# Vue框架需求

## 原始构想

最初采用Vue的构想，是以当前框架为主要模式，但是做如下重要改变：

* 原先页面的输出方式是，在输出控件的时候把对应的值一起放入。而Vue是有数据绑定的，所以新的方式，输出控件的时候没有对应的值，只有绑定的字段信息。这样的好处是，无论显示什么样的数据，这个模板是固定不变的，这样可以方便后端使用缓存等技术，提高性能。
* 页面的数据通过JSON直接返回，在前端也是存储为JSON，通过绑定，用户的输入自动会更新相应的JSON数据，这样提交时，直接使用这个JSON数据即可。这样就不用现有复杂的通过DOM求值的过程，效率要高。而且如果是多条记录，如果处理得当，可以考虑追踪每条记录的状态，即提交的数据是新增，原始（修改），删除等状态，这样后台处理会更加方便，更有效率。
* 原先从列表页面导航到其他页面（如新建，修改等），采用页面跳转的方式（比较传统）。现在考虑使用侧推的方式，这样的好处是解决页面地址跳转后难以保留的问题，而且用户体验也相对不错。
* 除此之外，吸收前一版框架的有点，即定义一套自己规范的标签，在PC和手机端分别实现。甚至可以考虑框架性代码也如此，这样生成的界面如果一致，会减轻后端大量的劳动。

## 名词解释

### Normal布局

基于单表单记录的常规布局，我们称之为Normal布局。Normal布局中，一行根据需要，可以布置1-4个字段（刚好适配当前12格布局）。每个字段通常占据一个单位，但是也可以根据需要占有多个单位，甚至一行。

如下图，就是Normal布局：



### Table布局

基于多记录的布局方式。比如主从表中的子表。该布局，每条记录就是HTML Table中一个tr，而每个字段则是td。该布局要求可以动态增加，减少行记录，当然也可以锁定行记录，禁止增加和减少。这个布局比较适合多记录子表的字段比较少的编辑方式。

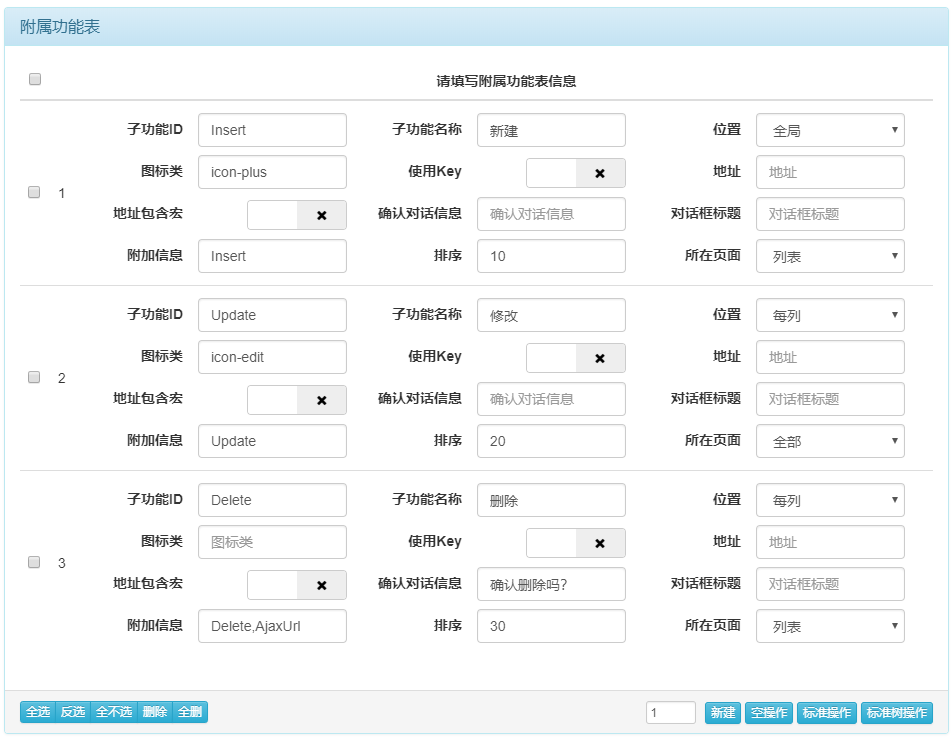
如下图，就是Table布局：



### NormalTable布局

NormalTable布局是Normal和Table的布局的混合体。主要是考虑部分子表的记录内容特别多，使用Table布局，势必造成横滚条拖得很长，界面效果差，因此，这种布局是将子表的每行变成Normal的布局方式，通过拉高每行的高度来解决字段多的问题。

如下图，就是NormalTable布局：



### 全部HTML

即一个完成的HTML文档，从<!DOCTYPE html>开始，包括完整的css和js。

### 局部HTML

即一个HTML片段，通常就是一段嵌入div的html，这段html通常不包括css和js

### 局刷

即js通过ajax从后台获取数据，然后动态刷新页面中的部分区域

### 平铺布局

多个表从上到下依次排列，这种方式称之为平铺

如下图，就是平铺布局：



### Tab布局

多个表采用Tab的方式，根据Tab页面的标签进行切换

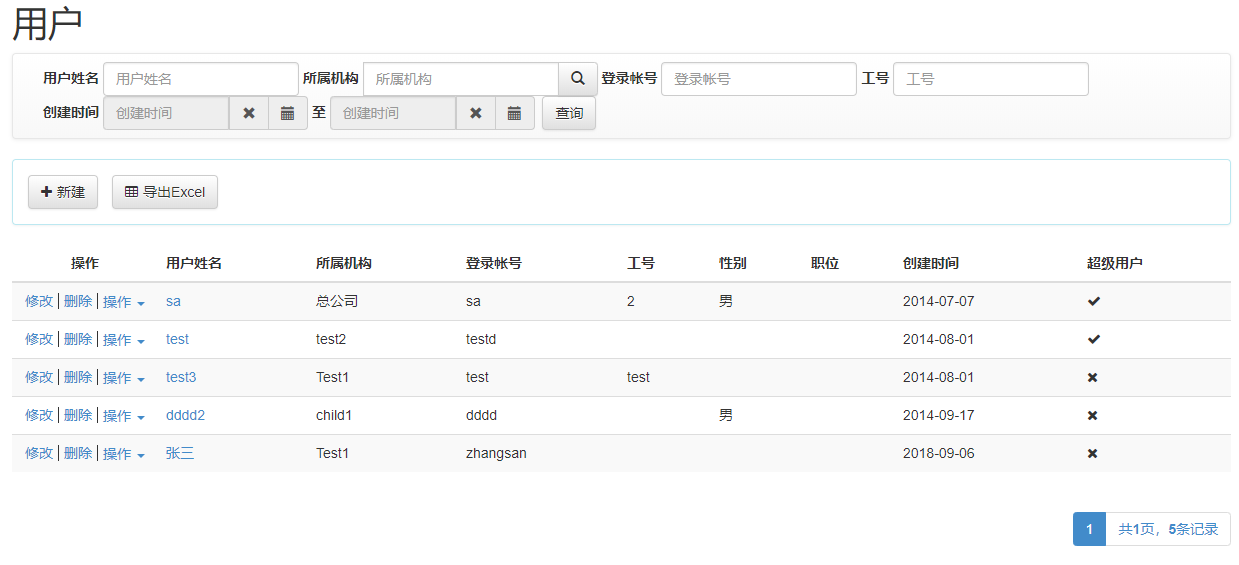
如下图，就是Tab布局：



### List页面

也称为列表页面，显示一个表的数据，可以提供查询，分页，排序等功能

如下图，就是List页面：



### Insert/Update页面

也称为编辑页面，处理表的新建和修改功能

如下图，就是Update页面：



### Detail页面

也称为详情页面，显示表的详细信息

如下图，就是Detail页面：



## 基本控件库

控件是构成表单的基本元素。比较丰富的控件可以带来良好的用户体验。这里基于ElementUI定义一组控件。

出于命名的统一性，所有的控件都以tk作为前缀。

### 普通文本控件(Input)

* 标签名：tk-input
* 标签总体：<tk-input></tk-input>
* 表现：
* 自定义属性：（必须的为必要属性）

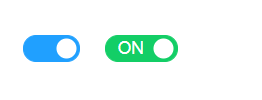
1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. placeholder：输入框提示信息，默认值为请输入...值(string)
5. classname：样式名
6. value：控件绑定值(string， number)
7. type：输入框的类型 默认值text
8. maxLength：最大输入长度(number)
9. minLength：最小输入长度(number)
10. disabled：禁用(boolean)
11. icon：输入框尾部图标(string)
12. readonly：原生属性，是否只读(boolean)
13. autoFocus：原生属性，自动获取焦点(boolean)
14. solt：input控件内添加其他元素，可配1.front添加到前面，2.end添加到后面（详情可参考element复合型输入框）
15. size：输入框尺寸(string)(large， small， mini)
16. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
17. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的。

