2017-4-10

组件规范 1.1.2

规范目的

无规矩不成方圆，组件规范的目的，在于规范行为，在于规范书写。规范后的代码，便于阅读，便于维护。我们要努力达到“即使出错，也是规范的错”的境界。

组件代码开头注释

组件JS代码 和 css代码中，头部需要添加注释 如：

/\*

\* cbbtn 1.0.0

\* author： niulei

\* Time: 2017.3.27

\* description: 按钮组件，用来发布事件或订阅事件。

\*/

组件JS代码规范

组件JS是组件的主体，自定义标签通过组件JS解析成浏览器认可的html元素。

1. 组件代码结构
2. 声明组件构造函数
3. 在公共模块-解析模块 注册组件
4. 组件Dom结构列表块
5. 组件默认配置块
6. 组件初始化函数 initTag
7. 组件自身属性设置函数 setAttr
8. 组件自身其他API
9. 组件JS详细规范 以cbbtn,js为例
10. 声明组件构造函数

组件的构造函数名和组件的名字一致，且首字母大写，如：

var Cbbtn = function () {};

1. 注册组件

因为自定义标签的统一解析时，需要告诉解析器要解析哪些模块，所以需要使用解析器的API来注册组件

// 第一个参数为自定义的标签名，全小写，且必须以cb开头

// 第二个参数为组件的构造函数

ParsingHelper.registerComponent("cbbtn", Cbbtn);

1. 组件Dom结构列表块

一个自定义标签会解析出一系列的Dom结构，这些解析出来的Dom结构的主体框架就缓存在这个结构列表中，解析时通过读取组件配置中的domDiv来读取指定的Dom结构，存在默认的dom结构。如：

//保存cbbtn组件结构的列表

(function () {

//cbbtn组件结构列表

var domDivList = {

default: ' <div class="cbbtn-div">' +

' <button class="cbbtn" hasbind="true"></button>' +

' </div>'

};

//根据 配置中的domDiv获得容器结构

Cbbtn.getDomDiv = function (domDivName) {

if (domDivName && typeof domDivName === "string") {

var componentDiv = domDivList[domDivName];

if (componentDiv) {

return componentDiv;

} else {

err("未找到" + domDivName + "组件结构，使用了默认结构",1);

return domDivList["default"];

}

} else {

return domDivList["default"];

}

}

})();

这个块为一个闭包结构，目的是为了将Dom结构列表封装在闭包中，只向外暴露一个读取列表的API。在书写这个闭包时，需注意：

1. 在返回Dom结构的API中，需要做校验，即没有传递Dom结构名或未找到Dom结构时，返回默认的Dom结构 。
2. 组件Dom结构都需要一层DIV来包裹，且这层DIV的class名为“组件名-div”，如“cbbtn-div”，当自定义标签上配置class属性时，会覆盖此层的class。
3. 由于事件绑定模块，校验模块的需要，要在需要事件绑定的DOM上配置 hasbind=”true”，在需要校验的表单元素上配置 hascheck=”true” 。
4. 组件Dom结构中，必须存在一直不变的class名，如cbbtn。
5. 组件默认配置块

自定义标签有了配置，就会有默认的配置，当自定义标签缺少配置项时，会使用默认配置项。默认配置块也是一个闭包结构，并向外暴露一个API来返回默认配置的副本。如：

(function () {

var options = {

domDiv: "default"，

type:"button"

};

Cbbtn.getOptions = function () {

return $.extend({}, options);

}

})();

每个默认的配置中，最少要有一项配置，domDiv – 组件默认结构

除此之外，还需注意 ，在返回默认配置时，必须返回配置的副本，负责会出大错。

1. 组件初始化函数initTag

解析器解析自定义标签时，会通过解析器的注册表来寻找组件构造函数原型链中的初始化函数 - initTag，并把解析到的自定义标签封装为Jquery对象传递给initTag函数，因此，所有组件的初始化函数必须为：

组件构造函数名.prototype.initTag($tag){} 如：

Cbbtn.prototype.initTag = function ($tag) {

//获得具体的配置项

var options = $.extend(Cbbtn.getOptions(), $tag.data("options"));

//获取Dom结构，且转化为Jquery对象

var $dom = $(Cbbtn.getDomDiv(options["domDiv"]));

//处理自定义标签中的内容

$dom.find(".cbbtn").text($tag.text());

//将处理好的Dom结构插在自定义标签后

$tag.after($dom);

//处理公共部位的宏命令

Macro.macroCommand($tag, $dom, options);

//设置自身的属性

Cbbtn.prototype.setAttr($dom, options);

//删除自定义标签

$tag.remove();

//返回解析后的Dom

return $dom;

};

组件的初始换函数，是整个组件JS中最终要的部分，因此需要注意：

1. 初始化函数的名称必须为 .prototype.initTag($tag)，且形参为$tag,用来接收解析器传来的自定义标签。
2. 初始化函数中的流程，强烈建议为示例中的流程。
3. 获取自定义标签的options配置时，必须与默认的属性合并。
4. 获取Dom结构后，必须转化为Jquery’对象。且按需处理自定义标签的内容。
5. 解析后的Dom必须插在自定义标签后。
6. 必须使用Macro.macroCommand($tag, $dom, options); 宏命令，此命令会为组件进行事件绑定，订阅，校验，生成cid，为组件外层设置内联样式等公共操作，且接受三个参数，分别为 自定义标签，解析后的dom，自定义标签配置。
7. 设置自身属性的函数名 建议为 .prototype.setAttr($dom , options)，接受两个参数 ，分别为 解析后的 dom 和 自定义标签的配置。
8. 解析完后，必须将自定义标签删除
9. 最后，必须将解析的Dom返回。
10. 要获得标签是第几次解析，可通过 $tag.data(“tagindex”) 来获得。
11. 组件自身属性设置函数 setAttr

当组件解析处理完公共部位后，就需要为自身的属性进行设置，因此，此处建议使用 . prototype.setAttr($dom , options) 来命名，如：

Cbbtn.prototype.setAttr = function($dom, options){

//设置按钮属性

options["type"]?$dom.find(".cbbtn").attr("type",options["type"]):err("无此属性",1);

//设置按钮是否失效

options["disabled"]?$dom.find(".cbbtn").prop("disabled",options["type"]).addClass("disabled"):err("无此属性",1);

};

这个函数中也有这几点建议：

1. 因为组件结构中存在一直不会改变的原子class名，因此，这里可以使用.find(“cbbtn”)来设置属性
2. 设置属性时建议使用三目运算符
3. 三木运算符中的err（）为错误抛出模块 ，err（”str”,type）接受两个参数，第一个为错误内容，第二个为错误严重程度，1位最低。
4. 组件自身其他API

组件除了以上部分外，还会有其他的内容，因每个组件都会不同，所以，这里至指定函数命名的规范：

组件构造函数名. prototype . xxx = function(){};

组件样式规范

1. 组件的样式费为两个维度，即 主题维度 和 具体样式的维度。
2. 组件样式名 组件js名 组件的自定义标签名 必须一致，且以cb开头
3. 因为自定义标签 class属性的存在 ，且此处的class属性会替换掉组件最外层Div的class属性， 另外组件内部的class名基本不变， 因此，书写样式时应该以最外层的class名为准，来区分各个区块。
4. 组件最外层的class名 应该以 组件名-xxx 来命名 如 cbbtn-div

还未增加的规范

1. 组件的配置中，需要增加css属性配置，如 height：xxx， 此处的规范还未定义。