## 后台手册

### 分页实现

前端基于Bootstrap的轻量级表格插件 Bootstrap Table  
后端分页组件使用Mybatis分页插件 PageHelper

#### 前端调用封装好的方法$.table.init，传入后台url。

1. var options = {
2. url: prefix + "/list",
3. columns: [{
4. field: 'id',
5. title: '主键'
6. },
7. {
8. field: 'name',
9. title: '名称'
10. }]
11. };
12. $.table.init(options);

#### 后台实现查询逻辑，调用startPage()方法即可自动完成服务端分页。

1. @PostMapping("/list")
2. @ResponseBody
3. public TableDataInfo list(User user)
4. {
5. startPage(); // 此方法配合前端完成自动分页
6. List<User> list = userService.selectUserList(user);
7. return getDataTable(list);
8. }

注意：启动分页关键代码startPage()（只对该语句以后的第一个查询语句得到的数据进行分页）  
如果改为其他数据库需修改配置application.yml helperDialect=你的数据库

### 导入导出

导入导出使用 Apache POI，目前支持参数如下

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| name | String | 空 | 导出到Excel中的名字 |
| dateFormat | String | 空 | 日期格式, 如: yyyy-MM-dd |
| readConverterExp | String | 空 | 读取内容转表达式 (如: 0=男,1=女,2=未知) |
| height | String | 14 | 导出时在excel中每个列的高度 单位为字符 |
| width | String | 16 | 导出时在excel中每个列的宽 单位为字符 |
| suffix | String | 空 | 文字后缀,如% 90 变成90% |
| defaultValue | String | 空 | 当值为空时,字段的默认值 |
| prompt | String | 空 | 提示信息 |
| combo | String | Null | 设置只能选择不能输入的列内容 |
| isExport | String | true | 是否导出数据,应对需求:有时我们需要导出一份模板,这是标题需要但内容需要用户手工填写 |
| targetAttr | String | 空 | 另一个类中的属性名称,支持多级获取,以小数点隔开 |
| type | Enum | Type.ALL | 字段类型（0：导出导入；1：仅导出；2：仅导入） |

#### 导出实现流程

##### 前端调用封装好的方法$.table.init，传入后台exportUrl。

1. var options = {
2. exportUrl: prefix + "/export",
3. columns: [{
4. field: 'id',
5. title: '主键'
6. },
7. {
8. field: 'name',
9. title: '名称'
10. }]
11. };
12. $.table.init(options);

##### 在实体变量上添加@Excel注解。

1. @Excel(name = "用户序号")
2. private Long id;
3. @Excel(name = "用户名称")
4. private String userName;

3、在Controller添加导出方法

1. @PostMapping("/export")
2. @ResponseBody
3. public AjaxResult export(User user)
4. {
5. List<User> list = userService.selectUserList(user);
6. ExcelUtil<User> util = new ExcelUtil<User>(User.class);
7. return util.exportExcel(list, "用户数据");
8. }

#### 导入实现流程

##### 前端调用封装好的方法$.table.init，传入后台importUrl。

1. var options = {
2. importUrl: prefix + "/importData",
3. columns: [{
4. field: 'id',
5. title: '主键'
6. },
7. {
8. field: 'name',
9. title: '名称'
10. }]
11. };
12. $.table.init(options);

##### 在实体变量上添加@Excel注解，默认为导出导入，也可以单独设置仅导入

Type.IMPORT

1. @Excel(name = "用户序号")
2. private Long id;
3. @Excel(name = "部门编号", type = Type.IMPORT)
4. private Long deptId;
5. @Excel(name = "用户名称")
6. private String userName;

3、在Controller添加导入方法，updateSupport属性为是否存在则覆盖（可选）

1. @PostMapping("/importData")
2. @ResponseBody
3. public AjaxResult importData(MultipartFile file, boolean updateSupport) throws Exception
4. {
5. ExcelUtil<SysUser> util = new ExcelUtil<SysUser>(SysUser.class);
6. List<SysUser> userList = util.importExcel(file.getInputStream());
7. String operName = ShiroUtils.getSysUser().getLoginName();
8. String message = userService.importUser(userList, updateSupport, operName);
9. return AjaxResult.success(message);
10. }

### 上传下载

首先创建一张上传文件的表，例如：

1. drop table if exists sys\_file;
2. create table sys\_file (
3. fileid int(11) not null auto\_increment comment '文件id',
4. filename varchar(50) default '' comment '文件名称',
5. filepath varchar(255) default '' comment '文件路径',
6. primary key (fileid)
7. ) engine=innodb auto\_increment=200 default charset=utf8 comment = '文件表';

#### 上传实现流程

##### 参考示例代码。

1. function submitHandler() {
2. if ($.validate.form()) {
3. uploadFile();
4. }
5. }
6. function uploadFile() {
7. var formData = new FormData();
8. if($('#file')[0].files[0] == null) {
9. $.modal.alertWarning("请先选择文件路径");
10. return false;
11. }
12. formData.append('fileName', $("#fileName").val());
13. formData.append('file', $('#file')[0].files[0]);
14. $.ajax({
15. url: prefix + "/add",
16. type: 'post',
17. cache: false,
18. data: formData,
19. processData: false,
20. contentType: false,
21. dataType: "json",
22. success: function(result) {
23. $.operate.successCallback(result);
24. }
25. });
26. }

