# 开发规范

目录

[开发规范 1](#_Toc16845989)

[1 命名规范 2](#_Toc16845990)

[2 application.yml文件 2](#_Toc16845991)

[3 多数据源 2](#_Toc16845992)

[3.1 数据源切换 3](#_Toc16845993)

[3.2 自定义数据源 3](#_Toc16845994)

[4 开发步骤 4](#_Toc16845995)

[4.1 前台 4](#_Toc16845996)

[4.1.1 布局 4](#_Toc16845997)

[4.1.2 列表按钮区 8](#_Toc16845998)

[4.1.3 查询条件区 9](#_Toc16845999)

[4.1.4 按钮 10](#_Toc16846000)

[4.1.5 表单 11](#_Toc16846001)

[4.1.6 文件导入 16](#_Toc16846002)

[4.1.7 数据表格 17](#_Toc16846003)

[4.1.8 弹出层 26](#_Toc16846004)

[4.1.9 树控件 30](#_Toc16846005)

[4.1.10 文件上传 32](#_Toc16846006)

[4.1.11 文件下载 33](#_Toc16846007)

[4.1.12 进度条 34](#_Toc16846008)

[4.1.13 单据打印 34](#_Toc16846009)

[4.1.14 选项卡 37](#_Toc16846010)

[4.1.15 Step控件 40](#_Toc16846011)

[4.2 后台 41](#_Toc16846012)

[4.2.1 Controller 41](#_Toc16846013)

[4.2.2 Service 43](#_Toc16846014)

[4.2.3 Dao 45](#_Toc16846015)

[4.2.4 工作流相关 45](#_Toc16846016)

[4.3 工具类 48](#_Toc16846017)

[4.4 事务规范 51](#_Toc16846018)

[4.4.1 spring事务使用规范 51](#_Toc16846019)

[4.4.2 分布式事务使用规范 52](#_Toc16846020)

# 命名规范

1. 文件夹和包名全部以小写英文字母命名。

例：com/joyintech/system/customerreport

1. 文件名需要使用“画面ID+编号+Controller/Service/Mapper/Dto（驼峰规则）”命名。

例：CustomerReportConfig001Controller.java、CustomerReportConfig001.html

1. JAVA代码中的变量名和方法名需遵守codeformat.xml和checkstyle.xml文件中的配置。此功能需要在eclipse中进行配置（参考开发环境搭建）。
2. 画面元素命名规则：画面ID\_btn\_功能

画面按钮必须遵循该命名规则，否则，部分按钮（如导出）功能将无法正常使用，按钮权限将无法被控制。

例： SysUser001\_btn\_add,SysUser001\_btn\_delete,SysUser001\_btn\_edit,SysUser002\_btn\_save等

<a id="SysUser001\_btn\_add" class="layui-btn layui-btn-sm btn-create">新建</a>

注：查询，导出按钮固定后缀为search和export。

查询按钮“画面ID\_btn\_search”

导出按钮“画面ID\_btn\_export”

1. 数据表格行内操作列按钮命名：画面ID\_TBLROW\_功能

当表格有操作列时，必须按照此命名规范进行画面元素基础数据的定义，否则，行内操作按钮的权限将无法正常使用。编辑，删除按钮固定后缀为edit和delete。

如果画面按钮中已经有了“删除”按钮，则无需定义行内的删除按钮，权限控制时，行内删除和删除按钮权限相同。

例：SysUser001\_TBLROW\_edit

# application.yml文件

application.yml的默认配置：



# 多数据源

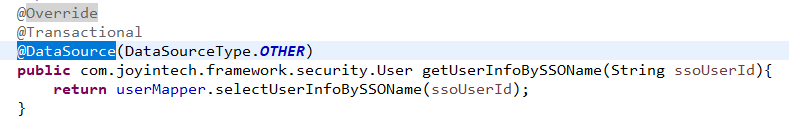
框架在理论上支持任意多数据源，每种数据源均可以在application.yml配置文件中指定。

框架默认配置了base和mast两个数据源。分别为base.datasource和spring.datasource。

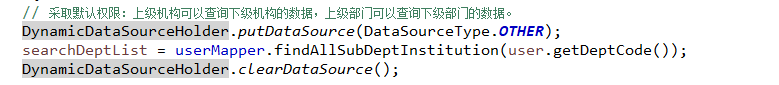
## 数据源切换

切换数据源有两种方式，如下代码所示：

第一种，@DataSource配合@Transaction一起使用。



第二种，使用DynamicDataSourceHolder切换数据。



## 自定义数据源

例：MyDataSourceAppender.java

@Component

public class MyDataSourceAppender extends DataSourceAppender {

@ConfigurationProperties(prefix = "my.datasource")

class BaseDataSourceProperties extends ConnectionPoolProperties {

}

@EnableConfigurationProperties(BaseDataSourceProperties.class)

class BaseDataSource {

/\*\*

\* @return DataSource

\*/

@Bean(name = "myDataSource", destroyMethod = "close")

public DataSource dataSource(BaseDataSourceProperties properties) {

DataSource dataSource = null;

if(StringUtil.isNotEmpty(properties.getJndiName())) {

JndiDataSourceLookup dataSourceLookup = new JndiDataSourceLookup();

dataSourceLookup.setResourceRef(true);

try {

dataSource = dataSourceLookup.getDataSource(properties.getJndiName());

} catch (DataSourceLookupFailureException e) {

logger.error("无法获取JNDI数据源：" + properties.getJndiName(), e);

throw e;

}

}

else {

dataSource = properties.initializeDataSourceBuilder().build();

}

if(dataSource instanceof BasicDataSource) {

DatabaseDriver databaseDriver = DatabaseDriver.fromJdbcUrl(properties.determineUrl());

String validationQuery = databaseDriver.getValidationQuery();

if (validationQuery != null) {

((BasicDataSource)dataSource).setTestOnBorrow(true);

((BasicDataSource)dataSource).setValidationQuery(validationQuery);

bindPropertiesForBasicDataSource((BasicDataSource)dataSource, properties);

}

}

initDataSource(DataSourceType.OTHER, properties);

return dataSource;

}

}

}

MyDataSourceType.java

public class MyDataSourceType {

/\*\*

\* 自定义数据源

\*/

public static final String MY = "myDataSource";

}

对应的yml文件

mydb:

datasource:

type: org.apache.commons.dbcp2.BasicDataSource

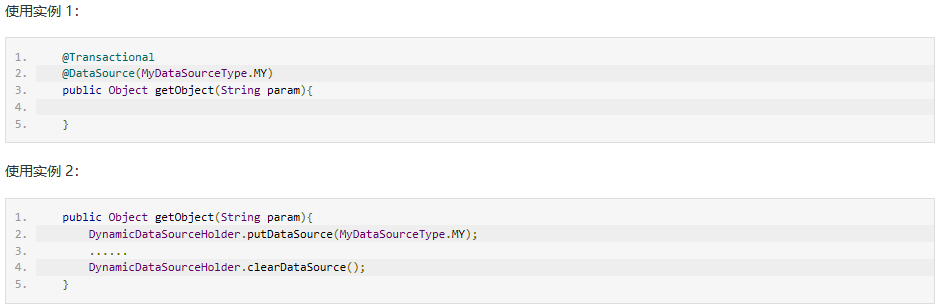
driverClassName: oracle.jdbc.driver.OracleDriver

url: jdbc:oracle:thin:@//192.168.70.180:1521/TAMS2

username: tams\_base

password: ENC(V4BaOp/IIOSK9SnSZE2oHCg9PB5612pH)

validationQuery: SELECT 1 FROM DUAL



# 开发步骤

系统中的每一个业务画面都是由前台和后台组成。前台包括JS、HTML文件以及CSS。后台主要包含Controller（控制器，负载处理画面的请求和跳转）、Service（业务逻辑层，具体的业务逻辑实现）、Dao（数据访问层，与数据库交互）、Dto（数据传输对象）、Validation（业务逻辑校验代码）。

## 前台

前台基于使用了Layui、JQuery、ECharts、Select2、lodash、ztree、suggest等第三方UI和类库，并在此基础上做了定制化开发。前台开发涉及静态画面以及和后台交互使用的JSON数据格式。需要在定义好数据结构的基础上进行开发。

### 布局

栅格系统（https://www.layui.com/doc/）

为了丰富网页布局，简化 HTML/CSS 代码的耦合，并提升多终端的适配能力，layui 在 2.0 的版本中引进了自己的一套具备响应式能力的栅格系统。将容器进行了 12 等分，预设了 4\*12 种 CSS 排列类，它们在移动设备、平板、桌面中/大尺寸四种不同的屏幕下发挥着各自的作用。

1. 栅格布局规则：
2. 采用 layui-row 来定义行
3. 采用类似 layui-col-md 这样的预设类来定义一组列（column），且放在行（row）内。其中：

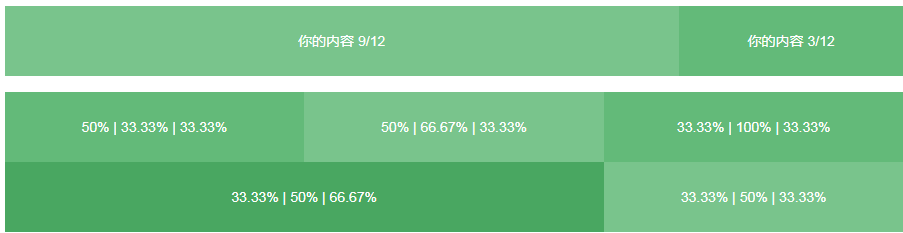
*变量md 代表的是不同屏幕下的标记（可选值见下文）*

*变量n代表的是该列所占用的12等分数（如6/12），可选值为 1 - 12*

如果多个列的“等分数值”总和等于12，则刚好满行排列。如果大于12，多余的列将自动另起一行。

1. 列可以同时出现最多四种不同的组合，分别是：xs（超小屏幕，如手机）、sm（小屏幕，如平板）、md（桌面中等屏幕）、lg（桌面大型屏幕），以呈现更加动态灵活的布局。
2. 可对列追加类似 layui-col-space5、 layui-col-md-offset3 这样的预设类来定义列的间距和偏移。
3. 最后，在列（column）元素中放入你自己的任意元素填充内容，完成布局！

示例（这里只是大致列举两个，更多实例请前往https://www.layui.com/ 查看）



<div class="layui-container">

常规布局（以中型屏幕桌面为例）：

<div class="layui-row">

<div class="layui-col-md9">

你的内容 9/12

</div>

<div class="layui-col-md3">

你的内容 3/12

</div>

</div>

移动设备、平板、桌面端的不同表现：

<div class="layui-row">

<div class="layui-col-xs6 layui-col-sm6 layui-col-md4">

移动：6/12 | 平板：6/12 | 桌面：4/12

</div>

<div class="layui-col-xs6 layui-col-sm6 layui-col-md4">

移动：6/12 | 平板：6/12 | 桌面：4/12

</div>

<div class="layui-col-xs4 layui-col-sm12 layui-col-md4">

移动：4/12 | 平板：12/12 | 桌面：4/12

</div>

<div class="layui-col-xs4 layui-col-sm7 layui-col-md8">

移动：4/12 | 平板：7/12 | 桌面：8/12

</div>

<div class="layui-col-xs4 layui-col-sm5 layui-col-md4">

移动：4/12 | 平板：5/12 | 桌面：4/12

</div>

</div>

</div>

1. 响应式规则：

栅格的响应式能力，得益于CSS3媒体查询（Media Queries）的强力支持，从而针对四类不同尺寸的屏幕，进行相应的适配处理



注：在开发画面的过程中，我们需要支持800-1920分辨率，在运用layer布局控件时，应该把这些layui-col-xs，layui-col-sm,layui-col-md,layui-col-lg响应式的布局都写全

代码示例：

<div class="layui-inline layui-col-md4 layui-col-xs4 layui-col-sm4 layui-col-lg4">

1. 响应式公共类：



1. 布局容器：

将栅格放入一个带有 class=”layui-container” 的特定的容器中，以便在小屏幕以上的设备中固定宽度，让列可控。

<div class="layui-container">

<div class="layui-row">

……

</div>

</div>

当然，你还可以不固定容器宽度。将栅格或其它元素放入一个带有 class=”layui-fluid” 的容器中，那么宽度将不会固定，而是 100% 适应

<div class="layui-fluid">

……

</div>

1. 列间距：

通过“列间距”的预设类，来设定列之间的间距。且一行中最左的列不会出现左边距，最右的列不会出现右边距。列间距在保证排版美观的同时，还可以进一步保证分列的宽度精细程度。我们结合网页常用的边距，预设了 12 种不同尺寸的边距，分别是：



下面是一个简单的例子，列间距为10px：

<div class="layui-row layui-col-space10">

<div class="layui-col-md4">

1/3

</div>

<div class="layui-col-md4">

1/3

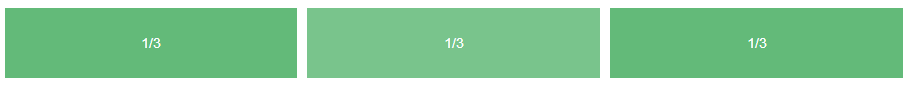
</div>

<div class="layui-col-md4">

1/3

</div>

</div>



如果需要的间距高于30px（一般不常见），请采用偏移。

1. 列偏移：

对列追加 类似 layui-col-md-offset 的预设类，从而让列向右偏移。其中 号代表的是偏移占据的列数，可选中为 1 - 12。  
如：layui-col-md-offset3，即代表在“中型桌面屏幕”下，让该列向右偏移3个列宽度

