

## 尼斯湖水鬼的博客

SpringBoot开发, SpringCloud开发, Java开发



RSS订阅

## 凸 原 SpringBoot开发详解（六）--异常统一管理以及AOP的使用

13

2017年05月07日 19:07:01

阅读数: 9626

目录

收藏

使用切面管理异常的原因:

评论

今天的內容干货满满哦~并且是我自己在平时工作中的一些问题与解决途径,对实际开发的作用很大,好,闲言少叙,让我们开始吧~~

微信

我们先看一张错误信息在APP中的展示图:

微博

QQ

```
### Error querying database.  
Cause: com.mysql.jdbc.  
exceptions.jdbc4.  
CommunicationsException: The  
last packet successfully  
received from the server was 52,  
930,923 milliseconds ago. The  
last packet sent successfully to  
the server was 52,930,923  
milliseconds ago. is longer than  
the server configured value of  
'wait_timeout'. You should  
consider either expiring and/or  
testing connection validity  
before use in your application,  
increasing the server configured  
values for client timeouts, or  
using the Connector/J  
connection property  
'autoReconnect=true' to avoid  
this problem.  
### The error may exist in "/usr/  
local/jboss/jboss-as-7.1.1.Final/
```

## 个人资料



尼斯湖水鬼

关注

原创

12

粉丝

185

喜欢

23

评论

43

等级: 博客 4

访问: 13万+

积分: 1090

排名: 4万+



双证在职研究生

## 最新文章

- SpringBoot开发详解（十二）--SpringBoot中执行定时任务
- SpringBoot开发详解（十一）--Redis在SpringBoot中的整合
- SpringBoot开发详解（十）--使用JPA访问数据库下篇及使用Page进行数据分页
- SpringBoot开发详解（九）--使用JPA访问数据库上篇
- SpringBoot开发详解（八）--使用Swagger 2构建API文档

## 个人分类

spring boot

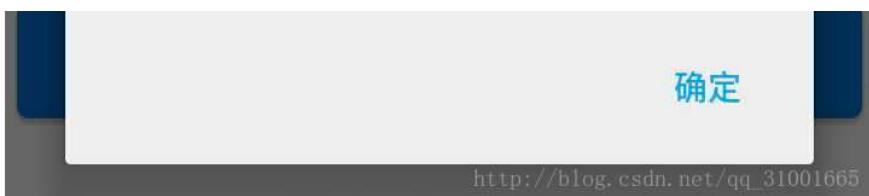
12篇

## 归档

- 2017年7月 2篇
- 2017年6月 1篇
- 2017年5月 5篇
- 2017年4月 3篇
- 2017年2月 1篇

## 热门文章

- SpringBoot开发详解（五）--Controller接收参数以及参数校验 阅读量: 55581
- SpringBoot开发详解（三）--SpringBoot配置文件YML注意事项 阅读量: 17424
- SpringBoot开发详解（十二）--SpringBoot中执行定时任务 阅读量: 12598



是不是体验很差，整个后台错误信息都在APP上打印了。

作为后台开发人员，我们总是在不停的写各种接口提供给前端调用，然而不可避免的，当后台出现BUG时，前端总是丑陋的讲错误信息直接暴露给用户，这样的用户体验想必是相当差的（不过后台开发一看就知道问题出现在哪里）。同时，在解决BUG时，我们总是要问前端拿到参数去调适，排除各种问题（网络，Json体错误，接口名写错.....BaLa.....BaLa.....BaLa）。在不考虑前端容错的情况下。我们自己后台有没有优雅的解决这个问题的方法呢，今天这篇我们就来使用AOP统一对异常进行记录以及返回。

## SpringBoot引入AOP

在SpringBoot中引入AOP是一件很方便的事，和其他引入依赖一样，我们只需要在POM中引入starter就可以了：

```
1 <!--spring切面aop依赖-->
2 <dependency>
3   <groupId>org.springframework.boot</groupId>
4   <artifactId>spring-boot-starter-aop</artifactId>
5 </dependency>
```

## 返回体报文定义

接下来我们先想一下，一般我们返回体是什么样子的呢？或者你觉得一个返回的报文应该具有哪些特征。

- 成功标示：可以用boolean型作为标示位。
- 错误代码：一般用整型作为标示位，罗列的越详细，前端的容错也就能做的更细致。
- 错误信息：使用String作为错误信息的描述，留给前端是否展示给用户或者进入其他错误流程的使用。
- 结果集：在无错误信息的情况下所得到的正确数据信息。一般是个Map，前端根据Key取值。

