**零、前言**

对于与数据库相关的 Spring MVC 项目，我们通常会把 **事务** 配置在 Service层，当数据库操作失败时让 Service 层抛出运行时异常，Spring 事物管理器就会进行回滚。

如此一来，我们的 Controller 层就不得不进行 try-catch Service 层的异常，否则会返回一些不友好的错误信息到客户端。但是，Controller 层每个方法体都写一些模板化的 try-catch 的代码，很难看也难维护，特别是还需要对 Service 层的不同异常进行不同处理的时候。例如以下 Controller 方法代码（非常难看且冗余）：

/\*\*

\* 手动处理 Service 层异常和数据校验异常的示例

\* @param dog

\* @param errors

\* @return

\*/

@PostMapping(value = "")

AppResponse add(@Validated(Add.class) @RequestBody Dog dog, Errors errors){

AppResponse resp = new AppResponse();

try {

// 数据校验

BSUtil.controllerValidate(errors);

// 执行业务

Dog newDog = dogService.save(dog);

// 返回数据

resp.setData(newDog);

}catch (BusinessException e){

LOGGER.error(e.getMessage(), e);

resp.setFail(e.getMessage());

}catch (Exception e){

LOGGER.error(e.getMessage(), e);

resp.setFail("操作失败！");

}

return resp;

}

本文讲解使用 @ControllerAdvice + @ExceptionHandler 进行全局的 Controller 层异常处理，只要设计得当，就再也不用在 Controller 层进行 try-catch 了！而且，@Validated 校验器注解的异常，也可以一起处理，无需手动判断绑定校验结果 BindingResult/Errors 了！

**一、优缺点**

* 优点：将 Controller 层的异常和数据校验的异常进行统一处理，减少模板代码，减少编码量，提升扩展性和可维护性。
* 缺点：只能处理 Controller 层未捕获（往外抛）的异常，对于 Interceptor（拦截器）层的异常，Spring 框架层的异常，就无能为力了。

**二、基本使用示例**

**2.1 @ControllerAdvice 注解定义全局异常处理类**

@ControllerAdvice

public class GlobalExceptionHandler {

}

请确保此 GlobalExceptionHandler 类能被扫描到并装载进 Spring 容器中。

**2.2 @ExceptionHandler 注解声明异常处理方法**

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(Exception.class)

@ResponseBody

String handleException(){

return "Exception Deal!";

}

}

方法 handleException() 就会处理所有 Controller 层抛出的 **Exception 及其子类的异常**，这是最基本的用法了。

被 **@ExceptionHandler** 注解的方法的参数列表里，还可以声明很多种类型的参数，[详见文档](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/ExceptionHandler.html" \t "_blank)。其原型如下：

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface ExceptionHandler {

/\*\*

\* Exceptions handled by the annotated method. If empty, will default to any

\* exceptions listed in the method argument list.

\*/

Class<? extends Throwable>[] value() default {};

}

如果 @ExceptionHandler 注解中未声明要处理的异常类型，则默认为参数列表中的异常类型。所以上面的写法，还可以写成这样：

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler()

@ResponseBody

String handleException(Exception e){

return "Exception Deal! " + e.getMessage();

}

}

参数对象就是 Controller 层抛出的异常对象！

**三、处理 Service 层上抛的业务异常**

有时我们会在复杂的带有数据库事务的业务中，当出现不和预期的数据时，直接抛出封装后的业务级运行时异常，进行数据库事务回滚，并希望该异常信息能被返回显示给用户。

**3.1 代码示例**

封装的业务异常类：

public class BusinessException extends RuntimeException {

public BusinessException(String message){

super(message);

}

}

Service 实现类：

@Service

public class DogService {

@Transactional

public Dog update(Dog dog){

// some database options

// 模拟狗狗新名字与其他狗狗的名字冲突

BSUtil.isTrue(false, "狗狗名字已经被使用了...");

// update database dog info

return dog;

}

}

其中辅助工具类 BSUtil

public static void isTrue(boolean expression, String error){

if(!expression) {

throw new BusinessException(error);

}

}

那么，我们应该在 GlobalExceptionHandler 类中声明该业务异常类，并进行相应的处理，然后返回给用户。更贴近真实项目的代码，应该长这样子：

/\*\*

\* Created by kinginblue on 2017/4/10.

\* @ControllerAdvice + @ExceptionHandler 实现全局的 Controller 层的异常处理

\*/

@ControllerAdvice

public class GlobalExceptionHandler {

private static final Logger LOGGER = LoggerFactory.getLogger(GlobalExceptionHandler.class);

/\*\*

\* 处理所有不可知的异常

\* @param e

\* @return

\*/

@ExceptionHandler(Exception.class)

@ResponseBody

AppResponse handleException(Exception e){

LOGGER.error(e.getMessage(), e);

AppResponse response = new AppResponse();

response.setFail("操作失败！");

return response;

}

/\*\*

\* 处理所有业务异常

\* @param e

\* @return

\*/

@ExceptionHandler(BusinessException.class)

@ResponseBody

AppResponse handleBusinessException(BusinessException e){

LOGGER.error(e.getMessage(), e);

AppResponse response = new AppResponse();

response.setFail(e.getMessage());

return response;