### 多行文本控件(Textarea)

* 标签名：tk-textarea
* 标签总体：<tk-textarea></tk-textarea>
* 表现：
* 自定义属性：（必须的为必要属性）

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. placehoder：输入框提示信息，默认值为请输入...值(string)
5. classname：样式名
6. value：控件绑定值(string)
7. autoSize：自适应内容高度
8. rows：输入框行数有效(number)(2)
9. disabled：禁用(boolean)
10. maxLength：最大输入长度(number)
11. minLength：最小输入长度(number)
12. readonly：原生属性，是否只读(boolean)
13. resize： 控制是否能被用户缩放(string)(默认：none)(可选：none，both)
14. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
15. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的。

### 开关控件(Switch)

* 标签名：tk-switch
* 标签总体： <tk-switch></tk-switch>
* 表现：
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. classname：样式名
5. value：数据绑定值(string，number)
6. readonly：是否禁用(boolean) 默认值 false
7. width：switch的宽度(number) 默认值 58（有文字）/46无文字
8. onText：switch打开时的文字(string) 默认值 开
9. offText：switch关闭时的文字(string) 默认值 关
10. onValue：switch打开时的值(string) 默认值 1
11. offValue：switch关闭时的值(string) 默认值 0
12. onColor：switch打开时的背景色(string) 默认值 #20A0FF
13. offColor：switch关闭时的背景色(string) 默认值 #C0CCDA
14. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容

### 选择框（CheckBox）

* 标签名：tk-checkbox
* 标签总体： <tk-checkbox></tk-checkbox>
* 表现：C:\Users\ADMINI~1\AppData\Local\Temp\1537860102(1).png
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. classname：样式名
5. value：数据绑定值(string，number)
6. readonly：是否禁用(boolean) 默认值 false
7. onValue：checkbox选中时的值(string) 默认值 1
8. offValue：checkbox没选中时的值(string) 默认值 0
9. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容

### 单选框组(RadioGroup)

* 标签名：tk-radio-group
* 标签总体：<tk-radio-group></tk-radio-group>
* 表现：
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false
5. value：控件绑定值(string， number)
6. classname：样式名
7. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
8. url：如果配置，那么说明RadioItem需要动态获取，从该url能够获取到代码表的数据。数据格式为json，内容为{"$regName”: :[{"Value":"1","Name":"xxx"},{"Value":"2","Name":"xxx"}...]}
9. regName：代码表的注册名，如果url配置内容，返回结果需要用该属性识别

#### 单选框项(RadioItem)

* 标签名：tk-radio-item
* 标签总体：<tk-radio-item>text</tk-radio-item>
* 表现：
* 自定义属性：

1. value：控件值
2. text：控件显示值

#### 单选框按钮(RadioButtonItem)

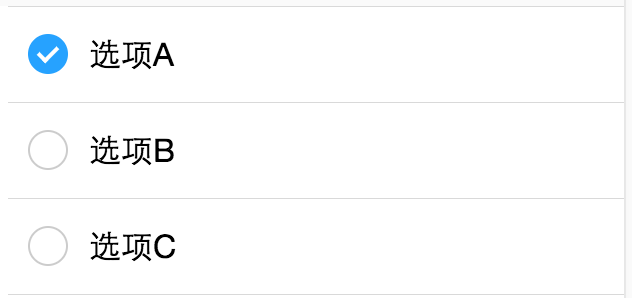
* 标签名：tk-radio-button-item
* 标签总体：<tk-radio-button-item>text</tk-radio-button-item>
* 表现：
* 自定义属性：

1. value：控件值

2. text：控件显示值

### 多选框组(CheckList)

* 标签名：tk-check-list
* 标签总体：<tk-check-list></tk-check-list>
* 表现(移动端)

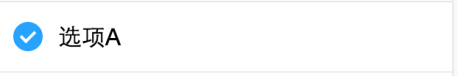


* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false
5. value：控件绑定值(string， number)，数据格式为Id添加双引号后，用逗号分隔。如："1","2"
6. classname：样式名
7. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
8. url：如果配置，那么说明CheckItem需要动态获取，从该url能够获取到代码表的数据。数据格式为json，内容为{"$regName”: :[{"Value":"1","Name":"xxx"},{"Value":"2","Name":"xxx"}...]}
9. regName：代码表的注册名，如果url配置内容，返回结果需要用该属性识别

#### 多选框项 (CheckItem)

* 标签名：tk-check-item
* 标签总体：<tk-check-item >选项</tk-check-item >
* 表现(移动端)：





* 自定义属性：

1. value（必须）： 控件值
2. text：控件显示值
3. align： 复选框对其位置(left/right)默认值left–移动端(PC端不用做)

### 下拉框(Select)

* 标签名：tk-select
* 标签总体：<tk-select></tk-select>
* 表现：
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false
5. value：控件绑定值(string， number)
6. placeholder：输入框提示信息，默认值为请输入...值(string)
7. classname：样式名
8. clearable：是否可清空选择项 默认值 true
9. addition：联动字段，数据格式如下： [{"DecoderNickName":"ParentId", "DataNickName":"Name"}]。它的含义就是当选中下拉框中某一项时，将该项中对应的"DecoderNickName"所对应的值，自动填充到界面上对应的"DataNickName"的控件中，达到类似联动填写的效果。
10. size：下拉框大小属性 （large/small/mini）
11. noDataText：选项为空时显示的文字
12. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
13. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的。
14. url：如果配置，那么说明ComboItem需要动态获取，从该url能够获取到代码表的数据。数据格式为json，内容为{"$regName”: :[{"Value":"1","Name":"xxx"},{"Value":"2","Name":"xxx"}...]}
15. regName：代码表的注册名，如果url配置内容，返回结果需要用该属性识别

#### 下拉框项(ComboItem)

* 标签名：tk-combo-item
* 标签总体：<tk-combo-item>text</tk-combo-item>
* 表现：
* 自定义属性：

1. value：控件绑定值
2. text：控件显示值
3. readonly：是否禁用，默认false
4. addition：联动相关 例如：{"ParentId"："1"} 联动的参数名称和参数值

### 搜索控件EasySearch

* 标签名：tk-easysearch
* 标签总体：<tk-easysearch></tk-easysearch>
* 表现：C:\Users\ADMINI~1\AppData\Local\Temp\1537695769(1).png
* 说明：EasySearch是通过在将输入框的内容通过ajax发送到后端，后端返回一些匹配的查询结果，前段以下拉列表的形态予以显示（这点类似于select控件），用户可以在下拉列表中选择合适的项目。此外，用户还可以点击右边的按钮弹出对话框进行搜索和选择（对话框支持树形结构）。此控件是单选控件，数据过程将在后面的章节予以说明。
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false，为true时，不仅输入框只读，后面的按钮也必须禁用。
5. value：控件绑定的显示值(string， number)
6. hiddenValue：绑定实际存储的Id值
7. placeholder：输入框提示信息，默认值为请输入...值(string)
8. classname：样式名
9. url：当input获取焦点时ajax请求数据的地址
10. dialogUrl：点击easysearch右侧按钮，弹出的页面地址
11. refField：级联字段，数据格式为[{"Field":"PartField",RefField:"Part"}]。在某些场景下，必须首先选择了部门，然后选择人员时，需要把人员约束在已经选择的部门内，这时就需要把当前记录下的Part字段对应的值取出，然后把PartField和Part字段的值通过ajax传回后端，后端在返回时，会对相应的结果做出过滤
12. regName：后端使用的EasySearch的注册名
13. addition：联动字段，含义同下拉框的addition
14. afterSelect：事件，在EasySearch选中某项后触发
15. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
16. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的

### 控件EasySearchText

* 标签名：tk-easysearch-text
* 标签总体：<tk-easysearch-text></tk-easysearch-text >
* 表现：
* 说明：EasySearchText在数据选择上和EasySearch类似，只是，它把选中的文本作为提交的数据，无需记录也无需提交对应的Id数据。本质上就是一个便捷输入的input控件
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false，为true时，不仅输入框只读，后面的按钮也必须禁用。
5. value：控件绑定的显示值(string， number)
6. placeholder：输入框提示信息，默认值为请输入...值(string)
7. classname：样式名
8. url：当input获取焦点时ajax请求数据的地址
9. refField：级联字段，意义同EasySearch的refField定义
10. regName：后端使用的EasySearch的注册名
11. addition：联动字段，含义同下拉框的addition
12. afterSelect：事件，在EasySearch选中某项后触发
13. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
14. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的

