R类 和 枚举

01、为什么学习统一返回？

在企业其实大部分的程序开发，或者接口开发，其实都返回一种所谓的JSON格式。而json格式是通过java面向对象的方式进行封装而得。

- 在统一异常处理的时候，我们使用的是@RestControllerAdvice是controller后置增强处理。使用@RestControllerAdvice后置处理可以达到
- 在前后端分离的项目架构中，全部都是异步返回 @RestController 也全部都是@ResponseBody的方法。以为你没有页面也没有所谓的freemarker。所以在开发我们肯定是希望把具体错误信息---进行拦截--进行改造（统一返回） -- 改造的信息返回给用户（接口的调用者）
- 进行改造的标准就显得非常的重要，一般做法就使用java面向对象的方式进行开发。面向类返回，而不要使用Map和Object.
- 因为使用Map的key用户随便定义，你达不到高度统一和规范。

常用的命名：

- R
- Result
- ResultResponse

- ServerResponse
- ApiResponse
- ApiResult

02、R类封装的过程

统一返回的格式：

> 成功只有一个情况，失败就N种情况。

```
# 成功的返回
{
  status:200,
  data:{id:1,name:"zhangsan"},
  msg:""
}


#  失败返回
{
  status:601,
  msg:"用户不存在!!",
  data:""
}

{
  status:602,
```

```
  msg:"支付用户已禁止!!",
  data:""
}
```

认识json格式

```
{}----Map或者Bean
[{Map},{Map},{Map}] List<Map>
[{Bean},{Bean},{Bean}] List<Bean>
```

R的雏形：

```
@Data
public class R {

    // 状态码
    private Integer status;
    // 返回信息
    private String msg;
    // 返回数据，因为返回数据是不确定类型，所以只能考虑Object或者泛型
    private Object data;
}
```

03、R的使用原因是什么？

```
1  package com.kuangstudy.web;
2
3  import com.kuangstudy.bean.User;
4  import com.kuangstudy.config.R;
5  import
   org.springframework.web.bind.annotation.GetMappin
   g;
6  import
   org.springframework.web.bind.annotation.PostMappi
   ng;
7  import
   org.springframework.web.bind.annotation.RequestBo
   dy;
8  import
   org.springframework.web.bind.annotation.RestContr
   oller;
9
10 import java.util.Date;
11 import java.util.HashMap;
12 import java.util.Map;
13
14 /**
15  * Description:
16  * Author: yykk Administrator
17  * Version: 1.0
18  * Create Date Time: 2021/12/16 20:33.
19  * Update Date Time:
20  *
21  * @see
```

```
 */
@RestController
public class UserController {

    @GetMapping("/saveuser")
    public R saveUser(){
        // 1:保存用户对象
        User user = new User();
        user.setUserid(1L);
        user.setUsername("yykk");
        user.setPassword("24534534");
        user.setCreateTime(new Date());
        // 2: 返回R
        R r = new R();
        r.setStatus(200);
        r.setMsg("");
        r.setData(user);
        return r;
    }

    @GetMapping("/saveuser2")
    public Map<String,Object> saveUser2(){
        // 1:保存用户对象
        User user = new User();
        user.setUserid(1L);
        user.setUsername("yykk");
        user.setPassword("24534534");
        user.setCreateTime(new Date());
        // 2: 返回R
        Map<String,Object> map = new HashMap<>();
        map.put("status",200);
```

```
        map.put("msg","");
        map.put("data",user);
        return map;
    }
}

```

http://localhost:8085/saveuser 返回得到如下效果：

```
var res = {
  "status": 200,
  "msg": "",
  "data": {
    "userid": 1,
    "username": "yykk",
    "createTime": "2021-12-16T12:39:10.037+00:00",
    "password": "24534534"
  }
}

// 如果取值
console.log(res.status)
console.log(res.data.userid)
console.log(res.data.username)
console.log(res.data.createTime)
console.log(res.data.password)
// 结构取值
var {userid,username,...yykk} = res.data;
console.log(userid)
console.log(username)
```

```
22 console.log(yykk.createTime)
23 console.log(yykk.password)
```

- 上面的代码R和Map,其实可以达到需要的效果。但是Map有问题，不灵活因为一个团队可能是很多开发人员在开发。如果每个人都一套标准和返回的话，就显得没有约束和规范。这个时候接口调用者就开始迷茫了、
- 使用R达到一个统一返回的目的。
  - 标准只有一种，推荐使用此方案
- 问题：返回书写不方便而且很繁琐，可以考虑使用方法进行封装，达到复用的目的和调用方便