下面是一个采用“列偏移”机制让两个列左右对齐的实例



<div class="layui-row">

<div class="layui-col-md4">

4/12

</div>

<div class="layui-col-md4 layui-col-md-offset4">

偏移4列，从而在最右

</div>

</div>

请注意，列偏移可针对不同屏幕的标准进行设定，比如上述的例子，只会在桌面屏幕下有效，当低于桌面屏幕的规定的临界值，就会堆叠排列。

1. 栅格嵌套：

理论上，你可以对栅格进行无穷层次的嵌套，这更加增强了栅格的表现能力。而嵌套的使用非常简单。在列元素（layui-col-md\*）中插入一个行元素（layui-row），即可完成嵌套。下面是一个简单的例子：

<div class="layui-row layui-col-space5">

<div class="layui-col-md5">

<div class="layui-row grid-demo">

<div class="layui-col-md3">

内部列

</div>

<div class="layui-col-md9">

内部列

</div>

<div class="layui-col-md12">

内部列

</div>

</div>

</div>

<div class="layui-col-md7">

<div class="layui-row grid-demo grid-demo-bg1">

<div class="layui-col-md12">

内部列

</div>

<div class="layui-col-md9">

内部列

</div>

<div class="layui-col-md3">

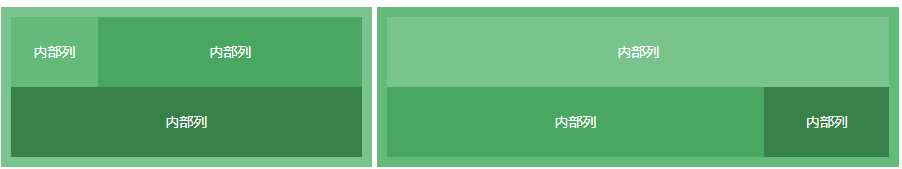
内部列

</div>

</div>

</div>

</div>



### 列表按钮区

组件对应的伪类：form-button-area

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 可选值 | 默认 | 注释 |
| filter | true/false | true | 筛选按钮是否显示 |
| export | true/false | true | 导出按钮是否显示 |
| position | left/right | left | 按钮位置 |

* filter：支持boolean类型的值，默认为true，显示“过滤”按钮
* export：支持boolean类型的值，默认为true，显示“导出”按钮
* position：可选值为left/right，默认为left，控件中的按钮对齐方式

*备注：过滤按钮默认放在所有按钮的最前面，导出按钮默认放置在最后。如果想要改变按钮的位置，请设置按钮不展示，并且对按钮进行自定义。按钮ID生成规则为“画面ID\_btn\_filter(export)”。*

代码实例：

<div class="form-button-area" export="false">

<a id="SysUser001\_btn\_add" class="layui-btn layui-btn-sm btn-create" data-th-text="#{SysCommon.create}">新建</a>

<a id="SysUser001\_btn\_export" class="layui-btn layui-btn-sm btn-export" data-th-text="#{SysCommon.export}">导出</a>

<a id="SysUser001\_btn\_delete" class="layui-btn layui-btn-sm btn-delete" data-th-text="#{SysCommon.delete}" style="display:none;">删除</a>

<a id="SysUser001\_btn\_pwdreset" class="layui-btn layui-btn-sm btn-pwdreset" data-th-text="#{SysCommon.pwdreset}" style="display:none;">密码重置</a>

<a id="SysUser001\_btn\_lock" class="layui-btn layui-btn-sm btn-frozen" data-th-text="#{SysCommon.frozen}" style="display:none;">冻结</a>

<a id="SysUser001\_btn\_unlock" class="layui-btn layui-btn-sm btn-unfrozen" data-th-text="#{SysCommon.unfrozen}" style="display:none;">解冻</a>

</div>

效果图：



### 查询条件区

组件对应的伪类：condition-area

1. 控件会在表单元素的右侧生成一个100像素宽度的按钮区域，该区域的按钮会根据表单中layui-form-item的个数进行动态生成（生成规则：1行查询条件，只生成一个“查询”按钮；2行查询条件，生成“查询”和“重置”按钮；3行及三行以上查询条件，生成“查询”、“重置”以及“更多”三个按钮，同时将超过2行的条件进行隐藏，默认只展示2行条件），并且按钮垂直排列。
2. 控件默认存在三种按钮（查询、重置、更多）。“查询”按钮事件需要自定义，重置和更多按钮在开发画面是无需关注，控件默认实现了这两个按钮的事件。按钮ID生成规则为“画面ID\_btn\_search(reset)”。
3. 针对画面开发，只需要书写condition-area下的表单元素即可，无需添加按钮。
4. 没有使用当前控件进行开发的画面需要修改为该控件，只需要将form最外层添加一个<div class="joyin-form-wrap joyin-filter-form condition-area"></div>(注：如果“列表按钮区域”控件中filter=“false”，则去除“joyin-filter-form”样式)，同时删除原表单的按钮，删除joyin-form-wrap、 joyin-filter-form样式，调整layui-col-md为合适的值即可。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 可选值 | 默认 | 注释 |
| line | Integer | 2 | 显示行数 |

* line：支持Integer类型的值，默认为2，查询条件默认显示的行数。当line等于1时，“更多”按钮将会显示到“查询”之后。

代码实例：

<div class="joyin-form-wrap joyin-filter-form condition-area">

<form class="layui-form" id="searchForm">

<div class="layui-form-item">

<div class="layui-inline layui-col-md4">

<label class="layui-form-label form-lable" data-th-text="#{SysUser001.userName}">用户名</label>

<div class="layui-input-inline input-inline">

<input type="text" id="userName" name="userName" class="layui-input form-input input-lenovoselect"

placeholder="请输入内容" data-th-placeholder="#{SysCommon.placeholder}"

data-th-data-url="@{/system/user/SysUser001/getUserName}"

data-idfield="code" data-keyfield="code"

data-th-data-effectivefieldsalias='|{"code":"#{SysUser001.userName}","shortName":"#{SysUser001.realName}"}|'>

</div>

</div>

<div class="layui-inline layui-col-md4">

<label class="layui-form-label form-lable" data-th-text="#{SysUser001.realName}">真实姓名</label>

<div class="layui-input-inline input-inline">

<input type="text" id="realName" name="realName" class="layui-input form-input" placeholder="请输入内容" data-th-placeholder="#{SysCommon.placeholder}">

</div>

</div>

<div class="layui-inline layui-col-md4">

<label class="layui-form-label form-lable" data-th-text="#{SysUser001.status}">数据状态</label>

<div class="layui-input-inline input-inline">

<input type="text" id="status" name="status"

class="layui-input form-input input-checkbox"

data-th-data-result='${STATUS}'

data-th-initValue='${statusE}'>

</div>

</div>

</div>

</form>

</div>

效果图：



### 按钮

按钮的几种形式

1. 含有btn-default类时，按钮的边框是灰色，背景是白色

代码实例：

<a id="SysDataDict001\_btn\_export" class="layui-btn layui-btn-sm btn-export btn-default>导出</a>

效果图：



1. 不含有btn-default类时，这时按钮就会依靠我们系统的主题色而变化，目前系统支持的主题颜色只有蓝色和绿色两种

代码实例：

<a id="SysDataDict001\_btn\_filter" class="layui-btn layui-btn-sm btn-filter">筛选</a>

效果图：



1. 按钮支持多种风格主题，目前只有暖色按钮和警告按钮两种形式

代码实例：

<a class="layui-btn layui-btn-sm layui-btn-warm">暖色按钮</a>

<a class="layui-btn layui-btn-sm layui-btn-warm">警告按钮</a>

效果图：



1. 按钮组，只需在一组按钮的加一个父类layui-btn-group

代码实例：

<div class="layui-btn-group">

<a id="SysInstitution001\_BTN\_add" class="layui-btn layui-btn-sm btn-create">新建</a>

<a id="SysInstitution001\_BTN\_export" class="layui-btn layui-btn-sm btn-export">导出</a>

<a id="SysInstitution001\_BTN\_delete" class="layui-btn layui-btn-sm btn-delete" style="display:none;">删除</a>

<a id="SysInstitution001\_btn\_filter" class="layui-btn layui-btn-sm btn-filter">筛选</a>

</div>

效果图：



1. 按钮组出现下拉选项，按钮下拉选项的前台展示直接用div的这一套，样式已经写好，直接在按钮组调用btn-select类

代码实例：

<div class="btn-select-wrapper">

<a class="layui-btn layui-btn-sm btn-select">下拉<i class="iconfont icon-down"></i></a>

<ul class="btn-select-wrap" style="display: none;">

<li><a href="javascript:;">导出XLS格式文件</a></li>

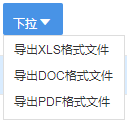
<li><a href="javascript:;">导出DOC格式文件</a></li>

<li><a href="javascript:;">导出PDF格式文件</a></li>

</ul>

</div>

效果图：



### 表单

注意：我们这里的表单只需要调用我们定义好的类就可以，这些特定的类是我们已经做好的初始化

1. 单行输入框，直接写上layui-input form-input类即可

代码实例：

<input type="text" class="layui-input form-input" placeholder="请输入内容" data-th-placeholder="#{SysCommon.placeholder}">

效果图：



1. 金额文本框，直接用text-money类进行初始化就可以

|  |  |
| --- | --- |
| 参数 | 说明 |
| data-pattern | 格式化表达式 |
| data-fixtype | 对于小数的处理(1：取整；2：四舍五入；3：向上取整；4:向下取整) |
| unit | 无单位，数值类型（格式化后的数据将会除以该单位） 默认值 1 |
| data-textrange | 具体保留几位小数 |

代码实例：

<input type="text" placeholder="请输入内容" data-pattern="" data-fixtype="2" unit="" class="layui-input form-input text-money">

效果图：



1. 数值类文本框，直接用text-number类进行初始化就可以

|  |  |
| --- | --- |
| 参数 | 说明 |
| data-pattern | 格式化表达式 |
| data-fixtype | 对于小数的处理(1：取整；2：四舍五入；3：向上取整；4:向下取整) |
| data-textrange | 具体保留几位小数 |

代码实例：

<input type="text" placeholder="请输入内容" data-pattern="" data-fixtype="2" class="layui-input form-input text-number">

效果图：



1. 时间选择器，时间选择器分为单日期选择和区间日期选择，用的是layer中时间控件

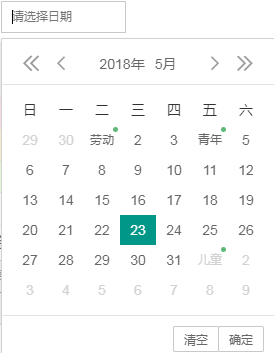
|  |  |
| --- | --- |
| 参数 | 说明 |
| type | 控件选择类型,year年选择器,month年月选择器,date日期选择器,time时间选择器,datetime日期时间选择器 |
| range | 开启左右面板范围选择,默认值：false |
| calendar | 是否显示公历节日默认值：false |
| showBottom | 是否显示底部栏默认值：true |
| btns | 工具按钮默认值：[‘clear’, ‘now’, ‘confirm’] |
| format | 自定义格式默认值：yyyy-MM-dd |
| single | 日期区间控件时才生效，默认是false，区间控件是否支持选择单边日期。目前支持的是选择年月日和年月日时分秒这两种，这个参数值为true时，日期控件会出现工具栏 |
| data-option | Json字符串，支持所有laydate参数。 |

* 1. 单日期时间选择器，直接用input-dateSelect类进行初始化就可以

代码实例：

<input class="layui-input form-input input-dateSelect" placeholder="请选择时间">

效果图：

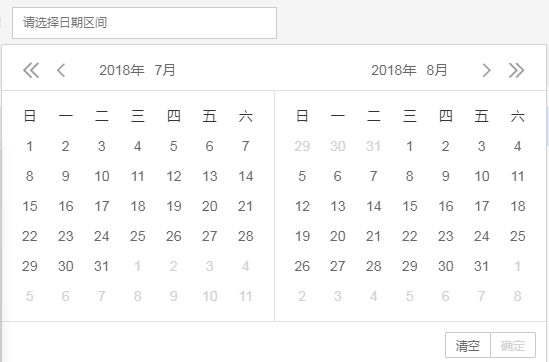


* 1. 日期区间选择器，直接用input-dateRangeSelect类进行初始化就可以

代码实例：

<input class="layui-input form-input input-dateRangeSelect" placeholder="请选择时间区间">

效果图：



1. 下拉框,下拉框包括单选下拉框和多选下拉框

|  |  |
| --- | --- |
| 参数 | 说明 |
| data-url | ajax请求后台url |
| initValue | 初始值（可选） |
| childrenIds | 指定联动子下拉框 |
| parentIds | 指定联动父下拉框 |
| multiple | 下拉框的多选 |
| allowClear | 是否允许清空（默认为true） |
| placeholder | 模糊查询输入框中的提示信息（可选） |
| data-result | 初始化model中获取，el表达式 |
| searchable | 如果该参数指定为false则不显示“搜索框” |
| formatoption | 是否格式化候选框，默认为true |
| showAll | 是否显示“全选”选项，默认为false，只对非data-url的多选下拉框生效。 |

注：data-result 和 data-url只需要指定一个，如果同时指定，则优先data-url

方法：

$selectElement.textSelect(‘setValue’, value) 是给data-url形式的下拉框赋值的。

$selectElement.select2(‘data’)[0] 是当需要选中值的额外属性时可以调用。

* 1. 单选下拉框，直接用input-select类进行初始化就可以

代码实例：

<select type="text" id="commonBusinessType" name="commonBusinessType" class="layui-input form-input input-select" data-th-data-result='${COMMON\_BUSINESS\_TYPE}' allowClear="true" placeholder="请输入内容" data-th-placeholder="#{SysCommon.placeholder}"> </select>