### 控件MultipleEasySearch

* 标签名：tk-multiple-easysearch
* 标签总体：<tk-multiple-easysearch></tk-multiple-easysearch>
* 表现：C:\Users\ADMINI~1\AppData\Local\Temp\1537767561(1).png
* 说明：MultipleEasySearch在单选的EasySearch基础上发展的，选择方式和EasySearch相同，但是和EasySearch不同的是，在选择完后它可以继续选择，从而达到多选的目的。
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false，为true时，不仅输入框只读，后面的按钮也必须禁用。
5. value：控件绑定的显示值，数据格式为Json格式，如{"Data":[{"Value":"310104","Name":"上海市徐汇区","DisplayName":"上海市徐汇区"},{"Value":"330104","Name":"浙江省杭州市江干区","DisplayName":"浙江省杭州市江干区"}]}
6. hiddenValue：绑定实际存储的Id值，数据格式为Id添加双引号后，用逗号分隔。如："310104","330104"
7. placeholder：输入框提示信息，默认值为请输入...值(string)
8. classname：样式名
9. url：当input获取焦点时ajax请求数据的地址
10. dialogUrl：点击easysearch右侧按钮，弹出的页面地址
11. refField：级联字段，意义同EasySearch的refField定义
12. regName：后端使用的EasySearch的注册名
13. addition：联动字段，含义同下拉框的addition
14. afterSelect：事件，在EasySearch选中某项后触发
15. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
16. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的

### 详情控件（Label）

* 标签名：tk-label
* 标签总体：<tk-label></tk-label>
* 表现
* 自定义属性：

1. name（必须）： json属性名
2. id：控件ID，json属性名，同Name
3. caption：标题，标题无时，外部标签不显示
4. value：控件绑定值(string， number)
5. classname：样式名

### 时间控件（Time）

* 标签名：tk-time
* 标签总体：<tk-time></tk-time>
* 表现：
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false
5. value：控件绑定值(string，time)
6. classname：样式名
7. selectableRange(可选) 限制可选时间范围 例如'18:30:00 - 20:30:00'或者传入数组['09:30:00 - 12:00:00'， '14:30:00 - 18:30:00']
8. placeholder：输入框提示信息，默认值为请输入时间...值(string)
9. format：时间格式化 默认值：HH:mm:ss
10. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
11. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的

### 日期控件（Date）

* 标签名：tk-date
* 标签总体：<tk-date></tk-date>
* 表现：
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false
5. value：控件绑定值(string，date)
6. classname：样式名
7. placeholder：选择日期...值(string)
8. defaultValue(可选)：打开时默认显示的时间
9. format：时间格式化 默认值：yyyy-MM-dd
10. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
11. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的
12. min：最小值，默认是1800-01-01（能否支持当前日期？）
13. max：最大值，默认是3000-01-01（能否支持当前日期？）

### 日期时间控件（DateTime）

* 标签名：tk-datetime
* 标签总体：<tk-datetime></tk-datetime>
* 表现：
* 自定义属性：

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. readonly：是否禁用(boolean) 默认值 false
5. value：控件绑定值(string， datetime)
6. classname：样式名
7. placeholder：默认值 选择日期时间...值(string)
8. defaultValue(可选)：打开时默认显示的时间
9. format：时间格式化 默认值：yyyy-MM-dd HH:mm
10. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
11. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的
12. min：最小值，默认是1800-01-01（能否支持当前日期？）
13. max：最大值，默认是3000-01-01（能否支持当前日期？）

### 密码控件(PassWord)

* 标签名：tk-password
* 标签总体：<tk-password></tk-password>
* 表现：
* 自定义属性：（必须的为必要属性）

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. placeholder：输入框提示信息，默认值为请输入...值(string)
5. classname：样式名
6. value：控件绑定值(string)
7. max-length：最大输入长度(number)
8. min-length：最小输入长度(number)
9. readonly：只读(boolean)
10. showicon：显示明文按钮(boolean) --mobile
11. size：输入框尺寸(string)(large， small， mini) --PC
12. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
13. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的

### 数字控件(Number)

* 标签名：tk-number
* 标签总体：<tk-number></tk-number>
* 表现：
* 自定义属性：（必须的为必要属性）

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. placeholder：输入框提示信息，默认值为请输入...值(string)
5. classname：样式名
6. value：控件绑定值(number)
7. readonly：原生属性，是否只读(boolean)
8. max：最大值(number)
9. min：最小值(number)
10. numbertype：数值类型(string：int、float)
11. afterpoint：小数点后位数限制(number)
12. size：输入框尺寸(string)(large， small， mini) --PC
13. step： 点击后加减值，默认为1
14. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
15. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的

### 上传控件（Upload）

* 标签名：tk-upload
* 标签总体：<tk-upload></tk-upload>
* 表现：
* 说明：上传控件按如下规则操作。当用户选取文件后，自动将文件按照传统上传的方式提交到uploadUrl上，这个地址接收到文件后，会返回一个json数据，需要把json数据的内容对应填充到value，fileSize，serverPath，contentType等属性绑定的值中，提交的时候，提交这些内容即可。
* 自定义属性：（必须的为必要属性）

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. caption：标题，标题无时，外部标签不显示
4. classname：样式名
5. value：绑定文件名字段
6. readonly：原生属性，是否只读(boolean)
7. uploadUrl：文件上传的地址(string)
8. maxSize：文件大小最大值限制(number)
9. fileType：文件类型限制(string) --针对PC端，移动端只有图片
10. fileSize：绑定文件大小字段
11. serverPath：绑定服务器文件路径字段
12. contentType：绑定mime类型字段
13. afterUpload：事件，在上传文件后触发
14. preview： 点击是否预览(boolean) – 仅当contentType为image时可用
15. tooltip：如果配置，控件的右上角显示一个帮助图标，以tooltip的方式显示配置的内容
16. required：是否必须录入，true就是必填项，同时界面要有明显标识，让用户知道该项是必填的

### 富文本控件（HtmlEditor）

* 标签名：tk-html-editor
* 标签总体：< tk-html-editor></ tk-html-editor>
* 说明：找一个合适html编辑器，然后进行相关封装。要求可以指定上传图片的地址，要求有readonly模式，要求相关的js能够根据具体需要加载。具体属性等选定相关控件后再定。

### 普通文本区间控件（InputRange）

* 标签名：tk-input-range
* 标签总体：< tk-input-range></ tk-input-range>
* 表现：
* 说明：Range系列控件都是为查询功能准备的，即由两个对应的单个控件组合成一个组件。大部分的属性都和单个控件相同，这里就不在重复，只说明不一样的属性。
* 自定义属性：

1. rangeSeparator：区间分隔符，默认为-
2. endValue：绑定区间的第二个值

### 数字区间控件（NumberRange）

* 标签名：tk-number-range
* 标签总体：< tk- number-range></ tk- number-range>
* 表现：
* 自定义属性：

1. rangeSeparator：区间分隔符，默认为-
2. endValue：绑定区间的第二个值

### 日期区间控件（DateRange）

* 标签名：tk-date-range
* 标签总体：< tk- date-range></ tk- date-range>
* 表现：
* 自定义属性：

1. rangeSeparator：区间分隔符，默认为-
2. endValue：绑定区间的第二个值

### 日期时间区间控件（DateTimeRange）

* 标签名：tk-datetime-range
* 标签总体：< tk- datetime-range></ tk- datetime-range>
* 表现：
* 自定义属性：

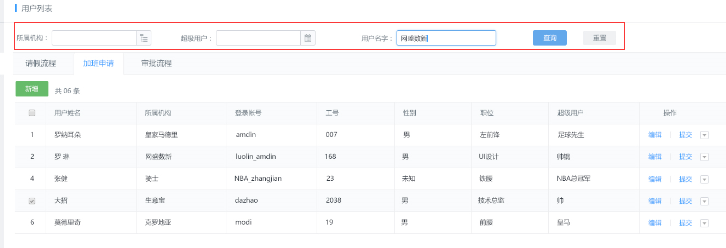
1. rangeSeparator：区间分隔符，默认为-
2. endValue：绑定区间的第二个值

## 基本页面

### 列表页面

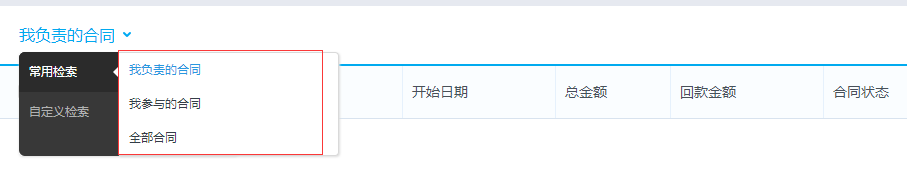
#### 查询条件

若设置了查询条件，查询条件都将在列表上方显示，如下图：



#### 工具栏

如果有多个固定查询条件，则使用下拉或者Tab的方式显示，默认显示第一个固定条件，可通过下拉方式切换其他查询固定条件，如果无固定查询条件，只显示该表名称，无下拉框或Tab。