以上是对一个返回体报文一个粗略的定义了，如果再细致点，可以使用签名进行验签功能活着对明文数据进行对称加密等等。这些我们今天先不讨论，我们先完成一个能够使用的接口信息定义。

我们再对以上提到这些信息做一个完善，去除冗余的字段，对差不多的类型进行合并于封装。这样的想法下，我们创建一个返回体报文的实体类。

```
1 public class Result<T> {
2
3     // error_code 状态值: 0 极为成功，其他数值代表失败
4     private Integer status;
5
6     // error_msg 错误信息，若status为0时，为success
7     private String msg;
8
9     // content 返回体报文的出参，使用泛型兼容不同的类型
10    private T data;
11
12    public Integer getStatus() {
13        return status;
14    }
15
16    public void setStatus(Integer code) {
17        this.status = code;
18    }
19
20    public String getMsg() {
21        return msg;
22    }
23
24    public void setMsg(String msg) {
25        this.msg = msg;
26    }
27
28    public T getData(Object object) {
29        return data;
```

SpringBoot开发详解（六）-- 异常统一管理

以及AOP的使用

阅读量：9583

SpringBoot开发详解（十一）-- Redis在SpringBoot中的整合

阅读量：8411

### 最新评论

SpringBoot开发详解（十一）...

xia\_zai\_pin\_dao\_：按照博主的思路写下来后，程序运行是正确的。但是在redis的客户端却是乱码，而且看不到值。但是用代...

SpringBoot开发详解（六）...

qq\_35981283：很好的文章

SpringBoot开发详解（一）...

qq\_31211685：很详细，赞一个

SpringBoot开发详解（六）...

weixin\_42096420：楼主写得很好，感谢，接下来的我都要看看，太棒了

SpringBoot开发详解（五）...

sss1342746626：感谢楼主。。终于发现一个接收数据好用的方法了。。。赞



### 联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

✉ QQ客服 客服论坛

关闭 报错 广告报错 帮助 反馈

```

30 }
31
32 public void setData(T data) {
33     this.data = data;
34 }
35
36 public T getData() {
37     return data;
38 }
39
40 @Override
41 public String toString() {
42     return "Result{" +
43         "status=" + status +
44         ", msg='" + msg + '\'' +
45         ", data=" + data +
46         '}';
47 }

```

现在我们已经有一个返回体报文的定义了，那接下来我们可以来创建一个枚举类，来记录一些我们已知的错误信息，可以在代码中直接使用。

```

1 public enum ExceptionEnum {
2     UNKNOW_ERROR(-1,"未知错误"),
3     USER_NOT_FIND(-101,"用户不存在"),
4 ;
5
6     private Integer code;
7
8     private String msg;
9
10    ExceptionEnum(Integer code, String msg) {
11        this.code = code;
12        this.msg = msg;
13    }
14
15    public Integer getCode() {
16        return code;
17    }
18
19    public String getMsg() {
20        return msg;
21    }
22 }

```

我们在这里把对于不再预期内的错误统一设置为-1，未知错误。以避免返回给前端大段大段的错误信息。

接下来我们只需要创建一个工具类在代码中使用：

```

1 public class ResultUtil {
2
3     /**
4      * 返回成功，传入返回体具体参数
5      * @param object
6      * @return
7      */
8     public static Result success(Object object){
9         Result result = new Result();
10        result.setStatus(0);
11        result.setMsg("success");
12        result.setData(object);
13        return result;
14    }
15
16    /**
17     * 提供给部分不需要出参的接口
18     * @return
19     */
20    public static Result success(){
21        return success(null);
22    }
23
24    /**
25     * 自定义错误信息
26     */
27 }

```

```

26     * @param code
27     * @param msg
28     * @return
29     */
30    public static Result error(Integer code, String msg) {
31        Result result = new Result();
32        result.setStatus(code);
33        result.setMsg(msg);
34        result.setData(null);
35        return result;
36    }
37
38    /**
39     * 返回异常信息，在已知的范围内
40     * @param exceptionEnum
41     * @return
42     */
43    public static Result error(ExceptionEnum exceptionEnum) {
44        Result result = new Result();
45        result.setStatus(exceptionEnum.getCode());
46        result.setMsg(exceptionEnum.getMsg());
47        result.setData(null);
48        return result;
49    }
50 }