效果图：



* 1. 多选下拉框，也是调用input-select类进行初始化就可以，但是加了一个属性multiple=”true”

代码实例：

<select type="text" id="flowType" name="flowType" class="layui-input form-input input-select" data-th-data-result='${PROCESS\_STATE}' allowClear="true" multiple="true"placeholder="请输入内容" data-th-placeholder="#{SysCommon.placeholder}"> </select>

效果图：



1. 联想框,联想框包括单选联相框和多选联想框

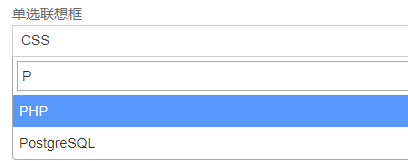
无论是单选还是多选，只要是通过data-url形式获取数据的，都会像后台发送initValue和search两个参数。initValue主要表明该控件需要显示的初始值，search则表示输入的过滤字符串。

* 1. 单选联想框,单选联想框用的和单选下拉框、多选下拉框是同一控件，都是调用input-select类进行初始化。

代码实例：

<input type="text" placeholder="请选择" class="layui-input form-input input-select" data-result='["PHP", "MySQL", "SQL", "PostgreSQL", "HTML", "CSS", "HTML5", "CSS3", "JSON"]'>

效果图：



* 1. 联想文本框,直接调用input-lenovoselect类进行初始化。该控件和下拉框的请求参数有所不同，

发送到后台的参数为param，并且不带有initValue。该控件大多用在查询条件中，不建议使用在新建和编辑页面。

代码实例：

<input type="text" id="taskGroup" name="taskGroup" class="layui-input form-input input-lenovoselect" placeholder="请输入内容" data-th-placeholder="#{SysCommon.placeholder}" data-th-data-url="@{/system/log/SysBatchLog001/getTaskGroup}" data-idfield="code" data-keyfield="shortName" data-th-data-effectivefieldsalias='|{"code":"#{SysBatchLog001.taskGroupId}","shortName":"#{SysBatchLog001.taskGroupName}"}|'>

效果图：



1. 特殊输入框

代码实例：

<input type="text" id="roleName" name="roleName" class="layui-input form-input input-fuzzy-text" placeholder="请输入账套代码或名称关键字" data-eventfunc="QueryFunc.serachKey">

<div class="layui-input-inline input-inline input-inline-wrap"> <input type="text" name="institutionName" class="layui-input form-input" placeholder="请输入内容"> <span class="input-right-show">%</span> </div>

<div class="layui-input-inline input-inline input-inline-wrap"> <span class="input-left-show">%</span> <input type="text" name="institutionName" class="layui-input form-input form-input-left" placeholder="请输入内容"> </div>

效果图：







1. 平铺选择框，input-checkbox

|  |  |
| --- | --- |
| 参数 | 说明 |
| initValue | 默认值（可选） |
| multiple | 多选（默认为单选） |
| data-result | 初始化显示值 |

代码实例：

<input type="text" id="openMode" class="layui-input form-input input-checkbox" intValue="01" data-result="[{"id":"01", "text":"开发式"}{"id":"02", "text":"封闭式"}]"/>

<script>

$('#openMode').val('02').trigger('change');

$('#openMode').prop('disabled', true).trigger('change');

</script>

效果图：



### 文件导入

文件导入，实现这一操作只需直接调用共通方法里的Joyin.uploadFile（）方法，调用共通的页面fileUpload.html

|  |  |
| --- | --- |
| 参数 | 说明 |
| type | 默认drag |
| singleImage | 单个图片上传 |
| mutiImage | 多图片上传 |
| allowFile | 允许上传指定后缀名的文件 |
| noAutoUpload | 选择文件后不自动上传文件 |
| drag | 允许拖拽文件上传 |
| list | 批量上传文件，同时显示文件列表 |
| size | 文件大小限制，单位是KB |
| auto | 选择文件后是否自动上传，默认为false。当type为noAutoUpload或list时，该参数不生效 |
| hideTpl | 是否隐藏模板下载按钮（如果显示，则下载当前pageId对应的模板文件） |
| pageId | 画面ID，默认为当前打开的画面ID（如果需要下载非当前画面模板时，需要指定pageId参数） |
| width | 文件上传画面的宽度 |
| height | 文件上传画面的高度 |
| uploadCallback | 上传成功后的回调函数，接收参数为上传后的文件名（多个文件名会是一个数组） |
| callback | 子画面关闭后调用 |



### 数据表格

参考文档 [*laytable*](http://www.layui.com/doc/modules/table.html)  
**基础参数**

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 数据类型 | 默认值 | 说明 |
| checkbox | Boolean | true | 是否显示checkbox |
| rowno | Boolean | false | 是否显示行号 |
| permissions | Boolean | true | 是否有权限控制 |
| contextmenu | Boolean | true | 是否显示右键菜单 |
| loadonce | Boolean | false | 是否前台分页 |
| height | String/Integer | full | 表格高度 |
| searchOnInit | Boolean | false | 是否在初始化的时候就加载数据 |
| checkOnClickRow | Boolean | true | 单击选中行 |
| tree | Boolean | false | 是否为树形表格 |
| lazyload | Boolean | false | 延迟加载, tree:true时生效 |
| addbtn | Boolean | false | 加号按钮是否显示 |
| addclick | Function |  | 工具栏加号按钮点击事件 |
| addToLast | Boolean | true | 点击加号按钮后，新添加行是否位于表格最后,在addbtn为true时生效 |
| addbtnPosition | String | top | 可选值为top和bottom，指定添加按钮位置，在addbtn为true时生效 |
| btns | Array | [] | 自定义表格工具栏按钮，上面的addbtn为true时只是该属性的一个特例。自定义按钮有三个属性{icon,title,action} |
| lazytime | Number | 300 | 延迟加载时间 |
| resize | Boolean | true | 是否自适应大小 |
| limits | Array | [10,20,30,40,50,60,70,80,90] | 每页条数的选择项 |
| rowSelectedClass | String | table-row-select | 行选中时的样式 |
| url | String | 详见异步数据接口 | 异步数据接口相关参数。其中详见异步接口 |
| page | Boolean | true | 开启分页 |
| limit | Number | 20 | 默认每页显示20条 |
| loading | Boolean | true | 显示加载条 |
| cols | Array | 详见表头参数 | 列定义 |
| done | Function | 详见done回调 | 数据渲染完的回调 |
| beforeEditCell | Function | 详见beforeEditCell | 编辑单元格之前调用的函数 |
| editCell | Function | 详见editCell | 单元格被编辑后的回调 |
| buttons | Object | 详见按钮设置 | 按钮设置 |
| even | Boolean | true | 开启隔行背景 |

异步数据接口

数据的异步请求由以下几个参数组成：

|  |  |
| --- | --- |
| 参数名 | 功能 |
| url | 数据接口访问地址 |
| type | AJAX请求方式 |
| data | 支持Function和普通对象 |
| dataType | 数据类型，可以为json和其它的contentType类型 |

调用示例：

var table=$("#dataTable").joyintable({

ajax: {

url: 'findPage',

type: 'POST',

data: function() {

return getSearchParams();

}

},

});

cols - 表头参数一览表

*cols参数为数组对象，如果在列比较少的时候，需要表格宽度自动铺满，则其中有至少一列的width不需要指定（action列不算）。*

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 数据类型 | 说明 | 示例值 |
| field | String | （必填项）设定字段名。字段名的设定非常重要，且是表格数据列的唯一标识 | institutionCode |
| width | Number/String | 设定列宽（默认自动分配）。支持填写：数字、百分比。请结合实际情况，对不同列做不同设定。 | 200或30% |
| title | String | （必填项）设定标题名称 | SysInstitution001.institutionCode |
| sortable | Boolean | 是否允许排序（默认：false）。如果设置 true，则在对应的表头显示排序icon，从而对列开启排序功能。 | false |
| edit | Boolean | 是否开启单元格编辑（默认不开启），如果开启，则默认会根据type设置值决定控件类型，如无type，则默认为text | false |
| editOption | Object | 详见列编辑设置 |  |
| fixed | String | 固定列。可选值有：left（固定在左）、right（固定在右）。一旦设定，对应的列将会被固定在左或右，不随滚动条而滚动。 注意：如果是固定在左，该列必须放在表头最前面；如果是固定在右，该列必须放在表头最后面。 | left |
| type | String | 设定列类型。可选值有：date、datetime、email、action、 money、number |  |
| digit | Number | 设定显示小数位，只有在type为money或者number时生效，默认值为2 |  |
| style | String | 自定义单元格样式。即传入 CSS 样式 | background-color: #5FB878; color: #fff; |
| rowspan | Number | 单元格所占行数（默认：1）。一般用于多级表头 | 2 |
| colspan | Number | 单元格所占列数（默认：1）。一般用于多级表头 | 3 |
| unresize | Boolean | 是否禁用拖拽列宽（默认：false） | false |
| minWidth | Number | 局部定义当前常规单元格的最小宽度（默认：60），一般用于列宽自动分配的情况 | 100 |
| render | Function | 自定义单元格内容，该方法接收的参数为[value,visible,row,options]，方法的返回值会被作为单元格内容 |  |

注：树形表格不允许排序

调用示例：

cols:[

{field:'institutionCode', width:100, title: 'SysInstitution001.institutionCode'}

,{field:'institutionName', width:200, title: 'SysInstitution001.institutionName'}

,{field:'institutionTypeName', width:100, title: 'SysInstitution001.institutionType', sortField:'institutionType'}

,{field:'upperInstitutionName', width:150, title: 'SysInstitution001.upperInstitutionName',sortField:'upperInstitutionCode'}

,{field:'insitituteCode', width:150, title: 'SysInstitution001.insitituteCode'}

,{field:'assPeople', width:100, title: 'SysInstitution001.assPeople'}

,{field:'assPhone', width:150, title: 'SysInstitution001.assPhone'}

,{field:'grdInfoName', width:150, title: 'SysInstitution001.grdInfo', sortField:'grdInfo'}

,{field:'provinceName', width:150, title: 'SysInstitution001.province', sortField:'province'}

,{field:'cityName', width:150, title: 'SysInstitution001.city', sortField:'city'}

,{field:'countiesName', width:150, title: 'SysInstitution001.counties', sortField:'counties'}

,{field:'other', width:150, title: 'SysInstitution001.other', sortable:false}

,{field:'remark', width:200, title: 'SysInstitution001.remark', sortable:false}

,{field:'operation', title: 'SysCommon.operation', type:'action', buttons: ['edit', 'delt']}

]

**done** - 数据渲染完的回调

类型：Function，默认值：无

无论是异步请求数据，还是直接赋值数据，都会触发该回调。你可以利用该回调做一些表格以外元素的渲染。

**beforeEditCell** - 单元格编辑前调用的函数

类型：Function，默认值：无，参数：preVal, field, rowData

在单元格被编辑之前调用，严格来说应该是在单元格开启的编辑属性后点击时调用。如果该方法返回值为false，将会阻止本次编辑。

**editCell** - 单元格编辑后调用的函数

类型：Function，默认值：无，参数：value, data, field, cell, column

在单元格被编辑后会调用该函数。比如，当单元格编辑类型为select时，可以用改方法来达到change事件的效果。

**按钮设置**

表格中的按钮一般在操作列中，目前定义了三种默认的按钮，如下所示：

buttons: {

edit: {

text: '&#xe642;',

clas: 'edit',

title: '编辑'

},

delt: {

text: '&#xe640;',

clas: 'delt',

title: '删除'

}

,

add: {

text: '&#xe654;',

clas: 'add',

title: '添加'

}

}

按钮也可以支持自定义添加

实例代码 ：

{

title: '操作', type: 'action', width: 20, buttons: [{

text: '&#xe614;',

clas: 'config',

title: '重新运行'

}]

}

**列编辑设置**

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 数据类型 | 说明 | 示例值 |
| type | String | 允许输入值为  text,number,money,date,datetime,dateRange,select | {type:’date’} |
| fixtype | Number | 取舍类型（1：整数，2：四舍五入，3：Math.ceil，4：Math.floor），在type为number或money时生效 | {type:’number’,fixtype:2} |
| pattern | String | 数值格式，在type为number或money时生效 | ‘#.00’ |
| textrange | Number | 显示小数位，该值的优先级高于col中设置的digit，在type为number或money时生效 | {type:’number’,textrange:6} |
| unit | Number | 金额单位，优先级高于表格动态设置的金额单位，仅在type为money时生效 | {type:’money’,unit: 10000} |
| formatter | String | 时间格式化字符串，在type为date、dateRange或datetime时生效，当type=date（type=dateRange）时为”yyyy-MM-dd”，当type=datetime时为”yyyy-MM-dd HH:mm:ss” |  |
| showField | String | 指定单元格显示值，可选值为id或text（默认），仅在type为select时生效 |  |
| url | String | 异步获取数据的接口，仅在type为select时生效，与result不共存 |  |
| result | String | 静态数据JSON字符串，仅在type为select时生效，与url不共存 |  |
| id | String | 下拉框ID，联动下拉框时必须指定 |  |
| childrenIds | String | 联动子下拉框ID（对应id属性值），多个可以用逗号分隔 |  |
| parentIds | String | 联动父下拉框ID（对应id属性值），多个可以用逗号分隔 |  |
| multiple | Boolean | 是否允许多选，默认值为false |  |
| searchable | Boolean | 是否显示搜索框，默认为true |  |
| initValue | String | 编辑框的初始值 |  |

