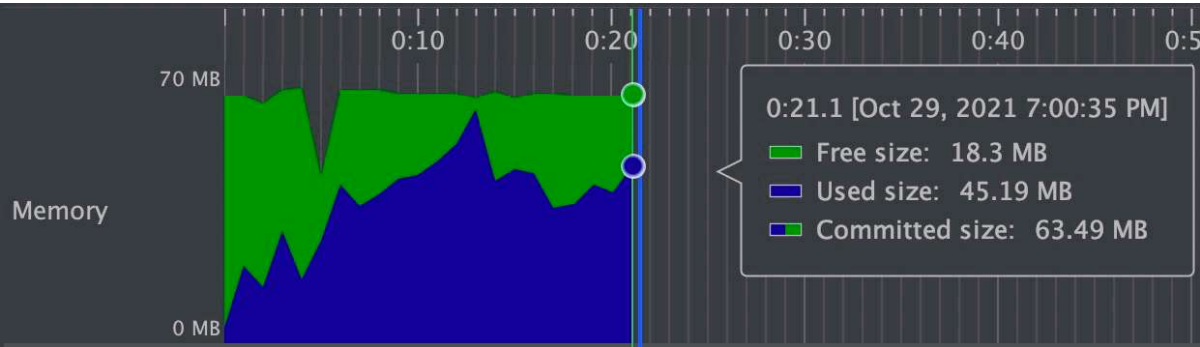


Excel 导入导出

项目的 [yudao-spring-boot-starter-excel](#) 技术组件，基于 EasyExcel 实现 Excel 的读写操作，可用于实现最常见的 Excel 导入导出等功能。

EasyExcel 的介绍?

EasyExcel 是阿里开源的 Excel 工具库，具有简单易用、低内存、高性能的特点。在尽可能节约内存的情况下，支持百万行的 Excel 读写操作。例如说，仅使用 64M 内存，20 秒完成 75M（46 万行 25 列）Excel 的读取。并且，还有极速模式能更快，但是内存占用会在 100M 多一点。



1. Excel 导出

以 [系统管理 -> 岗位管理] 菜单为例子，讲解它 Excel 导出的实现。



1.1 后端导入实现

在 `PostController` 类中，定义 `/admin-api/system/post/export` 导出接口。代码如下：

```

@GetMapping("/export")
@Operation(summary = "岗位管理")
@PreAuthorize("@ss.hasPermission('system:post:export')")
@ApiAccessLog(operateType = EXPORT)
public void export(HttpServletResponse response, @Validated PostPageReqVO reqVO) {
    // ① 查询数据
    reqVO.setPageSize(PageParam.PAGE_SIZE_NONE);
    List<PostDO> list = postService.getPostPage(reqVO).getList();
    // ② 导出 Excel
    ExcelUtils.write(response, "岗位数据.xls", "岗位列表", PostRespVO.class,
        BeanUtils.toBean(list, PostRespVO.class));
}

```

- ① 将从数据库中查询出来的列表，一般可以复用分页接口，需要设置 `.setPageSize(PageParam.PAGE_SIZE_NONE)` 不过滤分页。
- ② 将 `PostDO` 列表，转换成 `PostRespVO` 列表，之后通过 `ExcelUtils` 转换成 Excel 文件，返回给前端。

1.1.1 PostExcelVO 类

复用 `PostRespVO` 类，实现 岗位 Excel 导出的 VO 类。代码如下：

```

@Schema(description = "管理后台 - 岗位信息 Response VO")
@Data
@ExcelIgnoreUnannotated // ③
public class PostRespVO {

    @Schema(description = "岗位序号", requiredMode = Schema.RequiredMode.REQUIRED)
    @ExcelProperty("岗位序号") // ①
    private Long id;

    @Schema(description = "岗位名称", requiredMode = Schema.RequiredMode.REQUIRED)
    @ExcelProperty("岗位名称")
    private String name;

    @Schema(description = "岗位编码", requiredMode = Schema.RequiredMode.REQUIRED)
    @ExcelProperty("岗位编码")
    private String code;

    @Schema(description = "显示顺序不能为空", requiredMode = Schema.RequiredMode.REQUIRED)
    @ExcelProperty("岗位排序")
    private Integer sort;
}

