【Java系列】SpringCloudAlibaba统一返回体及全局异常捕获实现

> 本文将以实际代码展示如何实现SpringCloudAlibaba的统一返回体及全局异常捕获。
>
>
> **作者：后端小肥肠**

1. 前言

在构建微服务应用时，统一返回体和异常捕获机制的设计对于保持代码的整洁性和提高服务的可维护性至关重要。特别是在使用 Spring Boot 和 Spring Cloud Alibaba这样的现代开发框架时，这一点显得尤为重要。本文将重点介绍如何在Spring Cloud Alibaba环境中实现统一的响应体和异常处理策略。通过这种方式，无论是在单体应用还是在复杂的微服务架构中，开发者都能保证返回信息的一致性和异常的有效管理。

2. 开发环境搭建

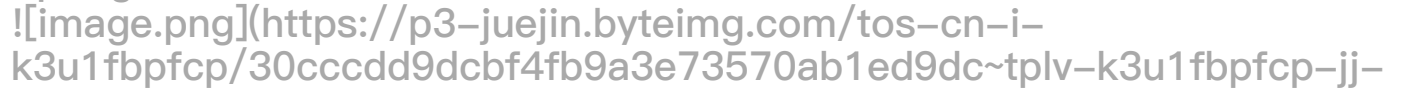
2.1. 所需版本依赖

依赖	版本
Spring Boot	2.6.3
Spring Cloud	2021.0.1
java	1.8以上
Spring Cloud Alibaba	2021.0.1.0

2.2. pom依赖

```
```

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.3</version>
        <relativePath/>
    </parent>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <spring-cloud.version>2021.0.1</spring-cloud.version>
        <spring-cloud-alibaba.version>2021.0.1.0</spring-cloud-
alibaba.version>
    </properties>
    <dependencies>
            <!-- springCloud -->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>com.alibaba.cloud</groupId>
                <artifactId>spring-cloud-alibaba-dependencies</artifactId>
                <version>${spring-cloud-alibaba.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
             <dependency>
                <groupId>org.jetbrains</groupId>
                <artifactId>annotations</artifactId>
                <version>17.0.0</version>
            </dependency>
     </dependencies>
```

```

3. 统一返回体实现
----------


### 3.1. Spring Cloud统一返回体实现


SpringCloud项目结构示例

```

mark:3024:0:0:0:q75.awebp#?w=539&h=307&s=18180&e=png&b=3c3f41)

在common中新建response包，将统一返回相关类放入其中即可。

![image.png](https://p6-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/1368ca09ba5d499cb0743cb735acb61f~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=540&h=502&s=25738&e=png&b=3c3f41)

3.1.1. 编写响应状态码枚举

```
public enum ResponseStatusCodeEnum implements IResponseStatusCode {
    //服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。
    SUCCESS(200, "OK"),
    //用户新建或修改数据成功。
    UPDATE_RETURN_201(201, "CREATED"),
    //表示一个请求已经进入后台排队（异步任务）
    ALL_RETURN_202(202, "Accepted"),
    //用户删除数据成功。
    DELETE_RETURN_204(204, "NO CONTENT"),
    //用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。
    UPDATE_RETURN_400(400, "INVALID REQUEST"),
    //用户发出的请求参数有误,服务器没有找到对应资源 这是新加的
    WRONG_PARAMETER_NOT_FIND_400(400,"请求参数有误，资源不存在"),
    //401 Unauthorized – [*]：表示用户没有权限（令牌、用户名、密码错误）。
    ALL_RETURN_401(401, "Unauthorized"),
    TOKEN_PAST(1401, "身份过期，请求重新登录！"),
    //403 Forbidden – [*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。
    ALL_RETURN_403(403, "Forbidden"),
    //404 NOT FOUND – [*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。
    ALL_RETURN_404(404, "NOT FOUND"),
    //406 Not Acceptable – [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。
    GET_RETURN_406(406, "Not Acceptable"),
    //410 Gone –[GET]：用户请求的资源被永久删除，且不会再得到的。
```

