这个文档编写 CtrlLib 的 文档

|  |
| --- |
| # CtrlLib 基类 该基类包含 CtrlLib 的部分功能实现和声明虚方法 原则上不允许直接使用这个类 |
| ## 构造函数 constructor(data) @param {Object} data 存入的数据 |

## ## CtrlLib 属性

## ### 静态属性

## idIndex {Number} 用于计算控件的id的属性, 不要在 addend 方法以外的地方修改它; 更不要把值改小

## ### 原型属性

## childCtrlType *{CtrlLib}* 子控件类的集合 用于添加子控件

## ### 实例属性

* name *String*  
  并没有什么用的属性
* ctrlLibID *Number*  
  用于区分所有控件的唯一id, 会在 new 一个 CtrlLib 或 它的派生类 时自增  
  this.ctrlLibID=CtrlLib.idIndex++;
* data *Object*  
  存放数据的地方
* rootNodes *Array< HTMLElement >*  
  控件的根级 dom 子元素
* parentNode *HTMLElement*  
  控件的父级 dom 元素
* parentCtrl *CtrlLib*  
  控件 的 父级控件
* childCtrl *Object*  
  子控件的集合
* elements *Object*  
  子元素的集合
* ctrlActionList *Object*  
  控件的动作集合
* childCtrlActionList *Object*  
  childCtrlActionList[ctrlID] *Array< Function >*  
  被挂起的子控件动作  
  因为子控件加载不一定是瞬间加载完成的(可能需要等待http请求完成后才能加载)  
  所以动作会被挂起

|  |
| --- |
| ## CtrlLib 原型属性 |
| \* childCtrlType < Object > 子控件 类的集合 |

## ## CtrlLib 方法

* addend(\_parentNode,…surplusArgument)  
  将控件加入到指定的dom元素内  
  @param {HTMLElement} \_parentNode 指定的父dom元素  
  @param surplusArgument 这些参数将会传递到 createContent 方法中
* createContent(surplusArgument) **虚方法**  
  创建内容  
  需要把元素赋值到 this.rootNodes 上
* callParent(\_fnc,…surplusArgument)  
  呼叫父级控件  
  @param {Function} \_fnc 执行的动作  
  @param {any} surplusArgument \_fnc 执行的参数
* callChild(childCtrlID,\_fnc,surplusArgument) **虚方法**  
  呼叫子控件, 如果兄弟控件没有加载完成将会被挂起  
  @param {String} childCtrlID 控件 在父控件的父元素 的 ctrlID  
  @param {Function} \_fnc 执行的动作  
  @param {any} surplusArgument \_fnc 执行的参数  
  **这个方法在 ExCtrl 中有一个实现**
* callBrother(childCtrlID,\_fnc,…surplusArgument) **虚方法**  
  呼叫兄弟控件, 如果兄弟控件没有加载完成将会被挂起  
  @param {String} childCtrlID 控件 在父控件的父元素 的 ctrlID  
  @param {Function} \_fnc 执行的动作  
  @param {any} surplusArgument \_fnc 执行的参数  
  **这个函数在 ExCtrl 中有一个实现**
* initialize(…argument) **虚方法**  
  addend刚开始执行的函数,  
  能调用到 addend 的 argument  
  通过返回字符串 “stop” 可以挂起 addend  
  当然你需要在这个方法中再使用 addend
* callback(…argument) **虚方法**  
  addend 后的回调  
  能调用到 addend 的 argument
* reRender() **虚方法**  
  重新渲染  
  根据data渲染部分需要渲染的内容  
  **这个方法在 ExCtrl 中有一个实现**
* reRender\_callback() **虚方法**  
  重新渲染完成后的回调
* touchCtrlAction(actionKey)  
  触发控件事件的方法  
  @param {String} actionKey 事件的类型
* addCtrlAction(actionKey,\_fnc)  
  添加控件事件  
  @param {String} actionKey 事件的类型  
  @param {Function} \_fnc 事件的执行函数 将会以控件为 this 指针

|  |
| --- |
| # ExCtrl 类 **派生于 CtrlLib** |
| 控件库派生类的基类,需要在派生时添加 **bluePrint** {DEF\_VirtualElementList} 原型属性 这个类继承了 CtrlLib 类 的 所有属性和方法 |

