自：<https://www.cnblogs.com/domi22/p/9098192.html>

https://www.cnblogs.com/domi22/category/1224454.html

[**springboot全局异常处理**](https://www.cnblogs.com/domi22/p/9098192.html)

基于springboot的全局异常处理

1 编写ResultBuilder类

[复制代码](javascript:void(0);)

package com.test.domi.common.utils;  
  
import org.apache.commons.lang.StringUtils;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.autoconfigure.web.ErrorProperties;  
import org.springframework.boot.autoconfigure.web.ServerProperties;  
import org.springframework.core.Ordered;  
import org.springframework.http.HttpStatus;  
import org.springframework.validation.BindingResult;  
import org.springframework.validation.FieldError;  
import org.springframework.web.bind.MethodArgumentNotValidException;  
import org.springframework.web.servlet.HandlerExceptionResolver;  
import org.springframework.web.servlet.ModelAndView;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.util.List;  
  
public class ResultBuilder implements HandlerExceptionResolver,Ordered {  
  
 private static final Logger LOGGER = LoggerFactory.getLogger(ResultBuilder.class);  
 private static final String ERROR\_NAME = "fp.error";  
 private ErrorProperties errorProperties;  
  
 public ErrorProperties getErrorProperties() {  
 return errorProperties;  
 }  
  
 public ResultBuilder(ServerProperties serverProperties){  
 LOGGER.info("serverProperties:{}",serverProperties.getError());  
 this.errorProperties = serverProperties.getError();  
 }  
  
 public ResultInfo getErrorInfo(HttpServletRequest request){  
 return this.getErrorInfo(request,this.getError(request));  
 }  
  
 /\*\*  
 \* 全局异常返回处理  
 \* @param request  
 \* @param error  
 \* @return  
 \*/  
 public ResultInfo getErrorInfo(HttpServletRequest request,Throwable error){  
 ResultInfo resultInfo = new ResultInfo();  
 //根据不同的error获取错误信息  
 String resultCode = "";  
 StringBuffer msg = new StringBuffer();  
 if (error instanceof MethodArgumentNotValidException) {  
 //1 参数校验异常  
 resultCode = getString2((MethodArgumentNotValidException) error, resultCode, msg);  
 }else {  
 //3 httpStatu枚举code对应的异常  
 resultCode = getString3(request, msg);  
 }  
 resultInfo.setCode(resultCode);  
 resultInfo.setMessage(msg.toString());  
 resultInfo.setData((Object)null);  
 return resultInfo;  
 }  
  
 private String getString3(HttpServletRequest request, StringBuffer msg) {  
 msg.append(this.getHttpStatus(request).getReasonPhrase());  
 return String.valueOf(this.getHttpStatus(request).value());  
 }  
  
 private String getString2(MethodArgumentNotValidException error, String resultCode, StringBuffer msg) {  
 BindingResult bindingResult = error.getBindingResult();  
 if (bindingResult.hasErrors()) {  
 List<FieldError> list = bindingResult.getFieldErrors();  
 resultCode =ResultCode.CONNECT\_ERROR.getCode();  
 for (FieldError fieldError : list) {  
 msg.append(fieldError.getDefaultMessage() + ";");  
 }  
 }  
 return resultCode;  
 }  
  
 private String getString(Throwable error, StringBuffer msg) {  
 msg.append(error.getMessage());  
 return ResultCode.INSERT\_ERROR.getCode();  
 }  
  
 /\*\*  
 \* 拿到最根部的error,携带手动抛出的异常信息  
 \* @param request  
 \* @return  
 \*/  
 public Throwable getError(HttpServletRequest request){  
 Throwable error = (Throwable)request.getAttribute(ERROR\_NAME);  
 if (error == null) {  
 error = (Throwable)request.getAttribute("javax.servlet.error.exception");  
 }  
  
 if (error != null) {  
 //while (error instanceof ServletException && ((Throwable) error).getCause() != null) {  
 while (error instanceof Exception && ((Throwable) error).getCause() != null) {  
 error = ((Throwable) error).getCause();  
 }  
 } else {  
 String message = (String)request.getAttribute("javax.servlet.error.message");  
 if (StringUtils.isNotEmpty(message)) {  
 HttpStatus status = this.getHttpStatus(request);  
 message = "Unknown Exception With" + status.value() + " " + status.getReasonPhrase();  
 }  
  
 error = new Exception(message);  
 }  
  
 return (Throwable)error;  
 }  
  
 public HttpStatus getHttpStatus(HttpServletRequest request){  
 Integer statusCode = (Integer)request.getAttribute("javax.servlet.error.status\_code");  
 try {  
 return statusCode != null ? HttpStatus.valueOf(statusCode.intValue()) : HttpStatus.INTERNAL\_SERVER\_ERROR;  
 } catch (Exception var4) {  
 return HttpStatus.INTERNAL\_SERVER\_ERROR;  
 }  
 }  
  