```
    GET_RETURN_410(410, ”Gone”),
    //422 Unprocessable entity – [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。
    UPDATE_RETURN_422(422, ”Unprocessable entity”),
    //500 INTERNAL SERVER ERROR – [*]：服务器发生错误，用户将无法判断发出的请求是否成功。*/
    GET_RETURN_500(500, ”INTERNAL SERVER ERROR”),
    CONFLICT_RETURN_409(409,”CONFLICT”);

    private final Integer code;
    private final String message;

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

    @Override
    public Integer getCode() {
        return this.code;
    }

    @Override
    public String getMessage() {
        return this.message;
    }
}


```

3.1.2. 编写响应状态码接口

```
public interface IResponseStatusCode {
    /**
     * 获取响应状态码
     *
     * @return 响应状态码
     */
    Integer getCode();

    /**
     * 获取响应消息
     *
     * @return 响应消息
```

```
     */
    String getMessage();
}
```

3.1.3. 编写统一返回结构体

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ResponseStructure<T>{
    private Integer code;
    private String status;
    private String message;
    private T data;

    private static final String SUCCESS = "success";
    private static final String FAIL = "fail";


    public static <T> ResponseStructure<T> success(T data) {
        return new ResponseStructure<>(
                ResponseStatusCodeEnum.SUCCESS.getCode(),
                SUCCESS,
                ResponseStatusCodeEnum.SUCCESS.getMessage(),
                data
        );
    }

    public static <T> ResponseStructure<T> created(String message) {
        return new ResponseStructure<>(
                ResponseStatusCodeEnum.UPDATE_RETURN_201.getCode(),
                SUCCESS,
                message,
                null);
    }

    public static <T> ResponseStructure<T> unauthorized(String
message) {
        return new ResponseStructure<>(
                ResponseStatusCodeEnum.ALL_RETURN_403.getCode(),
                FAIL,
                message,
                null);
```

```
    }

    public static <T> ResponseStructure<T> unauthenticated(String
message) {
        return new ResponseStructure<>(
                ResponseStatusCodeEnum.ALL_RETURN_401.getCode(),
                FAIL,
                message,
                null);
    }

    public static <T> ResponseStructure<T> success(String message, T
data) {
        return new ResponseStructure<>(
                ResponseStatusCodeEnum.SUCCESS.getCode(),
                SUCCESS,
                message,
                data);
    }

    public static <T> ResponseStructure<T> success(Integer code, String
message) {
        return new ResponseStructure<>(code, SUCCESS, message, null);
    }

    public static <T> ResponseStructure<T> success(Integer code, String
message, T data) {
        return new ResponseStructure<>(code, SUCCESS, message, data);
    }

    public static ResponseStructure<Object> failed() {
        return new ResponseStructure<>(
                ResponseStatusCodeEnum.GET_RETURN_500.getCode(),
                FAIL,
                ResponseStatusCodeEnum.GET_RETURN_500.getMessage(),
                null);
    }

    public static ResponseStructure<String> failed(String message) {
        return new ResponseStructure<>(
                ResponseStatusCodeEnum.GET_RETURN_500.getCode(),
                FAIL,
                message,
                null);
    }

    public static ResponseStructure<Object> failed(IResponseStatusCode
errorResult) {
```

```
        return new ResponseStructure<>(
                errorResult.getCode(),
                FAIL,
                errorResult.getMessage(),
                null);
    }

    public static ResponseStructure<Object> conflict(String message) {
        return new ResponseStructure<>(
ResponseStatusCodeEnum.CONFLICT_RETURN_409.getCode(),
   FAIL,
                message,
                null);
    }

    public static <T> ResponseStructure<T> instance(Integer code, String
message, T data) {
        ResponseStructure<T> responseStructure = new
ResponseStructure<>();
        responseStructure.setCode(code);
        responseStructure.setMessage(message);
        responseStructure.setData(data);

        if (code >= 300) {
            responseStructure.setStatus(FAIL);
        } else {
            responseStructure.setStatus(SUCCESS);
        }

        return responseStructure;
    }
    public static <T> ResponseStructure<T> instance(Integer code, String
message) {
        ResponseStructure<T> responseStructure = new
ResponseStructure<>();
        responseStructure.setCode(code);
        responseStructure.setMessage(message);
        responseStructure.setData(null);

        if (code >= 300) {
            responseStructure.setStatus(FAIL);
        } else {
            responseStructure.setStatus(SUCCESS);
        }

        return responseStructure;
    }
```

```
    public static <T> ResponseStructure<T>
instance(IResponseStatusCode code) {
        ResponseStructure<T> responseStructure = new
ResponseStructure<>();
        responseStructure.setCode(code.getCode());
        responseStructure.setMessage(code.getMessage());
        responseStructure.setData(null);

        if (code.getCode() >= 300) {
            responseStructure.setStatus(FAIL);
        } else {
            responseStructure.setStatus(SUCCESS);
        }

        return responseStructure;
    }
}
```