```

以上我们已经可以捕获代码中那些在编码阶段我们已知的错误了，但是却无法捕获程序出的未知异常信息。我们的代码应该写得漂亮一点，虽然很多时候我们会说时间太紧了，等之后我再来好好优化。可事实是，我们再也不会回来看这些代码了。项目总是一个接着一个，时间总是不够用的。如果真的需要你完善重构原来的代码，那你一定会非常痛苦，死得相当难受。所以，在第一次构建时，就将你的代码写完善了。

一般系统抛出的错误是不含错误代码的，除去部分的404, 400, 500错误之外，我们如果想把错误代码定义的更细致，就需要自己继承RuntimeException这个类后重新定义一个构造方法来定义我们自己的错误信息：

```

1  public class DescribeException extends RuntimeException{
2
3      private Integer code;
4
5      /**
6       * 继承exception，加入错误状态值
7       * @param exceptionEnum
8       */
9      public DescribeException(ExceptionEnum exceptionEnum) {
10         super(exceptionEnum.getMsg());
11         this.code = exceptionEnum.getCode();
12     }
13
14     /**
15      * 自定义错误信息
16      * @param message
17      * @param code
18      */
19     public DescribeException(String message, Integer code) {
20         super(message);
21         this.code = code;
22     }
23
24     public Integer getCode() {
25         return code;
26     }
27
28     public void setCode(Integer code) {
29         this.code = code;
30     }
31 }

```

同时，我们使用一个Handle来把Try, Catch中捕获的错误进行判定，是一个我们已知的错误信息，还是一个未知的错误信息，如果是未知的错误信息，那我们就用log记录它，便于之后的查找和解决：

```

1  @ControllerAdvice
2  public class ExceptionHandle {
3
4      private final static Logger LOGGER = LoggerFactory.getLogger(ExceptionHandle.class);
5

```

```

6     /**
7      * 判断错误是否是已定义的已知错误，不是则由未知错误代替，同时记录在log中
8      * @param e
9      * @return
10     */
11    @ExceptionHandler(value = Exception.class)
12    @ResponseBody
13    public Result exceptionGet(Exception e){
14        if(e instanceof DescribeException){
15            DescribeException MyException = (DescribeException) e;
16            return ResultUtil.error(MyException.getCode(),MyException.getMessage());
17        }
18
19        LOGGER.error("【系统异常】 {}",e);
20        return ResultUtil.error(ExceptionEnum.UNKNOW_ERROR);
21    }
22}

```

这里我们使用了 `@ControllerAdvice`，使Spring能加载该类，同时我们将所有捕获的异常统一返回结果Result这个实体。

此时，我们已经完成了对结果以及异常的统一返回管理，并且在出现异常时，我们可以不返回错误信息给前端，而是用未知错误进行代替，只有查看log我们才会知道真实的错误信息。

可能有小伙伴要问了，说了这么久，并没有使用到AOP啊。不要着急，我们继续完成我们剩余的工作。

我们使用接口若出现了异常，很难知道是谁调用接口，是前端还是后端出现的问题导致异常的出现，那这时，AOP久发挥了作用了，我们之前已经引入了AOP的依赖，现在我们编写一个切面类，切点如何配置不需要我多说了吧：

```

1  @Aspect
2  @Component
3  public class HttpAspect {
4
5      private final static Logger LOGGER = LoggerFactory.getLogger(HttpAspect.class);
6
7      @Autowired
8      private ExceptionHandle exceptionHandle;
9
10     @Pointcut("execution(public * com.zzp.controller.*.*(..))")
11     public void log(){
12
13     }
14
15     @Before("log()")
16     public void doBefore(JoinPoint joinPoint){
17         ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder
18         HttpServletRequest request = attributes.getRequest();
19
20         //url
21         LOGGER.info("url={}",request.getRequestURL());
22         //method
23         LOGGER.info("method={}",request.getMethod());
24         //ip
25         LOGGER.info("id={}",request.getRemoteAddr());
26         //class_method
27         LOGGER.info("class_method={},joinPoint.getSignature().getDeclaringTypeName() + "," +
28         //args[]
29         LOGGER.info("args={}",joinPoint.getArgs());
30     }
31
32     @Around("log()")
33     public Object doAround(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
34         Result result = null;
35         try {
36
37             } catch (Exception e) {
38                 return exceptionHandle.exceptionGet(e);
39             }
40             if(result == null){
41                 return proceedingJoinPoint.proceed();
42             }else {
43                 return result;
44             }
45         }
46
47     @AfterReturning(pointcut = "log()",returning = "object")//打印输出结果