以上属性值与对应控件的属性值一致  
※ 在select编辑时，获取的行数据中有个同field对应的Val的字段，存储了下拉框ID值

**基础方法**

1.绑定编辑事件

该方法可以对表格中的每一行数据进行编辑操作

table.bind('click', '.edit', function(data, index) {})

2.绑定删除事件

该方法是对表格中的数据进行删除

渲染方式

table.bind('click', '.delt', function(data, index) {})

3.双击查看事件

该方法是对表格中的每一行进行双击查看

渲染方式

dblclick:function(table, data) {}

4.重新加载

该方法是对表格重新加载

渲染方式:调用reload()方法，该方法只有在表格加载后调用之后才能正常使用

table.reload();

5.加载

该方法是对表格初始化加载

渲染方式:load()方法

table.load();

6.获取一行

该方法是获取表格的某一行

渲染方式:getRow(rowIndex)方法，参数rowIndex是行的索引

table.getRow(rowIndex);

7.获取所有选中行

该方法是获取表格的所有选中行

渲染方式:getAllCheckedRows()方法

table.getAllCheckedRows();

8.获取所有行

该方法是获取表格的所有行

渲染方式:getAllRows()方法

table.getAllRows();

9.选中行

该方法是获取表格的选中行

渲染方式:selectRow(rowData)方法，参数rowData是行内数据

table.selectRow(rowData);

10.添加行

该方法是对表格的行进行添加

渲染方式:addRow(rowData)方法，参数rowData是行内数据

table.addRow(rowData);

11.删除行（行数据）

该方法是对表格的行进行删除

渲染方式:removeRow(rowData)方法，参数rowData是行内数据

table.removeRow(rowData);

12.删除行（行号）

该方法是对表格的行进行删除

渲染方式:deleteRow(index)方法，参数index是行号

table.deleteRow(index);

13.bind事件

表格按钮绑定事件

渲染方式:参数：eventStr是事件的字符串，selector是dom选择器，fun函数

bind: function(eventStr, selector, fun) {})

14.判断表格是否有选中数据

该方法是判断表格中的数据是否被选中

渲染方式

table.isSearched();

15.选中行

选中表格中的指定行，参数index是行号

table.checkRow(index);

16.取消选中行

取消表格中的所有选中状态

table.uncheck();

17.清空表格

清空当前表格中的数据，可以传入一个函数，在表格清空后执行

table.clear(callback);

18.设置列显示和隐藏

在表格加载完成后，可以使用该方法动态设置表格的列展示与否

参数： Array 列的field属性，以及visible属性

例： table.showColumns([{field:'flowType', visible: true}, {field:'commonBusinessType',visible: false}]);

19.动态改变行内容

该方法可以修改行内容

参数： rowIndex 行号，data 被更新的数据，只需要包含被更改的属性

例： $('body').on('change', '#province', function(e, data) {

table.renderRow(data.rowIndex, {nowUser: data.value.join(',')});

});

**基本表格实例：**

<table class="layui-table" id="dataTable"></table>

var table=$("#dataTable").joyintable({

ajax: {

url: 'findPage',

type: 'POST',

data: function() {

return getSearchParams();

}

},

dblclick:function(table, data) {

Joyin.dialog(JoyinUtil.getLabel("SysCommon.refrence"),

basePath + "system/institution/SysInstitution002/init",

"95%", "90%", {pageModel:PAGE\_MODEL\_REFERENCE,

institutionCode:data.institutionCode,

status:data.status});

},

cols:[

{field:'institutionCode', width:100, title: 'SysInstitution001.institutionCode'}

,{field:'institutionName', width:200, title: 'SysInstitution001.institutionName'}

,{field:'institutionTypeName', width:100, title: 'SysInstitution001.institutionType', sortField:'institutionType'}

,{field:'upperInstitutionName', width:150, title: 'SysInstitution001.upperInstitutionName',sortField:'upperInstitutionCode'}

,{field:'insitituteCode', width:150, title: 'SysInstitution001.insitituteCode'}

,{field:'assPeople', width:100, title: 'SysInstitution001.assPeople'}

,{field:'assPhone', width:150, title: 'SysInstitution001.assPhone'}

,{field:'grdInfoName', width:150, title: 'SysInstitution001.grdInfo', sortField:'grdInfo'}

,{field:'provinceName', width:150, title: 'SysInstitution001.province', sortField:'province'}

,{field:'cityName', width:150, title: 'SysInstitution001.city', sortField:'city'}

,{field:'countiesName', width:150, title: 'SysInstitution001.counties', sortField:'counties'}

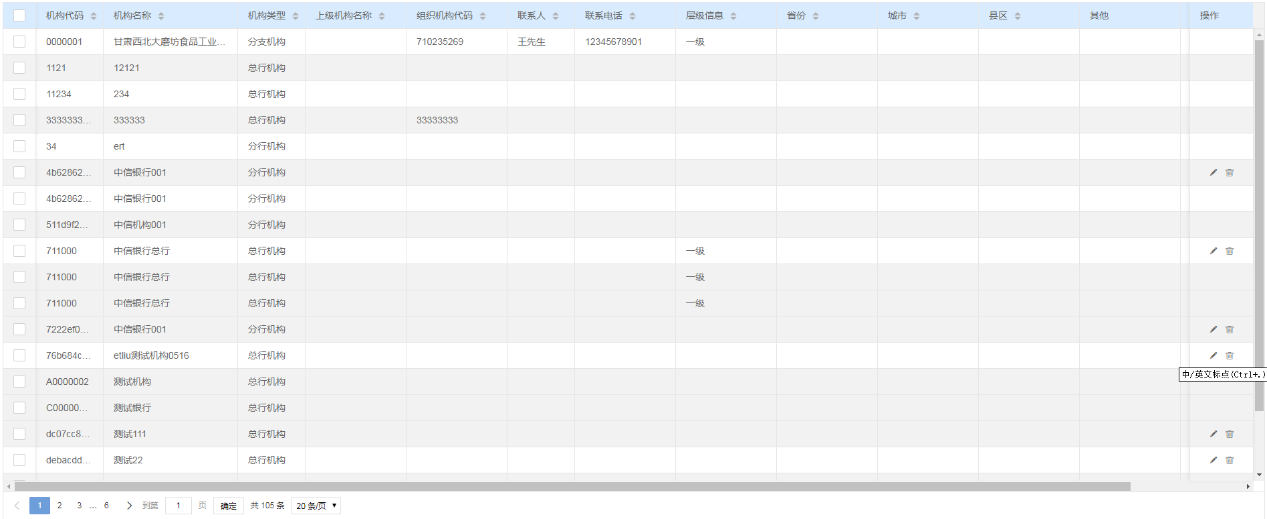
,{field:'other', width:150, title: 'SysInstitution001.other', sortable:false}

,{field:'remark', width:200, title: 'SysInstitution001.remark', sortable:false}

,{field:'operation', title: 'SysCommon.operation', type:'action', buttons: ['edit', 'delt']}

]

});

****

**可编辑表格实例：**

var table=$("#dataTable").joyintable({

ajax: {

url: 'queryList',

type: 'POST',

data: function() {

return getSearchParams();

}

},

cols:[

{field:'flowType', width:150, title: 'WorkflowMonitor001.flowType', edit:true, editOption:{type:'select', multiple: true,result: ADMINISTRATIVE\_TYPE1}}

,{field:'commonBusinessType', width:200, title: 'WorkflowMonitor001.commonBusinessType', edit:true, editOption:{type:'select', multiple: true, url: JoyinUtil.getRealPath('/system/institution/SysInstitution002/getCity')}}

,{field:'businessCode', width:200, title: 'WorkflowMonitor001.businessCode', edit:true, editOption:{type:'select', url: JoyinUtil.getRealPath('/system/institution/SysInstitution002/getCounties')}}

,{field:'nowUser', sortField:'nowUserId',width:200, title: 'WorkflowMonitor001.nowUser',edit:true,type:'money'}

,{field:'createUser', width:200, title: 'WorkflowMonitor001.createUser'}

,{field:'createTaskTime', title: 'WorkflowMonitor001.createTaskTime', type:'datetime', edit:true}

]

});

**树形表格实例：**

当开启树型表格时，只需将属性tree的值设为true ，lazyload:true只有在树形表格下生效

<table class="layui-table" id="dataTable"></table>

var table=$("#dataTable").joyintable({

ajax: {

url: 'findPage',

type: 'POST',

data: function() {

return getSearchParams();

}

},

tree:true,

lazyload:true,

dblclick:function(table, data) {

Joyin.dialog(JoyinUtil.getLabel("SysCommon.refrence"),

basePath + "system/department/SysDepartment002/init",

"95%", "80%", {pageModel:PAGE\_MODEL\_REFERENCE,

deptCode:data.deptCode,

status:data.status});

},

cols:[

{field:'id', width:150, title: 'SysDepartment001.deptCode'}

,{field:'name', width:170, title: 'SysDepartment001.deptName'}

,{field:'deptFunName', width:170, title: 'SysDepartment001.deptFun'}

,{field:'institutionName', width:170, title: 'SysDepartment001.institutionName'}

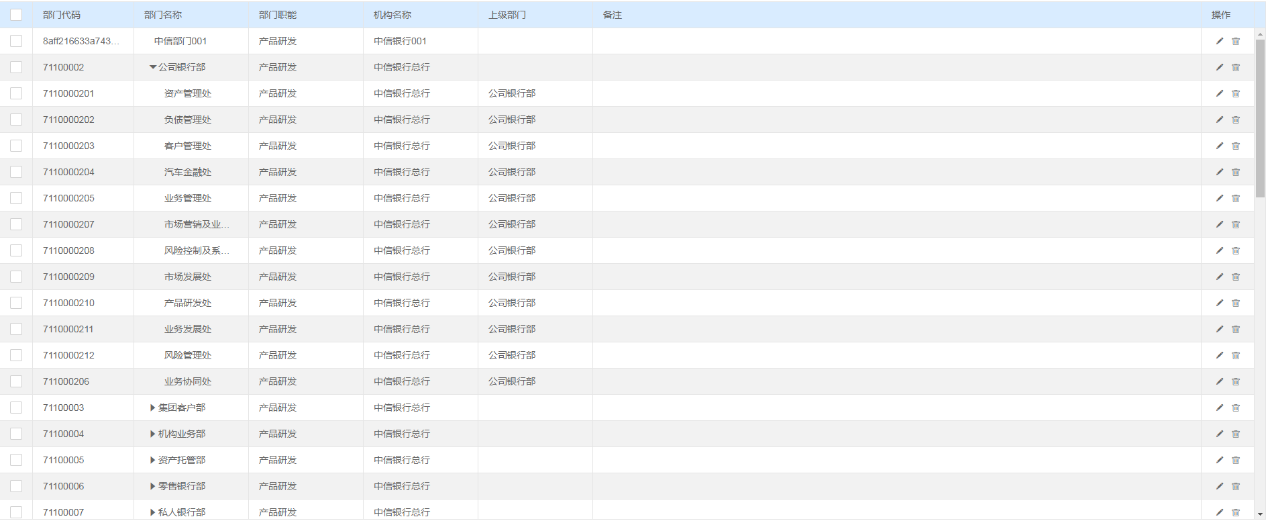
,{field:'upperDeptCodeName', width:170, title: 'SysDepartment001.upperDeptCode'}

,{field:'remark', title: 'SysDepartment001.remark', sortable:false}

,{field:'operation', title: 'SysCommon.operation', type:'action', buttons: ['edit', 'delt']}

]

});

****

### 弹出层

#### dialog对话框

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 数据类型 | 默认值 | 说明 |
| title | String/Array/Boolean |  | 对话框标题 |
| url | String |  | 对话框对应的URL地址 |
| width | String/Number | 50% | 对话框宽度 |
| height | String/Number | 90% | 对话框高度 |
| paramObj | JSON Object | {} | 对话框参数 |
| callback | Function |  | 对话框关闭时的回调函数 |
| btns | JSON Object | 对话框底部按钮 |  |

**title** - 标题

title支持三种类型的值，若你传入的是普通的字符串，如title :’我是标题’，那么只会改变标题文本；若你还需要自定义标题区域样式，那么你可以title: [‘文本’, ‘font-size:18px’], 数组第二项可以写任意css样式； 如果你不想显示标题栏，你可以title: false。

**url** - 地址

指向一个画面URL地址，地址可以采用JoyinUtil.getRealPath(url)写法，也可以直接写路径，在dialog内部会判断是否需要补充完整的绝对路径。

**width** - 宽度

弹出画面的宽度值，可以使用百分比或者是像素值，默认为window宽度的50%。

**height** - 高度

弹出画面的宽度值，可以使用百分比或者是像素值，默认为window高度的90%。

**paramObj** - 参数

该参数会添加到url地址的后面，以GET进行传递，所有该参数有限制，不能超过浏览器GET请求大小限制，否则会出错。

**callback** - 回调函数

弹出层画面关闭后就会执行该方法。该方法接收一个参数，弹出画面的返回值（子画面中设置的top.returnData）。

**btns** - 按钮

通过该参数指定的按钮，将固定显示在弹出画面底部，不随滚动条滚动。

btns中的key将会作为显示的按钮名称，value指定的是按钮点击的响应事件。

示例代码：

var height = '400px';

var btns = {};

var dialogIndex = -1;

btns[JoyinUtil.getLabel('userselect.btn.submit')] = function () {

JoyinUtil.executeDialogMethod(dialogIndex, 'setUser');

layer.close(dialogIndex);