##### 在Controller添加对应上传方法

1. @Autowired
2. private ServerConfig serverConfig;
3. @PostMapping("/add")
4. @ResponseBody
5. public AjaxResult addSave(MultipartFile file, SysFile sysFile) throws IOException
6. {
7. // 上传文件路径
8. String filePath = Global.getUploadPath();
9. // 上传并返回新文件名称
10. String fileName = FileUploadUtils.upload(filePath, file);
11. sysFile.setFilePath(fileName);
12. return toAjax(sysFileService.insertSysFile(sysFile));
13. }

##### 上传成功后需要预览可以对该属性格式化处理

1. {
2. title: '文件预览',
3. formatter: function(value, row, index) {
4. return '<a href="javascript:downloadFile(' + row.fileId + ')"><img style="width:30;height:30px;" src="/profile/upload/' + row.filePath + '"/></a>';
5. }
6. },

注意：如果只是单纯的上传一张图片没有其他参数可以使用通用方法 /common/upload  
请求处理方法 com.asiainfo.web.controller.common.CommonController

#### 下载实现流程

##### 参考示例代码。

1. function downloadFile(fileId){
2. window.location.href = ctx + "system/sysFile/downloadFile/" + fileId;
3. }

##### 在Controller添加对应上传方法

1. @GetMapping("/downloadFile/{fileId}")
2. public void downloadFile(@PathVariable("fileId") Integer fileId, HttpServletResponse response) throws Exception
3. {
4. SysFile sysFile = sysFileService.selectSysFileById(fileId);
5. String filePath = sysFile.getFilePath();
6. String realFileName = sysFile.getFileName() + filePath.substring(filePath.indexOf("."));
7. String path = Global.getUploadPath() + sysFile.getFilePath();
8. response.setCharacterEncoding("utf-8");
9. response.setContentType("multipart/form-data");
10. response.setHeader("Content-Disposition", "attachment;fileName=" + realFileName);
11. FileUtils.writeBytes(path, response.getOutputStream());
12. }

### 事务管理

在Spring Boot中，当我们使用了spring-boot-starter-jdbc或spring-boot-starter-data-jpa依赖的时候，框架会自动默认分别注入DataSourceTransactionManager或JpaTransactionManager。 所以我们不需要任何额外配置就可以用@Transactional注解进行事务的使用。

例如：新增用户时需要插入用户表、用户与岗位关联表、用户与角色关联表。就可以使用事务让它实现回退。  
做法非常简单，我们只需要在方法上添加@Transactional注解即可。事务可以用于Service和Controller

1. @Transactional
2. public int insertUser(User user)
3. {
4. // 新增用户信息
5. int rows = userMapper.insertUser(user);
6. // 新增用户岗位关联
7. insertUserPost(user);
8. // 新增用户与角色管理
9. insertUserRole(user);
10. return rows;
11. }

常见坑点1：遇到非检测异常时，事务开启，也无法回滚。 例如下面这段代码，账户余额依旧增加成功，并没有因为后面遇到检测异常而回滚！！

1. @Transactional
2. public void addMoney() throws Exception {
3. //先增加余额
4. accountMapper.addMoney();
5. //然后遇到故障
6. throw new SQLException("发生异常了..");
7. }

原因分析：因为Spring的默认的事务规则是遇到运行异常（RuntimeException）和程序错误（Error）才会回滚。如果想针对非检测异常进行事务回滚，可以在@Transactional 注解里使用 rollbackFor 属性明确指定异常。例如下面这样，就可以正常回滚：

1. @Transactional(rollbackFor = Exception.class)
2. public void addMoney() throws Exception {
3. //先增加余额
4. accountMapper.addMoney();
5. //然后遇到故障
6. throw new SQLException("发生异常了..");
7. }

常见坑点2： 在业务层捕捉异常后，发现事务不生效。 这是许多新手都会犯的一个错误，在业务层手工捕捉并处理了异常，你都把异常“吃”掉了，Spring自然不知道这里有错，更不会主动去回滚数据。 例如：下面这段代码直接导致增加余额的事务回滚没有生效。

1. @Transactional
2. public void addMoney() throws Exception {
3. //先增加余额
4. accountMapper.addMoney();
5. //谨慎：尽量不要在业务层捕捉异常并处理
6. try {
7. throw new SQLException("发生异常了..");
8. } catch (Exception e) {
9. e.printStackTrace();
10. }
11. }

推荐做法：在业务层统一抛出异常，然后在控制层统一处理。

1. @Transactional
2. public void addMoney() throws Exception {
3. //先增加余额
4. accountMapper.addMoney();
5. //推荐：在业务层将异常抛出
6. throw new RuntimeException("发生异常了..");
7. }

