登录功能后端接口

目录

目标任务 2

思路分析 2

1.后台接口开发思路分析 2

编码步骤 4

1.后端接口 Controller 页面 4

2.SXBUserServiceImpl 6

3.Dao 6

4.Model 8

源码 8

回马枪总结 9

目标任务

- 以接口的形式接收并判断前台数据，之后将结果返回给前台

思路分析

1.后台接口开发思路分析

If username && password 不为空

sxbUserService.get(username)

If 用户存在

If 密码正确

return {"code":"200","message":"成功","data":null};

else 密码错误

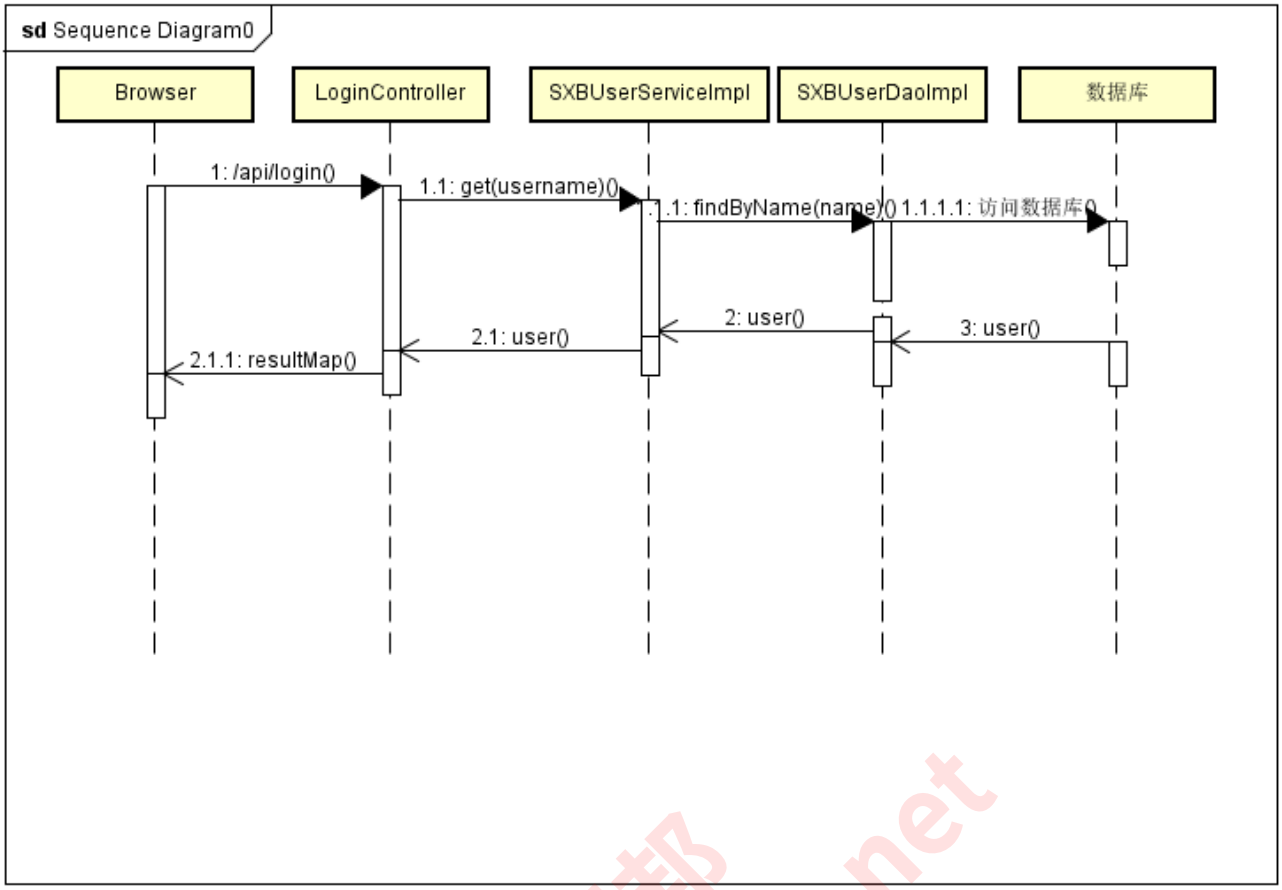
return {"code":"500201","message":"用户名或密码错误","data":null};

else 用户不存在

return {"code":"500201","message":"用户名或密码错误","data":null};

