拒绝写重复代码，试试这套开源的 SpringBoot 组件，效率翻倍~

1 简介

Graceful Response是一个Spring Boot技术栈下的优雅响应处理器，**提供一站式统一返回值封装、全局异常处理、自定义异常错误码** 等功能，使用Graceful Response进行web接口开发不仅可以节省大量的时间，还可以提高代码质量，使代码逻辑更清晰。

> 强烈推荐你花3分钟学会它！

本项目案例工程代码：`https://github.com/feiniaojin/graceful-response-example.git`，注意选择最新版本的分支。

Spring Boot版本

Graceful Response版本

graceful-response-example分支

2.x

3.2.1-boot2

3.2.0-boot2

3.x

3.2.1-boot3

3.2.0-boot3

> 注意，3.2.1-boot2版本的Graceful Response源码由单独的仓库进行维护，地址为：`https://github.com/feiniaojin/graceful-response-boot2`
>
>
> 3.2.1-boot2和3.2.1-boot3除了支持的SpringBoot版本不一样，其他实现完全一致，Maven引用时只需要根据对应的SpringBoot版本选择Graceful Response的version即可，两者的groupId、artifactId是一致的。

> 基于 Spring Boot + MyBatis Plus + Vue & Element 实现的后台管理系统 + 用户小程序，支持 RBAC 动态权限、多租户、数据权限、工作流、三方登录、支付、短信、商城等功能
>
>
> * 项目地址：[github.com/YunaiV/ruoy...](http://cxyroad.com/”https://github.com/YunaiV/ruoyi-vue-pro”)
> * 视频教程：[doc.iocoder.cn/video/](http://cxyroad.com/”https://doc.iocoder.cn/video/”)

2 快速入门

2.1 Spring Boot接口开发现状

目前，业界使用Spring Boot进行接口开发时，往往存在效率底下、重复劳动、可读性差等问题。以下伪代码相信大家非常熟悉，我们大部分项目的Controller接口都是这样的。

```
@Controllerpublic class Controller {    @GetMapping("/query")
@ResponseBody    public Response query(Map<String, Object>
paramMap) {        Response res = new Response();        try {
//1.校验params参数合法性，包括非空校验、长度校验等            if
(illegal(paramMap)) {                res.setCode(1);
res.setMsg("error");                return res;            }            //2.调用
Service的一系列操作，得到查询结果            Object data =
service.query(params);            //3.将操作结果设置到res对象中
res.setData(data);            res.setCode(0);            res.setMsg("ok");
  return res;        } catch (Exception e) {            //4.异常处理：一堆丑陋的
try...catch，如果有错误码的，还需要手工填充错误码
res.setCode(1);            res.setMsg("error");            return res;        }    }}
```

这段伪代码存在什么样的问题呢?

第一个问题，效率低下。 Controller层的代码应该尽量简洁，上面的伪代码其实只是为了将数据查询的结果进行封装，使其以统一的格式进行返回。例如以下格式的响应体：

```
{  "code": 0,  "msg": "ok",  "data": {    "id": 1,    "name": "username"  }}
```

查询过程中如果发生异常，需要在Controller进行手工捕获，根据捕获的异常人工地设置错误码，当然，也用同样的格式封装错误码进行返回。

可以看到，除了调用service层的query方法这一行，其他大部分的代码都执行进行结果的封装，大量的冗余、低价值的代码导致我们的开发活动效率很低。

第二个问题，重复劳动。 以上捕获异常、封装执行结果的操作，每个接口都会进行一次，因此造成大量重复劳动。

第三个问题，可读性低。 上面的核心代码被淹没在许多冗余代码中，很难阅读，如同大海捞针。

我们可以通过Graceful Response这个组件解决这样的问题。

2.2. 快速入门

2.2.1 引入Graceful Response组件

Graceful Response已发布至maven中央仓库，我们可以直接引入到项目中。

maven依赖如下：

```
<dependency>    <groupId>com.feiniaojin</groupId>
<artifactId>graceful-response</artifactId>
<version>{latest.version}</version></dependency>
```

Spring Boot版本

Graceful Response最新版本

2.x

3.2.1-boot2

3.x

3.2.1-boot3

2.2.2 启用Graceful Response

在启动类中引入`@EnableGracefulResponse`注解，即可启用Graceful Response组件。

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

2.2.3 Controller层

引入Graceful Response后，我们不需要再手工进行查询结果的封装，直接返回实际结果即可，Graceful Response会自动完成封装的操作。

Controller层示例如下。

```
@Controllerpublic class Controller {    @RequestMapping("/get") @ResponseBody    public UserInfoView get(Long id) { log.info("id={}", id);        return UserInfoView.builder().id(id).name("name" + id).build();    }}
```

在示例代码中，Controller层的方法直接返回了UserInfoView对象，没有进行封装的操作，但经过Graceful Response处理后，我们还是得到了以下的响应结果。

```
{  "status": {    "code": "0",    "msg": "ok"  }, "payload": {    "id": 1, "name": "name1"  }}
```