};

btns[JoyinUtil.getLabel('userselect.btn.cancel')] = function () {

layer.close(dialogIndex);

};

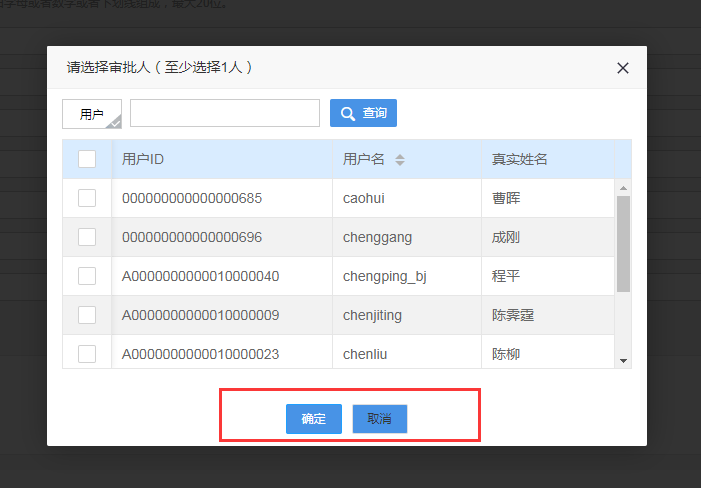
dialogIndex = Joyin.dialog(JoyinUtil.getLabel('userselect.title.user'), path, '600px', height, param, function (params) {

if (params && \_.isArray(params)) {

callbackFunc.call(callbackFunc, params[0], params[1], params[2], params[3]);

}

}, btns);



#### confirm确认框

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 说明 |
| message | String | 消息内容 |
| yesMethod | Function | 确认事件 |
| noMethod | Function | 取消事件 |

**message** - 消息内容

字符类型的消息内容，不允许为空。

**yesMethod** - 确认事件

确定按钮点击时的回调函数，不带参数。

**noMethod** - 取消事件

取消按钮点击时的回调函数，不带参数。

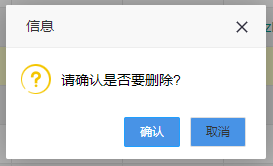
Joyin.confirm(JoyinUtil.getMessage("COMMON\_C\_0001",JoyinUtil.getLabel("SysCommon.delete")), function(){

console.log('确认');

}, function() {

console.log('取消')；

});



#### confirmMulti确认框

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 说明 |
| message | String | 消息内容 |
| btnArr | Array | 按钮组 |
| method1 | Function | 按钮1事件 |
| Method2 | Function | 按钮2事件 |
| Method3 | Function | 按钮3事件 |

**message** - 消息内容

字符类型的消息内容，不允许为空。

**btnArr** - 按钮组

接收数组类型的按钮，如[‘按钮1’,’按钮2’,’按钮3’]

**method1** - 事件

btnArr的第一个元素点击事件。

**method2** - 事件

btnArr的第二个元素点击事件。

**method3** - 事件

btnArr的第三个元素点击事件。

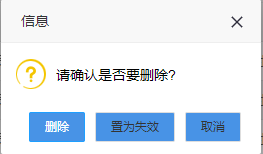
Joyin.confirmMulti(JoyinUtil.getMessage('COMMON\_C\_0001', JoyinUtil.getLabel('SysCommon.delete')), ['删除', operateType, '取消'], function () {

deleteBatchTask($('#batchSeqNo').val(), data.taskId, 1);

}, function () {

deleteBatchTask($('#batchSeqNo').val(), data.taskId, deleteType);

})；



#### prompt提示框

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 说明 |
| title | String | 标题 |
| formType | Integer | 输入框类型 |
| callback | Function | 确认回调 |
| initValue | String | 初始值 |
| maxlength | Integer | 最大长度 |

**title** - 标题

提示框标题，默认为“提示”。

**formType** - 输入框类型

输入框类型，支持0（文本）默认1（密码）2（多行文本）

**callback** - 确认回调

点击确定时的回调函数。该函数会接收输入框的值作为参数。

**initValue** - 初始值

输入框初始值。

**maxlength** - 最大长度

输入框允许输入的最大长度，默认为500。

Joyin.prompt("请为此次执行命名，以便在监控中查看", 0, function (result) {

...

});



#### alert消息框

|  |  |  |
| --- | --- | --- |
| 参数名 | 数据类型 | 说明 |
| message | String | 消息内容 |
| iconType | Integer | 图标类型 |
| callback | Function | 关闭事件 |
| result | Object | callback参数 |

**message** - 消息内容

字符类型的消息内容，不允许为空。

**iconType** - 图标类型

可以传入的值0-6，默认为1。

**callback** - 关闭事件

消息框关闭时执行的函数，参数有result指定。

**result** - callback参数

只有在设置callback参数后，该参数才会有效。

#### error错误提示

alert控件，在iconType为2时的特例。

#### warn警告

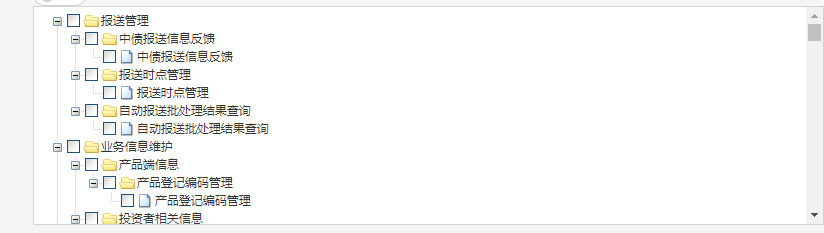
alert控件，在iconType为0时的特例。

### 树控件

#### 简单树

|  |  |
| --- | --- |
| 参数名 | 说明 |
| id | 树的dom元素属性id |
| canEdit | 定义的可编辑data-canedit属性 |
| chkStyle | 指定选择样式data-chkstyle属性，默认为checkbox |
| url | 定义的url属性，用来存放请求地址 |
| treeData | 定义的data-result属性 |
| addTreeNode | 增加节点方法 |
| expandTreeNode | 展开节点方法 |
| removeTreeNode | 删除节点方法 |
| editTreeNode | 编辑节点方法 |

实例代码:

****

<div class="layui-input" id="tree" url="/system/actorrole/SystemActorRole002/getAllTreeNode"></div>

$('#tree').joyinTree({});

#### 复杂树

1. html属性

data-canedit 是否可以编辑，可选值1

data-checkable 是否有checkbox框，默认为true

data-chkstyle 指定选择样式，默认为checkbox

data-checkall 是否有全选框，默认为true

url 数据来源URL地址

data-result 静态数据

1. 初始化方法

joyinMixTree(option);

参数介绍

settings 允许设置ztree的任何属性

onTreeAddSubFunction(parentNodeId,parentNodeName) 增加子节点方法

onTreeAddFunction 增加节点方法

onTreeEditFunction 编辑方法

onTreeLoadSuccess 树加载完成后的回调方法

editUrl 编辑url

delUrl 删除url

addUrl 增加url

col 快捷菜单定义

1. 具体用法

在参数中的col为控件的菜单展示列，其中支持的clas只有以上集中。

首先根据col中的配置可以划分为isdefine = true/false 两种，首先介绍为false以及缺省状态，此状态会执行默认方法以及发起对应功能的url（例如：edit->editUrl），但是不会执行对应的方法。

当isdefine=true的是候例如：{text:'编辑账套',clas:'edit',isdefine:true} ,会直接执行对应的onTreeEditFunction方法，但是editUrl则不会生效。

<div id='tree' style="border: 1px solid #fff; scroll-y: auto; scroll-x: hidden; height: 400px;" data-canedit='1' data-roottitle='账套组' data-th-url='|@{/system/actorrole/SystemActorRole002/getAllTreeNode}'></div>

var QueryFunc=function(){

return {

serachKey:function(value){

alert("执行你需要的方法,参数为"+value);

}

}

}();

$(function() {

$('#tree').joyinMixTree({

onTreeAddSubFunction: function(pcode,pname,callback){//isdefine:true

alert("增加子方法进来啦")

callback();

},

onTreeAddFunction: function(callback){

alert("增加方法进来啦")

callback();

},

onTreeEditFunction: function(callback){

alert("编辑方法进来啦")

callback();

},

onTreeLoadSuccess: function () {

if(pageModel == PAGE\_MODEL\_REFERENCE) {

var treeObj = $.fn.zTree.getZTreeObj("tree\_tree");

var nodes = treeObj.getNodes();

for(var i=0; i < nodes.length;i++) {

var node = nodes[i];

DisabledNode(treeObj, node);

}

}

},

col:[{text:'增加账套',clas:'add'},

{text:'编辑账套',clas:'edit'/\*,isdefine:true\*/},

{text:'删除账套',clas:'del'},

{text:'增加子账套',clas:'addsub'/\*,isdefine:true\*/}

]

});

function expandAll() {

$.fn.zTree.getZTreeObj('tree\_tree').expandAll(!$('#expandAll').is(':checked'));

};

function DisabledNode(treeObj, node) {

treeObj.setChkDisabled(node, true);

if(node.isParent) {

var children = node.children;

for(var i=0; i < children.length; i++) {

DisabledNode(treeObj, children[i]);

}

}

}

/\*\*

\* 画面加载完成

\*/

$(function() {

$('div.condition-area').find('a:first').attr('id','SysDepartment001\_btn\_search');

});

}());

树控件重载reloadTree

参数：params 重新加载数据使用的参数（画面查询条件）

调用方法如下

$('#tree').reloadTree(params)

### 文件上传

使用data-option属性指定，该属性是一个JSON字符串。

|  |  |
| --- | --- |
| 参数名 | 说明 |
| url | 文件上传的路径(默认值为/framework/fileUpload/uploadFile) |
| allowFileType | 允许上传的文件的类型，竖线分隔。（可选，默认值 doc/docx/xls/xlsx/ppt） |
| inline | 是否行内上传，指定附件控件的显示样式 |
| showDelete | 是否显示删除按钮(inline时生效，默认false） |
| autoUpload | 是否自动上传，默认为true |
| muiltiple | 是否支持多文件上传，默认为false |
| type | 上传方式，只支持list和drag，默认为drag（拖拽）。 |
| width | 宽度 |
| height | 高度 |
| bindAction | 上传动作触发者 |
| callback | 上传文件后的回调函数，多个文件会多次执行 |

**属性**

data-data：控件初始化显示值，com.joyintech.framework.base.dto.UploadFile列表的JSON对象。

disabled: 是否禁用,boolean类型的值

**方法**

获取文件上传控件当前已上传文件Array

$(‘#myUpload’).data(‘getSelectedFiles’)();

代码示例:

<div class="upload" th:disabled="${pageModel} eq ${pageModelReference}" th:data-data='${attachmentFiles}' data-option="{allowFileType: 'xls|xlsx', muiltiple: true, autoUpload: false, type: 'drag', callback: 'FileCommon.onUploaded'}">

</div>



### 文件下载

下载有两种，一种是模板下载，模拟表单提交，导出excel，另一种是普通下载

1. 模板下载，模拟表单提交，导出excel，调用共通方法里的Joyin.exportExcel（）方法

|  |  |
| --- | --- |
| 参数名 | 说明 |
| url | 提交的请求url |
| paramObj | 提交请求的参数对象 |
| callback | 导出成功后的回调函数 |
| millisecond | 回调函数延迟执行时间（默认为1000毫秒），设置为0时，回调函数不生 |

代码实例：

<button type="button" class="layui-btn" id="tplDowload">模板下载</button>

$('#tplDowload').click(function() {

Joyin.exportExcel('downloadTemplate', {

pageId: parent.pageId,

exts: exts

});

});

1. 普通下载，调用封装好的download（）方法

|  |  |
| --- | --- |
| 参数名 | 说明 |
| data-nameArray | 需要被下载的文件数组 |

代码实例：

<div data-nameArray="[{url:'www.baidu.com',name:'文件名1'},{url:'www.baidu.com',name:'文件名2'}，{url:'www.baidu.com',name:'文件名3'}]" class="download"></div>

$('.download').download();



### 进度条

嵌入式进度条，使用样式input-progressBar进行初始化

<div class="input-progressBar" data-key="progressBar"></div>

支持属性如下：

data-key: 进度条唯一标识，必须

data-width: 进度条宽度，默认为100%，支持百分比和像素

data-start: 进度条初始化后是否默认启动，默认值为true，可选值false

data-hide: 是否隐藏，默认为false，可选值true

进度条共通方法，使用JoyinProgress对象进行调用

方法一览：

showInvented: 开启虚拟进度条，支持参数为autoclose（是否自动关闭，默认为true，当进度值达到100%时自动关闭）。该虚拟进度条的位置固定显示在页面最底部，进度值通过随机数产生。

showCommon: 展示公共进度条，支持参数为autoclose（是否自动关闭，默认为true，当进度值达到100%时自动关闭）。该虚拟进度条的位置固定显示在页面最底部，进度值通过后台计算，和虚拟进度条的唯一区别在于进度值为后台计算。

hideCommon: 关闭共通进度条。

showInDialog: 在弹框中显示进度条，支持的参数为barId（为进度条指定ID，必须和后台计算进度时的进度条ID对应，默认值为dialogProgressBar）。该进度条在进度达到100%时会自动关闭弹窗。

### 单据打印

模板实例：

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<style>

.sys-table-wrap tbody tr td{

padding: 4px 15px;

}

</style>

</head>

<body>

<script id="prodBuild" type="text/html">

<h3 style="text-align:center;font-size:18px;">关于{{ d.prodSer}}人民币理财产品（{{ d.prodName}}）