 @Override  
 public int getOrder() {  
 return 0;  
 }  
  
 @Override  
 public ModelAndView resolveException(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e) {  
 httpServletRequest.setAttribute(ERROR\_NAME, e);  
 return null;  
 }  
}

[复制代码](javascript:void(0);)

2 编写ExceptionConfig类（传入ServerProperties ，实例化ResultBuilder。springboot中ErrorProperties类定义了异常自动映射路径@Value("${error.path:/error}")private String path = "/error"*）*

[复制代码](javascript:void(0);)

package com.test.domi.config;

import com.test.domi.common.system.ResultBuilder;

import org.springframework.boot.autoconfigure.web.ServerProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.Ordered;

import org.springframework.core.annotation.Order;

import javax.annotation.Resource;

@Configuration

@Order(Ordered.HIGHEST\_PRECEDENCE)

public class ExceptionConfig {

@Resource

private ServerProperties serverProperties;

@Bean

public ResultBuilder resultBuilder(){

return new ResultBuilder(serverProperties);

}

}

[复制代码](javascript:void(0);)

ErrorProperties：

[复制代码](javascript:void(0);)

//

// Source code recreated from a .class file by IntelliJ IDEA

// (powered by Fernflower decompiler)

//

package org.springframework.boot.autoconfigure.web;

import org.springframework.beans.factory.annotation.Value;

public class ErrorProperties {

@Value("${error.path:/error}")

private String path = "/error";

private ErrorProperties.IncludeStacktrace includeStacktrace;

public ErrorProperties() {

this.includeStacktrace = ErrorProperties.IncludeStacktrace.NEVER;

}

public String getPath() {

return this.path;

}

public void setPath(String path) {

this.path = path;

}

public ErrorProperties.IncludeStacktrace getIncludeStacktrace() {

return this.includeStacktrace;

}

public void setIncludeStacktrace(ErrorProperties.IncludeStacktrace includeStacktrace) {

this.includeStacktrace = includeStacktrace;

}

public static enum IncludeStacktrace {

NEVER,

ALWAYS,

ON\_TRACE\_PARAM;

private IncludeStacktrace() {

}

}

}

[复制代码](javascript:void(0);)

3 定义全局 异常Controller接管所有抛出的异常

[复制代码](javascript:void(0);)

package com.test.domi.controller;  
  
import com.test.domi.common.system.ResultBuilder;  
import com.test.domi.common.system.ResultInfo;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.autoconfigure.web.ErrorController;  
import org.springframework.http.MediaType;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.servlet.ModelAndView;  
import javax.annotation.Resource;  
import javax.servlet.http.HttpServletRequest;  
  
  
@Controller  
@RequestMapping("/error")  
public class GlobalErrorController implements ErrorController {  
  
 /\*\*  
 \* 1 ErrorController 接口的默认实现类是abstract：AbstractErrorController  
 \* 2 AbstractErrorController 的子类 BasicErrorController 才是真正干活儿的实现类（分html、json 两类处理）  
 \* 3 BasicErrorController 有 private final ErrorProperties errorProperties;属性  
 \* BasicErrorController的封装只能将状态码的提示信息返回前台，不能拿到手动抛异常的信息，因此需要实现HandlerExceptionResolver  
 \*/  
 private final static String DEFAULT\_ERROR\_VIEW = "/error";  
 private final static org.slf4j.Logger LOGGER = LoggerFactory.getLogger(GlobalErrorController.class);  
 /\*\*  
 \* ResultBuilder 实现 HandlerExceptionResolver 接口重写public ModelAndView resolveException(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e)  
 \* 通过httpServletRequest.setAttribute("fp.error", e);将Exception放到request中  
 \* 这种方法的好处是能拿到手动抛异常的信息  
 \*/  
 @Resource  
 private ResultBuilder resultBuilder;  
  
 /\*\*  
 \* 针对404的场景  
 \* @return  
 \*/  
 @Override  
 public String getErrorPath(){  
 return resultBuilder.getErrorProperties().getPath();  
 }  
  
 /\*\*  
 \* 如果请求头返回的类型是text/html，则返回到错误信息页面  
 \* @param request  
 \* @return  
 \*/  
 @RequestMapping(produces = MediaType.TEXT\_HTML\_VALUE)  
 @ResponseBody  
 public ModelAndView errorHtml(HttpServletRequest request) {  
 return new ModelAndView(DEFAULT\_ERROR\_VIEW,"errorInfo",resultBuilder.getErrorInfo(request));  
 }  
  
 /\*\*  
 \* 除了text/html的请求头信息，其它都返回json格式  
 \* @param request 请求对象  
 \* @return 错误信息字符串  
 \*/  
 @RequestMapping(produces = {MediaType.APPLICATION\_JSON\_VALUE})  
 @ResponseBody  
 public ResultInfo error(HttpServletRequest request){  
 return resultBuilder.getErrorInfo(request);  
 }  
  
  
  
}

[复制代码](javascript:void(0);)

配置完毕，后台的未被捕获的异常将从dao层到dervice层到controller层，然后被全局异常controller统一接管，封装之后返回给前台！