配置的全局操作在工具栏右边显示，支持平铺、下拉和混合显示（具体显示方式由设计者决定）。

平铺：全部的全局操作铺开水平显示；

下拉：全部的全局操作在下拉框中显示；

混合：一部分全局操作以平铺方式显示，剩下的以下拉方式显示。



#### 操作栏

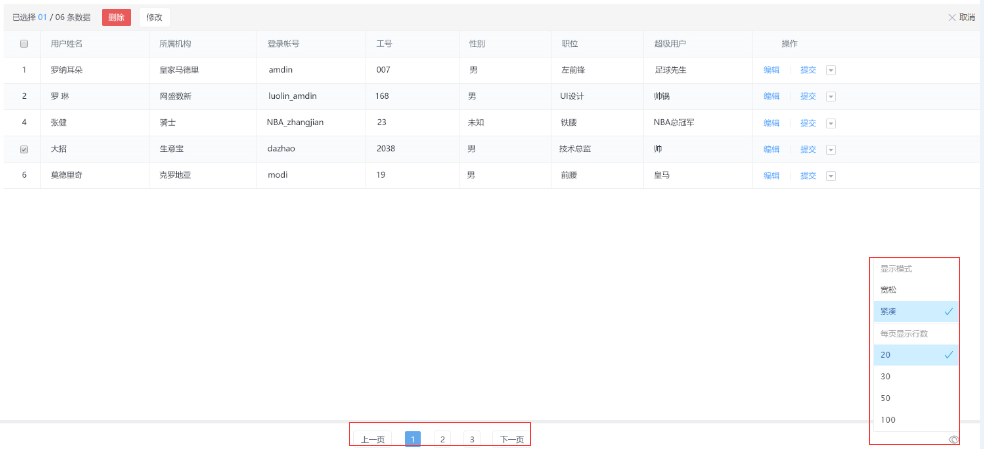
列表中的操作栏指的是行内操作，显示位置支持左边、右边和不显示，若显示则支持的显示格式与全局操作一样，同样支持平铺、下拉和混合三种显示方式（具体显示方式由设计者决定）。



#### 分页栏

分页栏固定在页面底部的中间显示，包括上一页、页码和下一页。

右侧可设置每页显示行数和显示模式，每页显示行数支持20、30、50、200，默认20，显示模式支持宽松和紧凑，默认紧凑。



#### 字段标题

点击某列标题，将自动对该列进行排序。如果再次点击该列，将进行反向排序。由于在数据库中，某些字段无法使用order by语句进行排序，因此需要提供屏蔽某列排序的功能，即该列字段无法通过点击排序。

### 编辑页面

#### 打开方式

新增/编辑页面打开方式支持模态窗口和侧推弹出两种方式。

* 模态窗口

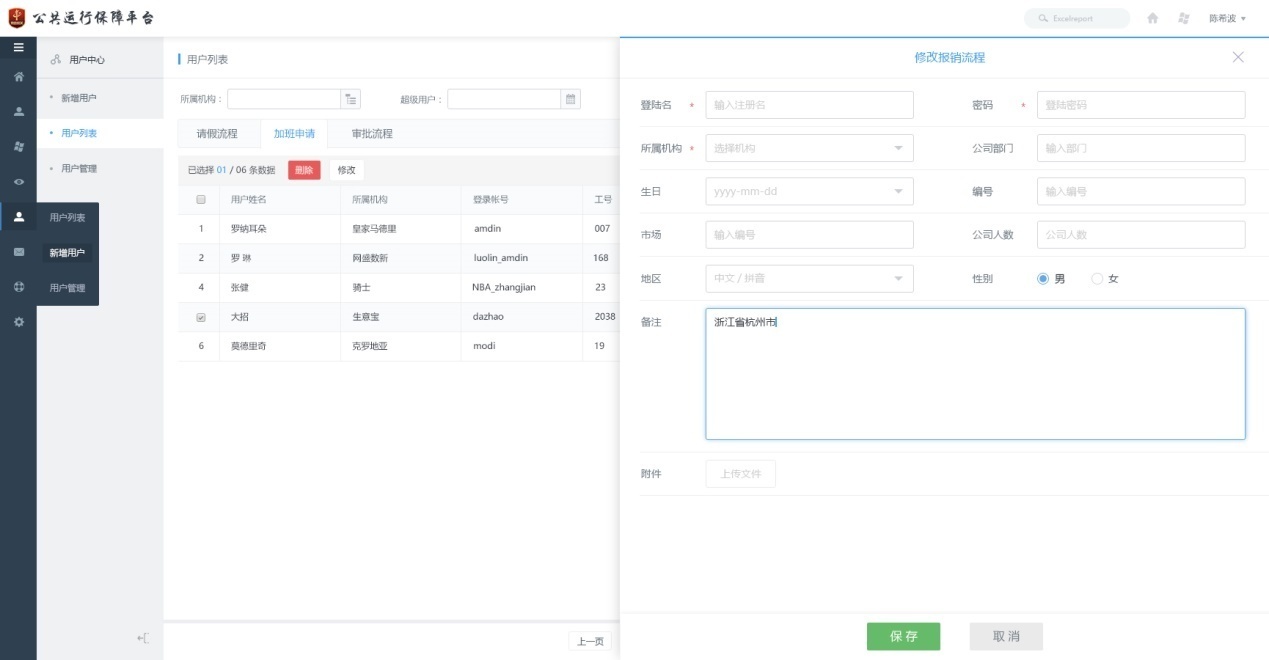
以模态窗口显示新增/编辑页面时，新增/编辑页面的底层页面全屏灰掉，不可操作。



新增/编辑页面布局支持一列、两列、三列显示，页面宽度根据显示列数自适应；页面高度根据显示行数自适应。

* 侧推弹出

以侧推弹出窗口的方式显示新增/编辑页面时，新增/编辑页面的底层页面均可操作，点击底层页面时新增/编辑页面自动关闭，编辑的信息不保存。



新增/编辑页面布局也支持一列、两列、三列显示，页面宽度根据显示列数自适应；页面高度固定为整高，超过整高时，新增/编辑页面显示滚动条。

#### 功能按钮

新增/编辑页面支持取消、保存和保存并新建的功能。



取消：不保存数据并关闭窗口返回列表页面；

保存并新建：保存该条数据信息并弹出新增/界面；

保存：保存该条数据信息并关闭窗口返回列表页面。

#### 字段分组

对于一张表中字段比较多，且可以进行二次分组的情况下，要求对该表中的字段进行分组显示，每组若干个字段，有组标签，可以展开/收缩，同时可以指定初始时是展开还是收缩。

#### 布局

在编辑界面中，如果一张表只有一条记录参与编辑，那么采用Normal布局。如果是多条记录，根据字段的多少，可以使用Table布局或者NormalTable布局。

通常情况下，在界面的上部，依照配置，显示Normal布局。在界面下部，显示Table/NormalTable布局。这里需要考虑，对所有Normal布局和所有Table/NormalTable布局采用平铺或者Tab布局两种方式。到底采用何种配置，由配置决定（提供相应的方式即可，不要求在运行中动态切换）

极端情况下，需要考虑，所有的Normal/Table/NormalTable布局都以Tab的方式显示，这点也需要提供一下。

### 详情页面

#### 打开方式

不管新增/编辑页面选择的打开方式是模态窗口还是侧推弹出，详情页面的打开方式统一为侧推弹出（从右侧推出）的方式。详情页面的底层页面（往往会是列表页面）所有功能均可操作，点击底层页面时详情页面自动关闭。



#### 功能按钮

详情页面含有功能区（功能区位置最终由设计者决定），支持修改和自定义的功能，功能布局同样支持平铺、下拉和混合三种显示方式（具体显示方式由设计者决定）。



#### 字段分组

同编辑页面中的字段分组

#### 布局

同编辑页面布局

### 单个树页面

单个树页面往往都是以对话框的方式显示树形结构，选择完毕后还需要和打开的页面进行相应的交互。

#### 展示方式



树的展示，要求根据后端返回结果，支持点击展开，或者全部展开。

要求可以在展示的时候，自动跳到选中节点的位置。

要求具有只能选择叶节点，不能枝节点的选项。

### 树维护页面

树维护页面主要是维护树形结构的数据表

#### 展示方式



通常情况下，左边显示树，右边显示选中节点的详情。同时可以通过点击修改，进入对应的修改页面进行修改，保存完成后，需要对左边的树进行局部刷新。

如果是新建节点，需要在左边树建立一个临时性节点，节点的名字可以配置，如果新建时，按取消按钮，那么这个临时性节点自动删除。如果新建保存，那么和修改一样，对左边的属性进行局部属性。