Transactional注解的常用属性表：

| **属性** | **说明** |
| --- | --- |
| propagation | 事务的传播行为，默认值为 REQUIRED。 |
| isolation | 事务的隔离度，默认值采用 DEFAULT |
| timeout | 事务的超时时间，默认值为-1，不超时。如果设置了超时时间(单位秒)，那么如果超过该时间限制了但事务还没有完成，则自动回滚事务。 |
| read-only | 指定事务是否为只读事务，默认值为 false；为了忽略那些不需要事务的方法，比如读取数据，可以设置 read-only 为 true。 |
| rollbackFor | 用于指定能够触发事务回滚的异常类型，如果有多个异常类型需要指定，各类型之间可以通过逗号分隔。{xxx1.class, xxx2.class,……} |
| noRollbackFor | 抛出 no-rollback-for 指定的异常类型，不回滚事务。{xxx1.class, xxx2.class,……} |
| .... |  |

TransactionDefinition传播行为的常量：

| **常量** | **含义** |
| --- | --- |
| TransactionDefinition.PROPAGATION\_REQUIRED | 如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。这是默认值。 |
| TransactionDefinition.PROPAGATION\_REQUIRES\_NEW | 创建一个新的事务，如果当前存在事务，则把当前事务挂起。 |
| TransactionDefinition.PROPAGATION\_SUPPORTS | 如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。 |
| TransactionDefinition.PROPAGATION\_NOT\_SUPPORTED | 以非事务方式运行，如果当前存在事务，则把当前事务挂起。 |
| TransactionDefinition.PROPAGATION\_NEVER | 以非事务方式运行，如果当前存在事务，则抛出异常。 |
| TransactionDefinition.PROPAGATION\_MANDATORY | 如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。 |
| TransactionDefinition.PROPAGATION\_NESTED | 如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于TransactionDefinition.PROPAGATION\_REQUIRED。 |

提示：事务的传播机制是指如果在开始当前事务之前，一个事务上下文已经存在，此时有若干选项可以指定一个事务性方法的执行行为。 即:在执行一个@Transactinal注解标注的方法时，开启了事务；当该方法还在执行中时，另一个人也触发了该方法；那么此时怎么算事务呢，这时就可以通过事务的传播机制来指定处理方式。

### 异常处理

在日常开发中程序发生了异常，往往需要通过一个统一的异常处理，来保证客户端能够收到友好的提示。  
通常情况下我们用try..catch..对异常进行捕捉处理，但是在实际项目中对业务模块进行异常捕捉，会造成代码重复和繁杂， 我们希望代码中只有业务相关的操作，所有的异常我们单独设立一个类来处理它。全局异常就是对框架所有异常进行统一管理 而这就表示在框架需要一个机制，将程序的异常转换为用户可读的异常。而且最重要的，是要将这个机制统一，提供统一的异常处理。 我们在可能发生异常的方法，全部throw抛给前端控制器；然后由前端控制器调用 全局异常处理器 对异常进行统一处理。 如此，我们现在的Controller中的方法就可以很简洁了。

1、统一返回实体定义

1. package com.asiainfo.common.core.domain;
2. import java.util.HashMap;
3. /\*\*
4. \* 操作消息提醒
5. \*
6. \* @author yichen
7. \*/
8. public class AjaxResult extends HashMap<String, Object>
9. {
10. private static final long serialVersionUID = 1L;
11. /\*\*
12. \* 返回错误消息
13. \*
14. \* @param code 错误码
15. \* @param msg 内容
16. \* @return 错误消息
17. \*/
18. public static AjaxResult error(String msg)
19. {
20. AjaxResult json = new AjaxResult();
21. json.put("msg", msg);
22. json.put("code", 500);
23. return json;
24. }
25. /\*\*
26. \* 返回成功消息
27. \*
28. \* @param msg 内容
29. \* @return 成功消息
30. \*/
31. public static AjaxResult success(String msg)
32. {
33. AjaxResult json = new AjaxResult();
34. json.put("msg", msg);
35. json.put("code", 0);
36. return json;
37. }
38. }

2、定义登录异常定义

1. package com.asiainfo.common.exception;
2. /\*\*
3. \* 登录异常
4. \*
5. \* @author yichen
6. \*/
7. public class LoginException extends RuntimeException
8. {
9. private static final long serialVersionUID = 1L;
10. protected final String message;
11. public LoginException(String message)
12. {
13. this.message = message;
14. }
15. @Override
16. public String getMessage()
17. {
18. return message;
19. }
20. }

3、基于@ControllerAdvice注解的Controller层的全局异常统一处理

1. package com.asiainfo.framework.web.exception;
2. import org.slf4j.Logger;
3. import org.slf4j.LoggerFactory;
4. import org.springframework.web.bind.annotation.ExceptionHandler;
5. import org.springframework.web.bind.annotation.RestControllerAdvice;
6. import com.asiainfo.common.core.domain.AjaxResult;
7. import com.asiainfo.common.exception.LoginException;
8. /\*\*
9. \* 全局异常处理器
10. \*
11. \* @author yichen
12. \*/
13. @RestControllerAdvice
14. public class GlobalExceptionHandler
15. {
16. private static final Logger log = LoggerFactory.getLogger(GlobalExceptionHandler.class);
17. /\*\*
18. \* 登录异常
19. \*/
20. @ExceptionHandler(LoginException.class)
21. public AjaxResult loginException(LoginException e)
22. {
23. log.error(e.getMessage(), e);
24. return AjaxResult.error(e.getMessage());
25. }
26. }