而对于命令操作（Command）尽量不返回数据，因此command操作的方法的返回值应该是void，Graceful Response对于对于返回值类型void的方法，也会自动进行封装。

```
public class Controller {    @RequestMapping(”/command”)
@ResponseBody    public void command() {        //业务操作    }}
```

成功调用该接口，将得到：

```
{  ”status”: {    ”code”: ”200”,    ”msg”: ”success”  },  ”payload”: {}}
```

2.2.4 Service层

在引入Graceful Response前，有的开发者在定义Service层的方法时，为了在接口中返回异常码，干脆直接将Service层方法定义为Response，淹没了方法的正常返回值。

Response的代码如下。

```
//lombok注解@Datapublic class Response {    private String code;
private String msg;    private Object data;}
```

直接返回Response的Service层方法：

```
/** * 直接返回Reponse的Service * 不规范 */public interface Service {
public Reponse commandMethod(Command command);}
```

```

Graceful Response引入`@ExceptionMapper`注解，通过该注解将异常和错误码关联起来，这样Service方法就不需要再维护Response的响应码了，直接抛出业务异常，由Graceful Response进行异常和响应码的关联。

`@ExceptionMapper`的用法如下。

```
/** * NotFoundException的定义，使用@ExceptionMapper注解修饰 * code:代表接口的异常码 * msg:代表接口的异常提示 */@ExceptionMapper(code = ”1404”, msg = ”找不到对象”)public class NotFoundException extends RuntimeException {}
```

Service接口定义：

```
public interface QueryService {    UserInfoView queryOne(Query query);}
```

Service接口实现：

```
public class QueryServiceImpl implements QueryService {    @Resource private UserInfoMapper mapper;    public UserInfoView queryOne(Query query) {        UserInfo userInfo = mapper.findOne(query.getId());        if (Objects.isNull(userInfo)) {            //这里直接抛自定义异常            throw new NotFoundException();        }        //……后续业务操作    }}
```

当Service层的queryOne方法抛出`NotFoundException`时，Graceful Response会进行异常捕获，并将NotFoundException对应的异常码和异常信息封装到统一的响应对象中，最终接口返回以下JSON。

```
```
```

{ ”status”: { ”code”: ”1404”, ”msg”: ”找不到对象” }, ”payload”: {}}

```

##### 2.2.5 参数校验

Graceful Response对JSR-303数据校验规范和Hibernate Validator进行了增强，Graceful Response自身不提供参数校验的功能，但是用户使用了Hibernate Validator后，Graceful Response可以通过`@ValidationStatusCode`注解为参数校验结果提供响应码，并将其统一封装返回。

例如以下的UserInfoQuery。

```
@Datapublic class UserInfoQuery { @NotNull(message = ”userName is null !”) @Length(min = 6, max = 12) @ValidationStatusCode(code = ”520”) private String userName;}
```

UserInfoQuery对象中定义了@NotNull和@Length两个校验规则，在未引入Graceful Response的情况下，会直接抛出异常；

在引入Graceful Response但是没有加入`@ValidationStatusCode`注解的情况下，会以默认的错误码进行返回；

在上面的UserInfoQuery中由于使用了`@ValidationStatusCode`注解，并指定异常码为520，则当userName字段任意校验不通过时，都会使用异常码520进行返回，如下。

```
{ ”status”: { ”code”: ”520”, ”msg”: ”userName is null !” }, ”payload”: {}}
```

而对于Controller层直接校验方法入参的场景，Graceful Response也进行了增强，如以下Controller。
```

```
public class Controller {    @RequestMapping(”/validateMethodParam”) @ResponseBody    @ValidationStatusCode(code = ”1314”)    public void validateMethodParam(            @NotNull(message = ”userId不能为空”) Long userId,            @NotNull(message = ”userName不能为空”) Long userName) {        //省略业务逻辑    }}
```

如果该方法入参校验触发了userId和userName的校验异常，将以错误码1314进行返回，如下

```
{  ”status”: {    ”code”: ”1314”,    ”msg”: ”userId不能为空”  }, ”payload”: {}}
```

2.2.6 自定义Response格式

Graceful Response内置了两种风格的响应格式，并通过`graceful-response.response-style`进行配置。

`graceful-response.response-style=0`，或者不配置（默认情况），将以以下的格式进行返回：

```
{  ”status”: {    ”code”: 1007,    ”msg”: ”有内鬼，终止交易”  }, ”payload”: {  }}
```

`graceful-response.response-style=1`，将以以下的格式进行返回：

```
{  ”code”: ”1404”,  ”msg”: ”not found”,  ”data”: {  }}
```

如果这两种格式均不满足业务需要，Graceful Response也支持用户自定义响应体，关于自定义响应体的技术实现，请到自定义Response格式进行了解。

本项目提供的进阶功能，包括

* 第三方组件汽车（Swagger、执行器等）
* 自定义响应
* 异常请求放行
* 异常别名
* 常用配置项

目前该组件在GitHub上已经有两百多Star，很多朋友已经开始用了

> 来源：[github.com/feiniaojin/...](http://cxyroad.com/”https://github.com/feiniaojin/graceful-response”)

原文链接: https://juejin.cn/post/7367195057559371788