```

```

48     public void doAfterReturning(Object object){
49         LOGGER.info("response={}",object.toString());
50     }
51 }

```

我们使用@Aspect来声明这是一个切面，使用@Before来定义切面所需要切入的位置，这里我们是对每一个HTTP请求都需要切入，在进入方法之前我们使用@Before记录了调用的接口URL，调用的方法，调用方的IP地址以及输入的参数等。在整个接口代码运作期间，我们使用@Around来捕获异常信息，并用之前定义好的Result进行异常的返回，最后我们使用@AfterReturning来记录我们的出参。

以上全部，我们就完成了异常的统一管理以及切面获取接口信息，接下来我们重新写一个ResultController来测试一下：

```

1  @RestController
2  @RequestMapping("/result")
3  public class ResultController {
4
5      @Autowired
6      private ExceptionHandle exceptionHandle;
7
8      /**
9       * 返回体测试
10      * @param name
11      * @param pwd
12      * @return
13      */
14     @RequestMapping(value = "/getResult",method = RequestMethod.POST)
15     public Result getResult(@RequestParam("name") String name, @RequestParam("pwd") String pw
16             Result result = ResultUtil.success();
17             try {
18                 if (name.equals("zzp")){
19                     result = ResultUtil.success(new UserInfo());
20                 }else if (name.equals("pzz")){
21                     result = ResultUtil.error(ExceptionEnum.USER_NOT_FIND);
22                 }else{
23                     int i = 1/0;
24                 }
25             }catch (Exception e){
26                 result = exceptionHandle.exceptionGet(e);
27             }
28             return result;
29     }
30 }

```

在上面我们设计了一个controller，如果传入的name是zzp的话，我们就返回一个用户实体类，如果传入的是pzz的话，我们返回一个没有该用户的错误，其他的，我们让他抛出一个by zero的异常。

我们用POSTMAN进行下测试：

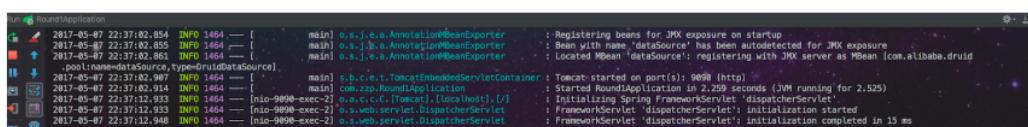
The screenshot shows the POSTMAN interface with the following details:

- Method:** POST
- URL:** http://localhost:9090/result/getResult
- Body (JSON):**
  - Key: name, Value: zzp
  - New key: value, Value: value
- Response:**
  - Status: 200 OK
  - Time: 153 ms
  - Size: 210 B
  - Content (Pretty):

```

1  {
2     "status": 0,
3     "msg": "success",
4     "data": {
5         "tel": null,
6         "nickName": null,
7         "passWord": null
8     }
9 }

```



```

2017-05-07 22:37:12,975 INFO 1464 --- [nio-9990-exec-2] com.zzp.util.HttpAspect : method:POST
2017-05-07 22:37:12,975 INFO 1464 --- [nio-9990-exec-2] com.zzp.util.HttpAspect : id:@0:0:0:0:0:1
2017-05-07 22:37:12,975 INFO 1464 --- [nio-9990-exec-2] com.zzp.util.HttpAspect : class_method=com.zzp.controller.ResultController.getResult
2017-05-07 22:37:12,977 INFO 1464 --- [nio-9990-exec-2] com.zzp.util.HttpAspect : args=zpz
2017-05-07 22:37:12,979 INFO 1464 --- [nio-9990-exec-2] com.zzp.util.HttpAspect : responseResult(status=0, msg='success', data=co...@ip:port/UserInfo@34f4e77)

```

我们可以看到，前端收到的返回体报文已经按我们要求同意了格式，并且在控制台中我们打印出了调用该接口的一些接口信息，我们继续测试另外两个会出现错误情况的请求：

POST http://localhost:9090/result/getResult

Authorization Headers (1) Body **Body** Pre-request Script Tests Cookies Code

form-data x-www-form-urlencoded raw binary

Key	Value
<input checked="" type="checkbox"/> name	pzz

Body Cookies (1) Headers (3) Tests

Pretty Raw Preview JSON

```

1 - [
2   "status": -101,
3   "msg": "用户不存在",
4   "data": null
5 ]