4、测试访问请求

1. @Controller
2. public class SysIndexController
3. {
4. /\*\*
5. \* 首页方法
6. \*/
7. @GetMapping("/index")
8. public String index(ModelMap mmap)
9. {
10. /\*\*
11. \* 模拟用户未登录，抛出业务逻辑异常
12. \*/
13. SysUser user = ShiroUtils.getSysUser();
14. if (StringUtils.isNull(user))
15. {
16. throw new LoginException("用户未登录，无法访问请求。");
17. }
18. mmap.put("user", user);
19. return "index";
20. }
21. }

根据上面代码含义，当我们在访问/index时就会发生LoginException业务逻辑异常，按照我们之前的全局异常配置以及统一返回实体实例化，访问后会出现AjaxResult格式JSON数据， 下面我们运行项目访问查看效果。  
界面输出内容如下所示：

1. {
2. "msg": "用户未登录，无法访问请求。",
3. "code": 500
4. }

这个代码示例写的非常浅显易懂，但是需要注意的是：基于@ControllerAdvice注解的全局异常统一处理只能针对于Controller层的异常，意思是只能捕获到Controller层的异常， 在service层或者其他层面的异常都不能捕获。

若依系统的全局异常处理器GlobalExceptionHandler  
注意：如果全部异常处理返回json，那么可以使用@RestControllerAdvice代替@ControllerAdvice，这样在方法上就可以不需要添加@ResponseBody。

### 系统日志

在实际开发中，对于某些关键业务，我们通常需要记录该操作的内容，一个操作调一次记录方法，每次还得去收集参数等等，会造成大量代码重复。 我们希望代码中只有业务相关的操作，所有的异常我们单独设立一个注解来处理它。

在需要被记录日志的controller方法上添加@Log注解，使用方法如下：

1. @Log(title = "用户管理", businessType = BusinessType.INSERT)

支持参数如下：

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| title | String | 空 | 操作模块 |
| businessType | BusinessType | BusinessType.OTHER | 操作功能 |
| operatorType | OperatorType | OperatorType.MANAGE | 操作人类别 |
| isSaveRequestData | boolean | true | 是否保存请求的参数 |

逻辑实现代码 com.asiainfo.framework.aspectj.LogAspect  
查询操作详细记录可以登录系统（系统管理-操作日志）

### 数据权限

在实际开发中，需要设置用户只能查看哪些部门的数据，一般称为数据权限  
默认系统管理员admin拥有所有数据权限（userId=1）  
在需要数据权限控制方法上添加@DataScope注解  
@DataScope(tableAlias = "u")，其中u用来表示表的别名

1. /\*\* 表的别名 \*/
2. String tableAlias() default "";

在mybatis查询标签中添加数据范围过滤 ${params.dataScope}

会生成如下关键代码：

1. select u.user\_id, u.dept\_id, u.login\_name, u.user\_name, u.email , u.phonenumber,
2. u.password, u.sex, u.avatar, u.salt, u.status, u.del\_flag, u.login\_ip,
3. u.login\_date, u.create\_by, u.create\_time, u.remark, d.dept\_name
4. from sys\_user u
5. left join sys\_dept d on u.dept\_id = d.dept\_id
6. where u.del\_flag = '0'
7. and u.dept\_id in ( select dept\_id from sys\_role\_dept where role\_id = 2 )

逻辑实现代码 com.asiainfo.framework.aspectj.DataScopeAspect

### 多数据源

在实际开发中，经常可能遇到在一个应用中可能需要访问多个数据库的情况  
在需要切换数据源Service或Mapper方法上添加@DataSource注解  
@DataSource(value = DataSourceType.MASTER)，其中value用来表示数据源名称

1. /\*\* 切换数据源名称 \*/
2. public DataSourceType value() default DataSourceType.MASTER;

注解实现数据源切换

1. @DataSource(value = DataSourceType.MASTER)
2. public List<SysUser> selectUserList(SysUser user)
3. {
4. return userMapper.selectUserList(user);
5. }

手动实现数据源切换

1. public List<SysUser> selectUserList(SysUser user)
2. {
3. DynamicDataSourceContextHolder.setDataSourceType(DataSourceType.MASTER.name());
4. List<SysUser> userList = userMapper.selectUserList(user);
5. DynamicDataSourceContextHolder.clearDataSourceType();
6. return userList;
7. }

逻辑实现代码 com.asiainfo.framework.aspectj.DataSourceAspect

注意：目前配置了一个从库，默认关闭状态。可新增多个从库，支持不同数据源（Mysql、Oracle、SQLServer）

### 代码生成

大部分项目里其实有很多代码都是重复的，几乎每个基础模块的代码都有增删改查的功能，而这些功能都是大同小异，如果这些功能都要自己去写，将会大大浪费我们的精力降低效率。所以这种重复性的代码可以使用代码生成。 1、修改代码生成配置  
编辑resources目录下的generator.yml  
author: # 开发者姓名，生成到类注释上  
packageName: # 默认生成包路径  
autoRemovePre: # 是否自动去除表前缀  
tablePrefix: # 表前缀