## ## ExCtrl 属性

## ### 静态属性

* attrKeyStr  
  保存用于编辑 bluePrint 的 xml 的关键字

|  |
| --- |
| ### 原型属性 |
| \* templateStringIsHTML 规定模板字符串能否为 html, 如果启用, 就别在 ctrl-if 的前面放模板字符串，否则可能会导致 ctrl-if 的渲染出错 |
| \* bluePrint **虚属性** 储存渲染用的蓝图。在实际用的派生类中一定要加入这个属性 |
| ### 实例属性 \* dataLinks {{}} 用于登记模板字符串的属性 |

## ## ExCtrl 原型方法

* getElementsByCtrlID(ctrlID)  
  通过 ctrlID 获取元素  
  @param {String} ctrlID  
  @returns {Array< Element >} 返回元素,包括ctrl-for 的
* attrHandle(key,elements,ves,i,\_attrVal,tname,k,forFlag)  
  控制标签的属性  
  @param {String} key 属性的key  
  @param {Array} elements 实例的 elements 的引用，用于添加新的子元素  
  @param {Array} ves DEF\_VirtualElement 的数组  
  @param {Number} i 当前的ves的下标  
  @param {String} \_attrVal 属性值  
  @param {String} tname 临时的元素名称，用作实例的 elements 当前的索引  
  @param {Number} k 当前的ves的下标  
  @param {String} forFlag 表示是不是 for 的  
  @returns {Number} 返回运算完成后的ves下标
* stringRender(str,ctrlID,type,ishtml,attrkey,tgt)  
  渲染 模板字符 内容  
  @param {String} str write TemplateKeyStr  
  @param {String} ctrlID 登记 ID  
  @param {String} type 登记 类型  
  @param {Boolean} ishtml 控制返回值, 默认将返回字符串 ，非0 将返回 DocumentFragment  
  @param {Array<>} attrkey 如果是登记的 标签的属性值 这个是属性的 key  
  @param {Element} tgt  
  @return {String||DocumentFragment} 字符串 或 包含内容的文档片段
* renderFor(elements,ves,i,forStr,tname)  
  渲染for 循环生成内容 @param {Array} elements  
  @param {Array} ves DEF\_VirtualEle  
  @param {Number} i 当前的ves的索引  
  @param {String} forStr 属性内容  
  @param {String} tname elements的索引  
  @returns {Number} 返回跳过子元素的索引
* ctrlIf(elements,ves,i,attrVal,tname)  
  用于控制元素是否出现  
  @param {Array} elements  
  @param {Array} ves DEF\_Virt  
  @param {Number} i 当前的ves的索引  
  @param {String} attrVal 属性内容  
  @param {String} tname elements的索引  
  @returns {Number} 返回跳过子元素的索引
* renderChildCtrl(element,ve,childCtrlType)  
  渲染子控件  
  @param {Element} element 加载子控件的元素  
  @param {DEF\_VirtualElement} ve  
  @param {String} childCtrlType 控件的类型
* itemVEToElement(ves,\_nameEX,forFlag)  
  把 ve 转换成 js 的 Element 对象;  
  @param {Array} ves DEF\_VirtualElement list  
  @param {String} \_nameEX 用来添加命名的  
  @return {Object{elements:{},fragment:DocumentFragment}}
* renderString()  
  重新渲染模板字符串内容
* reRender()  
  根据依赖项重新渲染所有内容 仅有在 stringRender 中登记过才能使用
* renderCtrl\_before(ctrlID)  
  重新渲染元素前面的文本
* renderCtrl\_innerEnd(ctrlID)  
  重新渲染模板字符串内容: 加在元素末尾的内容  
  @param {String} 目标的 ctrlID
* renderCtrl\_attr(ctrlID,attrkey)  
  重新渲染模板字符串内容: 元素 的 属性  
  @param {String} 目标的 ctrlID  
  @param {String} 目标的属性的 key
* renderStyle()  
  渲染styleElement内容
* 方法集 reRenderAttrCtrl
  + ctrl-for(bluePrint,tgtElem)  
    重新渲染由ctrl-for生成的内容 @param {DEF\_VirtualElementList} bluePrint 蓝图  
    @param {Element} tgtElem 将要渲染的目标
  + ctrl-if(bluePrint,tgtElem)  
    重新渲染被ctrl-if控制的内容  
    @param {DEF\_VirtualElementList} bluePrint  
    @param {Element} tgtElem