```

```
@Schema(description = "状态, 参见 CommonStatusEnum 枚举类", requiredMode = Schema.RequiredMode.REQUIRED)
// ②
@ExcelProperty(value = "状态", converter = DictConvert.class)
@DictFormat(DictTypeConstants.COMMON_STATUS)
private Integer status;

@Schema(description = "备注", example = "快乐的备注")
private String remark;

@Schema(description = "创建时间", requiredMode = Schema.RequiredMode.REQUIRED)
private LocalDateTime createTime;

}
```

- ① 每个字段上, 添加 `@ExcelProperty` 注解, 声明 Excel Head 头部的名字。每个字段的值, 就是它对应的 Excel Row 行的数据值。
- ② 如果字段的注解 `converter` 属性是 `DictConvert` 转换器, 用于字典的转换。例如说, 通过 `status` 字段, 将 `status = 1` 转换成“开启”列, `status = 0` 转换成“禁用”列。稍后, 我们会在「3. 字段转换器」小节来详细讲讲。
- ③ 在类上, 添加 `@ExcelIgnoreUnannotated` 注解, 表示未添加 `@ExcelProperty` 的字段, 不进行导出。

因此, 最终 Excel 导出的效果如下:

	A	B	C	D	E
1	岗位序号	岗位编码	岗位名称	岗位排序	状态
2	1	ceo	董事长	1	开启
3	2	se	项目经理	2	开启
4	3	hr	人力资源	3	开启
5	4	user	普通员工	4	开启

1.1.2 ExcelUtils 写入

ExcelUtils 的 `#write(...)` 方法, 将列表以 Excel 响应给前端。代码如下图:

```
/**
 * 将列表以 Excel 响应给前端
 *
 * @param response 响应
 * @param filename 文件名
 * @param sheetName Excel sheet 名
 * @param head Excel head 头
 */
@throws IOException 写入失败的情况
*/
public static <T> void write(HttpServletResponse response, String filename, String sheetName,
                             Class<T> head, List<T> data) throws IOException {
    // 输出 Excel
    EasyExcel.write(response.getOutputStream(), head)
        .autoCloseStream(false) // 不要自动关闭, 交给 Servlet 自己处理
        .registerWriteHandler(new LongestMatchColumnWidthStyleStrategy()) // 基于 column 长度, 自动适配。最大 255 宽度
        .sheet(sheetName).doWrite(data);
    // 设置 header 和 contentType。写在最后的原因是, 避免报错时, 响应 contentType 已经被修改了
    response.addHeader("Content-Disposition", "attachment;filename=" + URLEncoder.encode(filename, "UTF-8"));
    response.setContentType("application/vnd.ms-excel;charset=UTF-8");
}
```

1.2 前端导入实现

在 `post/index.vue` 界面，定义 `#handleExport()` 操作，代码如下图：

```
/** 导出按钮操作 */
handleExport() {
  const queryParams = this.queryParams;
  this.$modal.confirm( content: '是否确认导出所有岗位数据项?' ).then(() => {
    this.exportLoading = true;
    return exportPost(queryParams); 请求后端 RESTful API 接口
  }).then(response => {
    this.$download.excel(response, fileName: '岗位数据.xls');
    this.exportLoading = false;
  }).catch(() => {});
}
```

将后端的响应，下载成 Excel 文件

2. Excel 导入

以 [系统管理 -> 用户管理] 菜单为例子，讲解它 Excel 导出的实现。



2.1 后端导入实现

在 `UserController` 类中，定义 `/admin-api/system/user/import` 导入接口。代码如下：

```
@PostMapping("/import")
@ApiOperation("导入用户")
@ApiImplicitParams({
  @ApiImplicitParam(name = "file", value = "Excel 文件", required = true, dataTypeClass = MultipartFile.class),
  @ApiImplicitParam(name = "updateSupport", value = "是否支持更新，默认为 false", example = "true", dataTypeClass = Boolean.class)
})
@PreAuthorize("@ss.hasPermission('system:user:import')")
public CommonResult<UserImportRespVO> importExcel(@RequestParam("file") MultipartFile file,
  @RequestParam(value = "updateSupport", required = false, defaultValue = "false") Boolean updateSupport) throws Exception {
  List<UserImportExcelVO> list = ExcelUtils.read(file, UserImportExcelVO.class);
  return success(userService.importUsers(list, updateSupport));
}
```