```

Status: 200 OK Time: 139 ms Size: 181 B

http://blog.csdn.net/qq\_31001665

POST http://localhost:9090/result/getResult

Authorization Headers (1) Body **Body** Pre-request Script Tests Cookies Code

form-data x-www-form-urlencoded raw binary

Key	Value
<input checked="" type="checkbox"/> name	pzz

Body Cookies (1) Headers (3) Tests

Pretty Raw Preview JSON

```

1 - [
2   "status": -1,
3   "msg": "未知错误",
4   "data": null
5 ]

```

Status: 200 OK Time: 15 ms Size: 178 B

http://blog.csdn.net/qq\_31001665

```

2017-05-07 22:39:58.987 INFO 1495 --- [nio-9990-exec-3] com.zzp.util.HttpAspect : method:POST
2017-05-07 22:39:58.987 INFO 1495 --- [nio-9990-exec-3] com.zzp.util.HttpAspect : id:@0:0:0:0:0:1
2017-05-07 22:39:58.987 INFO 1495 --- [nio-9990-exec-3] com.zzp.util.HttpAspect : class_method=com.zzp.controller.ResultController.getResult
2017-05-07 22:39:58.987 INFO 1495 --- [nio-9990-exec-3] com.zzp.util.HttpAspect : args=zpz
2017-05-07 22:39:58.987 INFO 1495 --- [nio-9990-exec-3] com.zzp.util.HttpAspect : exceptionHandle : {异常处理}

```

```

java.lang.ArithmaticException: / by zero
at com.zzp.controller.ResultController.getResult(ResultController.java:35) ~[classes/:na]
at com.zzp.controller.ResultController$$FastClassBySpringCGLIB$$dab73a0.invoke(MethodGenerated) ~[classes/:na]
at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:248) [spring-core-4.3.7.RELEASE.jar:4.3.7.RELEASE]
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:157) [spring-aop-4.3.7.RELEASE.jar:4.3.7.RELEASE]
at org.springframework.aop.framework.adapter.MethodBeforeAdviceInterceptor.invoke(MethodBeforeAdviceInterceptor.java:52) [spring-aop-4.3.7.RELEASE.jar:4.3.7.RELEASE]
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:179) [spring-aop-4.3.7.RELEASE.jar:4.3.7.RELEASE]
at org.springframework.aop.framework.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:85) [spring-aop-4.3.7.RELEASE.jar:4.3.7.RELEASE]

```

我们可以看到，如是我们之前在代码中定义完成的错误信息，我们可以直接返回错误码以及错误信息，如果是程序出现了我们在编码阶段不曾预想到的错误，则统一返回未知错误，并在log中记录真实错误信息。

以上就是我们统一管理结果集以及使用切面来记录接口调用的一些真实情况，在平时的使用中，大家要清楚切点的优先级以及在不同的切点位置该使用哪些注解来帮助我们完成开发，并且在切面中，如果遇到同步问题该如何解决等等。希望这篇文章能让你更好的思考如何设计好接口，我们在实际开发中又是怎样一步步完善我们的功能与代码的。也希望大家能好好梳理这些内容，如果有疑惑的地方，还请留言，我如果看到，一定会解答的。这里预告下：下周，我们将使用ORM框架来做数据库的交互~~~

以上所有的代码我已经上传到GitHub

## round1-springboot

如果心急的小伙伴也可以去clone我已完成的项目，这个项目中把一些常用功能都写了，并且都写注释啦！！！

## MySpringBoot

版权声明：本文为博主原创文章，未经博主允许不得转载。 [https://blog.csdn.net/qq\\_31001665/article/details/71357825](https://blog.csdn.net/qq_31001665/article/details/71357825)

文章标签：[aop](#) [异常](#) [springboot](#)

个人分类：[spring boot](#)



那都不是事！学会AI这些技能，让你叱咤人工智能领域！

时代趋势，顺流而上！掌握人工智能，避免就业危机！成为稀缺人才我们有方法...