|  |
| --- |
| ## ExCtrl 静态方法 |
| \* getJsonData() 请求 api 并运行 json 反序列化 |
| \* xmlToCtrl()函数 将xml(html)代码制作成一个EXCtrl类 @param {String} htmlStr html代码 @param {Object} \_prototype 追加到派生控件的原型链 @returns {class} 返回一个 ExCtrl 的 派生类 |

# DEF\_VirtualElementList 类

这个是用于给 ExCtrl 类保存 蓝图的类  
采用的是序列化树(数组)的数据结构 ### 构造函数 constructor(ves,maxDepth,style) @param {Array} ves 序列化的虚拟元素树  
@param {Number} maxDepth 最大深度  
@param {DEF\_CSSVE} style 样式元素  
— ## DEF\_VirtualElementList 实例属性 — \* ves {Array<DEF\_VirtualElement>}  
存储着序列化树的虚拟元素的数组 \* maxDepth {Number}  
树的最大深度 \* style {DEF\_CSSVE}  
存储控件的样式的蓝本 — ## DEF\_VirtualElementList 静态属性 — \* voidElementsTagName {Array<String>}  
所有无内容元素(短标签)的tag name — ## DEF\_VirtualElementList 原型方法 — \* getCtrlIDByIndex(index)  
根据 ves 的下标, 查找ctrlID  
@param {Number} index  
@returns {String} ctrlID

* getByCtrlID(ctrlID)  
  根据 ctrlID 寻找 项  
  @param {String} ctrlID  
  @returns {DEF\_VirtualElement} 返回目标
* getIndexByCtrlID(ctrlID){ 根据 ctrlID 寻找 项 @param {String} ctrlID @returns {Number} 返回目标的下标
* getByLastDepth(start,depth,min=0)  
  向前寻找目标深度的 ves item  
  @param {Number} start 起点  
  @param {Number} depth 目标 深度  
  @param {Number} min 最小深度, 如果超过限制将返回最小深度的下标  
  @returns {Number} 返回目标的下标, 如果超过最小深度限制，将在返回中添加一个 flag=true 属性
* getChild(vesindex) 获取子元素  
  @param {Number} vesindex 目标ves的下标  
  @returns {{indexs:Array,ves:Array<,p:Number}}  
  @return {Array} indexs 在原蓝图中的下标集合  
  @return {Array} ves 子元素集合  
  @return {Number} p 下一个同级元素的下标
* getParent(vesIndex) 获取父元素  
  @param {Number} vesIndex 子元素在 ves 的下标  
  @returns {Number} 返回父元素的下标  
  — ### DEF\_VirtualElementList 静态方法 —
* xmlToVE(\_xmlStr) 把xml转换成 DEF\_VirtualElementList @param {String} xmlStr @return {DEF\_VirtualElementList} {ves:Array,maxDepth:Number} —- # DEF\_VirtualElement 类 作为虚拟元素树的叶子 供 htmlToControl 处理xml字符串 ### 构造函数 constructor(tagName,depth,attribute,before,innerEnd) @param {String} tagName 标签名 @param {Number} depth 深度 @param {Array} attribute 标签的属性 [{key,val}] @param {String} before 在标签前的内容 @param {String} innerEnd 最后一段内容 — ## DEF\_VirtualElement 实例属性
* tagName 标签名
* depth 深度
* attribute 标签的属性 [{key,val}]
* before 在标签前的内容
* innerEnd 最后一段内容 — ## DEF\_VirtualElement 原型方法
* setAttribute(key,val)  
  存入属性  
  @param {String} key 属性的 key  
  @param {String} val 属性的 value  
  @return {Number} 1:join; 2:update
* getAttribute(key)  
  获取属性  
  @param {String} key 属性的 key  
  @return {String} 返回属性的 value
* getAttributesByKeyBA(before="“,after=”")  
  用key的 before 或 after 获取属性  
  @param {String} before key 的开头的字符串  
  @param {String} after key 的结尾的字符串  
  @returns {Array} 返回 item 格式: {key:String, value:String}  
  —- —- # DEF\_CSSVE 类 给控件添加 style 样式标签 — ### 构造函数 constructor(cssString) @param {String} cssString css 格式的字符串 — ## DEF\_CSSVE 原型方法
* addString(cssString)  
  加入 css 字符串 @param {String} cssString css 格式的字符串
* createCssString(ctrlLibID,that)  
  创建 css 的文本  
  @param {Number} ctrlLibID 控件的id  
  @param {CtrlLib} that  
  @returns {String}
* getByLastDepth(start,depth)  
  向前寻找目标深度的 cssList’s item  
  @param {Number} start 起点  
  @param {Number} depth 目标 深度  
  @returns {Number} 返回目标的下标

