### 可视化编辑实现逻辑说明

采用iframe载入设计页面，然后父页面向iframe页面注入控制脚本和css样式文件。

依旧采用模块化开发模式，父页面use“site\_editor.js”，然后将页面参数传递进去。

{

url: "index.html", // 默认设计页面地址

basePath: "{{ rootUrl }}/", // basePath

assetPath: "{{ assetPath }}/" // 静态资源相对路径

}

\* site\_ediror.js主要是设计器逻辑上的代码、父页面相关事件操作代码和服务端数据交互的代码等。

注入脚本、样式表和操作盒子DOM结构：

iframe页面加载完毕后，将父页面隐藏的操作盒子DOM结构拷贝后注入到iframe页面，然后同步注入jquery-insert.js和site\_editor\_inject.js脚本，注入site\_editor\_inject.css样式表文件。

\* jquery-insert.js是修改后的jquery文件，为了避免注入jquery文件与iframe网站页面中可能已有的jquery定义冲突，脚本中将$和jQuery重新定义为$2和jQuery2，所以在父页面中使用\_editingWindow.$2。

\* site\_editor\_inject.js主要作用为iframe页面中操作盒子相关事件处理操作，例如“添加挂件”、“编辑挂件”、“删除挂件”和“移动挂件位置”等操作。

添加挂件数据、修改挂件数据和删除挂件数据只操作扁平格式挂件数据属性，不操作页面树结构格式数据，

site\_ediror.js中定义了一个全局变量“window.\_globeManage”，主要作用是能够让网站页面调用父页面的提供外部调用的接口。

$.extend(\_globeManage, {

/\*\* 删除挂件提醒窗口

@widgetId: 要删除的挂件id（用于删除挂件数据）

@callback: 删除成功后的回调函数[如果需要的话]

\*/

deleteWidgetConfirm: function(widgetId, callback){ },

/\*\* 打开编辑挂件面板

@widgetId: 要编辑的挂件id（如果传入的是widgetData对象，）

\*/

openEditWidgetPanel: function(widgetId){ },

// 打开追加组件面板

openAppendWidgetPanel: function(){ },

// 更新挂件数据（挂件设置项面板中调用）

updateWidgetData: function(widgetData){ },

// 添加挂件数据（添加挂件面板调用）

addWidget: function(widgetData){ }

});

其中下面三个函数是给操作盒子相关元素调用的：

1. deleteWidgetConfirm
2. openEditWidgetPanel
3. openAppendWidgetPanel

其中下面两个函数是给弹出面板iframe页面调用的：

1. updateWidgetData
2. addWidget

下面列出关键属性及函数：

\_pageData：页面数据，包括页面名称，所用模板，挂件列表等信息，树结构

\_widgetsData：包含此页面所有挂件数据的map

\_widget\_identify：挂件标识

\_container\_identify：容器标识

\_operatebox\_widget\_relation\_identify：操作盒子与挂件关联的属性标识

\_editingWindow：iframe页面的window对象

\_editWidgetDataDialog：编辑挂件数据的dialog对象

\_addWidgetDialog：添加挂件的dialog对象

function \_getWidgetData(widgetId)：传入挂件id从扁平数据中获取挂件数据

function \_\_getWidgetsTreeDataBySiteDom ()：通过页面dom结构获取挂件树结构格式数据

function \_changeDesignPage(pageAddr)：跳转设计页面地址

function \_validateEmptyContainer($container)：校验容器是否为空容器

function \_getFrameDomByProperty(prop, val)：通过属性获取编辑页面dom对象