</h3>

<h3 style="text-align:center;font-size:18px;">成立通知及划款指令</h3>

<div style="text-align:right">期数 :{{ d.noPeriods}}

</div >

<div style="text-align:center;font-size:13px;">{{ d.prodSer}}人民币理财产品（{{ d.prodName}}）于{{ d.prodBuildDay}}起息，起息要素见下表。</div>

<table class='layui-table sys-table-wrap' style="align:center;background:#b8ecc663;width: 90%;margin:0 auto;font-size:13px;">

<tbody>

<tr><td style="width:25%" >产品系列</td><td style="width:75%">{{ d.prodSer}}</td></tr>

<tr><td style="width:25%">客户类型</td><td style="width:75%"><span style='display: inline-block;width:32%;'>{{ d.customType}}</span>

<span style='border-left:2px solid #ddd; display: inline-block;width:31%;padding-left: 2%;'>产品类型</span><span style='border-left:2px solid #ddd;display: inline-block;width:31%;padding-left: 2%;'>{{ d.prodType}}</tr>

<tr><td style="width:25%" >产品代码</td><td style="width:75%">{{ d.prodId}}</td></tr>

<tr><td style="width:25%" >产品名称及期数</td><td style="width:75%">平安理财-{{ d.prodName}}（{{ d.noPeriods}}期）</td></tr>

<tr><td style="width:25%" >本金（大写）</td><td style="width:75%">{{ d.upCasePrincipal}}</td></tr>

<tr><td style="width:25%" >本金（小写）</td><td style="width:75%">{{ d.lowCasePrincipal}}</td></tr>

<tr><td style="width:25%" >上期滚入金额</td><td style="width:75%">{{ d.rollOnAmount}}</td></tr>

<tr><td style="width:25%" >本期新认购金额</td><td style="width:75%">{{ d.subscriptAmount}}</td></tr>

<tr><td style="width:25%" >分层金额</td><td style="width:75%">{{ d.layerAmount}}</td></tr>

<tr><td style="width:25%" >产品起息日</td><td style="width:75%">{{ d.prodVdate}}</td></tr>

<tr><td style="width:25%" >产品到期日</td><td style="width:75%">{{ d.prodMdate}}</td></tr>

<tr><td style="width:25%" >期限</td><td style="width:75%">{{ d.term}}</td></tr>

<tr><td style="width:25%" >销售分行</td><td style="width:75%">{{ d.salesBranch}}</td></tr>

<tr><td style="width:25%" >产品收益率</td><td style="width:75%">{{ d.prodYield}}</td></tr>

<tr><td style="width:25%" >分层收益率</td><td style="width:75%">{{ d.layerYield}}</td></tr>

<tr><td style="width:25%">销售费率</td><td style="width:75%"><span style='display: inline-block;width:32%;'>{{ d.salesFee}}</span>

<span style='border-left:2px solid #ddd; display: inline-block;width:31%;padding-left: 2%;'>托管费率</span>

<span style='border-left:2px solid #ddd;display: inline-block;width:30%;padding-left: 2%;'>{{ d.HostFee}}</tr>

<tr><td style="width:25%">资产管理费</td><td style="width:75%"><span style='display: inline-block;width:32%;'>{{ d.assetManagementFee}}</span>

<span style='border-left:2px solid #ddd; display: inline-block;width:31%;padding-left: 2%;'>其他费率</span>

<span style='border-left:2px solid #ddd;display: inline-block;width:30%;padding-left: 2%;'>{{ d.otherFee}}</tr>

</tbody>

</table>

<div style="text-align:center;margin-top:20px;">

<span style='display: inline-block;width:31%;'>经办：{{d.runName}}</span>

<span style='display: inline-block; width:31%;'>复核：{{d.reviewName}}</span>

<span style='display: inline-block;width:31%;'>授权：{{d.authorization}}</span>

</div>

<div style="text-align:right;margin-top:40px;margin-right:5%">

<span>资产管理事业部</span><br/>

<span>{{d.prodBuildDay}}</span>

</div>

<div style="text-align:left;margin-top:20px;margin-left:7%">

<span>经办：{{d.runAndDate}}</span><br/>

<span>复核：{{d.reviewAndDate}}</span>

</div>

<div style="text-align:right;margin-top:40px;margin-right:5%">

<span>运营管理部</span><br/>

</div>

</script>

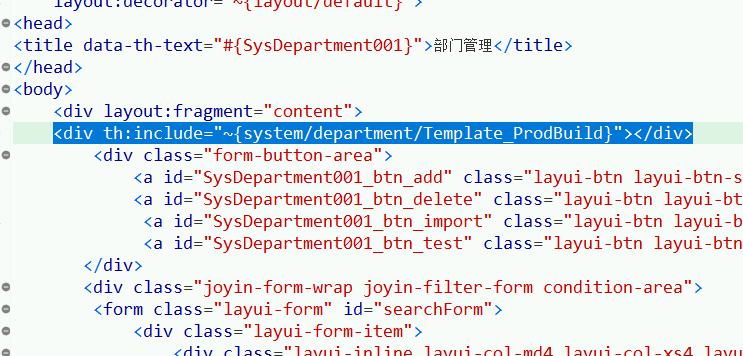
</body>

</html>

模板显示效果：



模板的引用和使用方式：



直接将模板的路径放入即可引入模板

var data = {

"prodSer":"和盈滚滚添利30天",

"prodName":"和盈资产管理类滚滚添利30天",

"noPeriods":"0030",

prodBuildDay:"2018年05月08日",

customType:"公司客户",

prodType:"和盈开放式N天",

prodId:"DGN170030",

upCasePrincipal:"伍佰贰拾万圆整",

lowCasePrincipal:"5,200,000.00",

rollOnAmount:"0.00",

subscriptAmount:"5,200,000.00",

layerAmount:"5,200,000.00",

prodVdate:"2018年05月08日",

prodMdate:"2018年06月07日",

term:"30天",

salesBranch:"集合",

prodYield:"4.5%",

layerYield:"4.5%",

salesFee:"0.45%",

HostFee:"0.03%",

assetManagementFee:"",

otherFee:"0",

runName:"",

reviewName:"",

authorization:"",

runAndDate:"XXX 2018年05月08日",

reviewAndDate:"XXX 2018年05月08日"

};

模板的数据源示例。这个需要配合模板使用模板；

关于生成打印画面需要使用以下的语法。

Joyin.printTemplate({templateHtml:prodBuild.innerHTML,data:data});

### 选项卡

Tab选项卡提供多套风格，支持响应式，支持删除选项卡等功能。面包屑结构简单，支持自定义分隔符。

#### 默认Tab选项卡

Tab广泛应用于Web页面，因此我们也对其进行了良好的支持。Layui内置多种Tab风格，支持删除选项卡、并提供响应式支持。

这是一个最基本的例子：

实例代码：

<div class="layui-tab">

<ul class="layui-tab-title">

<li class="layui-this">网站设置</li>

<li>用户管理</li>

<li>权限分配</li>

<li>商品管理</li>

<li>订单管理</li>

</ul>

<div class="layui-tab-content">

<div class="layui-tab-item layui-show">内容1</div>

<div class="layui-tab-item">内容2</div>

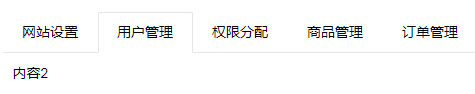
<div class="layui-tab-item">内容3</div>

<div class="layui-tab-item">内容4</div>

<div class="layui-tab-item">内容5</div>

</div>

</div>



#### 带图标的TAB

改实例代码为了简单的展示效果，采用的是layui-icon，这些图标完全可以有iconfont代替。 添加图标只需要添加<i class="图标样式"></i>即可。

实例代码：

<div class="layui-tab">

<ul class="layui-tab-title">

<li class="layui-this"><i class="layui-icon layui-icon-form"></i> 网站设置</li>

<li><i class="layui-icon layui-icon-user"></i> 用户管理</li>

<li><i class="layui-icon layui-icon-engine"></i> 权限分配</li>

<li><i class="layui-icon layui-icon-cart"></i> 商品管理</li>

<li><i class="layui-icon layui-icon-rmb"></i> 订单管理</li>

</ul>

<div class="layui-tab-content">

<div class="layui-tab-item layui-show">内容1</div>

<div class="layui-tab-item">内容2</div>

<div class="layui-tab-item">内容3</div>

<div class="layui-tab-item">内容4</div>

<div class="layui-tab-item">内容5</div>

</div>

</div>



#### Tab简洁风格

通过追加class：layui-tab-brief 来设定简洁风格。

实例代码：

<div class="layui-tab layui-tab-brief" lay-filter="docDemoTabBrief">

<ul class="layui-tab-title">

<li class="layui-this">网站设置</li>

<li>用户管理</li>

<li>权限分配</li>

<li>商品管理</li>

<li>订单管理</li>

</ul>

<div class="layui-tab-content"></div>

</div>



#### Tab卡片风格

通过追加class：layui-tab-card来设定卡片风格

实例代码：

<div class="layui-tab layui-tab-card">

<ul class="layui-tab-title">

<li class="layui-this">网站设置</li>

<li>用户管理</li>

<li>权限分配</li>

<li>商品管理</li>

<li>订单管理</li>

</ul>

<div class="layui-tab-content" style="height: 100px;">

<div class="layui-tab-item layui-show">1</div>

<div class="layui-tab-item">2</div>

<div class="layui-tab-item">3</div>

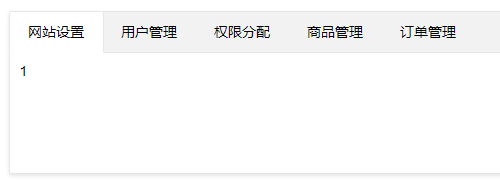
<div class="layui-tab-item">4</div>

<div class="layui-tab-item">5</div>

<div class="layui-tab-item">6</div>

</div>

</div>



#### 带删除的Tab

你可以对父层容器设置属性 lay-allowClose=”true” 来允许Tab选项卡被删除

实例代码：

<div class="layui-tab" lay-allowClose="true">

<ul class="layui-tab-title">

<li class="layui-this">网站设置</li>

<li>用户基本管理</li>

<li>权限分配</li>

<li>全部历史商品管理文字长一点试试</li>

<li>订单管理</li>

</ul>

<div class="layui-tab-content">

<div class="layui-tab-item layui-show">1</div>

<div class="layui-tab-item">2</div>

<div class="layui-tab-item">3</div>

<div class="layui-tab-item">4</div>

<div class="layui-tab-item">5</div>

<div class="layui-tab-item">6</div>

</div>

</div>



### Step控件

代码中需要指定步骤

通过伪类page-step初始化，其中的步骤标签需要指定step伪类，该控件会自动为每个步骤创建“上一步”和“下一步”以及“取消”按钮。

<div class="pro-module-wrapper pro-module-base step" title="基本信息" acitve="true">

...

</div>

<div class="pro-module-wrapper pro-module-base step" title="中债报送信息">

...

<div class="row text-center stepbar">

<a class="layui-btn layui-btn-sm btn-save" >保存</a>

<a class="layui-btn layui-btn-sm btn-submit">提交</a>

<a class="layui-btn layui-btn-sm btn-cancel">取消</a>

</div>

</div>

支持属性如下：

title: 步骤名称，必须

acitve: 非必须，默认激活第一个步骤

stepbar：必须，写在最后一个步骤中，为了到达最后一步的时候才显示其中定义的各按钮（如果需要单独控制是否显示，请写到步骤外层）。 pageStep控件会在自动在最后一个stepbar最前面添加一个“上一步”按钮。





## 后台

### Controller

基本约定

controller一般继承基础Controller类（BaseController）

类级注解

@PageId(“SysDepartment001”)

@Controller

@RequestMapping(“/system/department/SysDepartment001”)

代码如下：此代码中的画面访问地址为/system/department/SysDepartment001/init

@Controller

@PageId("SysDepartment001")

@RequestMapping("/system/department/SysDepartment001")

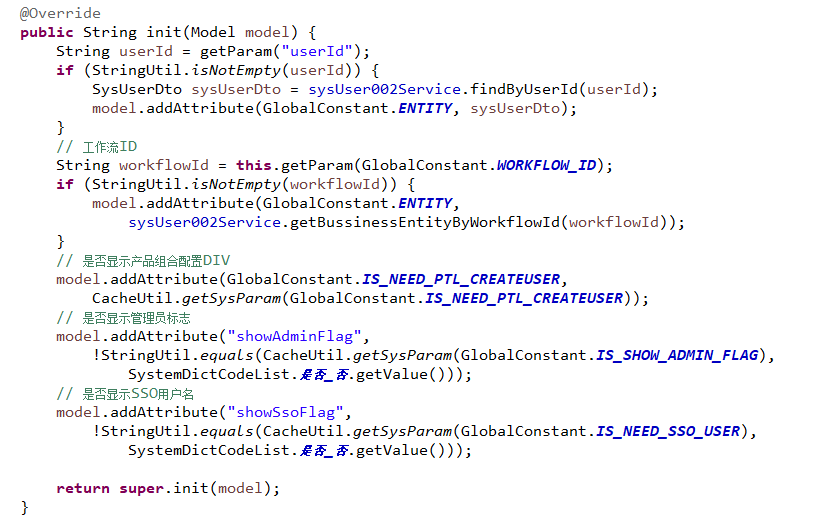
public class DepartmentController001 extends BaseController {