3.1.4. 编写GlobalResponseBodyAdvice

```
@RestControllerAdvice
public class GlobalResponseBodyAdvice implements
ResponseBodyAdvice<Object> {
    @Override
    public boolean supports(@NotNull MethodParameter returnType,
                            @NotNull Class<? extends
HttpMessageConverter<?>> converterType) {
        GlobalResponse globalResponse =
returnType.getMethodAnnotation(GlobalResponse.class);
        return globalResponse == null || globalResponse.format();
    }

    @Override
    public Object beforeBodyWrite(Object body,
                                  @NotNull MethodParameter returnType,
                                  @NotNull MediaType selectedContentType,
                                  @NotNull Class<? extends
HttpMessageConverter<?>> selectedConverterType,
                                  @NotNull ServerHttpRequest request,
                                  @NotNull ServerHttpResponse response) {
        // 如果是 actuator 请求，直接返回
        if (isActuatorRequest(request)) {
            return body;
```

```
        }

        /* 如果是 Feign 请求，直接返回
         */
        if (Objects.requireNonNull(request.getHeaders().get("user-
agent")).get(0).startsWith("Java")) {
            return body;
        }

        // 以下代码主要解决和 Swagger 的冲突
        if (body instanceof ResponseStructure || body instanceof Json ||
body instanceof UiConfiguration ||
                (body instanceof ArrayList && !((ArrayList<?>)
body).isEmpty() &&
                        ((ArrayList<?>) body).get(0) instanceof
SwaggerResource)) {
            return body;
        }

        ResponseStructure<Object> responseStructure;

        // 如果是 POST 请求，业务状态码统一设置为 201
        if ("POST".equals(((ServletServerHttpRequest)
request).getServletRequest().getMethod())) {
            responseStructure = ResponseStructure.created("OK");
        } else {
            responseStructure = ResponseStructure.success(null);
        }

        // 如果返回值是字符串类型，则用其替换 message
        if (body instanceof String) {
            responseStructure.setData(body);
            return JSON.toJSONString(responseStructure);
        }

        if (body instanceof byte[]) {
            return body;
        }

        responseStructure.setData(body);

        return responseStructure;
    }

    private boolean isActuatorRequest(ServerHttpRequest request) {
        return ((ServletServerHttpRequest)
request).getServletRequest().getRequestURI().endsWith("/actuator");
    }
```