2、新建数据库表结构（需要表注释）

1. drop table if exists sys\_test;
2. create table sys\_test (
3. test\_id int(11) auto\_increment comment '测试id',
4. test\_name varchar(30) default '' comment '测试名称',
5. primary key (test\_id)
6. ) engine=innodb auto\_increment=1 default charset=utf8 comment = '测试表';

3、登录系统-系统工具 -> 代码生成  
找到sys\_test表，点击生成代码会得到一个asiainfo.zip  
执行sql文件，覆盖文件到对应目录即可

所有代码生成的相关业务逻辑代码在yichen-generator模块，可以自行调整或剔除

## 前端手册

### 前端组件

yichen封装了一些常用的JS组件方法。

| **名称** | **代码** | **介绍** |
| --- | --- | --- |
| 表格 | $.table | 表格封装处理 |
| 表格树 | $.treeTable | 表格树封装处理 |
| 表单 | $.form | 表单封装处理 |
| 弹出层 | $.modal | 弹出层封装处理 |
| 操作 | $.operate | 操作封装处理 |
| 校验 | $.validate | 校验封装处理 |
| 树插件 | $.tree | 树插件封装处理 |
| 通用方法 | $.common | 通用方法封装处理 |

### 通用方法

支持属性

| **方法** | **参数** | **介绍** |
| --- | --- | --- |
| $.table.init(); | options（选项参数） | 初始化表格参数 |
| $.table.search(); | formId（表单ID） | 搜索-默认第一个form |
| $.table.exportExcel(); | formId（表单ID） | 导出-默认第一个form |
| $.table.importExcel(); | formId（表单ID） | 导入-默认importForm |
| $.table.importTemplate(); | formId（表单ID） | 摸版下载 |
| $.table.refresh(); | 无 | 刷新表格 |
| $.table.selectColumns(); | column（查询列值） | 查询表格指定列值 |
| $.table.selectFirstColumns(); | 无 | 查询表格首列值 |
| $.table.destroy(); | tableId（表格ID） | 销毁表格-默认options.id |
| $.table.serialNumber(); | index（序号） | 序列号生成 |
| $.table.dropdownToggle(); | value（内容） | 下拉按钮切换 |
| $.table.showColumn(); | column（列值） | 显示表格特定的列 |
| $.table.hideColumn(); | column（列值） | 隐藏表格特定的列 |
| $.table.tooltip(); | value（内容）, length（截取长度） | 超出指定长度浮动提示 |
| $.table.selectDictLabel(); | datas（字典列表）, value（当前值） | 回显数据字典 |
| $.treeTable.init(); | options（选项参数） | 初始化表格树参数 |
| $.treeTable.search(); | formId（表单ID） | 搜索-默认第一个form |
| $.treeTable.refresh(); | 无 | 刷新表格树 |
| $.form.reset(); | formId（表单ID） | 表单重置 |
| $.form.selectCheckeds(); | name（name名称） | 获取选中复选框项 |
| $.form.selectSelects(); | name（id名称） | 获取选中下拉框项 |
| $.modal.icon | type（图标类型） | 显示图标 |
| $.modal.msg | content（内容）, type（图标类型） | 消息提示 |
| $.modal.msgError(); | content（内容） | 错误消息 |
| $.modal.msgSuccess(); | content（内容） | 成功消息 |
| $.modal.msgWarning(); | content（内容） | 警告消息 |
| $.modal.alert | content（内容）, type（图标类型） | 消息提示 |
| $.modal.msgReload | msg（消息）, type（图标类型） | 消息提示并刷新父窗体 |
| $.modal.alertError(); | content（内容） | 错误提示 |
| $.modal.alertSuccess(); | content（内容） | 成功提示 |
| $.modal.alertWarning(); | content（内容） | 警告提示 |
| $.modal.close(); | 无 | 关闭窗体 |
| $.modal.closeAll | 无 | 关闭全部窗体 |
| $.modal.confirm(); | content（内容）, callBack（回调函数） | 确认窗体 |
| $.modal.open(); | title, url, width, height, callBack（回调函数） | 弹出层指定宽度 |
| $.modal.openOptions(); | options（选项参数） | 弹出层指定参数选项 |
| $.modal.openFull(); | title, url, width, height | 弹出层全屏 |
| $.modal.openTab | title（标题）, url（地址） | 选卡页方式打开 |
| $.modal.disable | 无 | 禁用按钮 |
| $.modal.enable | 无 | 启用按钮 |
| $.modal.loading(); | message（提示消息） | 打开遮罩层 |
| $.modal.closeLoading(); | 无 | 关闭遮罩层 |
| $.modal.reload(); | 无 | 重新加载 |
| $.operate.submit(); | url, type, dataType, data | 提交数据 |
| $.operate.post(); | url（地址）, data（数据） | post方式请求提交数据 |
| $.operate.get(); | url（地址） | get请求传输数据 |
| $.operate.detail(); | id（数据ID） | 详细信息 |
| $.operate.remove(); | id（数据ID） | 删除信息 |
| $.operate.removeAll(); | 无 | 批量删除信息 |
| $.operate.clean(); | 无 | 清空信息 |
| $.operate.add(); | id（数据ID） | 添加信息 |
| $.operate.addTab(); | id（数据ID） | 添加信息（选项卡方式） |
| $.operate.addFull(); | id（数据ID） | 添加信息 全屏 |
| $.operate.addUrl(); | id（数据ID） | 添加访问地址 |
| $.operate.edit(); | id（数据ID） | 修改信息 |
| $.operate.editTab(); | id（数据ID） | 修改信息（选项卡方式） |
| $.operate.editFull(); | id（数据ID） | 修改信息 全屏 |
| $.operate.editUrl(); | id（数据ID） | 修改访问地址 |
| $.operate.save(); | url（地址）, data（数据） | 保存信息 |
| $.operate.saveTab(); | url（地址）, data（数据） | 保存选项卡信息 |
| $.operate.ajaxSuccess(); | result（返回结果） | 保存结果弹出msg刷新table表格 |
| $.operate.saveSuccess(); | result（返回结果） | 保存结果提示msg |
| $.operate.successCallback(); | result（返回结果） | 成功回调执行事件（静默更新） |
| $.operate.successTabCallback(); | result（返回结果） | 选项卡成功回调执行事件（静默更新） |
| $.validate.unique(); | value（返回标识） | 判断返回标识是否唯一 |
| $.validate.form(); | formId（表单ID） | 表单验证-默认第一个form |
| $.tree.init(); | options（选项参数） | 初始化树结构 |
| $.tree.searchNode(); | 无 | 搜索节点 |
| $.tree.selectByIdName(); | treeId, treeName, node | 根据Id和Name选中指定节点 |
| $.tree.showAllNode(); | nodes（全部节点数据） | 显示所有节点 |
| $.tree.hideAllNode(); | nodes（全部节点数据） | 隐藏所有节点 |
| $.tree.showParent(); | treeNode（节点数据） | 显示所有父节点 |
| $.tree.showChildren(); | treeNode（节点数据） | 显示所有孩子节点 |
| $.tree.updateNodes(); | nodeList（全部节点数据） | 更新节点状态 |
| $.tree.getCheckedNodes(); | column（列值） | 获取当前被勾选集合 |
| $.tree.notAllowParents(); | \_tree（树对象） | 不允许根父节点选择 |
| $.tree.toggleSearch(); | 无 | 隐藏/显示搜索栏 |
| $.tree.collapse(); | 无 | 树折叠 |
| $.tree.expand(); | 无 | 树展开 |
| $.common.isEmpty(); | value（值） | 判断字符串是否为空 |
| $.common.isNotEmpty(); | value（值） | 判断一个字符串是否为非空串 |
| $.common.nullToStr(); | value（值） | 空对象转字符串 |
| $.common.visible(); | value（值） | 是否显示数据 为空默认为显示 |
| $.common.trim(); | value（值） | 空格截取 |
| $.common.random(); | min（最小）, max（最大） | 指定随机数返回 |
| $.common.startWith(); | value（值）, start（开始值） | 判断字符串是否是以start开头 |
| $.common.endWith(); | value（值）, end（结束值） | 判断字符串是否是以end结尾 |