用户名或密码为空

return {"code":"500","message":"出错","data":null};;



url:/api/login

method: post

逻辑：根据 user 查找相对应 username 和 password，如果前后 username 和 password 相同，则跳转到课程页面

参数：

参数名	类型	是否必须	说明
username	String	Y	用户名
password	String	Y	密码

返回值：

参数名	类型	是否必须	说明
code	Int	Y	200 为成功 500***为失败

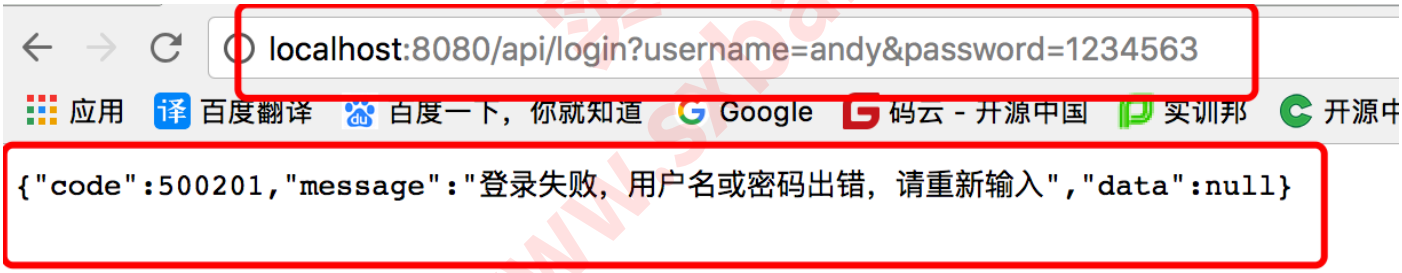
message	String	Y	成功或失败的消息内容
data	object	N	