}

}

Controller 层的代码，就不需要进行异常处理了：

@RestController

@RequestMapping(value = "/dogs", consumes = {MediaType.APPLICATION\_JSON\_UTF8\_VALUE})

public class DogController {

@Autowired

private DogService dogService;

@PatchMapping(value = "")

Dog update(@Validated(Update.class) @RequestBody Dog dog){

return dogService.update(dog);

}

}

**3.2 代码说明**

Logger 进行所有的异常日志记录。

@ExceptionHandler(BusinessException.class) 声明了对 BusinessException 业务异常的处理，并获取该业务异常中的错误提示，构造后返回给客户端。

@ExceptionHandler(Exception.class) 声明了对 Exception 异常的处理，起到**兜底**作用，不管 Controller 层执行的代码出现了什么未能考虑到的异常，都返回统一的错误提示给客户端。

备注：以上 GlobalExceptionHandler 只是返回 Json 给客户端，更大的发挥空间需要按需求情况来做。

**四、处理 Controller 数据绑定、数据校验的异常**

在 Dog 类中的字段上的注解数据校验规则：

@Data

public class Dog {

@NotNull(message = "{Dog.id.non}", groups = {Update.class})

@Min(value = 1, message = "{Dog.age.lt1}", groups = {Update.class})

private Long id;

@NotBlank(message = "{Dog.name.non}", groups = {Add.class, Update.class})

private String name;

@Min(value = 1, message = "{Dog.age.lt1}", groups = {Add.class, Update.class})

private Integer age;

}

说明：@NotNull、@Min、@NotBlank 这些注解的使用方法，不在本文范围内。如果不熟悉，请查找资料学习即可。

其他说明：

@Data 注解是 \*\*Lombok\*\* 项目的注解，可以使我们不用再在代码里手动加 getter & setter。

在 Eclipse 和 IntelliJ IDEA 中使用时，还需要安装相关插件，这个步骤自行Google/Baidu 吧！

Lombok 使用方法见：[Java奇淫巧技之Lombok](http://blog.csdn.net/ghsau/article/details/52334762" \t "_blank)

SpringMVC 中对于 RESTFUL 的 Json 接口来说，数据绑定和校验，是这样的：

/\*\*

\* 使用 GlobalExceptionHandler 全局处理 Controller 层异常的示例

\* @param dog

\* @return

\*/

@PatchMapping(value = "")

AppResponse update(@Validated(Update.class) @RequestBody Dog dog){

AppResponse resp = new AppResponse();

// 执行业务

Dog newDog = dogService.update(dog);

// 返回数据

resp.setData(newDog);

return resp;

}

使用 @Validated + @RequestBody 注解实现。

当使用了 @Validated + @RequestBody 注解但是没有在绑定的数据对象后面跟上 Errors 类型的参数声明的话，Spring MVC 框架会抛出 MethodArgumentNotValidException 异常。

所以，在 GlobalExceptionHandler 中加上对 MethodArgumentNotValidException 异常的声明和处理，就可以全局处理数据校验的异常了！加完后的代码如下：

/\*\*

\* Created by kinginblue on 2017/4/10.

\* @ControllerAdvice + @ExceptionHandler 实现全局的 Controller 层的异常处理

\*/

@ControllerAdvice

public class GlobalExceptionHandler {

private static final Logger LOGGER = LoggerFactory.getLogger(GlobalExceptionHandler.class);

/\*\*

\* 处理所有不可知的异常

\* @param e

\* @return

\*/

@ExceptionHandler(Exception.class)

@ResponseBody

AppResponse handleException(Exception e){

LOGGER.error(e.getMessage(), e);

AppResponse response = new AppResponse();

response.setFail("操作失败！");

return response;

}

/\*\*

\* 处理所有业务异常

\* @param e

\* @return

\*/

@ExceptionHandler(BusinessException.class)

@ResponseBody

AppResponse handleBusinessException(BusinessException e){

LOGGER.error(e.getMessage(), e);

AppResponse response = new AppResponse();

response.setFail(e.getMessage());

return response;

}

/\*\*

\* 处理所有接口数据验证异常

\* @param e

\* @return

\*/

@ExceptionHandler(MethodArgumentNotValidException.class)

@ResponseBody

AppResponse handleMethodArgumentNotValidException(MethodArgumentNotValidException e){

LOGGER.error(e.getMessage(), e);

AppResponse response = new AppResponse();

response.setFail(e.getBindingResult().getAllErrors().get(0).getDefaultMessage());

return response;

}

}

注意到了吗，所有的 Controller 层的异常的日志记录，都是在这个 GlobalExceptionHandler 中进行记录。也就是说，Controller 层也不需要在手动记录错误日志了。

**五、总结**

本文主要讲 @ControllerAdvice + @ExceptionHandler 组合进行的 Controller 层上抛的异常全局统一处理。

其实，被 [@ExceptionHandler](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/ExceptionHandler.html) 注解的方法还可以声明很多参数，详见文档。

[@ControllerAdvice](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/ControllerAdvice.html) 也还可以结合 @InitBinder、@ModelAttribute 等注解一起使用，应用在所有被 @RequestMapping 注解的方法上，详见搜索引擎。

**六、附录**

本文示例代码已放到 [Github](https://github.com/kinginblue/KingBlogSamples/tree/master/exception-handler" \t "_blank)。