需要考虑在树上预留一个搜索框，一旦后续开发树的搜索功能，可以使用。

#### 功能按钮

C:\Users\ADMINI~1\AppData\Local\Temp\1537857329(1).png

树本身有如下功能，上移/下移是改变当前树节点在兄弟之间的位置。当处于第一个兄弟时，上移按钮无作用，同样，当处于最后一个兄弟时，下移按钮无作用。在上下移动节点时，需要触动后端地址，进行相应的数据操作。

新建根节点和新建子节点功能类似，只是它没有父节点而已。

此外，树还需要进行节点移动的功能，需要有开关进行配置。

### 框架页面

开发1-2种类型的框架页面（比如页头部菜单类型和页左侧菜单类型），供选择。框架页打开菜单考虑Tab页方式。

## 数据流程

**注意：所有的数据操作主要都是描述当前系统的，和新系统可能一致，可能不一致。将根据实际情况进行相应的调整。**

### EasySearch的数据操作

当在EasySearch的输入框，每输入一个字符时，EasySearch就要构造一个Json提交到后端。提交的json格式为：

var postdata = { "RegName": $input.regName, "Text":$input. 6. hiddenValue, "RefField": [] }

其中RegName是EasySearch配置的regName属性，Text是当前输入框的Id值。如果EasySearch配置了refField，那么按照配置，依次获取当前记录中对应的RefField的值，然后和配置的Field一起作为数组的一个元素加入到RefField中。例如：postdata.RefField.push({ "NickName": item.Field, "Value": refValue })

提交的url是EasySearch配置的url属性，提交方式post，提交数据为json。

返回的Json格式可能是：

[{

"Value": "1",

"Name": "总公司",

"DisplayName": "总公司",

"Data": {

"Value": "1",

"Name": "总公司",

"Code": "zgs",

"City": "110000",

"Layer": "000",

"ParentId": "-1",

"IsLeaf": "0",

"Active": "1",

"CreateId": "1",

"CreateDate": "2014/7/7 18:34:57",

"UpdateId": "1",

"UpdateDate": "2014/9/2 14:01:38",

"ROWNUMBER\_": "1"

}

}]

[{

"Value": "340100",

"Name": "安徽省合肥市",

"DisplayName": "340100 - 安徽省合肥市",

"Data": {

"Value": "340100",

"Name": "安徽省合肥市",

"ROWNUMBER\_": "1"

}

}, {

"Value": "340200",

"Name": "安徽省芜湖市",

"DisplayName": "340200 - 安徽省芜湖市",

"Data": {

"Value": "340200",

"Name": "安徽省芜湖市",

"ROWNUMBER\_": "2"

}

},...

}]

其中，每条记录的Value，Name，DisplayName都是固定的。正常情况下都是显示DisplayName，如果选中这条记录，那么分别将Value填充到EasySearch的hiddenValue属性中，Name填充到EasySearch的Value属性中。

如果配置了addition属性，那么就要在Data中查找配置的DecoderNickName的值，将这个值填充到对应的DataNickName中去。

### Upload的数据操作

当用户选择文件上传到upload配置的uploadUrl中后，将返回如下json格式

{

"NewWindow": false,

"UploadInfo": {

"FileName": "新建文本文档.txt",

"ContentType": "text/plain",

"ServerPath": "E:\\VS2014\\Mitu.CXCS\\ManageWeb\\WebSite\\UploadTemp\\15c7bd21-691a-4c4a-828c-a1f1ca4c964d.txt",

"FileSize": "29",

"WebPath": "/UploadTemp/15c7bd21-691a-4c4a-828c-a1f1ca4c964d.txt"

},

"Result": {

"Message": "",

"Result": "Success"

}

}

这里，需要处理UploadInfo中的数据，FileName的值给Upload控件value属性绑定的值，ContentType的值给Upload控件contentType属性绑定的值，ServerPath的值给Upload控件serverPath属性绑定的值，FileSize的值给Upload控件fileSize属性绑定的值。WebPath的值，如果Upload控件需要预览，可以使用。

上述这些值在提交的时候，确保能上传到服务器上即可，后台对相关信息作出适当的处理。

### 列表的数据操作

#### 列表的QueryString

列表的QueryString有如下参数（上述都是当前系统状况，如果觉得不妥，可以修订）：

* Page：当前的页码
* PageSize：一页的记录数
* Condition：已进行BASE64编码的查询条件
* Tab：固定查询条件的Id
* TotalCount：总记录数，不写或者值为0都会迫使后台重新计算总记录数，通常分页为了提高效率不会修改这个值，但是查询条件改变时需要将其改为0
* Sort：对第n列进行排序
* Order：升序或者降序排列
* GetData：当其值为Page时，返回局部html，否则返回全部html

当前系统处理列表上代码如下，可供参考：

Toolkit.list.\_queryParams = function (listView) {

var data = {

"Page": listView.attr("data-page"), "Condition": listView.attr("data-condition"),

"Tab": listView.attr("data-tab"), "TotalCount": listView.attr("data-totalcount"),

"TotalPage": listView.attr("data-totalpage"), "Sort": listView.attr("data-sort"),

"Order": listView.attr("data-order")

};

return data;

};

Toolkit.list.\_loadPage = function (options) {

var listView = $("#pageList");

if (listView.length === 0)

return;

var url = listView.attr("data-url");

var path = url.getPathName();

var param = url.getQueryJson();

var data = Toolkit.list.\_queryParams(listView);

data.GetData = "Page";

$.extend(param, data);

$.extend(param, options);

delete param[''];

param.\_ = (new Date()).getTime();

$.ajax({

type: 'get', dataType: 'text', url: path, data: param,

success: Toolkit.list.\_queryResult,

error: function () {

Toolkit.page.showError('数据提交失败！请稍候重试。');

}

});

};

Toolkit.list.\_loadPageByTab = function (tab) {

Toolkit.list.\_loadPage({ "Tab": tab, "Page": 0, "TotalCount": 0 });

};

Toolkit.list.\_loadPageByOrder = function (data) {

Toolkit.list.\_loadPage({ 'Sort': data.sort, 'Order': data.order });

};

Toolkit.list.\_loadPageByPage = function (page) {

Toolkit.list.\_loadPage({ "Page": page });

};

#### 查询

当前系统根据用户的输入，以及配置，自动构造出查询的JSON数据，例如：

{

"Condition": {

"Name": "test",

"OrgId": "",

"hdOrgId": "",

"LoginName": "",

"WorkNo": "",

"CreateDate": "",

"CreateDateEND": ""

},

"IsEqual": false

}:

其中，Condition中的Json数据都是以字段名/值对存在。IsEqual的含义是是否使用精确查询还是模糊查询。False表示模糊查询，true表示精确查询（界面上根据配置，可能会有一个表示精确查询的checkbox）。

查询是通过Post的方式，将查询条件提交到后端，后端根据提交的查询条件作出相应的反应，返回局部HTML刷新当前界面。

下面是当前系统的js代码，可供参考：

Toolkit.list.query = function (frmId) {

var frm = Toolkit.data.\_getForm(frmId, $(this), "QueryForm");

if (!frm)

return;

Toolkit.ui.enableButton(false);

var table = frm.attr("data-post").parseJSON();

var postData = { "Condition": {}, "IsEqual": false };

postData.Condition = Toolkit.data.getTableData(table, false);

postData.IsEqual = $("#\_searchMethod").prop("checked");

var listView = $("#pageList");

if (listView.length === 0)

return;

var listParams = Toolkit.list.\_queryParams(listView);

listParams.TotalCount = 0;

listParams.Page = 0;

var url = Toolkit.page.adjustUrl(listParams, "RetURL");

$.ajax({

type: 'post', dataType: 'text', url: url,

data: Toolkit.json.stringify(postData),

success: Toolkit.list.\_queryResult,

error: function () {

Toolkit.ui.enableButton(true);

//Toolkit.page.showError('数据提交失败！请稍候重试。');

}

});

return false;

};

#### 列表的刷新

列表的刷新比较特别，不能简单的调用js的reload刷新，这样会丢失页码，查询条件，排序等一系列的信息。因此，为了解决这个问题，当前系统使用jquery绑定了一个ListRefresh的事件，然后用trigger，来调用此事件，并通过返回值来感知是否真的调用。

当前的JS示例如下：

$("body").bind("ListRefresh", Toolkit.list.refreshCurrent);

Toolkit.list.refreshCurrent = function (elm) {

var listView = $("#pageList");

if (listView.length === 0)

return;

var listParams = Toolkit.list.\_queryParams(listView);

listParams.TotalCount = 0;

var url = Toolkit.page.adjustUrl(listParams, "RetURL");

document.location.href = url;

return true;

};

retValue = $("body").triggerHandler("ListRefresh");

if (!retValue)

document.location.reload();

#### 对Vue系统的设想

考虑到Vue支持数据绑定，因此页面的渲染分成两步，第一步先生成页面框架，如：

<el-table :data="listDataModule.UR\_USERS" border style="width:100%" class="">

<el-table-column prop="Name" label="Name" sortable width="120">

</el-table-column>

<el-table-column prop="UserName" label="UserName" sortable width="220">

<template scope="scope">

<div v-html="scope.row.UserName"></div>

</template>

</el-table-column>

<el-table-column prop="" label="操作" width="">

<template scope="scope">

...