### 表格使用

表格组件基于bootstrap table组件进行封装，轻松实现数据表格。

* 表格初始化 $.table.init

表的各项(Table options )

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String | Null | 请求后台的URL |
| uniqueId | String | Null | 指定唯一列属性 配合删除/修改使用 未指定则使用表格行首列 |
| createUrl | String | Null | 新增URL 配合使用 $.operate.add()，$.operate.addTab() |
| updateUrl | String | Null | 修改URL 配合使用 $.operate.edit()，$.operate.editTab() |
| removeUrl | String | Null | 删除URL 配合使用 $.operate.remove() |
| exportUrl | String | Null | 导出URL 配合使用 $.table.exportExcel() |
| importUrl | String | Null | 导入URL 配合使用 $.table.importExcel() |
| detailUrl | String | Null | 详细URL 配合使用 $.operate.detail() |
| cleanUrl | String | Null | 清空URL 配合使用 $.operate.clean() |
| importTemplateUrl | String | Null | 摸版URL 配合使用 $.table.importTemplate() |
| height | String | undefined | 表格的高度 |
| striped | String | false | 是否显示行间隔色 |
| sortName | String | Null | 排序列名称 |
| sortOrder | String | Null | 排序方式 asc 或者 desc |
| pagination | Boolean | true | 默认为true表格的底部工具栏会显示分页条，设为false不显示 |
| pageSize | int | 10 | 每页的记录行数（\*） |
| id | String | bootstrap-table | 表格ID属性 |
| toolbar | String | toolbar | 表格工具栏ID属性 |
| escape | Boolean | false | 是否转义HTML字符串 |
| showFooter | Boolean | false | 默认为false隐藏表尾，设为true显示 |
| sidePagination | String | server | server启用服务端分页client客户端分页 |
| search | Boolean | true | 默认为true显示搜索框功能，设为false隐藏 |
| showSearch | Boolean | true | 默认为true显示检索信息，设为false隐藏 |
| showPageGo | Boolean | false | 默认为false不显示跳转页，设为true显示 |
| showRefresh | Boolean | true | 默认为true显示刷新按钮，设为false隐藏 |
| showColumns | Boolean | true | 默认为true显示某列下拉菜单，设为false隐藏 |
| showToggle | Boolean | true | 默认为true显示视图切换按钮，设为false隐藏 |
| showExport | Boolean | true | 默认为true显示导出文件按钮，设为false隐藏 |
| clickToSelect | Boolean | false | 默认为false不启用点击选中行，设为true启用 |
| fixedColumns | Boolean | false | 默认为false禁用冻结列，设为true启用冻结列（左侧） |
| fixedNumber | int | 0 | 冻结列的个数，当fixedColumns设为true有效（左侧） |
| rightFixedColumns | Boolean | false | 默认为false禁用冻结列，设为true启用冻结列（右侧） |
| rightFixedNumber | int | 0 | 冻结列的个数，当fixedColumns设为true有效（右侧） |
| params | Array | Null | 当请求数据时，你可以通过修改queryParams向服务器发送参数 |
| columns | Array | Null | 默认空数组，在JS里面定义 参考列的各项(Column options ) |
| responseHandler | Array | Null | 在加载服务器发送来的数据之前，处理数据的格式 |
| onLoadSuccess | Array | Null | 当所有数据被加载时触发处理函数 |

