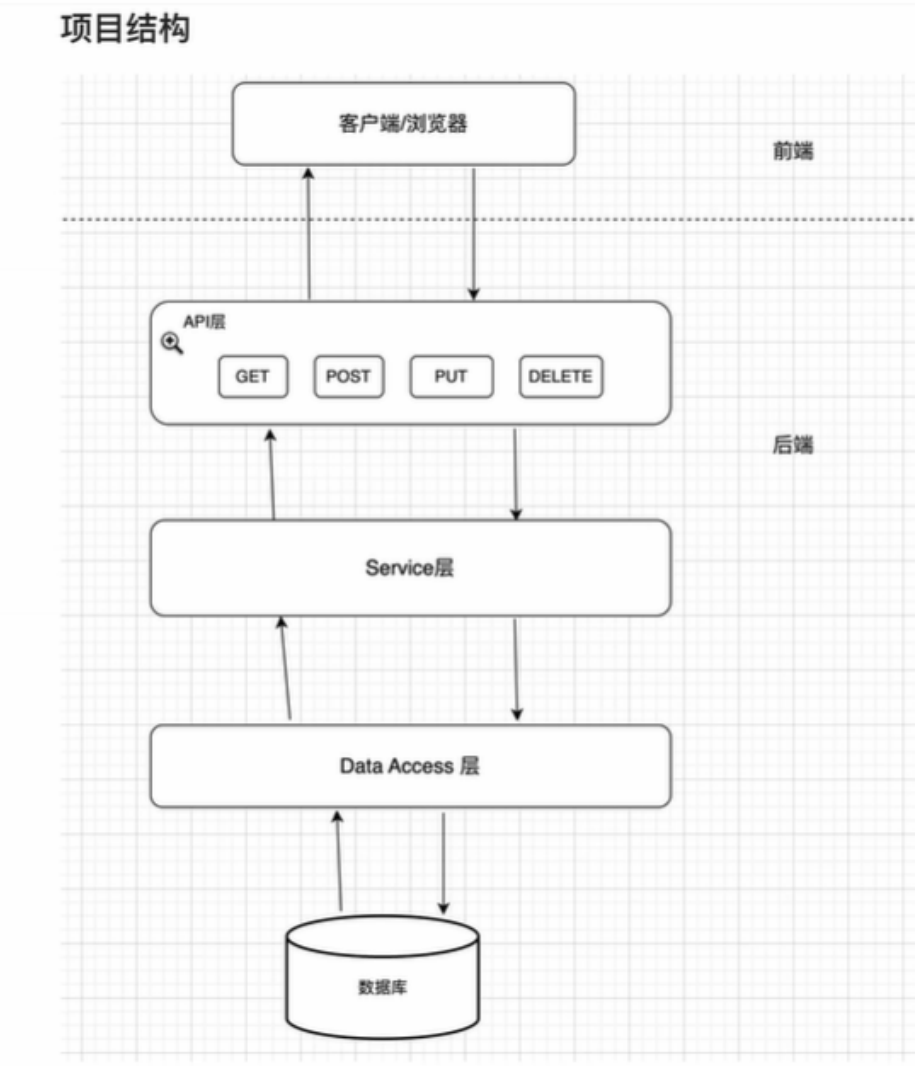
`application.properties`里面配置了连接`MySQL`数据库的文件，如下：

```
spring.application.name=trial-demo

spring.datasource.url=jdbc:mysql://localhost:3306/test?charactetEncoding=utf-8
spring.datasource.username=root
spring.datasource.password=123456
```

## 三、`java-com.zsh.trial_demo`结构分析

**前后端交互时序图展示**




- `dao(mapper)`：持久层，用于将数据持久化，也就是向**数据库**中永久性地写入数据保存起来。
  - 持久层框架比如`Mybatis`可以简化持久层的开发，让我们专注于写`SQL`语句。
- `service`：逻辑层，用于实现业务数据的逻辑处理，可以向**持久层**要数据（无需关心数据库是什么类型），也可以向**接口层**提供经过逻辑处理后的数据。
- `controller(API)`：接口层，用于跟外部做交互，对外提供项目入口。

### 1. dao包

- Student类

```java
package com.zsh.trial_demo.dao;

import jakarta.persistence.*;

@Entity//实体标注
@Table(name="student")//与数据库中相应的"student"表建立映射关系
public class Student {

    @Id//主键标注
    @GeneratedValue(strategy = GenerationType.IDENTITY)//自增
    @Column(name="id")//与数据库中相应表的字段名建立映射关系
    private long id;

    @Column(name="name")//与数据库中相应表的字段名建立映射关系
    private String name;

    @Column(name="email")//与数据库中相应表的字段名建立映射关系
    private String email;

    @Column(name="age")//与数据库中相应表的字段名建立映射关系
    private int age;

    /*以下都是成员变量的get&set方法*/
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
}
```