将前端上传的 Excel 文件，读取成 `UserImportExcelVO` 列表。

2.1.1 UserImportExcelVO 类

创建 `UserImportExcelVO` 类，用户 Excel 导入的 VO 类。它的作用和 Excel 导入是一样的，代码如下：

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Accessors(chain = false) // 设置 chain = false, 避免用户导入有问题
public class UserImportExcelVO {

    @ExcelProperty("登录名称")
    private String username;

    @ExcelProperty("用户名称")
    private String nickname;

    @ExcelProperty("部门编号")
    private Long deptId;

    @ExcelProperty("用户邮箱")
    private String email;

    @ExcelProperty("手机号码")
    private String mobile;

    @ExcelProperty(value = "用户性别", converter = DictConvert.class)
    @DictFormat(DictTypeConstants.USER_SEX)
    private Integer sex;

    @ExcelProperty(value = "账号状态", converter = DictConvert.class)
    @DictFormat(DictTypeConstants.COMMON_STATUS)
    private Integer status;
}
```

注意，一定要添加。
原因，EasyExcel 使用了 cglib，
而 cglib 读取链接调用方法存在问题

对应使用的 Excel 导入文件如下：

	A	B	C	D	E	F	G
1	登录名称	用户名称	部门编号	用户邮箱	手机号码	用户性别	账号状态
2	yunai	芋道	1	yunai@iocoder.cn	15601691300	男	开启
3	yuanma	源码	2	yuanma@iocoder.cn	15601701300	女	关闭
4							

2.1.2 ExcelUtils 读取

ExcelUtils 的 `#read(...)` 方法，读取 Excel 文件成列表。代码如下图：

```
public static <T> List<T> read(MultipartFile file, Class<T> head) throws IOException {
    return EasyExcel.read(file.getInputStream(), head, readListener: null)
        .autoCloseStream(false) // 不要自动关闭，交给 Servlet 自己处理
        .doReadAllSync();
}
```

2.2 前端导入实现

在 `user/index.vue` 界面，定义 Excel 导入的功能，代码如下图：


```
174 <!-- 用户导入对话框 -->
175 <el-dialog :title="upload.title" :visible.sync="upload.open" width="400px" append-to-body>
176   <el-upload ref="upload" :limit="1" accept=".xlsx, .xls" :headers="upload.headers"
177     :action="upload.url + '?updateSupport=' + upload.updateSupport" :disabled="upload.isUploading"
178     :on-progress="handleFileUploadProgress" :on-success="handleFileSuccess" :auto-upload="false" drag>
179     <i class="el-icon-upload"></i>
180     <div class="el-upload__text">将文件拖到此处，或<em>点击上传</em></div>
181     <div class="el-upload__tip text-center" slot="tip">
182       <div class="el-upload__tip" slot="tip">
183         <el-checkbox v-model="upload.updateSupport" /> 是否更新已经存在的用户数据
184       </div>
185       <span>仅允许导入xls、xlsx格式文件。</span>
186       <el-link type="primary" :underline="false" style="..." @click="importTemplate">下载模板</el-link>
187     </div>
188   </el-upload>
189   <div slot="footer" class="dialog-footer">
190     <el-button type="primary" @click="submitFileForm">确定</el-button>
191     <el-button @click="upload.open = false">取消</el-button>
192   </div>
193 </el-dialog>
```

下面，还有一些 JavaScript 方法，需要你自己瞅瞅

3. 字段转换器

EasyExcel 定义了 [Converter](#) 接口，用于实现字段的转换。它有两个核心方法：

- ① `#convertToJavaData(...)` 方法：将 Excel Row 对应表格的值，转换成 Java 内存中的值。例如说，Excel 的“状态”列，将“状态”列转换成 `status = 1`，“禁用”列转换成 `status = 0`。
- ② `#convertToExcelData(...)` 方法：恰好相反，将 Java 内存中的值，转换成 Excel Row 对应表格的值。例如说，Excel 的“状态”列，将 `status = 1` 转换成“开启”列，`status = 0` 转换成“禁用”列。

3.1 DictConvert 实现

以项目中提供的 [DictConvert](#) 举例子，它实现 Converter 接口，提供字典数据的转换。代码如下：