列的各项(Column options )

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| radio | Boolean | false | 默认false不显示radio（单选按钮），设为true则显示，radio宽度是固定的 |
| checkbox | Boolean | false | 默认false不显示checkbox（复选框），设为true则显示，checkbox的每列宽度已固定 |
| field | String | Null | 是每列的字段名，不是表头所显示的名字，通过这个字段名可以给其赋值，相当于key，表内唯一 |
| title | String | Null | 这个是表头所显示的名字，不唯一，如果你喜欢，可以把所有表头都设为相同的名字 |
| titleTooltip | String | true | 当悬浮在某控件上，出现提示 - 参考 Bootstrap 提示工具（Tooltip）插件 |
| class | boolean | false | 表格列样式 |
| rowspan | Number | true | 每格所占的行数 |
| colspan | Number | true | 每格所占的列数 |
| align | String | true | 每格内数据的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| halign | String | true | table header（表头）的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| falign | String | true | table footer（表脚，的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| valign | String | true | 每格数据的对齐方式，有：top（靠上）、middle（居中）、bottom（靠下） |
| width | Number | Null | 每列的宽度。如果没有自定义宽度自适应 |
| sortable | Boolean | false | 默认false就默认显示，设为true则会被排序 |
| order | String | asc | 默认的排序方式为"asc（升序）"，也可以设为"desc（降序）"。 |
| visible | Boolean | true | 默认为true显示该列，设为false则隐藏该列 |
| cardVisible | Boolean | true | 默认为true显示该列，设为false则隐藏。 |
| switchable | Boolean | true | 默认为true显示该列，设为false则禁用列项目的选项卡。 |
| clickToSelect | Boolean | true | 默认true不响应，设为false则当点击此行的某处时，不会自动选中此行的checkbox（复选框）或radiobox（单选按钮） |
| formatter | Function | Null | 某格的数据转换函数，需要三个参数： -value： field（字段名） -row：行的数据 -index：行的（索引）index |
| footerFormatter | Function | Null | 某格的数据转换函数，需要一个参数： -data： 所有行数据的数组 函数需要返回（return）footer某格内所要显示的字符串的格式 |
| events | Object | Null | 当某格使用formatter函数时，事件监听会响应，需要四个参数： -event：-value：字段名 -row：行数据 -index：此行的index |
| sorter | Function | Null | 自定义的排序函数，实现本地排序，需要两个参数： - a：第一个字段名 - b：第二个字段名 |
| sortName | String | Null | 排序列名称 |
| cellStyle | Function | Null | 对某列中显示样式改变 |
| searchable | Boolean | true | 默认true，表示此列数据可被查询 |
| searchFormatter | Boolean | true | 默认true，可使用格式化的数据查询 |
| escape | Boolean | false | 是否转义HTML字符串 |