- `StudentRepository`接口

```java
package com.zsh.trial_demo.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository//用于标记DAO组件，可以让Spring自动检测和管理这些DAO组件
public interface StudentRepository extends JpaRepository<Student,Long> {
    /*
        JpaRepository<T,ID>是一个泛型接口，其中T代表泛型。
        该接口拥有基本的CRUD（增删改查）功能。
    */
    //自定义了一个方法，通过邮箱查找拥有该邮箱的学生List集合
    //只要自定义格式符合以下规范，那么java会为我们自动生成对应的方法，无需我们手写SQL语句
    List<Student> findByEmail(String email);
}
```

---

### 2.`dto`包

- `DTO(Data Transefer Object)`：数据传输对象
  - 作用：在不同层之间传输数据
  - `DTO`类型的对象，不应该掺杂任何业务逻辑，只包含成员变量和相应的`get&set`方法
  - 通常在`Service`层使用，在`Service`层与`Controller`层之间传输数据
- `StudentDTO`类

```java
package com.zsh.trial_demo.dto;

public class StudentDTO {

    /*成员变量*/
    private long id;
    private String name;
    private String email;

    /*以下都是成员变量的get&set方法*/
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
```

```
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

### 3. `converter`包

> 用于DAO和DTO之间的相互转换

```
package com.zsh.trial_demo.converter;

import com.zsh.trial_demo.dao.Student;
import com.zsh.trial_demo.dto.StudentDTO;

public class StudentConverter {

    //将DAO从数据库中拿到的student对象，
    //转换成我们想要返回给前端的studentDTO对象（序列化）
    public static StudentDTO converterStudent(Student student){
        StudentDTO studentDTO=new StudentDTO();
        studentDTO.setId(student.getId());
        studentDTO.setName(student.getName());
        studentDTO.setEmail(student.getEmail());
        return studentDTO;
    }

    //将前端拿到的studentDTO对象，
    //转换成我们想要写入数据库的student对象（反序列化）
    public static Student converterStudent(StudentDTO studentDTO){
        Student student=new Student();
        student.setId(studentDTO.getId());
        student.setName(studentDTO.getName());
        student.setEmail(studentDTO.getEmail());
        return student;
    }
}
```

---

### 4. `Response`类

> 项目设计应遵守的规范，方便统一处理

```
package com.zsh.trial_demo;

public class Response <T>{//Response是一个泛型类

    /*成员变量*/
```

```
    private T data;//T代表泛型，所以data中存放的是泛型数据对象，比如Student对象
    private boolean success;//指示前端发送的请求是否成功
    private String errorMsg;//记录错误信息

    /*封装请求成功时返回的响应*/
    public static <K> Response<K> newSuccess(K data){
        //首先new一个Response对象出来
        Response<K> response=new Response<>();

        //再来设置这个Response对象的各种属性
        response.setData(data);
        response.setSuccess(true);

        //最后返回这个Response对象
        return response;
    }

    /*封装请求失败时返回的响应*/
    public static Response<Void> newFail(String errorMsg){
        //首先new一个Response对象出来
        Response<Void> response=new Response<>();

        //再来设置这个Response对象的各种属性
        response.setSuccess(false);
        response.setErrorMsg(errorMsg);

        //最后返回这个Response对象
        return response;
    }

    /*以下都是成员变量的get&set方法*/
    public T getData() {
        return data;
    }
    public void setData(T data) {
        this.data = data;
    }
    public boolean isSuccess() {
        return success;
    }
    public void setSuccess(boolean success) {
        this.success = success;
    }
    public String getErrorMsg() {
        return errorMsg;
    }
    public void setErrorMsg(String errorMsg) {
        this.errorMsg = errorMsg;
    }
}
```

### 5. service包

- `StudentService`接口

```java
package com.zsh.trial_demo.service;

import com.zsh.trial_demo.dao.Student;
import com.zsh.trial_demo.dto.StudentDTO;

public interface StudentService {
    /*在接口中预先定义好准备实现的各种方法，如下：*/
    //查
    StudentDTO getStudentById(long id);
    //增
    Long addNewStudent(StudentDTO studentDTO);
    //删
    void deleteStudentById(long id);
    //改
    StudentDTO updateStudentById(long id, String name, String email);
}
```

- `StudentServiceImpl`接口实现类

```java
package com.zsh.trial_demo.service;

import com.zsh.trial_demo.converter.StudentConverter;
import com.zsh.trial_demo.dao.Student;
import com.zsh.trial_demo.dao.StudentRepository;
import com.zsh.trial_demo.dto.StudentDTO;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.util.CollectionUtils;
import org.springframework.util.StringUtils;

import java.beans.Transient;
import java.util.List;

@Service//逻辑层标注
public class StudentServiceImpl implements StudentService{

    @Autowired//自动注入
    private StudentRepository studentRepository;