</template>

</el-table-column>

</el-table>

根据后端的配置，将生成table的架构和相关的绑定字段，由于这个页面没数据，因此，后端在处理的时候可以考虑使用缓存，这样提高运行效率。

第二步，可以考虑在页面上放个指令，当碰到指令触发后，就去后端申请JSON数据，这种数据的抓取应该是一次性的，后端将返回可能的所有数据，因此可能根据需要做相应的绑定。比如，下面就是可能的示例：

{

"ListOperator": [{

"Id": "Insert",

"Caption": "新建",

"Info": "Insert",

"Content": "/insert/razortest\_user.c",

"ConfirmData": "",

"IconClass": "icon-plus",

"DialogTitle": "新建"

}],

"Count": [{

"TotalCount": "4",

"TotalPage": "0",

"CurrentPage": "0",

"PageSize": "15"

}],

"Sort": [{

"SortField": "-1",

"Order": "ASC",

"SqlCon": ""

}],

"UR\_USERS": [{

"Id": "1",

"Name": "<a v-tk-float-url='/detail/razortest\_user.c?Id=1' href='#'>sa</a>",

"OrgId": "总公司",

"LoginName": "sa",

"WorkNo": "",

"Sex": "男",

"Title": "",

"Active": "1",

"Admin": "<i class='icon-ok'></i>",

"ROWNUMBER\_": "1",

"\_OPERATOR\_RIGHT": "<li><a href=\"#\" v-tk-float-url=\"/update/razortest\_user.c?Id=1\">修改</a></li><li><a href=\"#\" v-tk-ajax-url=\"/delete/razortest\_user.c?Id=1\" data-confirm=\"确定删除吗？\">删除</a></li><li><a href=\"#\" data-title=\"重置密码\" data-dialog-url=\"/Library/WebUpdateXmlPage.tkx?Source=UserManager/ChangeUserPasswd&amp;UserId=1&amp;UserName=&lt;a v-tk-float-url='/detail/razortest\_user.c?Id=1' href='#'>sa&lt;/a>\">重置密码</a></li><li><a href=\"#\" data-title=\"设置角色\" data-dialog-url=\"/Library/WebUpdateXmlPage.tkx?Source=UserManager/UserPart&amp;UserId=1\">设置角色</a></li>",

"OrgId\_Name": "总公司",

"Sex\_Name": "男"

}, {

"Id": "1001",

"Name": "<a v-tk-float-url='/detail/razortest\_user.c?Id=1001' href='#'>test</a>",

"OrgId": "test2",

"LoginName": "testd",

"WorkNo": "",

"Sex": "",

"Title": "",

"Active": "1",

"Admin": "<i class='icon-ok'></i>",

"ROWNUMBER\_": "2",

"\_OPERATOR\_RIGHT": "<li><a href=\"#\" v-tk-float-url=\"/update/razortest\_user.c?Id=1001\">修改</a></li><li><a href=\"#\" v-tk-ajax-url=\"/delete/razortest\_user.c?Id=1001\" data-confirm=\"确定删除吗？\">删除</a></li><li><a href=\"#\" data-title=\"重置密码\" data-dialog-url=\"/Library/WebUpdateXmlPage.tkx?Source=UserManager/ChangeUserPasswd&amp;UserId=1001&amp;UserName=&lt;a v-tk-float-url='/detail/razortest\_user.c?Id=1001' href='#'>test&lt;/a>\">重置密码</a></li><li><a href=\"#\" data-title=\"设置角色\" data-dialog-url=\"/Library/WebUpdateXmlPage.tkx?Source=UserManager/UserPart&amp;UserId=1001\">设置角色</a></li>",

"OrgId\_Name": "test2",

"Sex\_Name": ""

}, {

"Id": "1002",

"Name": "<a v-tk-float-url='/detail/razortest\_user.c?Id=1002' href='#'>test3</a>",

"OrgId": "Test1",

"LoginName": "test",

"WorkNo": "test",

"Sex": "",

"Title": "",

"Active": "1",

"Admin": "<i class='icon-remove'></i>",

"ROWNUMBER\_": "3",

"\_OPERATOR\_RIGHT": "<li><a href=\"#\" v-tk-float-url=\"/update/razortest\_user.c?Id=1002\">修改</a></li><li><a href=\"#\" v-tk-ajax-url=\"/delete/razortest\_user.c?Id=1002\" data-confirm=\"确定删除吗？\">删除</a></li><li><a href=\"#\" data-title=\"重置密码\" data-dialog-url=\"/Library/WebUpdateXmlPage.tkx?Source=UserManager/ChangeUserPasswd&amp;UserId=1002&amp;UserName=&lt;a v-tk-float-url='/detail/razortest\_user.c?Id=1002' href='#'>test3&lt;/a>\">重置密码</a></li><li><a href=\"#\" data-title=\"设置角色\" data-dialog-url=\"/Library/WebUpdateXmlPage.tkx?Source=UserManager/UserPart&amp;UserId=1002\">设置角色</a></li>",

"OrgId\_Name": "Test1",

"Sex\_Name": ""

}, {

"Id": "1010",

"Name": "<a v-tk-float-url='/detail/razortest\_user.c?Id=1010' href='#'>dddd2</a>",

"OrgId": "child1",

"LoginName": "dddd",

"WorkNo": "",

"Sex": "男",

"Title": "",

"Active": "1",

"Admin": "<i class='icon-remove'></i>",

"ROWNUMBER\_": "4",

"\_OPERATOR\_RIGHT": "<li><a href=\"#\" v-tk-float-url=\"/update/razortest\_user.c?Id=1010\">修改</a></li><li><a href=\"#\" v-tk-ajax-url=\"/delete/razortest\_user.c?Id=1010\" data-confirm=\"确定删除吗？\">删除</a></li><li><a href=\"#\" data-title=\"重置密码\" data-dialog-url=\"/Library/WebUpdateXmlPage.tkx?Source=UserManager/ChangeUserPasswd&amp;UserId=1010&amp;UserName=&lt;a v-tk-float-url='/detail/razortest\_user.c?Id=1010' href='#'>dddd2&lt;/a>\">重置密码</a></li><li><a href=\"#\" data-title=\"设置角色\" data-dialog-url=\"/Library/WebUpdateXmlPage.tkx?Source=UserManager/UserPart&amp;UserId=1010\">设置角色</a></li>",

"OrgId\_Name": "child1",

"Sex\_Name": "男"

}],

"Info": [{

"UserId": "1",

"RoleId": "1",

"Source": "razortest\_user",

"Module": "true",

"IsHttpPost": "false",

"Guid": "5a9f9c4c-e96d-4888-8368-359eab8c9852",

"PageX": "c",

"SessionId": "oozkzqasik3g0v5q3f0zzlag",

"Culture": "zh-CHS",

"Style": "List"

}],

}

其中：

* ListOperator：工具栏中的全局操作按钮
* Count：页码，分页信息
* Sort：排序，查询条件
* UR\_USERS，列表的数据源，不同的功能可能这个表名也是不同的。

上面是源于对当前系统的转化，所以，最终数据不做定论，可以根据具体需要进行调整。

这两步做完，就能够展示出相应的列表数据了。

### 编辑页面的数据操作

#### 数据提交的JSON格式

当前系统，最复杂的莫过于编辑页面中构造提交的JSON数据。因为不同的控件获取值的方式不同，获取值的个数也不同，而且如果扩展控件，还需要尝试了解如何取得扩展控件对应的值。如果采用数据绑定，可能这个问题就迎刃而解了。

当前系统的JSON数据示例如下：

{

"UR\_USERS": [{

"Id": "1001",

"Name": "test",

"OrgId": "1005",

"hdOrgId": "test2",

"Phone": "",

"Mobile": "",

"Email": "",

"WorkNo": "",

"Sex": "",

"Birthday": "",

"Worked": "",

"Education": "",

"Origin": "",

"hdOrigin": "",

"Title": "",

"WorkLimit": "10",

"IdentNo": "",

"InDate": "",

"Area": "",

"hdArea": "",

"Address": "",

"Postal": "",

"Active": "1",

"Out": "0",

"OutDate": "",

"Note": "",

"CreateDate": "2014-08-01",

"Admin": "1"

