# 项目说明

# 版本一说明

# 单点登陆

# 全局异常

## 4.1 Spring异常处理

<https://www.cnblogs.com/junzi2099/p/7840294.html>

Spring 统一异常处理有 3 种方式，分别为：

1. 使用 @ ExceptionHandler 注解
2. 实现 HandlerExceptionResolver 接口
3. 使用 @controlleradvice 注解

### 4.1.1 使用@ ExceptionHandler 注解

使用该注解有一个不好的地方就是：进行异常处理的方法必须与出错的方法在同一个Controller里面。使用如下：

|  |
| --- |
| @Controller  public class GlobalController {  /\*\*  \* 用于处理异常的  \* @return  \*/  @ExceptionHandler({MyException.class})  public String exception(MyException e) {  System.out.println(e.getMessage());  e.printStackTrace();  return "exception";  }  @RequestMapping("test")  public void test() {  throw new MyException("出错了！");  }  } |

### 4.1.2 实现 HandlerExceptionResolver 接口

这种方式可以进行全局的异常控制。例如：

|  |
| --- |
| @Slf4j  @Component  public class ExceptionResolver implements HandlerExceptionResolver{  @Override  public ModelAndView resolveException(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e) {  log.error("{} Exception",httpServletRequest.getRequestURI(),e);  ModelAndView modelAndView = new ModelAndView(new MappingJacksonJsonView());  //当使用是jackson2.x的时候使用MappingJackson2JsonView，课程中使用的是1.9。  modelAndView.addObject("status", ResponseCode.ERROR.getCode());  modelAndView.addObject("msg","接口异常,详情请查看服务端日志的异常信息");  modelAndView.addObject("data",e.toString());  return modelAndView;  }  } |

### 4.1.3 使用 @ControllerAdvice+ @ ExceptionHandler 注解

上文说到 @ ExceptionHandler 需要进行异常处理的方法必须与出错的方法在同一个Controller里面。那么当代码加入了 @ControllerAdvice，则不需要必须在同一个 controller 中了。这也是 Spring 3.2 带来的新特性。从名字上可以看出大体意思是控制器增强。 也就是说，@controlleradvice + @ ExceptionHandler 也可以实现全局的异常捕捉。

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

|  |
| --- |
| @ControllerAdvice  @ResponseBody  public class WebExceptionHandle {  private static Logger logger = LoggerFactory.getLogger(WebExceptionHandle.class);  /\*\*  \* 400 - Bad Request  \*/  @ResponseStatus(HttpStatus.BAD\_REQUEST)  @ExceptionHandler(HttpMessageNotReadableException.class)  public ServiceResponse handleHttpMessageNotReadableException(HttpMessageNotReadableException e) {  logger.error("参数解析失败", e);  return ServiceResponseHandle.failed("could\_not\_read\_json");  }    /\*\*  \* 405 - Method Not Allowed  \*/  @ResponseStatus(HttpStatus.METHOD\_NOT\_ALLOWED)  @ExceptionHandler(HttpRequestMethodNotSupportedException.class)  public ServiceResponse handleHttpRequestMethodNotSupportedException(HttpRequestMethodNotSupportedException e) {  logger.error("不支持当前请求方法", e);  return ServiceResponseHandle.failed("request\_method\_not\_supported");  }  /\*\*  \* 415 - Unsupported Media Type  \*/  @ResponseStatus(HttpStatus.UNSUPPORTED\_MEDIA\_TYPE)  @ExceptionHandler(HttpMediaTypeNotSupportedException.class)  public ServiceResponse handleHttpMediaTypeNotSupportedException(Exception e) {  logger.error("不支持当前媒体类型", e);  return ServiceResponseHandle.failed("content\_type\_not\_supported");  }  /\*\*  \* 500 - Internal Server Error  \*/  @ResponseStatus(HttpStatus.INTERNAL\_SERVER\_ERROR)  @ExceptionHandler(Exception.class)  public ServiceResponse handleException(Exception e) {  if (e instanceof BusinessException){  return ServiceResponseHandle.failed("BUSINESS\_ERROR", e.getMessage());  }    logger.error("服务运行异常", e);  e.printStackTrace();  return ServiceResponseHandle.failed("server\_error");  }  } |

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！

继承 ResponseEntityExceptionHandler 类来实现针对 Rest 接口 的全局异常捕获，并且可以返回自定义格式：

|  |
| --- |
| @Slf4j  @ControllerAdvice  public class ExceptionHandlerBean extends ResponseEntityExceptionHandler {  /\*\*  \* 数据找不到异常  \* @param ex  \* @param request  \* @return  \* @throws IOException  \*/  @ExceptionHandler({DataNotFoundException.class})  public ResponseEntity<Object> handleDataNotFoundException(RuntimeException ex, WebRequest request) throws IOException {  return getResponseEntity(ex,request,ReturnStatusCode.DataNotFoundException);  }  /\*\*  \* 根据各种异常构建 ResponseEntity 实体. 服务于以上各种异常  \* @param ex  \* @param request  \* @param specificException  \* @return  \*/  private ResponseEntity<Object> getResponseEntity(RuntimeException ex, WebRequest request, ReturnStatusCode specificException) {  ReturnTemplate returnTemplate = new ReturnTemplate();  returnTemplate.setStatusCode(specificException);  returnTemplate.setErrorMsg(ex.getMessage());  return handleExceptionInternal(ex, returnTemplate,  new HttpHeaders(), HttpStatus.OK, request);  }  } |

### 4.1.4 区别

@ControllerAdvice只是标注当前bean是对Controller增强，具体是增强什么，则是配合@ExceptionHandler对异常进行精确处理。HandlerExceptionResolver则可以进行全局的异常控制，因此，所有的controller发生的异常都会进入其实现类，我们需要做通用处理。查看源码，可以发现HandlerExceptionResolver接口的抽象类AbstractHandlerMethodExceptionResolver和其子类ExceptionHandlerExceptionResolver一起组成使用@ExceptionHandler注释的方法进行异常解析的功能。现在基本都使用@ControllerAdvice+@ExceptionHandler注解来实现全局的异常捕获。

# 拦截器

## 5.1 使用Spring MVC拦截器实现权限统一校验

# 六、RESTful改造

# 七、定时关单