    /*以下是接口中预定义的方法的具体实现*/
    //查
    @Override
    public StudentDTO getStudentById(long id) {
        //先检查该id是否存在，若不存在则抛出RuntimeException异常
        //findById是studentRepository继承自父类的方法，我们直接调用即可
        Student student =
studentRepository.findById(id).orElseThrow(RuntimeException::new);

```

```
        //返回将student序列化后的studentDTO对象
        return StudentConverter.converterStudent(student);
    }

    //增
    @Override
    public Long addNewStudent(StudentDTO studentDTO) {
        //先检查新增学生的email是否已被他人占用
        //此处调用的findByEmail方法是一个自定义方法，但由于我们自定义格式符合规范，故无需手写SQL语句，java会帮我们自动生成该            方法
        List<Student> studentList=studentRepository.findByEmail(studentDTO.getEmail());
        if(!CollectionUtils.isEmpty(studentList)){
            throw new IllegalStateException("email:"+studentDTO.getEmail()+" has been taken.");
        }

        //将新增的studentDTO对象序列化为student对象，再保存进数据库
        //save是studentRepository继承自父类的方法，我们直接调用即可
        Student student=studentRepository.save(StudentConverter.converterStudent(studentDTO));

        //返回序列化后的student对象的id
        return student.getId();
    }

    //删
    @Override
    public void deleteStudentById(long id) {
        //先检查该id是否存在，若不存在则抛出异常
        studentRepository.findById(id).orElseThrow(() ->
                new IllegalArgumentException("id:"+id+" doesn't exist!"));

        //deleteById是studentRepository继承自父类的方法，我们直接调用即可
        studentRepository.deleteById(id);
    }

    //改
    @Transactional//若更新失败，则进行事务回滚
    @Override
    public StudentDTO updateStudentById(long id, String name, String email) {
        //先检查该id是否存在，若不存在则抛出异常
        Student studentInDB=studentRepository.findById(id).orElseThrow(() ->
                new IllegalArgumentException("id:"+id+" doesn't exist!"));

        //如果传进来的name不为空且与数据库中的name不相同，我们才更新数据库中的name
        if(StringUtils.hasLength(name) && !studentInDB.getName().equals(name)){
            studentInDB.setName(name);
        }

        //如果传进来的email不为空且与数据库中的email不相同，我们才更新数据库中的email
```

```
        if(StringUtils.hasLength(email) &&
!studentInDB.getEmail().equals(email)){
            studentInDB.setEmail(email);
        }
        //将修改后的student对象保存进数据库
        //save是studentRepository继承自父类的方法，我们直接调用即可
        Student student=studentRepository.save(studentInDB);

        //返回将student反序列化后的studentDTO对象
        return StudentConverter.converterStudent(student);
    }
}
```

---

### 6. `controller` 包

易错！！！ `public Response<Long> addNewStudent(@RequestBody StudentDTO studentDTO)`

注意此处必须要有 `RequestBody` 标注，否则前后端无法交互，导致 `add` 接口失效

- `StudentController` 类

```
package com.zsh.trial_demo.controller;

import com.zsh.trial_demo.Response;
import com.zsh.trial_demo.dao.Student;
import com.zsh.trial_demo.dto.StudentDTO;
import com.zsh.trial_demo.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController//接口层标注
public class StudentController {

    @Autowired//自动注入
    private StudentService studentService;

    /*以下是增删改查4个接口*/
    //查
    @GetMapping("/student/{id}")//GET方法标注
    public Response<StudentDTO> getStudentById(@PathVariable long id){//路径
变量标注，指代{id}
        //调用Service层中的相关方法处理请求，并将响应结果封装成一个响应对象返回
        return Response.newSuccess(studentService.getStudentById(id));
    }

    //增
    @PostMapping("/student")//POST方法标注
    public Response<Long> addNewStudent(@RequestBody StudentDTO studentDTO)
{//RequestBody标注
        //调用Service层中的相关方法处理请求，并将响应结果封装成一个响应对象返回
```

```
        return
Response.newSuccess(studentService.addNewStudent(studentDTO));
    }

    //删
    @DeleteMapping("/student/{id}")//DELETE方法标注
    public void deleteStudentById(@PathVariable long id){//路径变量标注，指代
{id}
        //调用Service层中的相关方法处理请求，并将响应结果封装成一个响应对象返回
        studentService.deleteStudentById(id);
    }

    //改
    @PutMapping("/student/{id}")//PUT方法标注
    public Response<StudentDTO> updateStudentById
            (@PathVariable long id,//路径变量标注，指代{id}
             @RequestParam(required = false)String name,//请求参数name标注
             @RequestParam(required = false)String email){//请求参数email标注
        //调用Service层中的相关方法处理请求，并将响应结果封装成一个响应对象返回
        return
Response.newSuccess(studentService.updateStudentById(id,name,email));
    }
}
```

---

### 7. `TrialDemoApplication` 类

> 项目启动入口

```
package com.zsh.trial_demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TrialDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(TrialDemoApplication.class, args);
    }
}
```