04、R类的改造

```
1  package com.kuangstudy.config;
2
3  import lombok.Data;
4
5  /**
6   * Description:
7   * Author: yykk Administrator
8   * Version: 1.0
9   * Create Date Time: 2021/12/15 23:20.
10  * Update Date Time:
11  *
12  * @see
```

```
 */
@Data
public class R {

    // 状态码
    private Integer status;
    // 返回信息
    private String msg;
    // 返回数据，因为返回数据是不确定类型，所以只能考虑Object或者泛型
    private Object data;

    // 全部约束使用方法区执行和返回，不允许到到外部去new
    private R() {

    }

    /**
     * 方法封装 :
     * - R 大量的大量的入侵
     * - 解决调用方便的问题
     *
     * @param obj
     * @return
     */
    public static R success(Object obj) {
        // 200写死，原因很简单：在开发成功只允许只有一种声音，不允许多种
        return restResult(obj, 200, "success");
    }

```

```
42      public static R success(Object obj, String
    msg) {
43          return restResult(obj, 200, msg);
44      }
45
46      // 错误为什么传递status。成功只有一种，但是错误有N状
    态
47      public static R error(Integer status, String
    msg) {
48          return restResult(null, status, msg);
49      }
50
51      public static R error(Integer status, String
    msg, Object obj) {
52          return restResult(obj, status, msg);
53      }
54
55      private static R restResult(Object data,
    Integer status, String msg) {
56          R r = new R();
57          r.setStatus(status);
58          r.setMsg(msg);
59          r.setData(data);
60          return r;
61      }
62  }
63
```

05、R的构造函数为什么要私有化?

其实不用，只不过从规范约束的角度思考话，建议是私有化：让调用只有一种方式。

- 只通过方法区处理返回
- 不允许使用创建对象的对象的方式进行返回
- 从而得到高度的统一和标准

06、R类和枚举恋爱关系

统一返回和枚举：黄金搭档

在统一返回的时候，其实我们很多错误信息，都是让我们程序开发人员去定义，状态也好还是返回信息也好。比如：

```
R.error(601,"用户属于有误");
R.error(602,"支付失败");
R.error(10001,"密码有误");
R.error(500,"服务器忙中");
```

上面会存在一个问题，随着项目越来越庞大，开发人员也越来越多。这个时候就必须高度对着提示和状态控制做集中式管理。方便修改和统一调整。

集中管理的方式

用常量类 & 用接口

```java
package com.kuangstudy.consts;

/**
 * Description:
 * Author: yykk Administrator
 * Version: 1.0
 * Create Date Time: 2021/12/16 20:57.
 * Update Date Time:
 *
 * @see
 */
public interface ResultConstants {
    // return R.error(601,"用户密码有误");
    Integer USER_PWD_STATUS = 601;
    String USER_PWD_MESSAGE = "用户密码有误!";


    //R.error(500,"服务器忙中");
    Integer SERVER_ERROR_STATUS = 500;
    String SERVER_ERROR_MESSAGE = "服务器忙中!";

}

```

上面的静态常量区做集中管理信息是没有任何问题，但是存在几个毛病：

- 状态管理会非常的多 一个接口定义状态会很多，如果一旦开发不标准不规范就造成谁和谁。
- 命名都是一种灾难

枚举

枚举在学习中：它是一个类，类继承接口，继承枚举。

枚举在程序中：它是一个常量类多值匹配（成双入对，三三两两）的时候一种面向对象的替代。

枚举的定义

- 可以定义属性和方法
- 构造函数 - 私有
    - 说明什么：它的对象只能通过方法去暴露。枚举并选择通过方法区暴露对象
    - 而是通过一种自定义构造函数名字的方式，进行暴露。

```
1
```

枚举的定义在什么情况下是单列的：

在枚举中只有一个对象的时候就单列的。

```java
package com.kuangstudy.enums;

/**
 * Description:
 * Author: yykk Administrator
 * Version: 1.0
 * Create Date Time: 2021/12/16 21:05.
 * Update Date Time:
 *
 * @see
 */
public enum ResultStatusEnum {
    INSTANCE();
    ResultStatusEnum() {
    }
    public Integer age;
}

```

```java
ResultStatusEnum.INSTANCE.age = 10;
ResultStatusEnum.INSTANCE2.age = 20;
```

07、R类和枚举恋爱

R.java

```java
package com.kuangstudy.config;