## —-

# DEF\_CSSVEItem 类

一个 style 选择器和样式 ### 构造函数 constructor(selectors,cssString,depth) @param {Array} selectors 选择器的数组 @param {String} cssString css 的内容 @param {Number} depth 深度 ## DEF\_CSSVEItem 属性  
selectors {Array} 选择器的数组  
cssString {String} css 的内容  
depth {Number} 深度  
## DEF\_CSSVEItem 方法  
\* toString(\_ctrlLibID,\_that)  
将对象渲染成css语句  
@param {String} \_ctrlID  
@param {CtrlLib} \_that  
@returns {String}  
—- —- # 编辑 XML 以 为ExCtrl 提供 bluePrint —-

* 使用 模板字符串 在 内容 或者在 属性的 value 里使用 ${\*表达式\*} 即可食用。 表达式的this指针指向当前控件 示例: ``` xml <div class="${this.data.class}">
* ```

*以下的 bluePrint 的关键字存在于 Exctrl.attrKeyStr中* \* 使用 循环生成子元素 在父元素中使用属性 “ctrl-for” 值为 for 的体  
示例:  
XML <div ctrl-for="this.i=0;this.i<10;++this.i"> <div> ${this.i} </div> </div> 控制的值一定要写在 this 中，因为运行时的 初始化 和 判读体 和 回调 以及 内容 都是不同上下文的

* 为元素添加 dom 事件, 在元素中使用属性 pa-${actType}, 值为操作的 js 代码  
  示例:
* <div pa-click="console.log(this,tgt,e);"></div>  
   <!-- 点击后会在控制台打印: 当前控件, 当前元素, 当前事件的 event 对象 -->
* 此处的 this 指针为控件, 如果需要用当前元素 应该使用 tgt  
  e 指向当前的 事件对象

### **特殊的 dom 事件**

控件提供了一些特殊的事件, 这些事件是强化后的 dom 事件  
\* pa-keyup[{Number}] & pa-keydown[{Number}]

当按下或抬起组合键时组合键触发时间  
方括号中的值为按键的 keyCode

示例:  
xml <div pa-keydown[38]="console.log('这是方向键向上按下')" pa-keydown[38,40]="console.log('这是方向键上下一起按下')"></div> 和一般的dom事件一样，它也能得到 **this** *控件* , **tgt** *当前元素* ,**e** *事件对象*

* pa-resize  
  元素大小缩放时触发的事件,和普通的dom事件一样使用，但是这个事件没有e参数