}]

}:

这是单表的JSON结构

{

"SYS\_FUNCTION": [{

"Id": "6",

"ShortName": "Org",

"Name": "组织机构",

"Url": "~/usermanager\_organization.c",

"ParentId": "1"

}],

"SYS\_SUB\_FUNC": [{

"Id": "12",

"FnId": "6",

"NameId": "Update",

"Name": "修改节点",

"Position": "Global",

"Icon": "icon-edit",

"UseKey": "0",

"Content": "",

"UseMarco": "0",

"ConfirmData": "",

"DialogTitle": "",

"Info": "Update,Dialog",

"OperOrder": "10",

"Page": "Detail"

}, {

"Id": "13",

"FnId": "6",

"NameId": "Delete",

"Name": "删除节点",

"Position": "Global",

"Icon": "icon-remove",

"UseKey": "0",

"Content": "",

"UseMarco": "0",

"ConfirmData": "确认删除吗？",

"DialogTitle": "",

"Info": "Delete,AjaxUrl",

"OperOrder": "20",

"Page": "Detail"

}, {

"Id": "14",

"FnId": "6",

"NameId": "Insert",

"Name": "新建子节点",

"Position": "Global",

"Icon": "icon-plus",

"UseKey": "1",

"Content": "~/CNewChild/~{CcSource}.c",

"UseMarco": "1",

"ConfirmData": "",

"DialogTitle": "",

"Info": "Dialog",

"OperOrder": "30",

"Page": "Detail"

}, {

"Id": "15",

"FnId": "6",

"NameId": "\_Empty",

"Name": "空操作",

"Position": "Row",

"Icon": "",

"UseKey": "0",

"Content": "避免角色在配置权限时，出现需要选中该功能，但不要所有子操作的情况无法实现的BUG",

"UseMarco": "0",

"ConfirmData": "",

"DialogTitle": "",

"Info": "",

"OperOrder": "-1",

"Page": "List"

}]

}:

这个是主从表的JSON。

#### 提交后返回的JSON格式

编辑页面通过Post将数据提交，如果成功，返回的Json如下：

{

"NewWindow": false,

"Result": {

"Message": "http://localhost:24885/usermanager\_function.c?InitValue=6",

"Result": "Success"

}

}

其中Message是如下常量时，会做如下动作：

* WeixinClose：在微信中关闭窗口；
* Refresh：刷新当前页面；
* Back：返回上一页等同于浏览器的back按钮；
* CloseDialog：关闭对话框；
* ListRefresh：刷新列表页面（和刷新不一样，会考虑列表分页，查询条件等因素）；CloseDialogAndRefresh：关闭对话框并使用列表刷新的方式刷新页面
* 如无法识别，认为它是Url，进行跳转

上述问题，也就是为什么我需要插件的缘由之一吧。写死了不利于扩展。

如果提交的数据存在逻辑校验错误，那么返回的JSON如下

{

"Result": {

"Message": "校验错误",

"Result": "Error"

},

"FieldInfo": [{

"TableName": "UR\_USERS",

"NickName": "Origin",

"Message": "籍贯不存在",

"Position": "0"

}]

}

FieldInfo可能会返回很多内容，TableName表示表名，NickName是字段名，Position是第几条记录，Message是错误信息。上述信息，JS处理后，在界面上显示



上述结构，对于传统的DOM访问应该没什么问题，但是如果采用绑定，可能是个大问题。这块，可以根据实际情况进行调整。

#### 对Vue系统的设想

和列表一样，编辑页面也是分两步走，第一步，根据表的配置信息，获取表的模板，这个模板只有数据绑定，没有具体数据。如下所示（只是参考，将根据具体需要改变）

<div>

<div data-type="edit" class="data-edit float-edit" data-col="2" v-tk-data-url="../../common/js/data2.json" slot="content">

<tk-form ref="form" :model="editDataModule" label-width="80px" :col="2" module="editDataModule" v-cloak>

<tk-row module="editDataModule">

<div>{{editDataModule['UR\_USERS'][0].UserName}}</div>

<tk-input v-model="editDataModule['UR\_USERS'][0].UserName" caption="测试" col="1" module="editPageModule"

:error="errorDataModule['UR\_USERS'][0].UserName"></tk-input>

<tk-input v-model="editDataModule['UR\_USERS'][0].Phone" caption="手机" col="1" module="editPageModule"

:error="errorDataModule['UR\_USERS'][0].Phone"></tk-input>

<tk-check-list module="editPageModule" v-model="editDataModule['UR\_USERS'][0].Education" col="2"

caption="性别">

<tk-check-item value="01">女生</tk-check-item>

<tk-check-item value="02">男生</tk-check-item>

<tk-check-item value="03">其他</tk-check-item>

</tk-check-list>

</tk-row>

</tk-form>

</div>

</div>

假设由于指令的触发，将通过配置的相关地址去获取数据，数据格式如下，当获取数据后，和现有界面上绑定，即是需要保存的数据。提交的时候，把这份数据提交上去即可。

{

"UR\_USERS": [

{

"Id": "1",

"Name": "sa",

"OrgId": "1",

"LoginName": "sa",

"LoginPasswd": "gsP3ok1cS6qzxyKhaWcU9g==",

"LoginDate": "2018/5/28 16:35:51",

"Email": "gfdgfd",

"Sex": "01",

"Origin": "340207",

"InDate": "2014-08-07",

"Area": "330102",

"Postal": "230022",

"Active": "1",

"Out": "0",

"CreateId": "1",

"CreateDate": "2014/7/7 18:34:57",

"UpdateId": "1",

"UpdateDate": "2016/9/22 20:11:37",

"Admin": "1",

"OrgId\_Name": "总公司",

"Sex\_Name": "男",

"Origin\_Name": "安徽省芜湖市鸠江区",

"Area\_Name": "浙江省杭州市上城区",

"CreateId\_Name": "sa",

"UpdateId\_Name": "sa"

}

],

"URL": [

{

"SelfURL":

"http%3a%2f%2flocalhost%3a24885%2fupdate%2frazortest\_user.c%3fId%3d1",

"DSelfURL": "http://localhost:24885/update/razortest\_user.c?Id=1"

}

],

"Info": [

{

"UserId": "1",

"RoleId": "1",

"Source": "razortest\_user",

"Module": "true",

"IsHttpPost": "false",

"Guid": "8b37d634-0dc7-4ca5-bac5-adf931e0af82",

"PageX": "c",

"SessionId": "oozkzqasik3g0v5q3f0zzlag",

"Culture": "zh-CHS",

"Style": "Update"

}

],

"QueryString": [{ "Id": "1" }]

}

在处理数据时，有如下两个建议：

* 对取得的数据进行浅层clone保存，这样如果要实现重置按钮，这个数据就可以使用了。
* 如果是主从表，能否给从表记录扩展一个状态的字段，比如叫\_Status，初始值都是unchanged，如果是新的记录，\_Status是Insert，如果是删除的记录，\_Status是Delete，当然如果能监控记录修改，可以把\_Status改为changed。这样做可以大大提高后台的处理效率，不过可能会增加前端的难度，这个看情况实现吧。

### 详情页面的数据操作

#### 子表的操作

详情页面如果要显示从表，它将借助现在的List界面，从List获取数据源。这需要前端进行相关的操作，而且可以对从表的记录进行分页，排序。由于当前系统js写的不太好，所以，所有的从表都是共用一个div，这样最大限度的共享了原有的js代码，但是效率，展示都不太好。希望今后的系统，都是每一个从表一个div，div上有标记，没有加载向后台申请展示和数据，如果加载过，直接显示即可。这样效果更好。

#### 对Vue系统的设想

和其他页面一样，详情页面也是先有页面框架，如下示例

<div>

<div data-type="detail" class="data-detail float-detail" data-col="2" v-tk-data-url="../../common/js/data.json" slot="content">

<div slot="content">

<tk-form ref="form" label-width="80px" :col="1" module="detailDataModule" v-cloak>

<tk-label caption="用户名" :value="detailDataModule['table'][0].UserName" module="detailDataModule"></tk-label>

<tk-label caption="教育" :value="detailDataModule['table'][0].Education" module="detailDataModule"></tk-label>

<tk-label caption="手机" :switchs="false" :value="detailDataModule['table'][0].Phone" module="detailDataModule"></tk-label>

</tk-form>

</div>

<div slot="button">

<el-button type="" size="small" class="dnlayer-foot-mt9 dnlayer-foot-ml10">

<i class="el-icon-edit mr5"></i>修改</el-button>

<a href="" class="el-button mr5" size="small" type="dialog" data-url="edit.html" data-confirm="是否删除?"