}

#### 显示画面

显示画面只需要继承BaseController，然后访问init地址即可。

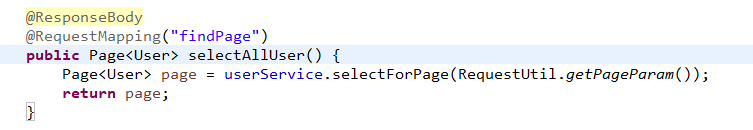
在BaseController中提供了一个默认方法init，该方法会根据@PageId注解加载相应的模板，并展示到画面上。此方法需要初始化SYS\_PAGE数据。如果需要给画面的添加额外属性时，需要重写init方法。



#### 返回表格数据

数据表格要求后台返回一个Page或List类型的JSON数据。

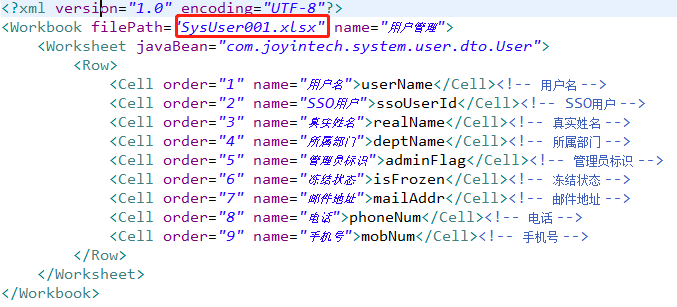
如果表格开启前台分页，则后台需返回List。

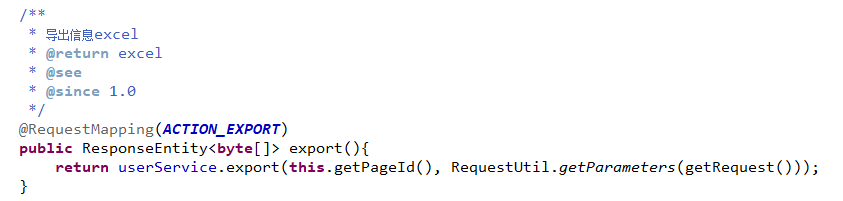


#### 导出功能实现

导出要求方法的返回值为ResponseEntity<byte[]>。

同时还需要export\_config.xml文件和一个导出模板（此部分在service中用到）。

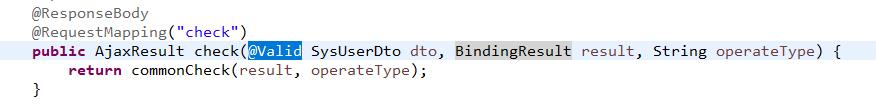




#### 业务逻辑验证

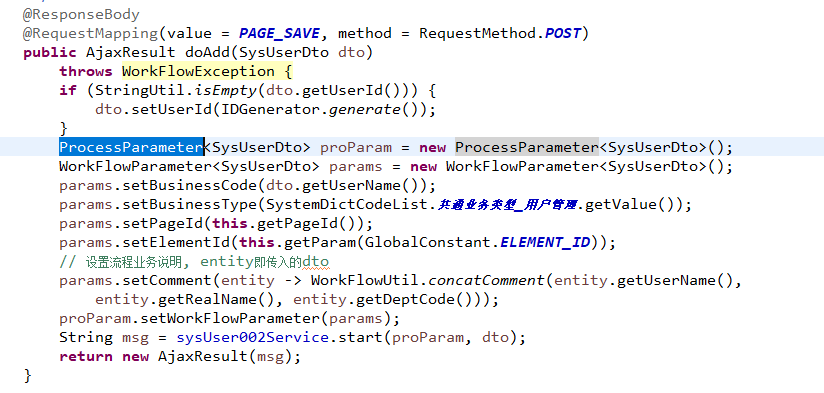
验证结果需返回一个AjaxResult的JSON对象，该对象中包含了验证消息，前台页面将会自动给出错误信息提示。

注意验证方法的Dto对象参数需要添加@Valid注解，并紧接其后添加一个BindingResult参数。



#### 工作流发起

构造一个ProcessParameter对象，调用service的start方法，返回一个AjaxResult的JSON对象

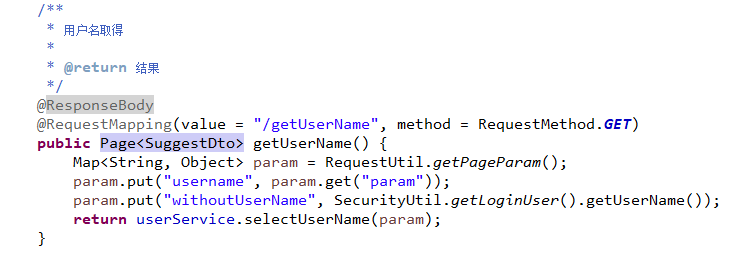


#### 联想文本框与下拉框实现

联想框和下拉框后台都有对应的数据结构，分别为SuggestDto和SelectOptionDto。后台必须回对应数据结构的集合。后台需要做的事情是根据前台控件的请求参数，返回对应的结果即可。

注1：后台必须返回JSON数据。

注2：如果是data-url形式的联想框或下拉框，后台应该返回Page而非List。



### Service

业务逻辑层主要就是实现功能块所要要求的具体业务的实现。所以就需要我们的servcie具有一些具体的功能或者特征来满足我们的界面层的功能要求。

框架提供了一个带有导出功能的父类CommonExportService和BaseService。CommonExportService继承了BaseService。BaseService中提供了多种分页查询的实现，能满足绝大部分查询需求，同还提供了国际化Message的获取方法。

Service必须添加@Service注解。

#### 查询画面数据实现(实现类的编写)

需要service继承BaseService或者CommonExportService

然后就可以调用父类的queryForPage方法来实现加载页面表格的作用。

queryForPage有多种参数的方法，现在对于这个方法的使用可以使用以下的方式：

例：

//这个方法为比较常用的方法中的一个，参数为λ表达式（格式为()->{查询方法}）一个匿名方法，

/\*\*

\* 使用给定的sqlID检索分页数据。返回类型是Service的泛型。

\*

\* @param select 查询方法

\* @return Page类

\* @throws Exception 异常信息

\*/

public Page<User> selectForPage(Map<String, Object> pageParam)

throws Exception {

return super.queryForPage(() -> userMapper.select(pageParam));

}

/\*\*

\* 使用给定的sqlID检索分页数据。返回类型是Service的泛型。

\*

\* @param mapperClass 相关dao类的class

\* @param sqlId 数据库检索sqlId

\* @param paramMap 检索参数

\* @return Page类

\* @throws Exception 异常信息

\*/

queryForPage(Class<?> mapperClass, String sqlId,

Map<String, Object> paramMap);

/\*\*

\* 联想文本框service层实现

\*

\* @param param 检索参数

\* @return SuggestDto

\*/

public List<SuggestDto> getInstitutionCode(String param) {

return queryForSuggest(institutionMapper::getInstitutionName, param, null);

}

/\*\*

\* data-url 方式的下拉框

\*

\* @param param 检索参数

\* @return SuggestDto

\*/

public List<SelectOptionDto> getAddressSelectOption(Map<String,Object> param) {

return institutionMapper.getAddressSelectOption(param);

}

/\*\*

\*

\* 工作流（留痕操作结束的时候执行的方法）

\*/

@Override

public void doComplete(SysInstitutionDto entity, OperationType type) {

//首先根据操作类型来区分在此操作中需要进行处理的操作

//处理业务entity,转化为我们最终需要的entity

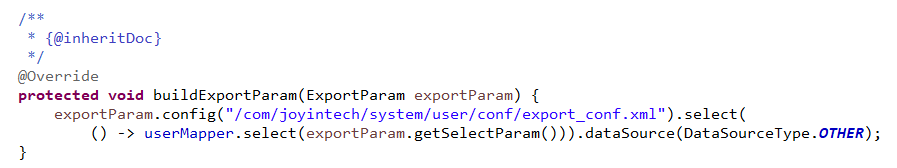
//把业务entity持久化到我们的系统

}

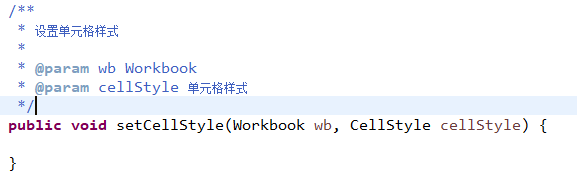
#### 导出逻辑实现

要实现导出功能需继承CommonExportService，然后buildExportParam方法即可。

该方法中将设置导出配置文件、数据查询接口、数据源（默认使用主数据源）。



特殊导出需求可通过重写父类方法实现，如设置单元格样式：



### Dao

持久层使用的是mybatis，所以我们的代码将分为两部分，数据接口和xml文件。

mapper接口文件放置在dao目录下，其xml文件放置在dao包下的sqlmap文件中。

#### 接口文件

使用简介：需要添加类级注解@Repository，

在传入多个参数时需要使用@Param(“参数名”) 来做标注



#### xml文件

sqlId对应方法名称，parameterType：为参数值类型，resultType：返回值类型



### 工作流相关

1. 所有的参与流程的画面元素都可以添加workflow属性，该属性值将决定元素在工作流画面中是否显示。

workflow属性说明如下：

|  |  |
| --- | --- |
| 属性值 | 说明 |
| hide | 在工作流中该元素将会被隐藏 |
| remove | 在工作流中该元素将会被移除 |
| approve:hide | 在工作流中审批时隐藏该元素 |
| approve:remove | 在工作流中审批时移除该元素 |

例：



1. JS工作流方法说明

WorkFlow（工作流全局工具类 常用方法）

发起保存 doSave(url, formElement, elementId, pid, callback),

发起批量保存 doSaveList(url, formElement, elementId, pid, callback),

发起撤销 doRevoke(url, formElement, elementId, pid, callback),

发起撤销提交 doRevokeList(url, formElement, elementId, pid, callback),

发起删除doDelete(url, formElement, elementId, pid, callback),

发起批量删除 doDeleteList(url, formElement, elementId, pid, callback),

发起提交 doSubmit(url, formElement, elementId, pid, callback),

发起批量提交 doSubmitList(url, formElement, elementId, pid, callback),

弹框选择用户界面 selectUser(elementId, pid, param, callback)

参数名介绍：

url, formElement 表单控件, elementId 控件Id, pid 页面Id, callback 回调函数,param 参数

方法的额外说明，在实际的开发过程，上述的一些启动流程会额外的帮助我们维护一些工作流所必须的一些参数，而不用我们手动的去做这些。所以我们在需要发起工作流的时候请使用上述的流程发起函数来进行工作流的发起。

1. 后台发起及工作流基本要求
2. 首先需要系统满足以下条件：

* 系统中的流程模板已部署
* 流程模板已经配置了审批人

1. 模块需要走工作流需要自身满足以下几点：

* SYS\_PAGE中对应的模块的PAGE\_ID对应的IS\_NEED\_WORKFLOW字段为01
* SYS\_WFL\_BUSINESS\_MODULE中对应的画面的工作流的配置已经配置现在已经部署的流程
* 业务service实现BaseProcessService接口，并重写doComplete方法（传递参数为业务实体与操作类型）
* 业务entity 实现ProcessEntity接口,需要加类级注解@PageId(“VALUE”),其VALUE值为画面的PageId,需要加字段与方法级注解@PrimaryKey(value=Field.PK\_1)帮助共通获取字段值，其中Field.PK\_1代表业务关键字1，一共存在PK\_1->PK\_10（注解添加位置，要添加在业务主键，即能确保为唯一条可识别业务数据，根据提供的标注能在业务上确保为同一条记录）

1. 工作流发起：

* 需要在工作流的Controller层中发起，发起流程所需参数ProcessParameter、WorkFlowParameter，其中ProcessParameter为留痕参数,WorkFlowParameter为工作流参数（在工作流参数中已经设置的参数不需要在ProcessParameter中重新设置，ProcessParameter为基础的留痕数据所需的参数。）WorkFlowParameter工作流程参数中必须设置的参数为PageId（页面Id），ElementId(控件Id)，businessCode（业务代码编码/支持function模式），businessType（业务类型/支持function模式），comment（工作流备注/仅function模式）。仅为留痕流程的参数设置必须设置的字段为operationType（操作类型），pageId（画面Id）。

一次发起多个工作流程是允许的但是需要传递的参数为启动参数和数组。

1. 工作流的service方法需要实现以下方法来完成工作流的终结操作：



1. 带业务流程表模式的工作流实现

带业务流程表的工作流主要的功能是为了实现在业务管理界面可以查看和匹配一些业务流程表中存在的数据。在我们的以上的不含业务流程变中的普通的操作流程的基础上，需要再重写以下方法：



上述的方法是在工作流程在执行的过程中会在各个阶段执行的一部分方法的入口。

doInsert()方法会在工作流发起的时候执行，其参数会有业务实体，以及操作类型。

doUpdate()方法会在经办人的代办中修改本地保存的数据的时候执行，其参数为修改后的业务实体。

doDelete方法会在我们的系统点击作废按钮的时候执行，其参数为工作流Id。

doComplete为工作流终结方法，其参数为操作类型，与业务实体。

在实际的开发过程中主要的就是明确两点，第一点是否需要走流程，第二点是否需要业务流程表。确定完之后就是设计dto（dto需要满足的一点就是满足画面的初始化的信息，满足业务逻辑的所需要的参数，在实际的设置过程中对于比较复杂的页面，可以将dto设计为组合结构。即将所有的画面关联的模块分为多个模型然后在组合成一个模型，这个主模型就是我们实际上在进行业务处理及画面所需的业务实体），只要我们设计完成dto或者说是业务模型之后我们的关于工作流的开发应该是已经完成了一半，因为我们的工作流主要是根据dto（数据传输对象）展开的。

