Atitit springboot 全局异常处理

## @ControllerAdvice 不起作用

**public** **class** ExceptionHandle {

@ExceptionHandler(value = Exception.**class**)

@ResponseBody

估计是因为已经有了一个，不能覆盖，，

## 所以使用aop法

 上述配置为AOP配置代码片段，其中expression部分为定义切点的表达式部分，如下：

execution(\* com.loongshawn.method.ces..\*.\*(..))

  注意：markdown中符号“\*”是加粗，因此输出“\*”符号需要进行转义“\*”。

  表达式结构解释如下：

标识符 含义

execution() 表达式的主体

第一个“\*”符号 表示返回值的类型任意

com.loongshawn.method.ces AOP所切的服务的包名，即，需要进行横切的业务类

包名后面的“..” 表示当前包及子包

第二个“\*” 表示类名，\*即所有类

.\*(..) 表示任何方法名，括号表示参数，两个点表示任何参数类型

————————————————

版权声明：本文为CSDN博主「loongshawn」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/loongshawn/article/details/72303040

## 定义切面**MyAllAspect**

***@Component***

***@Aspect*** // AOP 切面u

***@SuppressWarnings***("all")

**public** **class** **MyAllAspect** {

\* 后置异常通知 不是这个，这个不行

\* **@param** jp

\*/

@AfterThrowing(value = "execution(\* aoppkg.UserService.login(..))", )

**public** **void** throwss(JoinPoint jp){

System.***out***.println("方法异常时执行.....");

}

//   切入点（Pointcut）：用于定义通知应该切入到哪些连接点上

***@AfterReturning***(value = "execution(\* aoppkg.UserService.login(..))", returning = "result")

**public** **Object** **afterReturning**(JoinPoint joinPoint, **Object** result) {

**System**.***out***.println("log.....");

**return** result;

}

/\*\*

\* 环绕通知,环绕增强，相当于MethodInterceptor

\* **@param** pjp

\* **@return**

\*/

@Around("pointcutname()")

**public** Object arround(ProceedingJoinPoint pjp) {

**try** {

Object o = pjp.proceed();

**return** o;

} **catch** (Throwable e) {

**return** **new** Result("500", "服务端调用异常", e);

// return null;

}

}

/\*\*

# codee

package com.kok.sport.utils.constant;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.Signature;

import org.aspectj.lang.annotation.\*;

import org.aspectj.lang.reflect.MethodSignature;

import org.springframework.stereotype.Component;

import org.springframework.web.context.request.RequestContextHolder;

import org.springframework.web.context.request.ServletRequestAttributes;

import com.alibaba.fastjson.JSON;

import com.kok.sport.base.Result;

import com.kok.sport.controller.ApiController;

import javax.servlet.http.HttpServletRequest;

import java.util.Arrays;

/\*\*

\* @Auther: cookie

\* @Date: 2018/7/27 10:17

\* @Description: 使用AOP统一处理Web请求日志

\*/

@Aspect

@Component

public class ErrorProcessAspect {

public static void main(String[] args) throws Exception {

System.out.println(new ApiController().queryPage());;

}

/\*\*

\* 指定切点

\* 匹配 com.example.demo.controller包及其子包下的所有类的所有方法

\*/

@Pointcut("execution(public \* com.kok.sport.controller.ApiController.\*(..))")

public void pointcutname(){

}

/\*\*

\* 前置通知，方法调用前被调用

\* @param joinPoint

\*/

@Before("pointcutname()")

public void doBefore(JoinPoint joinPoint){

System.out.println("我是前置通知!!!");

//获取目标方法的参数信息

Object[] obj = joinPoint.getArgs();

Signature signature = joinPoint.getSignature();

//代理的是哪一个方法

System.out.println("方法："+signature.getName());

//AOP代理类的名字

System.out.println("方法所在包:"+signature.getDeclaringTypeName());

//AOP代理类的类（class）信息

signature.getDeclaringType();

MethodSignature methodSignature = (MethodSignature) signature;

String[] strings = methodSignature.getParameterNames();

System.out.println("参数名："+Arrays.toString(strings));

System.out.println("参数值ARGS : " + Arrays.toString(joinPoint.getArgs()));

// 接收到请求，记录请求内容

ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();

HttpServletRequest req = attributes.getRequest();

// 记录下请求内容

System.out.println("请求URL : " + req.getRequestURL().toString());

System.out.println("HTTP\_METHOD : " + req.getMethod());

System.out.println("IP : " + req.getRemoteAddr());

System.out.println("CLASS\_METHOD : " + joinPoint.getSignature().getDeclaringTypeName() + "." + joinPoint.getSignature().getName());

}

/\*\*

\* 处理完请求返回内容

\* @param ret

\* @throws Throwable

\*/

@AfterReturning(returning = "ret", pointcut = "pointcutname()")

public void doAfterReturning(Object ret) throws Throwable {

// 处理完请求，返回内容

System.out.println("方法的返回值 : " + ret);

}

/\*\*

\* 后置异常通知

\* @param jp

\*/

@AfterThrowing("pointcutname()")

public void throwss(JoinPoint jp){

System.out.println("方法异常时执行.....");

}

/\*\*

\* 后置最终通知,final增强，不管是抛出异常或者正常退出都会执行

\* @param jp

\*/

@After("pointcutname()")

public void after(JoinPoint jp){

}

/\*\*

\* 环绕通知,环绕增强，相当于MethodInterceptor

\* @param pjp

\* @return

\*/

@Around("pointcutname()")

public Object arround(ProceedingJoinPoint pjp) {

try {

Object o = pjp.proceed();

return o;

} catch (Throwable e) {

return new Result("500", "服务端调用异常", e);

// return null;

}

}

}

Spring面向切面编程(AOP-execution表达式)\_Java\_loongshawn的博客-CSDN博客