编码步骤

1. 运行效果截图

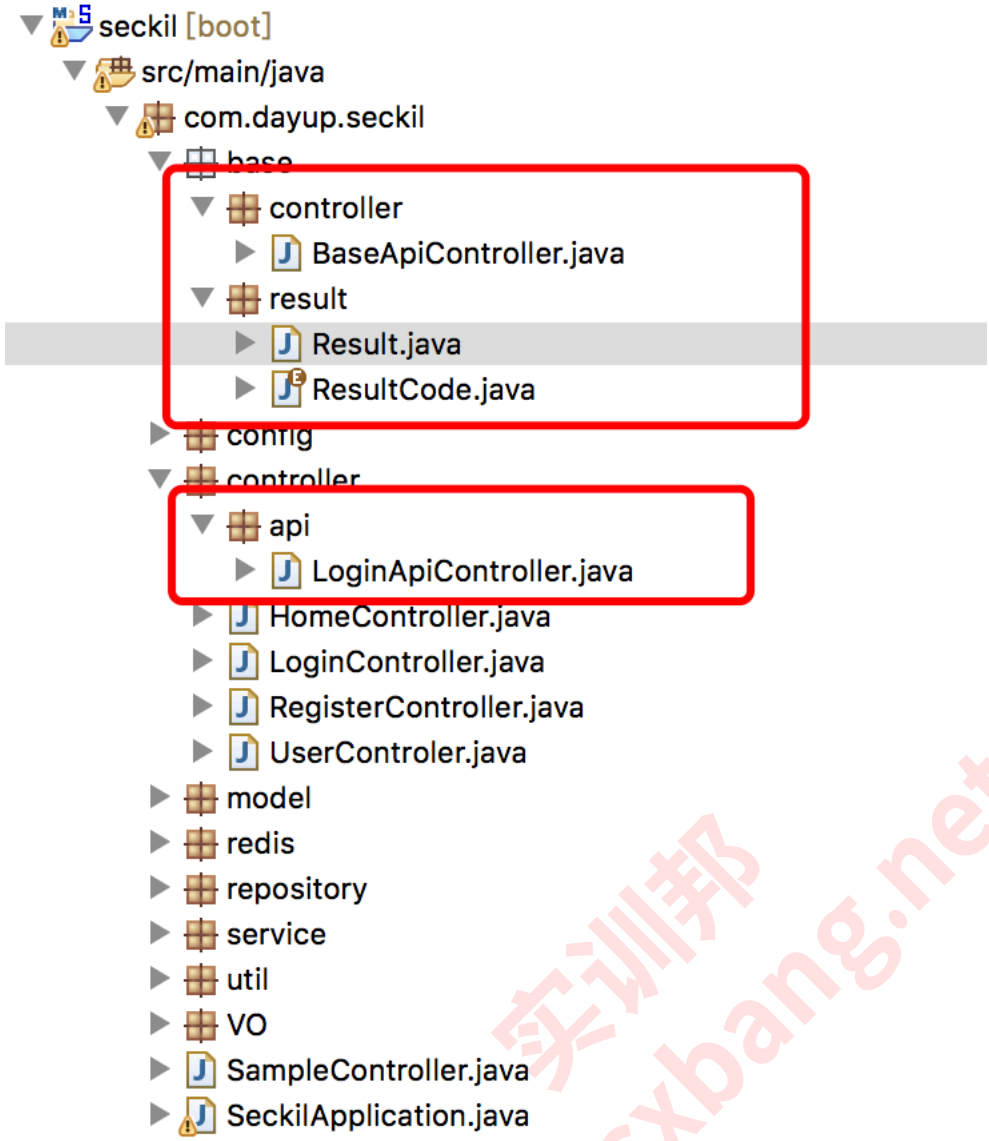
成功时

密码错时



用户名或密码为空

2. 文件目录结构



---------- 正题分隔线 ----------

3. BaseApiController.java

LoginApiController.java | BaseApiController.java | Result.java

```java
package com.dayup.seckil.base.controller;

import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestMapping;

@RequestMapping("/api")
@CrossOrigin(origins = "*", allowCredentials = "true")
public class BaseApiController {

}

```

- 新建 BaseApiController 类，定义所有 api 接口的 URL 前缀，并开放跨域访问权限
- @CrossOrigin 是用来处理跨域请求让你能访问不同域的文件的注解，controller 要以接口形式为前端调用，必须继承 BaseApiController

4. Result.java

LoginApiController.java | BaseApiController.java | Result.java

```java
package com.dayup.seckil.base.result;

public class Result<T> {

    private Integer code;

    private String message;

    private T data;

    public static <T> Result<T> failure() {
        Result<T> result = new Result<T>();
        result.setResultCode(ResultCode.FAIL);
        return result;
    }

    public static <T> Result<T> failure(T data) {
        Result<T> result = new Result<T>();
        result.setResultCode(ResultCode.FAIL);
        result.setData(data);
        return result;
    }

    public static <T> Result<T> failure(ResultCode resultCode) {
        Result<T> result = new Result<T>();
        result.setResultCode(resultCode);
        return result;
    }

    public static <T> Result<T> failure(ResultCode resultCode, T data) {
        Result<T> result = new Result<T>();
        result.setResultCode(resultCode);
        result.setData(data);
        return result;
    }

    public static <T> Result<T> success() {
        Result<T> result = new Result<T>();
        result.setResultCode(ResultCode.SUCCESS);
        return result;
    }

    public static <T> Result<T> success(T data) {
        Result<T> result = new Result<T>();
        result.setResultCode(ResultCode.SUCCESS);
        result.setData(data);
        return result;
```

```
49
50      public void setResultCode(ResultCode resultCode) {
51          this.code = resultCode.getCode();
52          this.message = resultCode.getMessage();
53      }
54
55      public Integer getCode() {
56          return code;
57      }
58
59      public void setCode(Integer code) {
60          this.code = code;
61      }
62
63      public String getMessage() {
64          return message;
65      }
66
67      public void setMessage(String message) {
68          this.message = message;
69      }
70
71      public T getData() {
72          return data;
73      }
74
75      public void setData(T data) {
76          this.data = data;
77      }
78
79  }
80
```

- 静态方法可以帮助我们快速为 javaBean 赋值，得到我们想要的格式结果。
- 这个 javaBean 转成 json 将会是{"code":"500201","message":"用户名或密码错误","data": ...}