```
20 public class DictConvert implements Converter<Object> {
21
22   @Override Excel => Java: 开启 => 1; 禁用 => 0
23   public Object convertToJavaData(CellData cellData, ExcelContentProperty contentProperty, GlobalConfiguration globalConfiguration) {
24     // 使用字典解析
25     String type = getType(contentProperty);
26     String label = cellData.getStringValue();
27     DictDataRespDTO dictData = DictFrameworkUtils.parseDictDataFromCache(type, label);
28     if (dictData == null) {
29       log.error("[convertToJavaData][type({}) 解析不掉 label({})]", type, label);
30       return null;
31     }
32     // 将 String 的 value 转换成对应的属性
33     Class<?> fieldClazz = contentProperty.getField().getType();
34     return Convert.convert(fieldClazz, dictData.getValue());
35   }
36
37   @Override Java => Excel: 1 => 开启; 0 => 禁用
38   public CellData<String> convertToExcelData(Object object, ExcelContentProperty contentProperty, GlobalConfiguration globalConfiguration) {
39     // 空时，返回空
40     if (object == null) {
41       return new CellData<>("");
42     }
43     // 使用字典格式化
44     String type = getType(contentProperty);
45     String value = String.valueOf(object);
46     DictDataRespDTO dictData = DictFrameworkUtils.getDictDataFromCache(type, value);
47     if (dictData == null) {
48       log.error("[convertToExcelData][type({}) 转换不了 label({})]", type, value);
49       return new CellData<>("");
50     }
51     // 生成 Excel 小表格
52     return new CellData<>(dictData.getLabel());
53   }
54 }
```

实现的代码比较简单，自己看看就可以明白。

3.1 DictConvert 使用示例

在需要转换的字段上，声明注解 `@ExcelProperty` 的 `converter` 属性是 DictConvert 转换器，注解 `@DictFormat` 为对应的字典数据的类型。示例如下：

PostExcelVO.java

```
1 package cn.iocoder.yudao.module.system.controller.admin.dept.vo.post;
2
3 import ...
4
5 /**
6  * 岗位 Excel 导出响应 VO
7  */
8 @Data
9 public class PostExcelVO {
10
11     @ExcelProperty("岗位序号")
12     private Long id;
13
14     @ExcelProperty("岗位编码")
15     private String code;
16
17     @ExcelProperty("岗位名称")
18     private String name;
19
20     @ExcelProperty("岗位排序")
21     private Integer sort;
22
23     @ExcelProperty(value = "状态", converter = DictConvert.class)
24     @DictFormat(DictTypeConstants.COMMON_STATUS)
25     private String status;
26
27 }
28
```

UserImportExcelVO.java

```
12 /**
13  * 用户 Excel 导入 VO
14  */
15 @Data
16 @Builder
17 @AllArgsConstructor
18 @NoArgsConstructor
19 @Accessors(chain = false) // 设置 chain = false, 避免用户导入有问题
20 public class UserImportExcelVO {
21
22     @ExcelProperty("登录名称")
23     private String username;
24
25     @ExcelProperty("用户名称")
26     private String nickname;
27
28     @ExcelProperty("部门编号")
29     private Long deptId;
30
31     @ExcelProperty("用户邮箱")
32     private String email;
33
34     @ExcelProperty("手机号码")
35     private String mobile;
36
37     @ExcelProperty(value = "用户性别", converter = DictConvert.class)
38     @DictFormat(DictTypeConstants.USER_SEX)
39     private Integer sex;
40
41     @ExcelProperty(value = "账号状态", converter = DictConvert.class)
42     @DictFormat(DictTypeConstants.COMMON_STATUS)
43     private Integer status;
44
45 }
46
```

4. 更多 EasyExcel 注解

基于《EasyExcel 中的注解》文章，整理相关注解。

4.1 @ExcelProperty

这是最常用的一个注解，注解中有三个参数 `value`、`index`、`converter` 分别代表列明、列序号、数据转换方式。`value` 和 `index` 只能二选一，通常不用设置 `converter`。

最佳实践

```
public class ImeiEncrypt {

    @ExcelProperty(value = "imei")
    private String imei;

