【禁止血压飙升】如何拥有一个优雅的 controller

前言

见过几千行代码的 controller吗？我见过。

见过全是 try catch 的 controller 吗，我见过。

见过全是字段校验的 controller 吗，我见过。

见过全是业务代码的 controller 吗？不好意思，我们公司很多业务写在 controller 的。

看见这些我真的血压高。

正文

不优雅的 controller

```
@RestController
@RequestMapping(”/user/test”)
public class UserController {

    private static Logger logger =
LoggerFactory.getLogger(UserController.class);

    @Autowired
    private UserService userService;

    @Autowired
    private AuthService authService;
```

```
    @PostMapping
    public CommonResult userRegistration(@RequestBody UserVo
userVo) {
        if (StringUtils.isBlank(userVo.getUsername())){
            return CommonResult.error("用户名不能为空");
        }
        if (StringUtils.isBlank(userVo.getPassword())){
            return CommonResult.error("密码不能为空");
        }
        logger.info("注册用户：{}" , userVo.getUsername());
        try {
            userService.registerUser(userVo.getUsername());
            return CommonResult.ok();
        }catch (Exception e){
            logger.error("注册用户失败：{}", userVo.getUsername(), e);
            return CommonResult.error("注册失败");
        }
    }

    @PostMapping("/login")
    @PermitAll
    @ApiOperation("使用账号密码登录")
    public CommonResult<AuthLoginRespVO> login(@RequestBody
AuthLoginReqVO reqVO) {
        if (StringUtils.isBlank(reqVO.getUsername())){
            return CommonResult.error("用户名不能为空");
        }
        if (StringUtils.isBlank(reqVO.getPassword())){
            return CommonResult.error("密码不能为空");
        }
        try {
            return success(authService.login(reqVO));
        }catch (Exception e){
            logger.error("注册用户失败：{}", reqVO.getUsername(), e);
            return CommonResult.error("注册失败");
        }
    }

}


```

优雅的controller

```
@RestController
@RequestMapping(”/user/test”)
public class UserController1 {

    private static Logger logger =
LoggerFactory.getLogger(UserController1.class);

    @Autowired
    private UserService userService;

    @Autowired
    private AuthService authService;

    @PostMapping(”/userRegistration”)
    public CommonResult userRegistration(@RequestBody @Valid UserVo
userVo) {
        userService.registerUser(userVo.getUsername());
        return CommonResult.ok();
    }

    @PostMapping(”/login”)
    @PermitAll
    @ApiOperation(”使用账号密码登录”)
    public CommonResult<AuthLoginRespVO> login(@RequestBody
@Valid AuthLoginReqVO reqVO) {
        return success(authService.login(reqVO));
    }

}
```

> 代码量直接减一半呀，这还不算上有些直接把业务逻辑写在 controller 的，看到这些我真的直接吐血

改造流程

校验方式

> 这个 if 校验看得我哪哪都不爽。好歹给我写一个断言吧。
Assert.notNull(userVo.getUsername(), ”用户名不能为空”);
>

>
> 这不香吗？确实不香。
>
>
> 使用 spring 提供的@Valid

* 在入参时使用@Valid注解，并且在 vo 中使用校验注解，如AuthLoginReqVO

```
@ApiModel(value = "管理后台 - 账号密码登录 Request VO")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class AuthLoginReqVO {

    @ApiModelProperty(value = "账号", required = true, example = "user")
    @NotEmpty(message = "登录账号不能为空")
    @Length(min = 4, max = 16, message = "账号长度为 4-16 位")
    @Pattern(regexp = "^[A-Za-z0-9]+$", message = "账号格式为数字以及字母")
    private String username;

    @ApiModelProperty(value = "密码", required = true, example = "password")
    @NotEmpty(message = "密码不能为空")
    @Length(min = 4, max = 16, message = "密码长度为 4-16 位")
    private String password;

}
```

@Valid

在SpringBoot中，@Valid是一个非常有用的注解，主要用于数据校验。以下是关于@Valid的一些详细信息：

1. `为什么使用 @Valid 来验证参数`：在编写接口时，我们经常需要验证请求参数。通常，我们可能会写大量的 if 和 if else 代码来进行判断。但这样的代码不仅不优雅，而且如果存在大量的验证逻辑，这会使代码看起来混乱，大大降低代码可读性。为了简化这个过程，我们可以使用 @Valid 注解来帮助我们简化

验证逻辑。
2. `@Valid 注解的作用`：@Valid 的主要作用是用于数据效验，可以在定义的`实体中的属性上`，添加不同的注解来完成不同的校验规则，而在接口类中的接收数据参数中添加 @valid 注解，这时你的实体将会开启一个校验的功能。
3. `@Valid 的相解`：在实体类中不同的属性上添加不同的注解，就能实现不同数据的效验功能。
4. `使用 @Valid 进行参数效验步骤`：整个过程如下，用户访问接口，然后进行参数效验，因为 @Valid 不支持平面的参数效验（直接写在参数中字段的效验）所以基于 GET 请求的参数还是按照原先方式进行效验，而 POST 则可以以实体对象为参数，可以使用 @Valid 方式进行效验。如果效验通过，则进入业务逻辑，否则抛出异常，交由全局异常处理器进行处理。
5. `@Validated与@Valid的区别`：`@Validated` 是 `@Valid` 的变体。通过声明实体中属性的 `groups` ，再搭配使用 `@Validated` ，就能决定哪些属性需要校验，哪些不需要校验。