* 表单搜索 $.table.search
  1. <a onclick="$.table.search();">搜索</a>
* 表格数据导出 $.table.exportExcel
  1. <a onclick="$.table.exportExcel();">导出</a>
* 数据模板下载 $.table.importTemplate
  1. <a onclick="$.table.importTemplate();">下载模板</a>
* 表格数据导入 $.table.importExcel
  1. <a onclick="$.table.importExcel();">导入</a>
  2. <form id="importForm" enctype="multipart/form-data" class="mt20 mb10" style="display: none;">
  3. <div class="col-xs-offset-1">
  4. <input type="file" id="file" name="file"/>
  5. <div class="mt10 pt5">
  6. <input type="checkbox" id="updateSupport" name="updateSupport" title="如果登录账户已经存在，更新这条数据。"> 是否更新已经存在的用户数据
  7. <a onclick="$.table.importTemplate()" class="btn btn-default btn-xs"><i class="fa fa-file-excel-o"></i> 下载模板</a>
  8. </div>
  9. <font color="red" class="pull-left mt10">
  10. 提示：仅允许导入“xls”或“xlsx”格式文件！
  11. </font>
  12. </div>
  13. </form>
* 表格销毁 $.table.destroy
  1. <a onclick="$.table.destroy();">销毁</a>
* 表格数据刷新 $.table.refresh
  1. <a onclick="$.table.refresh();">刷新</a>
* 选择表格行具体列 $.table.selectColumns
  1. var loginName = $.table.selectColumns("loginName");
* 选择表格行首列 $.table.selectFirstColumns
  1. var firstColumn = $.table.selectFirstColumns();
* 显示表格特定的列 $.table.showColumn
  1. $.table.showColumn("userName");
* 隐藏表格特定的列 $.table.hideColumn
  1. $.table.hideColumn("userName");
* 序列号生成 $.table.serialNumber
  1. {
  2. title: "序号",
  3. formatter: function (value, row, index) {
  4. return $.table.serialNumber(index);
  5. }
  6. },
* 超出指定长度浮动提示 $.table.tooltip
  1. {
  2. field: 'remark',
  3. title: '备注',
  4. align: 'center',
  5. formatter: function(value, row, index) {
  6. return $.table.tooltip(value);
  7. }
  8. },
* 回显数据字典 $.table.selectDictLabel
  1. var datas = [[${@dict.getType('sys\_common\_status')}]];
  2. {
  3. field: 'status',
  4. title: '用户状态',
  5. align: 'center',
  6. formatter: function(value, row, index) {
  7. return $.table.selectDictLabel(datas, value);
  8. }
  9. },
* 下拉按钮切换 $.table.dropdownToggle
  1. formatter: function(value, row, index) {
  2. var actions = [];
  3. actions.push('<a class="' + editFlag + '" href="#" onclick="$.operate.edit(\'' + row.deptId + '\')"><i class="fa fa-edit"></i>编辑</a>');
  4. actions.push('<a class="' + removeFlag + '" href="#" onclick="$.operate.remove(\'' + row.deptId + '\')"><i class="fa fa-trash"></i>删除</a>');
  5. actions.push('<a class="' + addFlag + '" href="#" onclick="$.operate.add(\'' + row.deptId + '\')"><i class="fa fa-plus"></i>添加下级部门</a>');
  6. return $.table.dropdownToggle(actions.join(''));
  7. }

### 弹层使用

弹层组件目前基于layer组件进行封装,提供了弹出、消息、提示、确认、遮罩处理等功能。

* 提供成功、警告和错误等反馈信息
  1. $.modal.msg("默认反馈");
  2. $.modal.msgError("错误反馈");
  3. $.modal.msgSuccess("成功反馈");
  4. $.modal.msgWarning("警告反馈");
* 提供成功、警告和错误等提示信息
  1. $.modal.alert("默认提示");
  2. $.modal.alertError("错误提示");
  3. $.modal.alertSuccess("成功提示");
  4. $.modal.alertWarning("警告提示");
  5. $.modal.confirm("确认信息", function() {});
* 提供弹出层信息
  1. // title 标题 url 请求链接 width 宽度 height 高度 options 选项
  2. $.modal.open(title, url, width, height);
  3. $.modal.openTab(title, url);
  4. $.modal.openOptions(options);
  5. $.modal.openFull(title, url, width, height);
  6. $.modal.close();
  7. $.modal.closeAll();
  8. $.modal.reload();
* 提供遮罩层信息
  1. $.modal.loading("正在导出数据，请稍后...");
  2. $.modal.closeLoading();

### 权限使用

使用thymeleaf模板整合了shiro标签 - 界面可以直接使用。（此处简单介绍两个，更多请参考官方文档）

1. <a href="#" shiro:hasPermission="system:user:add">包含权限字符串才能看到</a>
2. <a href="#" shiro:hasRole="admin">管理员才能看到</a>

如果需要在JS中使用权限，使用封装方法

1. var addFlag = [[${@permission.hasPermi('system:user:add')}]];
2. <a class="btn btn-success btn-xs ' + editFlag + '">包含权限字符串才能看到</a>

### 字典使用

配置好相关的数据字典信息即可正常使用（系统管理-字典管理）

1. <select name="status" th:with="type=${@dict.getType('sys\_normal\_disable')}">
2. <option value="">所有</option>
3. <option th:each="dict : ${type}" th:text="${dict.dictLabel}" th:value="${dict.dictValue}">
4. </option>
5. </select>

如果在想Table表格数据使用字典，使用formatter格式化

1. // 获取数据字典数据
2. var datas = [[${@dict.getType('sys\_normal\_disable')}]];
3. // 格式化数据字典
4. formatter: function(value, row, index) {
5. return $.table.selectDictLabel(datas, value);
6. }

### 参数使用

配置好相关的参数信息即可正常使用（系统管理-参数管理）

1. <body th:classappend="${@config.getKey('sys.index.skinName')}">

如果需要在JS中使用参数，使用封装方法

1. var skinName = [[${@config.getKey('sys.index.skinName')}]];
2. $("#id").val(skinName);