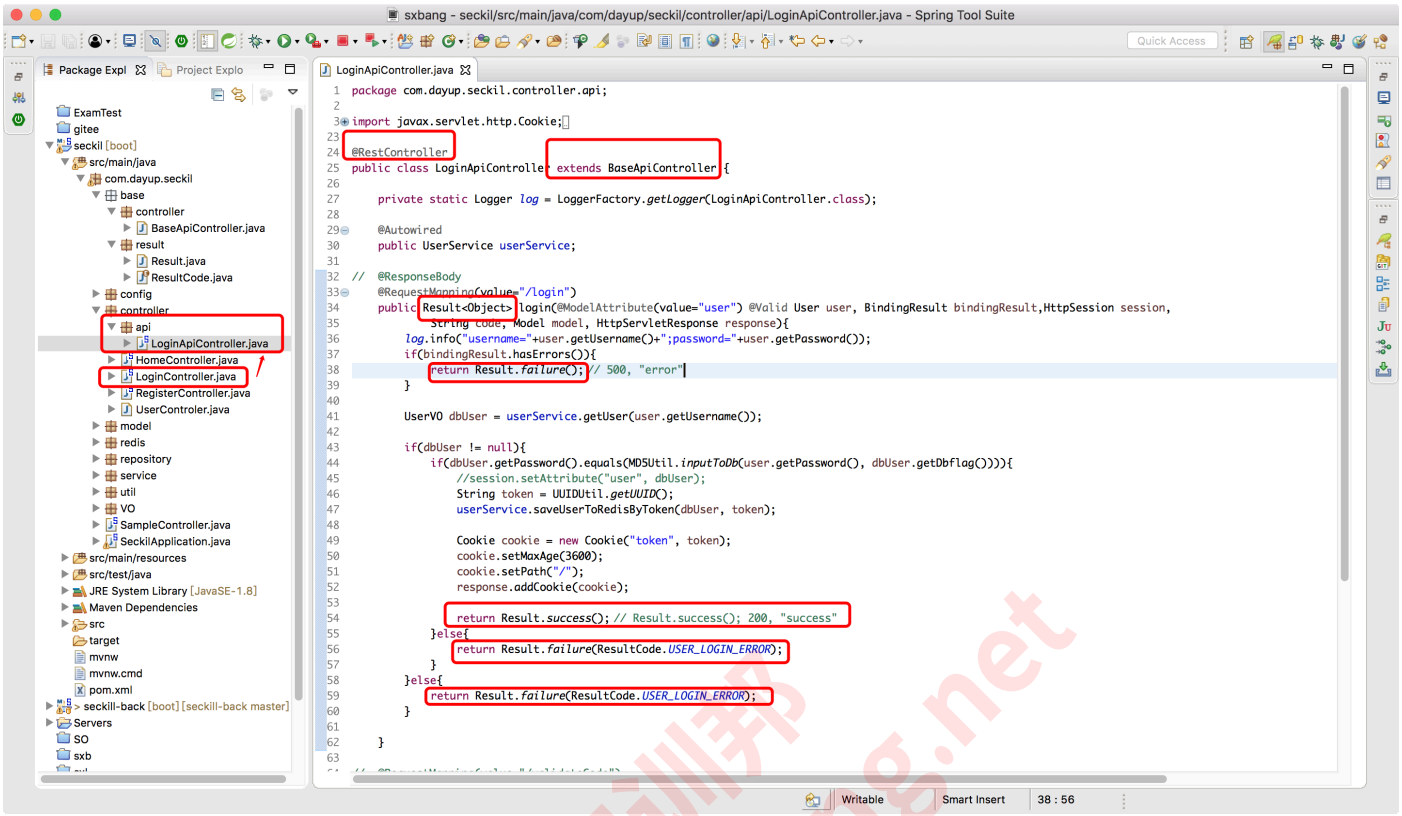
5. ResultCode.java

LoginApiController.java | BaseApiController.java | Result.java | ResultCode.java

```java
package com.dayup.seckil.base.result;

public enum ResultCode {

    // 全局成功
    SUCCESS(200, "成功"),
    // 全局失败
    FAIL(500, "失败"),

    /**
     * 跟用户有关的错误
     * USER_ 开头
     * 错误码 500200 - 500299
     */
    USER_LOGIN_ERROR(500201, "登录失败，用户名或密码出错，请重新输入。"),
    USER_HAS_EXISTED(500202, "用户已存在，请试试其他用户名。"),
    USER_NOT_LOGIN(500203, "用户未登录或登录已失效，请重新登录。");

    private Integer code;

    private String message;

    ResultCode(Integer code, String message) {
        this.code = code;
        this.message = message;
    }

    public Integer getCode() {
        return code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

}
```


- 使用枚举类型定义静态错误信息，方便修改，查找，添加。
- 我们约定，200 为成功，500 为失败，因为失败的情况和业务较多，我们用 500 开头，如 5002**表示用户相关业务出错等。

2.LoginApiController



- 复制 LoginController 到 api 包中命名为 LoginApiController ,并继承 BaseApiController ,改写 login 方法。
- 返回值为 Result<Object>
- 调用 Result 中封装的静态成功与失败方法去返回。
- Controller 上的 @Controller 改为 @RestController, @RestController 相关于 @Controller 和 @ResponseBody 的合并注解 ,如果类注解了@RestController ,下面的方法就不用每个都写@ReponseBody 了。

回马枪总结

1. 使用 Result 返回固定格式结果
2. 使用注解@CrossOrigin：处理跨域请求的注解
3. 使用注解@RestController：返回 json 数据