◎ 12522

[查看更多>>](#)

想对作者说点什么？ [我来说一句](#)

月未明 2018-05-08 15:31:28 #6楼

很好的文章

weixin\_42096420 2018-04-28 14:57:56 #5楼

楼主写得很好，感谢，接下来的我都要看看，太棒了

deamonCORW 2018-03-22 21:19:42 #4楼

谢谢博主，学习了

[查看 6 条热评](#)

## SpringBoot全局异常处理

欢迎使用Markdown编辑器写博客本Markdown编辑器使用StackEdit修改而来，用它写博客，将会带来全新的体验哦： Markdown和扩展Markdown简洁的语法 代码块高亮 图片链接和...

tianyaleixiaowu 2017-04-12 17:18:44 阅读数：10868

## SpringBoot之集成Spring AOP

在开始之前，我们先把需要的jar包添加到工程里。新增Maven依赖如下： org.springframework.boot sprin...

zknxx 2016-11-20 15:09:55 阅读数：26680

## springboot统一异常处理 - CSDN博客

springboot提供了一个默认映射 /error,当处理中抛出异常之后,会转到该请求中处理,并且该请求有一个全局的错误页面用来展示异常内容 也可以自己创建全局异常处理类,...

2018-2-13

## SpringBoot之统一异常处理 - CSDN博客

我们在做Web应用的时候,请求处理过程中发生错误是非常常见的情况。 Spring Boot提供了一个默认的映射:/error,当处理中抛出异常之后,会转到该请求中处理,并且该请求...

2018-4-17

## Spring Boot 使用 Spring AOP 处理拦截器

在 spring boot 中，简单几步，使用 spring AOP 实现一个拦截器。

ClementAD 2016-07-26 14:08:13 阅读数：43403

## Springboot(统一异常处理) - CSDN 博客

logger.error("系统异常={}",e); ...[springboot统一异常](#)处理机制 1.自定义异常的处理在controller层捕获的自定义异常时...

2018-4-19

## SpringBoot初始教程之统一异常处理(三) - CSDN 博客

在日常开发中发生了[异常](#),往往是需要通过一个[统一的异常](#)处理处理所有[异常](#),来保证客户端能够收到友好的提示。[SpringBoot](#)在页面发生[异常](#)的时候会自动把请求转到/error...

2018-5-6

## springboot配置aop切面详解

1 引入依赖 org.springframework.boot spring-boot-starter-aop 2 是否需要在程序主类中增加@EnableAspectJAut...

wabiaozia 2018-01-28 22:54:57 阅读数：563

## spring boot aop 的使用

spring boot aop 的使用 1.aop 的官网介绍 AOP concepts Let us begin by defining some central AOP concepts and...

long290046464 2017-08-05 19:04:15 阅读数：1530

## springboot统一异常处理 - CSDN 博客

我们在做Web应用的时候,请求处理过程中发生错误是非常常见的情况。[Spring Boot](#)提供了一个默认的映射:/error,当处理中抛出[异常](#)之后,会转到该请求中处理,并且该请求有...

2018-4-10

## SpringBoot之统一异常处理 - CSDN 博客

不良信息举报 举报内容: [SpringBoot之统一异常](#)处理 举报原因: 色情 政治 抄袭 广告 招聘 骂人 其他 原文地址: 原因补充: 最多只允许输入30个字...

2018-5-15

## springBoot aop

在[使用springboot](#)中[使用AOP](#)时需要在自己需要在方法上添加@Component 和 @Aspect 两个注解 //匹配cn.com.epicc.ecommerce.wcar.contr...

github\_39557053 2017-12-13 15:14:18 阅读数：88

## spring boot 教程(四) 统一异常处理 - CSDN 博客

我们在做Web应用的时候,请求处理过程中发生错误是非常常见的情况。[Spring Boot](#)提供了一个默认的映射:/error,当处理中抛出[异常](#)之后,会转到该请求中处理,并且该请求有...

2018-4-11

## springboot统一异常处理机制 - CSDN 博客

springboot提供了默认的[统一异常](#)处理,basicErrorController,这个controller提供了默认了错误处理方法,包括错误跳转的路径和渲染方法。因为默认的错误处理方法可能会不适合...

2018-5-5

## spring boot 中使用AOP

spring boot 中[使用AOP](#), 需要在pom.xml中添加依赖: org.springframework.boot spri...

xzmeasy 2017-07-25 21:17:23 阅读数：98

## springboot干货——（十八）AOP

AOP这个名词相信大家都很陌生，尤其是在面试的过程中，面试官多多少少都会问到一些关于他的问题，这玩意儿有用吗？答案是必然的，只是在日常业务逻辑中用的不多，一般像想在某个写好的代码之前插入一些内容，这种...