```

```java
import com.kuangstudy.enums.ResultStatusEnum;
import lombok.Data;

/**
 * Description:
 * Author: yykk Administrator
 * Version: 1.0
 * Create Date Time: 2021/12/15 23:20.
 * Update Date Time:
 *
 * @see
 */
@Data
public class R {

    // 状态码
    private Integer status;
    // 返回信息
    private String msg;
    // 返回数据，因为返回数据是不确定类型，所以只能考虑Object或者泛型
    private Object data;

    // 全部约束使用方法区执行和返回，不允许到到外部去new
    private R() {

    }

    /**
     * 方法封装 :
     * - R 大量的大量的入侵
```

```
33          * - 解决调用方便的问题
34          *
35          * @param obj
36          * @return
37          */
38         public static R success(Object obj) {
39             // 200写死，原因很简单：在开发成功只允许只有一种声音，不允许多种
40             return restResult(obj, ResultStatusEnum.SUCCESS_STATUS.getStatus(), ResultStatusEnum.SUCCESS_STATUS.getMessage());
41         }
42
43         public static R success(Object obj, String msg) {
44             return restResult(obj, ResultStatusEnum.SUCCESS_STATUS.getStatus(), ResultStatusEnum.SUCCESS_STATUS.getMessage());
45         }
46
47         // 错误为什么传递status。成功只有一种，但是错误有N状态
48         @Deprecated
49         public static R error(Integer status, String msg) {
50             return restResult(null, status, msg);
51         }
52
53         @Deprecated
54         public static R error(Integer status, String msg, Object obj) {
```

```
55          return restResult(obj, status, msg);
56      }
57
58      public static R error(ResultStatusEnum
    resultStatusEnum) {
59          return restResult(null,
    resultStatusEnum.getStatus(),resultStatusEnum.get
    Message());
60      }
61
62      private static R restResult(Object data,
    Integer status, String msg) {
63          R r = new R();
64          r.setStatus(status);
65          r.setMsg(msg);
66          r.setData(data);
67          return r;
68      }
69  }
70
```

ResultStatusEnum.java

```
1  package com.kuangstudy.enums;
2
3  /**
4   * Description:
5   * Author: yykk Administrator
6   * Version: 1.0
7   * Create Date Time: 2021/12/16 21:05.
8   * Update Date Time:
```

```
 *
 * @see
 */
public enum ResultStatusEnum {

    SUCCESS_STATUS(200,"SUCCESS"),
    USER_PWR_STATUS(601,"用户密码有误"),
    ORDER_ERROR_STATUS(602,"订单有误");

    ResultStatusEnum(Integer status,String message) {
        this.status = status;
        this.message = message;
    }

    private Integer status;
    private String message;

    public Integer getStatus() {
        return status;
    }

    public String getMessage() {
        return message;
    }

}

```

08、枚举还有必要**set**方法吗**?**

本身统一返回用枚举，本身就去做集中管理错误信息，你现在提供一个set暴露出去让程序员进行去修改，那不是自相矛盾吗？所以这种场景下，枚举的set就应去除。

09、**SpringMvc**框架提供**ResponseEntity**为什么不用？

在一些开源项目中，未来学习一个项目课程的时候，可能会看到有一些项目返回的并不是R。而是ResponseEntity

ResponseEntity 是springmvc官方提供。它sprignmvc的方法做统一管理的一个机制。但是这个存在很大的局限性。不方便扩展和以及内部提供HttpStatus枚举类。满足不了我们业务需求。因为它会脱离业务，不方便控制。所以一般不使用。

其实R类+自定义枚举的参考：ResponseEntity + HttpStatus

```java
@GetMapping("/saveuser3")
public ResponseEntity<User> saveuser3(){
    // 1:保存用户对象
    User user = new User();
    user.setUserid(1L);
    user.setUsername("yykk");
    user.setPassword("24534534");
    user.setCreateTime(new Date());
    // 2: 返回R
    return
ResponseEntity.status(HttpStatus.OK).body(user);
}
```

010、R存在和枚举的存在问题?

在企业开发一定多人协同的开发模式，但是如果团队突然一下激增几十人或者上百人的时候，就存在一个问题。这个时候团队就出现一个声音，很多觉得这种统一返回有点点：

- 太过于约束。
- 来了一个新人，又和沟通一遍。沟通成本会越来越高。

解决方案

springmvc利用aop思想，其实在controller调用方法时候做很多的增强处理。aop：对象执行方法

- 如果出错：sprignmvc统一异常处理---Aop. ----@AfterThrowing----切面
- springmvc还提供统一返回返回的处理----Aop----@AfterReturing---切面
- springmvc还提供统一参数处理----Aop----@Before---切面