## 工具类

框架中为日常开发提供了一些常用的工具类来满足基本的开发需求。日常开发中尽量使用工具类，特别是字符串处理，日期转换，BigDecimal计算，JSON转换等。

1. PageUtil（画面工具类）

根据pageId获取page对象

根据pageId,elementId获取PageElement

1. ReflectUtils（反射工具类）

循环向上转型, 获取Field

将字符串值转为目标属性类型，赋值到目标实体内

将目标实体中的指定属性做为字符串返回

将目标实体中的指定属性转为JSON串返回

将目标实体中的指定属性强转为指定数据类型返回

1. ArithmeticUtil（BigDecimal操作类）

提供精确的加法运算，如果某个参数为空，则不作为加算数据

提供精确的减法运算，参数为空的作为0处理

提供精确的减法运算，参数为空的作为0处理

提供精确乘法运算方法

提供精确的除法运算方法

提供精确的除法运算方法

提供精确的除法运算方法

调整值精度

1. BeanUtil（Bean工具类）

把map转化为java bean

采用反射将map转为object

java bean 转 map

java bean 转 map

复制一个dto的字段内容到另外一个dto

获取对象的所有null的属性

CacheUtil(缓存处理工具类)

注入缓存管理器(CacheManager)

添加app缓存，将value缓存到指定的key中

从app缓存中获取key对应的缓存值，获取{@link CacheUtil#putAppCache(Object, Object)}方法设置的缓存

从app缓存中获取缓存值

从app缓存中获取系统参数缓存值

从app缓存中获取default对应的缓存值，并返回指定类型

从app缓存中获取key对应的缓存值，并返回指定类型

1. DateUtil(日期处理工具类)

取得当前日期（返回值：DATA,STRING 多种格式）

取得当前时间

获取当前的微秒数

获取当月第一天 yyyy-MM-dd

获取本月最后一天 yyyy-MM-dd

增加日期

计算两个日期之间月中的日相相差的天数，可以用来判断是否为每个月同一天

将日期字符串转为日期形式

根据格式化字符串格式化日期

根据两个日期，取得相隔的天数

比较两个日期的大小，精确到天数

得到两个日期的间隔时间，可以是年月日，时分秒

判断两个日期是否相等，如果都是null，则返回true。

主要功能:获取月份 注意事项:无

…

1. CookieUtil（Cookie工具类）

读取cookie

创建cookie，并将新cookie添加到“响应对象”response中。

修改cookie，可以根据某个cookie的name修改它（不只是name要与被修改cookie一致，path、domain必须也要与被修改cookie一致）

删除cookie

1. GenericsUtils(泛型工具类)

通过反射,获得指定类的父类的泛型参数的实际类型

通过反射,获得指定类的父类的第一个泛型参数的实际类型

通过反射,获得方法返回值泛型参数的实际类型

通过反射,获得方法返回值第一个泛型参数的实际类型

通过反射,获得方法输入参数第index个输入参数的所有泛型参数的实际类型

通过反射,获得方法输入参数第一个输入参数的所有泛型参数的实际类型

通过反射,获得Field泛型参数的实际类型

通过反射,获得Field泛型参数的实际类型

1. IDGenerator（ID生成工具）

获取一个32位长度的UUID

获取一个指定长度的UUID

1. JsonUtils（Json工具类）

对象转JSON

JSON转对象

将节点转换为数组节点

将字符串转换为一个json节点

将对象串格式化为JSON字符串

将 json 字段串转换为 对象

将 json 字段串转换为 List

1. RequestUtil(Request工具类)

将request查询参数封装至Map

获取请求方IP

获取完整的请求URL

获得请求路径（去除URL中的参数）

获取request

获取Web的真实路径

获取session

从request中获取请求参数

获取分页参数

获取请求的浏览器信息

是否为ajax请求

是否为ajaxJson请求

1. ResponseUtil（响应内容，主要负责构建画面路径）

spring环境中获取HttpServletResponse对象

1. SpringContextUtil（spring 上下文）

实现ApplicationContextAware接口的回调方法，设置上下文环境

获取bean

获取bean并转换对象

获取bean

获取message

1. **StringUtil（字符串工具类）**

判断两个字符串是否相等

删除html标签，防止js注入

取得一个非空的整数值

获取一个对象的字符串的值，如果该字符串为空则取默认值

数字转字符串

判断对象是否为空

自定义的分隔字符串函数

判断list是否为空

sou中是否存在finds 如果指定的finds字符串有一个在sou中找到

过滤特殊字符

将list转为指定分隔符分割的字符串

将list转为指定分隔符分割的字符串，同时提供前缀和后缀

文本替换

截取字符串 超出的字符用symbol代替

取得字符串的字节长度（考虑了汉字的情况）

1. SysCodeUtil（子系统代码获取工具类）
2. SysDataUtil（共通数据字典工具类）

从数据字典中获取参数值

获取数据字典明细列表

获取数据字典明细

1. ZipFileUtil（压缩文件处理工具类）

压缩zip格式文件

解压zip格式文件

## 事务规范

### spring事务使用规范

事务处理，只允许使用注解[@Transactional](https://github.com/Transactional)(rollbackFor = Exception.class)

1. 通常情况下不允许在Controller添加[@Transactional](https://github.com/Transactional)注解。
2. 如果需要添加事务控制，请在service中进行添加，特殊情况可添加到Controller中。
3. Service中事务使用规则  
   3.1 尽量不要添加类级别的事务注解，除非该类中的所有公共方法都需要事务控制

3.2 事务注解只能应该到public可见度的方法上。如果应用在protected、private或者package可见度的方法上，也不会报错，但事务设置不生效。

3.3 事务注解不能使用在static修饰的方法上。如果应用在静态方法上，事务设置将不生效。

3.4 为了让事务能够正确回滚，[@Transactional](https://github.com/Transactional" \t "_blank" \o "@Transactional)注解中必须指定rollbackFor = Exception.class。因为，默认情况下，spring会对unchecked异常进行回滚；如果是checked异常则不回滚。

3.5 方法通常不要对异常进行捕获（除非业务需要，即使捕获也需要重新抛出一个异常）。因为方法抛出异常才会引起事务回滚，如果被捕获了，可能会导致事务无法回滚，会造成数据不一致。

3.6 Service中调用同类中的方法，被调用的方法事务不生效。如果被调用的方法需要单独做事务处理，请将该方法提取到一个新的Service中。

3.7 工作流方法（doInsert、doUpdate、doDelete、doComplete）通常不需要添加事务控制，因为这些方法默认和工作流表在同一个事务中，工作流共通已开启事务。  
如果工作流方法中需要做特殊处理，请给[@Transactional](https://github.com/Transactional)添加propagation属性,不指定传播属性的事务和不添加事务是一个效果。

3.8 在循环中处理业务表数据的时候，需要根据业务逻辑，考虑是否要添加try catch方法让失败一条后继续处理下一条。  
try catch代码块中如果需要事务处理，请捕获异常后在throw一个异常。

3.9 [@Transactional](https://github.com/Transactional)注解不能胡乱添加，因为一旦添加该注解就会导致线程开销，影响性能。

3.10 通常情况下，事物只需要在数据库插入、更新、删除的方法上添加，查询方法除非需要利用事物去切换数据源，请添加只读事物，否则不要添加。

1. [@Transactional](https://github.com/Transactional)不建议添加在接口上
2. 启动事务会增加线程开销,通常情况下，仅是读取数据时，不必设置事务。
3. 避免大事务。  
   在进行复杂业务处理时，如果一个方法逻辑复杂，处理步骤多，应该拆分为多个小方法进行调用。  
   将数据库的查询和更新操作分开，事务只添加到负责更新DB的方法上，避免事务执行时间过长。
4. 不要自定义事务，手动控制事务是很容易忘记提交或回滚。

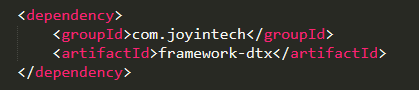
### 分布式事务使用规范

分布式事务拦截器，只拦截bean名称后缀为“\_dtx”的类中同时添加了@Transaction和@ DTXTransaction注解的方法。如果当前事务组已存在，则加入，不存在则创建。

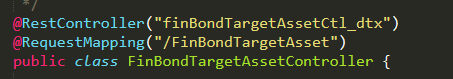
工作流的启动方法中默认添加了分布式事务，当业务代码中引入framework-dtx依赖，则分布式事务被开启，移除则不开启，业务代码中已添加的分布式事务代码无需修改。

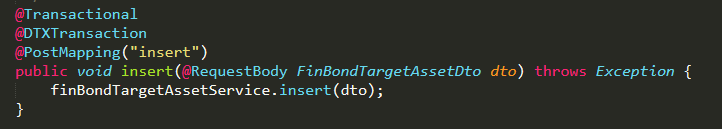
添加分布式事务支持示例：

pom.xml：



java代码：





application.yml



TXLCN分布式事务原理:

TX-LCN由两大模块组成, TxClient、TxManager。 TxClient作为模块的依赖框架，提供TX-LCN的标准支持，TxManager作为分布式事务的服务器，协调和控制client的事务。事务发起方或者参与方都由TxClient端来控制。 TXLCN本身并不会创建事务，只是控制本地事务的提交和回滚。

流程描述如下：

事务发起方请求TxManager创建事务组，TxManager通知事务发起方，告知事务组创建成功或者失败。

事务参与方执行业务代码，业务代码执行结束以后，将该模块的事务信息通知给TxManager。

发起方执行完业务代码以后，将发起方执行结果状态通知给TxManager,TxManager将根据事务最终状态和事务组的信息来通知相应的参与模块提交或回滚事务，并返回结果给事务发起方。

以上流程中，事务是在发起方业务代码执行完毕后，才会通知TxManager，然后有TxManager通知各事物参与方事物提交还是回滚。

添加分布式事务后将会有带来以下影响：

1. 事务的提交和回滚延迟到事务发起方的业务代码执行完。

2. 各服务的事务提交顺序不固定。

3. 系统性能会有所下降。

4. 所有业务代码必须在分布式事务超时时间内完成，否则将会被回滚。

在分布式系统中，事物发起方和参与方可能是不同的微服务，现假设存在如下情况：

各服务没有做数据库拆分，所有的服务都使用了同一个关系型数据库。

1. 服务A往表A里插入了一条数据，同时服务B也往表A里插入了同一条数据。

>> 此种操作，无论有没有分布式事务，都会报错。 没有添加分布式事务的情况下，报错后依旧可以从表A中查询到该条数据。

添加分布式事务后，一旦事物回滚，那么表A中将查询不到该条数据。

2. 服务A往表A里插入了一条数据，同时服务B从表A里查询该条数据做后续处理。

>> 在没有添加分布式事务的场景下，该业务流程没有问题，服务B中的业务代码正常运行。

添加分布式事务后，执行到服务B时，由于服务A的事务并没有提交，所以服务B中将查询不到该条数据，无法正常进行后续操作。

3. 服务A往表A里插入了一条数据，同时服务B服务去更新表A里中该条数据。

>> 在没有添加分布式事务的场景下，该业务流程没有问题，服务B中的业务代码正常运行。

添加分布式事务后，在事务组提交时，无法保证服务A的事务比服务B的事务先提交，导致服务B中的更新逻辑有时生效有时不生效。

4. 服务A更新表A里的一条数据，服务B也更新表A的同一条数据。

>> 在没有添加分布式事务的场景下，该业务流程没有问题，服务B中的业务代码正常运行。

添加分布式事务后，当服务B更新表A时，服务A对表A的行锁未释放，导致服务B将卡在获取行锁的逻辑，直到超时设置（数据库连接超时、分布式事务超时、网关超时、熔断器超时等）生效而回滚。 一旦超时设置不合理，就会导致WAS线程挂起。 当并发量大的时候，连接池、线程池等资源会耗尽，系统将无法对外提供任务服务，表现给用户的就是所有画面卡死。

5. 服务A更新表A里的一条数据，服务B更新表A的另一条数据。

>> 在没有添加分布式事务的场景下，该业务流程没有问题，服务B中的业务代码正常运行。

添加分布式事务后，如果系统并发量高，系统响应缓慢，一旦数据库超时设置比较小的时候，会表现出系统频繁报错，业务数据被回滚的比较多。

6. 事务参与方需要调用第三方系统的接口，等待第三方系统反馈，但该系统反馈时间无法预知。

>> 在没有添加分布式事务的场景下，在各种超时允许的范围内，该业务流程最多就是响应慢，但业务可以正常运作。

添加分布式事务后，如果第三方系统反馈延迟，导致整个调用链事务资源无法释放，可能会产生锁表、线程挂起、系统崩溃等问题。

经过对以上情况的分析，数据库未做合理拆分的情况下，必须要保证每个服务对数据的DML操作是原子性的。

每个表的增删改只能由某一服务进行操作。避免多服务DML同一张表。避免同一个事务组（调用链）操作时间过长。

分布式事务部署注意事项：

1. 如果各服务集群部署，不能在发起方和参与方之间使用反向代理，否则将会导致TxManager无法正确将事务通知给参与方和发起方，将会导致事务无法提交。

2. 需要先启动TxManager，再启动TxClient。 TxClient分布式事务的超时时间是从TxManager同步的。TxManager支持集群，连接到同一个redis的TxManager将会构成一个集群，给集群添加新的TxManager，TxClient配置可以不用修改，TxClient将会自动发现同一TxManager集群中的所有服务。

3. TxClient的IP和端口不能一致，也就是不能在同一个容器中通过区分contextPath部署多个TxClient。

4. TxManager支持Redis3.2及以上版本。