}
```

4.2 @ColumnWith

用于设置列宽度的注解，注解中只有一个参数 `value`。`value` 的单位是字符长度，最大可以设置 255 个字符，因为一个 Excel 单元格最大可以写入的字符个数，就是 255 个字符。

最佳实践

```
public class ImeiEncrypt {  
  
    @ColumnWidth(value = 18)  
    private String imei;  
  
}
```

4.3 @ContentFontStyle

用于设置单元格内容字体格式的注解。参数如下：

参数	含义
fontName	字体名称
fontHeightInPoints	字体高度
italic	是否斜体
strikeout	是否设置删除水平线
color	字体颜色
typeOffset	偏移量
underline	下划线
bold	是否加粗
charset	编码格式

4.4 @ContentLoopMerge

用于设置合并单元格的注解。参数如下：

参数	含义
eachRow	
columnExtend	

4.5 @ContentRowHeight

用于设置行高。参数如下：

参数	含义
value	行高， -1 代表自动行高

4.6 @ContentStyle

设置内容格式注解。参数如下：

参数	含义
dataFormat	日期格式
hidden	设置单元格使用此样式隐藏
locked	设置单元格使用此样式锁定
quotePrefix	在单元格前面增加`符号，数字或公式将以字符串形式展示
horizontalAlignment	设置是否水平居中
wrapped	设置文本是否应换行。将此标志设置为 <code>true</code> 通过在多行上显示使单元格中的所有内容可见
verticalAlignment	设置是否垂直居中
rotation	设置单元格中文本旋转角度。03版本的Excel旋转角度区间为-90°~90°，07版本的Excel旋转角度区间为0°~180°
indent	设置单元格中缩进文本的空格数
borderLeft	设置左边框的样式
borderRight	设置右边框样式
borderTop	设置上边框样式
borderBottom	设置下边框样式
leftBorderColor	设置左边框颜色
rightBorderColor	设置右边框颜色
topBorderColor	设置上边框颜色
bottomBorderColor	设置下边框颜色
fillPatternType	设置填充类型
fillBackgroundColor	设置背景色
fillForegroundColor	设置前景色
shrinkToFit	设置自动单元格自动大小

4.7 @HeadFontStyle

用于定制标题字体格式。参数如下：

参数	含义
fontName	设置字体名称
fontHeightInPoints	设置字体高度
italic	设置字体是否斜体

参数	含义
strikeout	是否设置删除线
color	设置字体颜色
typeOffset	设置偏移量
underline	设置下划线
charset	设置字体编码
bold	设置字体是否家畜

4.8 @HeadRowHeight

设置标题行行高。参数如下：

参数	含义
value	设置行高，-1代表自动行高

4.9 @HeadStyle

设置标题样式。参数如下：

参数	含义
dataFormat	日期格式
hidden	设置单元格使用此样式隐藏
locked	设置单元格使用此样式锁定
quotePrefix	在单元格前面增加`符号，数字或公式将以字符串形式展示
horizontalAlignment	设置是否水平居中
wrapped	设置文本是否应换行。将此标志设置为 <code>true</code> 通过在多行上显示使单元格中的所有内容可见
verticalAlignment	设置是否垂直居中
rotation	设置单元格中文本旋转角度。03版本的Excel旋转角度区间为-90°~90°，07版本的Excel旋转角度区间为0°~180°
indent	设置单元格中缩进文本的空格数
borderLeft	设置左边框的样式
borderRight	设置右边框样式
borderTop	设置上边框样式
borderBottom	设置下边框样式
leftBorderColor	设置左边框颜色

参数	含义
rightBorderColor	设置右边框颜色
topBorderColor	设置上边框颜色
bottomBorderColor	设置下边框颜色
fillPatternType	设置填充类型
fillBackgroundColor	设置背景色
fillForegroundColor	设置前景色
shrinkToFit	设置自动单元格自动大小

4.10 @ExcelIgnore

不将该字段转换成 Excel。

4.11 @ExcelIgnoreUnannotated

没有注解的字段都不转换

← 文件存储（上传下载）

操作日志、访问日志、异常日志→



Theme by Vdoing | Copyright © 2019-2024 芋道源码 | MIT License