全局异常处理

* 这个全局异常处理，可以根据自己的异常，自定义异常处理，并设置一个兜底的异常处理

```

```
@ResponseBody
@RestControllerAdvice
public class ExceptionHandlerAdvice {
    protected Logger logger = LoggerFactory.getLogger(getClass());

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public CommonResult<Object>
handleValidationExceptions(MethodArgumentNotValidException ex) {
        logger.error("[handleValidationExceptions]", ex);
        StringBuilder sb = new StringBuilder();
        ex.getBindingResult().getAllErrors().forEach(error -> {
            String fieldName = ((org.springframework.validation.FieldError)
error).getField();
            String errorMessage = error.getDefaultMessage();
sb.append(fieldName).append(":").append(errorMessage).append(";");
    });
        return CommonResult.error(sb.toString());
    }

    /**
     * 处理系统异常，兜底处理所有的一切
     */
```
```

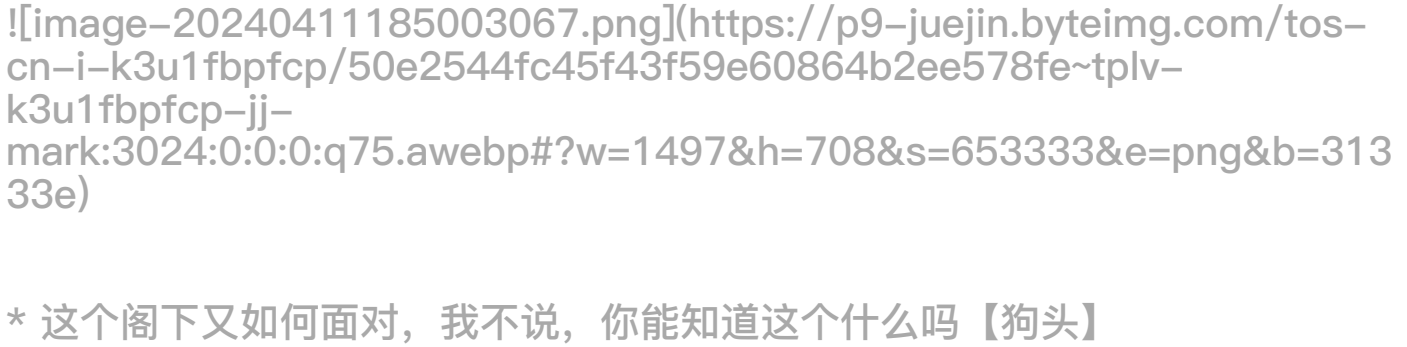
```
    @ExceptionHandler(value = Exception.class)
    public CommonResult<?> defaultExceptionHandler(Throwable ex) {
        logger.error("[defaultExceptionHandler]", ex);
        // 返回 ERROR CommonResult
        return CommonResult.error(INTERNAL_SERVER_ERROR.getCode(),
INTERNAL_SERVER_ERROR.getMsg());
    }

}
```

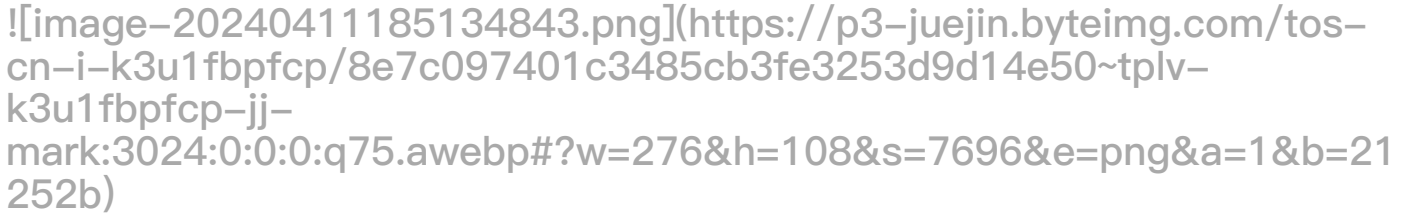
> 就这么多，搞定，这样就拥有了漂流优雅的 controller 了

在日常开发中，还有那些血压飙升瞬间
---

* 我拿出下图阁下如何面对



* 这个阁下又如何面对，我不说，你能知道这个什么吗【狗头】



总结
--

* 不是很明白为什么有些喜欢在 controller 写业务逻辑的，曾经有个同事问我（就是喜欢在 controller 写业务的），你这个接口写在那里，我需要调一下你这个接口。我满脸问号？？不是隔壁的模块吗，为什么要调我的接口？直接引用的我的 service 去调方法就好了。

* 这个就是痛点，各写各的，冗余代码一堆。
* 曾经看到一个同事写一个保存的方法，虽然逻辑挺多，我滑动了好久都还没有方法还没有结束。一个方法整整几百行……
* 看过 spring 源码都知道，spring 源码难啃，就是因为 spring 无限往下套娃，基本每个方法干每个方法的事情。比如我保存用户时，就只是保存用户，至于什么校验丢给校验的方法处理，什么发送消息丢给发送消息处理，这些就不能耦合在一起。
* 对于看到一些 if 下面一丢逻辑，然后 if 再一丢逻辑，看代码时很多情况不需要知道这个逻辑怎么实现的，知道入参出参就大概这里做什么了。即使想知道详细情况点进去就知道了。突出这个当前方法要做的事情就好了。
* 阿里的开发手册就推荐一个方法不能超过 80 行，超过可以根据业务具体调整一下。

原文链接: https://juejin.cn/post/7357172505961578511