# 9-SpringMVC基于注解使用：异常处理

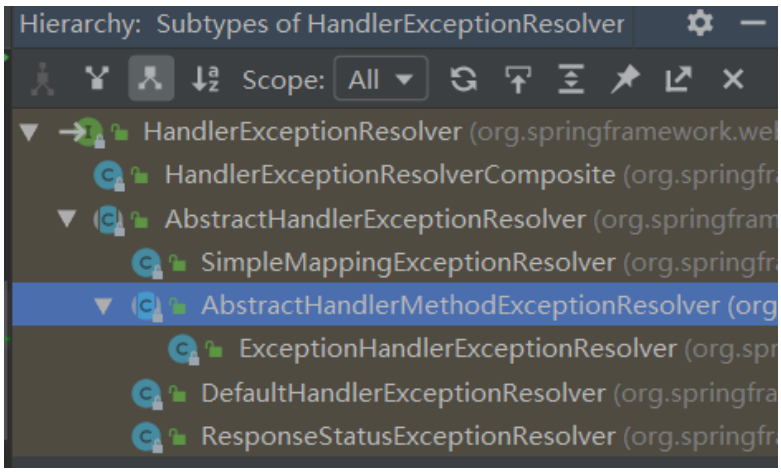


## 1、内置异常处理解析器

在J2EE项目的开发中，不管是对底层的数据库操作过程，还是业务层的处理过程，还是控制层的处理过程，都不可避免会遇到各种可预知的、不可预知的异常需要处理。每个过程都单独处理异常，系统的代码耦合度高，工作量大且不好统一，维护的工作量也很大。

那么，能不能将所有类型的异常处理从各处理过程解耦出来，这样既保证了相关处理过程的功能较单一，也实现了异常信息的统一处理和维护？答案是肯定的。下面将介绍使用Spring MVC统一处理异常的解决和实现过程。

在SpringMVC中拥有一套非常强大的异常处理机制，SpringMVC通过HandlerExceptionResolver处理程序的异常，包括请求映射，数据绑定以及目标方法的执行时发生的异常。




在容器启动好，进入DispatcherServlet之后，会对HandlerExceptionResolver进行初始化操作：

```java
private void initHandlerExceptionResolvers(ApplicationContext context) {
    this.handlerExceptionResolvers = null;

    if (this.detectAllHandlerExceptionResolvers) {
        // Find all HandlerExceptionResolvers in the ApplicationContext, including ancestor contexts.
        Map<String, HandlerExceptionResolver> matchingBeans = BeanFactoryUtils
                .beansOfTypeIncludingAncestors(context, HandlerExceptionResolver.class, includeNonSingletons: true, allowEagerInit: false);
        if (!matchingBeans.isEmpty()) {
            this.handlerExceptionResolvers = new ArrayList<>(matchingBeans.values());
            // We keep HandlerExceptionResolvers in sorted order.
            AnnotationAwareOrderComparator.sort(this.handlerExceptionResolvers);
        }
    }
    else {
        try {
            HandlerExceptionResolver her =
                    context.getBean(HANDLER_EXCEPTION_RESOLVER_BEAN_NAME, HandlerExceptionResolver.class);
            this.handlerExceptionResolvers = Collections.singletonList(her);
        }
        catch (NoSuchBeanDefinitionException ex) {
            // Ignore, no HandlerExceptionResolver is fine too.
        }
    }

    // Ensure we have at least some HandlerExceptionResolvers, by registering
    // default HandlerExceptionResolvers if no other resolvers are found.
    if (this.handlerExceptionResolvers == null) {
        this.handlerExceptionResolvers = getDefaultStrategies(context, HandlerExceptionResolver.class);
        if (logger.isTraceEnabled()) {
            logger.trace("No HandlerExceptionResolvers declared in servlet '" + getServletName() +
                    "': using default strategies from DispatcherServlet.properties");
        }
    }
}
```

会默认的从DispatcherServlet.properties中找到对应的异常处理类：

```
#默认的处理类
org.springframework.web.servlet.HandlerExceptionResolver=
#处理@ExceptionHandler
org.springframework.web.servlet.mvc.method.annotation.ExceptionHandlerExceptionResolver,\
#解析@ResponseStatus注释类型的异常
org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver,\
#按照不同类型分别对异常进行解析
org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver
```

AbstractHandlerMethodExceptionResolver是ExceptionHandlerExcpetionResolver父类

SimpleMappingExceptionResolver：通过配置的异常类和view的对应关系解析异常

**@ExceptionHandler:**

```
* 通过@ExceptionHandler可以在方法中记录日志
*       转发一个友好的界面进行提示：
*          经验：1.记录日志中
*                2.可以将异常转发错误也没，  将错误信息在一个隐藏的div中
*  如果@ExceptionHandler写在@Controller中只能处理当前控制器类的处理方法
```

```
@ExceptionHandler(Exception.class)
```

```
public ModelAndView handleException(Exception ex){
    System.out.println("@Controller异常处理");
    ModelAndView modelAndView=new ModelAndView();
    modelAndView.setViewName("error");
    modelAndView.addObject("ex",ex);
    StringWriter sw = new StringWriter();
    PrintWriter pw = new PrintWriter(sw);
    ex.printStackTrace(pw);
    System.out.println(sw.toString());   // 日志记录
    return modelAndView;
}
```

## 2、统一异常处理

@ControllerAdvice 是Spring3.2提供的新注解,它是对Controller的增强,可对controller中被 @RequestMapping注解的方法加一些逻辑处理:

**1. 全局异常处理**

2. 全局数据绑定

3. 全局数据预处理

@ExceptionHandler

加在Controller中 :只处理当前控制器的异常，优先级比全局高

加在ControllerAdvice中 :处理全局异常

```

@ControllerAdvice
public class GeneralExceptionHandler {

    @ExceptionHandler(Exception.class)
    public ModelAndView handleException(HttpServletRequest request,
                                        HttpServletResponse reponse, Exception ex,
                                        HandlerMethod handle){
        System.out.println("全局异常处理");
        // 如果当前请求是ajax就返回json

        // 1.根据用户请求的处理方法，是否是一个返回json的处理方法
        //RestController restAnnotation = handle.getClass().getAnnotation(RestController.class); // 获得类上面
        //ResponseBody responseBody = handle.getMethod().getAnnotation(ResponseBody.class);//获得方法上面的某个
        // if(restAnnotation!=null || responseBody!=null){ }

        // 2.可以根据请求头中的类型Content-Type包含application/json



        if(request.getHeader("Accept").indexOf("application/json")>-1){
          // 可以直接输出json  reponse.getWriter().write();  或者集成jackson

            // 集成jackson的方式:
            //ModelAndView 同时支持视图返回和json返回
```

```
            // 这种方式就是返回json
            ModelAndView modelAndView = new ModelAndView(new MappingJackson2JsonView());
            // 通常会根据不同的异常返回不同的编码
            modelAndView.addObject("code", HttpStatus.INTERNAL_SERVER_ERROR.value());
            modelAndView.addObject("message",ex.getMessage());
            return  modelAndView;
        }
        else{
            ModelAndView modelAndView = new ModelAndView();
            modelAndView.setViewName("error");
            modelAndView.addObject("ex", ex);
            StringWriter sw = new StringWriter();
            PrintWriter pw = new PrintWriter(sw);
            ex.printStackTrace(pw);
            System.out.println(sw.toString());   // 日志记录
            return modelAndView;
        }
    }
}
```