实现步骤

定义一个统一返回的类：ResultResponseHandler实现ResponseBodyAdvice接口如下：

```
1  package com.kuangstudy.config;
2
3  import
   com.fasterxml.jackson.databind.ObjectMapper;
4  import com.xq.common.result.handler.ErrorHandler;
5  import
   com.xq.controller.login.WeixinLoginController;
6  import
   com.xq.controller.login.WeixinOpenReplyController
   ;
7  import
   com.xq.controller.pay.alipay.AliPayController;
8  import
   com.xq.controller.pay.weixin.WeixinNavtiveControl
   ler;
9  import org.springframework.core.MethodParameter;
10 import org.springframework.http.MediaType;
```

```
11 import
   org.springframework.http.converter.HttpMessageCon
   verter;
12 import
   org.springframework.http.server.ServerHttpRequest
   ;
13 import
   org.springframework.http.server.ServerHttpRespons
   e;
14 import
   org.springframework.web.bind.annotation.RestContr
   ollerAdvice;
15 import
   org.springframework.web.servlet.mvc.method.annota
   tion.ResponseBodyAdvice;
16
17 /**
18  * @author 飞哥
19  * @Title: 学相伴出品
20  * @Description: 我们有一个学习网站:
   https://www.kuangstudy.com
21  * @date 2021/6/2 11:16
22  * <p>
23  * bug: (basePackages = "com.kuangstudy")建议扫包
24  * 为什么?
25  * 如果你项目中没有使用Swagger，你可以扫包也可以不扫。都
   是正常的。
26  * 但是如果你项目使用了Swagger，因为Swagger本身也是一个
   springmvc的项目，他里面也是一个个http请求
27  * 这个请求的时候如果你项目中配置了拦截器，或者一些通知类
   xxxAdvice，那么就会把Swagger都会进行拦截。
```

```
28  * 就会造成Swagger失效。
29  */
30 @RestControllerAdvice
31 public class ResultResponseHandler implements
   ResponseBodyAdvice<Object> {
32     /**
33      * 是否支持advice功能，true是支持 false是不支持
34      *
35      * @param methodParameter
36      * @return
37      */
38     @Override
39     public boolean supports(MethodParameter
   methodParameter, Class<? extends
   HttpMessageConverter<?>> aClass) {
40         return true;
41     }
42
43
44     // 参数o 代表其实就是springmvc的请求的方法的结果
45     @Override
46     public Object beforeBodyWrite(Object o,
   MethodParameter methodParameter, MediaType
   mediaType, Class<? extends HttpMessageConverter<?
   >> aClass, ServerHttpRequest serverHttpRequest,
   ServerHttpResponse serverHttpResponse) {
47         // 对请求的结果在这里统一返回和处理
48         if (o instanceof ErrorHandler) {
49             // 1、如果返回的结果是一个异常的结果，就把异
   常返回的结构数据倒腾到R.fail里面即可
```

```
50              ErrorHandler errorHandler =
    (ErrorHandler) o;
51              return
    R.error(errorHandler.getStatus(),
    errorHandler.getMsg());
52          } else if (o instanceof String) {
53              try {
54                  // 2、因为springmvc数据转换器对
    String是有特殊处理 StringHttpMessageConverter
55                  ObjectMapper objectMapper = new
    ObjectMapper();
56                  R r = R.success(o);
57                  return
    objectMapper.writeValueAsString(r);
58              } catch (Exception ex) {
59                  ex.printStackTrace();
60              }
61          }
62          return R.success(o);
63      }
64  }
```

正常路线：--- 请求-----controller对象----saveuser-----return user -----supports(true)-- beforeBodyWrite---R.success(user)

异常路线：--- 请求-----controller对象----saveuser-----异常ex---GlobalExceptionControllerHandler---ErrorHandler----- supports(true)-- beforeBodyWrite---R.error(ex)

springclould

A------->提供用户查询接口-------Fegin <-----调用--------B----得到的是什么结果：R

10、为什么要学习自定义异常

```
1  package com.kuangstudy.config;//package
   com.kuangstudy.config;
2
3  /**
4   * Description: 自定义异常
5   * Author: yykk Administrator
6   * Version: 1.0
7   * Create Date Time: 2021/12/15 21:48.
8   * Update Date Time:
9   *
10  * @see
11  */
12 public class BussinessException extends
   RuntimeException {
13
14     private Integer status;
15     private String msg;
16
17     public BussinessException(Integer status,
   String msg) {
18         super(msg);
19         this.msg = msg;
```

```
        this.status = status;
    }

    public BussinessException(String msg) {
        super(msg);
        this.status = 500;
        this.msg = msg;
    }

    public Integer getStatus() {
        return status;
    }

    public String getMsg() {
        return msg;
    }

}

```

```
package com.kuangstudy.config;//package com.kuangstudy.config;
/**
 * Description: 自定义异常
 * Author: yykk Administrator
 * Version: 1.0
 * Create Date Time: 2021/12/15 21:48.
 * Update Date Time:
 *
 * @see
 */
```

```java
public class OrderException extends RuntimeException {

    private Integer status;
    private String msg;

    public OrderException(Integer status, String msg) {
        super(msg);
        this.msg = msg;
        this.status = status;
    }

    public OrderException(String msg) {
        super(msg);
        this.status = 500;
        this.msg = msg;
    }

    public Integer getStatus() {
        return status;
    }

    public String getMsg() {
        return msg;
    }

}

```

- 自定义异常可以方便后续分析业务 便于后续分析和定位

- 自定义不建议继承：Exception 和 Throwable 如果你集成是顶级异常。你throw必须要throws.