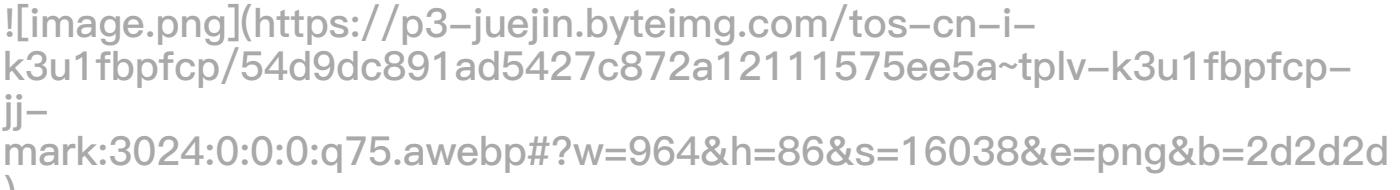
}

```

在上述类中有个核心注解 @RestControllerAdvice，`@RestControllerAdvice` 注解是 Spring Framework 提供的一种特殊的增强型注解，用于全局性地处理 RESTful API 控制器抛出的异常。它结合了 `@ControllerAdvice` 和 `@ResponseBody` 注解的功能，因此适用于 REST 控制器。

这个注解的原理和作用可以分为以下几个方面：

1. **拦截异常处理**: 在应用程序中，`@RestControllerAdvice` 注解的类会被 Spring 容器识别为全局的异常处理器。当 REST 控制器中的方法抛出异常时，Spring 会查找匹配的 `@ExceptionHandler` 方法，并调用它来处理异常。因此，这个注解提供了一种便捷的方式来集中管理和处理异常，避免了在每个控制器中都编写相同的异常处理逻辑。
2. **自定义异常处理逻辑**: 在 `@RestControllerAdvice` 注解的类中，可以定义多个 `@ExceptionHandler` 方法，每个方法处理特定类型的异常。这样就可以根据具体的异常类型，提供相应的错误响应，比如返回 JSON 格式的错误消息或自定义的错误码。这种灵活性使得开发者能够根据实际需求定制化地处理异常情况。
3. **统一数据格式**: RESTful API 通常使用 JSON 格式来传递数据，因此在异常处理过程中，`@RestControllerAdvice` 注解可以确保所有的错误响应都采用统一的 JSON 格式。这样客户端能够更方便地解析和处理错误信息，提高了接口的可用性和易用性。
4. **全局数据绑定和响应处理**: 除了异常处理之外，`@RestControllerAdvice` 注解还可以用于全局数据绑定和响应处理。通过在类中定义 `@InitBinder` 和 `@ModelAttribute` 方法，可以实现全局性的请求参数验证、数据预处理以及通用模型属性的添加等功能，从而进一步提高了代码的复用性和可维护性。

综上所述，`@RestControllerAdvice` 注解通过拦截并统一处理 RESTful API 控制器中的异常、数据绑定和响应，提供了一种简洁而强大的机制来增强应用程序的异常处理和数据处理能力。

#### 3.1.5. 编写GlobalResponse

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface GlobalResponse {
```
```

```
    boolean format() default true;
}

```

3.1.6. 效果测试

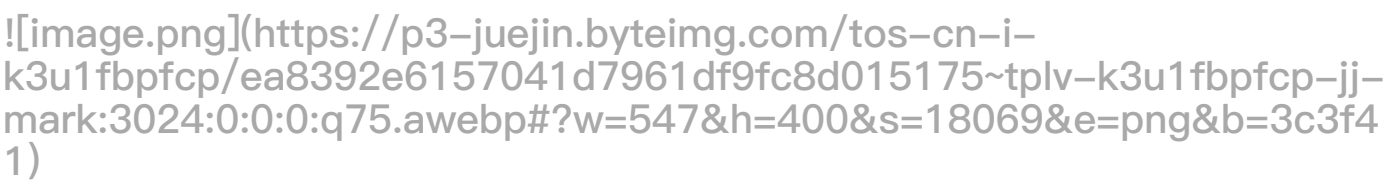
1. controller层方法编写，只需要返回实际数据结构即可



2. 返回结构测试

3.2 Spring Cloud异常捕获实现

在common中新建expection包，将异常捕获类类放入其中即可。



3.2.1. 编写ExceptionAdvice

```
@Slf4j
@RestControllerAdvice
public class ExceptionAdvice {
    @ExceptionHandler(BindException.class)
    public Object bindException(BindException bindException) {
        bindException.printStackTrace();
        String message =
Objects.requireNonNull(bindException.getBindingResult().getFieldError()).
getDefaultMessage();
        return ResponseStructure.conflict(message);
```