data-title="删除" data-width="-1" data-height="-1" v-acustom="12345" data-ajax="../../common/data.json">

<i class="el-icon-delete2 mr5"></i>删除dialog</a>

<el-button type="primary" size="small" class="dnlayer-foot-mt9 dnlayer-foot-ml10">

<i class="el-icon-plus mr5"></i>新建子节点</el-button>

</div>

</div>

</div>

然后根据配置，去获取相应的Json数据和界面绑定。Json数据示例如下：

{

"UR\_USERS": [

{

"Id": "1",

"Name": "sa",

"OrgId": "总公司",

"LoginName": "sa",

"LoginPasswd": "gsP3ok1cS6qzxyKhaWcU9g==",

"LoginDate": "2018/9/26 22:04:53",

"Phone": "",

"Mobile": "",

"Email": "gfdgfd",

"WorkNo": "",

"Sex": "男",

"Birthday": "",

"Worked": "",

"Education": "",

"Origin": "安徽省芜湖市鸠江区",

"Title": "",

"WorkLimit": "",

"IdentNo": "",

"InDate": "2014-08-07",

"Area": "浙江省杭州市上城区",

"Address": "",

"Postal": "230022",

"Active": "1",

"Out": "<i class='icon-remove'></i>",

"OutDate": "",

"Note": "",

"CreateId": "1",

"CreateDate": "2014/7/7 18:34:57",

"UpdateId": "1",

"UpdateDate": "2016/9/22 20:11:37",

"PasswdChangeDate": "",

"UnlockDate": "",

"Admin": "<i class='icon-ok'></i>"

}

],

"DetailOperator": [

{

"Id": "Update",

"Caption": "修改",

"Info": "Update",

"Content": "/update/razortest\_user.c?Id=\*Id\*",

"ConfirmData": "",

"IconClass": "icon-edit",

"DialogTitle": "修改"

}

],

"URL": [

{

"RetURL": "http%3a%2f%2flocalhost%3a24885%2flist%2frazortest\_user.c%3fPage%3d0%26Condition%3d%26Tab%3d%26TotalCount%3d8%26TotalPage%3d0%26Sort%3d-1%26Order%3dASC",

"SelfURL": "http%3a%2f%2flocalhost%3a24885%2fdetail%2frazortest\_user.c%3fId%3d1",

"DRetURL": "http://localhost:24885/list/razortest\_user.c?Page=0&Condition=&Tab=&TotalCount=8&TotalPage=0&Sort=-1&Order=ASC",

"DSelfURL": "http://localhost:24885/detail/razortest\_user.c?Id=1"

}

],

"Info": [

{

"UserId": "1",

"RoleId": "1",

"Source": "razortest\_user",

"Module": "true",

"IsHttpPost": "false",

"Guid": "92d29fd8-89b2-4a3f-b56c-203f870d927e",

"PageX": "c",

"SessionId": "pheaidqgto3z340qlolzjshe",

"Culture": "zh-CHS",

"Style": "Detail"

}

]

}

另外，详情中有明细表的，能否有些明细表依旧采用Ajax获取的方式获取，而有些则直接从当前的数据绑定中获取？

## 指令

### v-tk-url

* 附着标签：a，button
* 主要作用：跳转到指定页面，通常情况下，需要在跳转的url上自动添加RetUrl={转义后的本页url地址}
* 参数：string | {url：string，[confirm：string， newwindow：bool， noRetUrl：bool]}
* 参数含义：url是跳转的地址，如果没有其他参数，直接写就行；confirm如果配置，则在跳转前进行询问，得到yes回答才能跳转；newwindow为true时，打开新的页面进行跳转，默认为false；noRetUrl为true时，不需要再url自动添加RetUrl的参数，默认为false

### v-tk-dialog-url

* 附着标签：a，button
* 主要作用：弹出对话框，对话框中访问url，显示其返回内容
* 参数：string | {url：string，[title：string，height：int，width：int，useiFrame：bool， closeAction：string]}
* 参数含义：url是对话框显示内容的地址，如果没有其他参数，直接写就行；title是对话框的标题，height为对话框的宽度。其中0为默认值，-1为最大屏幕高度，>0为实际像素点高度。width为对话框的宽度，含义同width相同。useiFrame，当为true时，提供iFrame的方式显示url中的内容，此时url返回应该是全部HTML。closeAction是关闭对话框后需要做的操作，设想以插件的方式处理，这里提供的是注册名。（另：该参数仅是设想，可能会根据对话框的API进行适当的调整）

### v-tk-ajax-url

* 附着标签：a，button
* 主要作用：当对象点击时，用ajax的方式访问该地址
* 参数：string | {url：string，[confirm：string，method：string]}
* 参数含义：url是ajax访问的地址，如果没有其他参数，直接写就行；confirm如果配置，则在跳转前进行询问，得到yes回答才能跳转；method为http的方式，默认为get

### v-tk-float-url

* 附着标签：a，button
* 主要作用：读取url的内容，加载到侧推的浮层，并打开侧推浮层
* 参数：string | {url：string，[floatId：string]}
* 参数含义：url是加载内容的地址，如果没有其他参数，直接写就行；floatId，浮层DIV的Id（默认假定为\_tk\_float）

### v-tk-load-url

* 附着标签：div
* 主要作用：将url的内容自动加载到div中
* 参数：string
* 参数含义：需要加载内容的url地址

## 其他页面组件以及相关

### 对话框API

提供一套打开对话框的API，可以使用js打开对话框。

要求：

* 对话框是模态的
* 能够在对话框上再次弹出对话框

### 简易对话框

提供alert，question，prompt三种简易对话框的API

### Menu

参考ElementUI的控件进行相应封装

### Tab

参考ElementUI的控件进行相应封装

### Button

参考ElementUI的控件进行相应封装

### Badge

* 标签名：tk-badge
* 标签总体：<tk-badge></tk-badge>
* 表现：1504506143(1) 1504506166(1)
* 自定义属性：（必须的为必要属性）

1. name（必须）： json属性名
2. ~~id：控件ID，json属性名，同Name~~
3. value：显示值（Number，String）String格式时，max不起作用
4. max：最大值（Number），value的类型必须为数字，否则不显示，当超过最大值时会显示为’{max}+’
5. is-dot：模式改为点模式（Boolean）默认为false

### Tooltip

* 标签名：tk-tooltip
* 标签总体：<tk-tooltip></tk-tooltip>
* 表现：1504837432(1)
* 自定义属性：（必须的为必要属性）

1. ~~id：可传入的id~~
2. classname：需传入的class，或者可直接写class=”需写入的class”
3. value：提示的内容
4. placement：属性值为（方向--对齐位置），四个方向：top，left，right，bottom；
5. 三个对齐位置：start，end，默认为空；
6. slot：根据name分发，name为tooltip
7. effect：主题色，默认为dark，可选light（白色）

## 特别要求

### 将普通js函数转化为上下文函数

由于这个是框架程序，一旦写好后，在开发时，就基本不会动，如果要动，也只能在外部动，而不能在内部改。因此，需要考虑以下的情况。

通常后端在写一些前端代码的时，往往采用传统的js写法，即function funcName() {…}的方式来处理。那么，如果一个button的click事件需要挂接函数，我们生成配置的时候，会写@click=”funcName”，但是@click事件是要求配置在Vue对象中的函数，而普通的funcName不是。这就需要提供一个相应函数，能把普通的js函数转换为上下文函数，比如@click=”convertToContext(‘funcName’)”，这样可以方便后端程序员的开发。

### 插件结构

定义某种插件接口，这样，在二次开发的时候，能够方便的创建一个插件工厂的容器。同时，写的插件可以方便的注册到插件工厂中去。通常要求的插件是(string，func)，当然如果是ts，可以是(string，class)或者(string，interface)。最好可以使用自动注册的方式，建议考虑ts中的装饰器。

### URL处理

URL的处理最好采用现有的开源库，原则上不要使用字符串拼接的方式去构造URL。比如原先，我就从网上找了一个URI.js来处理URL。用字符串拼接的方式，处理QueryString的替换，转义等会比较麻烦。

### JWT的支持

可能在不久的将来，整个系统将采用JWT进行认证（暂时现在还不支持）。因此，前端到时候需要在提交的head中添加一些东西，或者提供一个机构，方便以后修改。

## 编程规范要求

* 遵循约定俗称的js编码规范
* 安装eslint或者tslint（如果使用TypeScript，优先建议tslint），基本上需要按照lint提出的建议修改（比如使用===，而不是==。尽量使用let，而不是var）。如有特别情况，请予以说明
* 建议安装某种插件，保存时自动对齐。
* 原则上，一个控件一个文件，无特别情况，不要形成一个超大的文件，不方便阅读。