## SpringBoot 统一异常处理 - CSDN博客

在用spring Boot做web后台时,经常会出现异常,如果每个异常都自己去处理很麻烦,所以我们创建一个全局异常处理类来统一处理异常一

2018-4-18

## SpringBoot统一异常处理 - CSDN博客

有个完善的项目,应该尽量保证自己自己的接口出参数据接口应当尽量一致,包括出现异常也应该如此,那么我们在springboot里面如何处理呢? 1.首先定义一个接口状态的枚举类...

2018-4-14

## SpringBoot中AOP的配置

AOP目的: 面向切面编程 (aspect-oriented programming, AOP) 主要实现的目的是针对业务处理过程中的切面进行提取, 诸如日志、事务管理和安全这样的系统服务, 从而使得业务逻辑...

qq\_29614459 2017-02-04 11:23:33 阅读数: 7065

## SpringBoot中aop的使用

步骤 编写使用注解的被拦截类, 加了这个注解的类或者方法就会被拦截 使用@Aspect声明一个切面, 并通过@Component让此切面成为Spring容器管理的Bean 使用@After、@Before、...

limenghua9112 2016-12-18 13:41:10 阅读数: 2748

## 第四十四章： 基于SpringBoot & AOP完成统一资源自动查询映射

本章内容比较偏向系统设计方面, 简单的封装就可以应用到系统中使用, 从而提高我们的编码效率以及代码的可读性。统一资源在系统内是不可避免的模块, 资源分类也有很多种, 比较常见如: 图片资源、文本资源、视频资源等...

weixin\_42033269 2018-04-23 11:19:16 阅读数: 42

## spring boot-aop的使用

一、添加aop starter依赖

liuchuanhong1 2017-02-14 22:32:02 阅读数: 3726

## springboot中aop应用

aop的理解: 我们传统的编程方式是垂直化的编程, 即A->B->C->D这么下去, 一个逻辑完毕之后执行另外一段逻辑。但是AOP提供了另外一种思路, 它的作用是在业务逻辑不知情 (即业务逻辑不需要做任何的改动...)

zlxls 2018-03-30 13:48:29 阅读数: 34

## 高效、准确、低成本的文字识别api

证件、文件一键扫描, 准确率高达99%。经过海量用户和多种复杂场景考验



## springboot使用AOP

1、添加依赖 org.springframework.boot spring-boot-starter-aop 2、设置切面 @Aspect @Configuration public...

qq\_21033663 2017-06-12 22:06:44 阅读数: 1940

## springboot基于AOP将web请求写入日志

2016年11月25日 11KB 下载



## Spring Boot 开启AOP的方法

Spring Boot与普通的Spring JavaConfig项目还有有所区别的, 如果出现无效的问题, 八成是加了多余的配置导致混乱。Spring Boot开启AOP的方法其实相对简单, 分以下两个...

gefangshuai 2015-12-16 15:27:50 阅读数: 5083

## 46. Spring Boot中使用AOP统一处理Web请求日志【从零开始学Spring Boot】

在之前一系列的文章中都是提供了全部的代码, 在之后的文章中就提供核心的代码进行讲解。有什么问题大家可以给我留言或者加我QQ, 进行咨询。 AOP为Aspect Oriented Progra...

linxingliang 2016-08-26 08:53:44 阅读数: 8434

## SpringBoot 实战之全局异常捕获、全局参数校验和全局异常处理

package com.eparty.ccp.order.aop; import com.joindata.inf.common.util.log.Logger; import org.apache...

 qq\_32719003 2017-05-08 11:18:17 阅读数: 3894

### 高端网站设计

一个成功的高端网站建设对访客有很大影响



百度广告

### Spring Boot实战之全局异常捕获 实现参数异常检查返回统一错误信息

Spring Boot实战之全局异常处理实现参数非法性检查 在一个项目中的异常我们都会统一进行处理的，本文实现对接口中传入的参数进行非法性检查，当参数非法时，抛出异常，然后返回统一的...

 sun\_t89 2016-07-28 17:07:02 阅读数: 24682

### Spring Boot 全局异常处理

一、ErrorPageRegistrar 异常处理 1.1ErrorPageRegistrar 详解 Spring Boot 为我们提供了统一异常处理方式。通过实现ErrorPageRegistr...

 codejas 2018-02-14 15:15:31 阅读数: 227

### springboot统一异常处理机制