```
    }

    @ResponseStatus(value = HttpStatus.CONFLICT)
    @ExceptionHandler({ValidationException.class})
    public ResponseStructure<Object>
handleValidationException(ValidationException validationException) {
        validationException.printStackTrace();
        return (ResponseStructure<Object>)
ResponseStructure.conflict(validationException.getMessage());
    }

    @ResponseStatus(value = HttpStatus.CONFLICT)
    @ExceptionHandler({MaxUploadSizeExceededException.class})
    public ResponseStructure<Object>
handleMaxUploadSizeException(MaxUploadSizeExceededException
maxUploadSizeExceededException) {
        maxUploadSizeExceededException.printStackTrace();
        return (ResponseStructure<Object>) ResponseStructure.conflict("当
前文件大小已超过限制大小，请重新上传文件");
    }

    /**
     * 顶级异常捕获，当其他异常无法处理时选择使用
     */
    @ResponseStatus(value = HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler({Exception.class})
    public ResponseStructure<String> handle(Exception exception) {
        exception.printStackTrace();
        return (ResponseStructure<String>)
ResponseStructure.failed(exception.getMessage());
    }

    /**
     * 认证异常捕获
     */
    @ResponseStatus(value = HttpStatus.UNAUTHORIZED)
    @ExceptionHandler({AuthenticationException.class})
    public ResponseStructure<String>
handleUnAhthorized(AuthenticationException exception) {
        exception.printStackTrace();
        return ResponseStructure.instance(ALL_RETURN_401.getCode(),
exception.getMessage());
    }
}

```

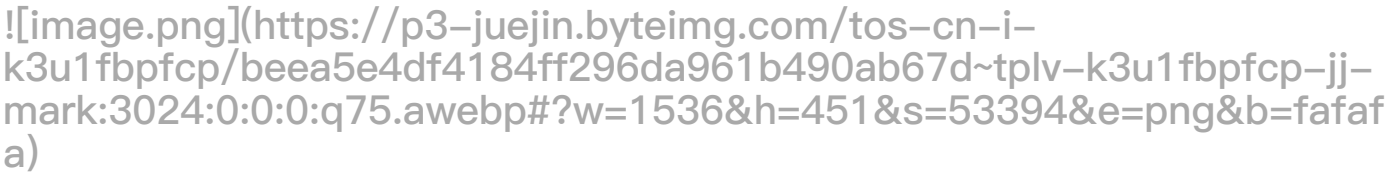
`ExceptionAdvice` 类通过定义一系列异常处理器，使得应用能够在抛出异常时

提供友好的用户反馈，而不是让用户面对不友好的原始错误信息或空白页面。这样的处理机制不仅提高了应用的可用性和可维护性，还能通过日志记录帮助开发者快速定位和解决问题。此外，通过统一异常处理和响应格式，开发者可以更容易地保持前后端的一致性和同步。

3.2.2. 效果测试

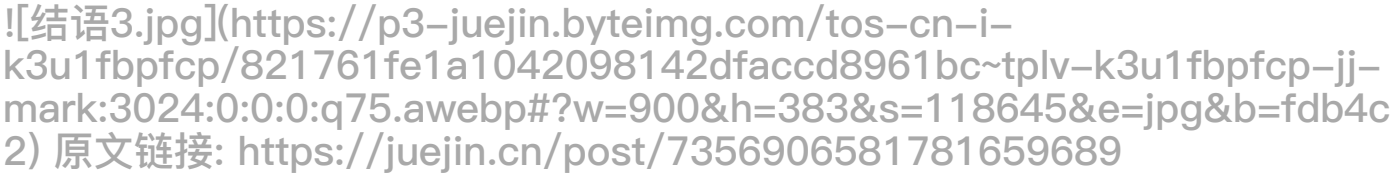
1. 编写异常测试controller层方法

2. 返回结构测试



5. 结语

本文以代码实例展示了如何在SpringCloudAlibaba中实现统一返回及全部异常捕获，如您有更好观点欢迎在评论区留言探讨~



) 原文链接: https://juejin.cn/post/7356906581781659689