atitit.atiHtmlUi web组件化方案与规范v1

未来的WEB开发，将会效仿今天桌面软件的开发路子，那就是“组件化”。

目前组件化最好的就是React angular了。。

React 的最大问题是以js为核心，嵌入html

儿anrular最大问题是啰嗦，繁琐。

# 如何在现有html 标签基础上定义自己的组件

推荐像bootstrap哪些使用oocss的方式，在class属性上增加一个特殊的classname来表明组件的类型....

也可以使用扩展属性

# 组件的构成与定义

Html css构成组件的外观,js添加act..

# 组件的加载

使用html文件作为组件，小型组件可以使用ajax读取载入，这样组件可以方便dw的设计。。大型组件可以使用ajax载入，也可以iframe方式载入。。

# 操作组件getVal,SetVal

需要当html dom当主要的...

comboxUtil(“#id”).getVal(); 相同当特别的的jq,,专门化的jq

这导致了界面可视化设计极差。。应该以html dom为基础，在html的基础上内嵌js。。Js本身就是代码，可视化要求相对较低。。Dom 标签使用常用的class 增加一个特点的oocss class来表明一个组件的类型，以及复合组件，或者可增加一个扩展属性。。

2.3.组件的属性以js为主  
这带来查看属性值的调试的问题，如果可以在html标签上扩展属性，就方便调试，毕竟浏览器查看工具可以一目了然的查看属性的值

Web Components提供了一种组件化的推荐方式，具体来说，就是：

* 通过shadow DOM封装组件的内部结构
* 通过Custom Element对外提供组件的标签
* 通过Template Element定义组件的HTML模板
* 通过HTML imports控制组件的依赖加载

这几种东西，会对现有的各种前端框架/库产生很巨大的影响：

* 由于shadow DOM的出现，组件的内部实现隐藏性更好了，每个组件更加独立，但是这使得CSS变得很破碎，LESS和SASS这样的样式框架面临重大挑战。
* 因为组件的隔离，每个组件内部的DOM复杂度降低了，所以选择器大多数情况下可以限制在组件内部了，常规选择器的复杂度降低，这会导致人们对jQuery的依赖下降。
* 又因为组件的隔离性加强，致力于建立前端组件化开发方式的各种框架/库（除Polymer外），在自己的组件实现方式与标准Web Components的结合，组件之间数据模型的同步等问题上，都遇到了不同寻常的挑战。
* HTML imports和新的组件封装方式的使用，会导致之前常用的以JavaScript为主体的各类组件定义方式处境尴尬，它们的依赖、加载，都面临了新的挑战，而由于全局作用域的弱化，请求的合并变得困难得多。

# 组件化的本质目的并不一定是要为了可复用，而是提升可维护性。 不具有复用性的组件”

大量的业务界面，这块东西很显然复用价值很低，基本不存在复用性，但仍然有很多方案中把它们“组件化”了，使得它们成为了“不具有复用性的组件”。为什么会出现这种情况呢？

组件化的本质目的并不一定是要为了可复用，而是提升可维护性。这一点正如面向对象语言，Java要比C++纯粹，因为它不允许例外情况的出现，连main函数都必须写到某个类里，所以Java是纯面向对象语言，而C++不是。

另外有一些框架/库偏爱用函数逻辑来生成界面，早期的ExtJS，现在的React（它内部还是可能使用模板，而且对外提供的是组件创建接口的进一步封装——jsx）等，这种实现技术的优势是不同平台上编程体验一致，甚至可以给每种平台封装相同的组件，调用方轻松写一份代码，在Web和不同Native平台上可用。但这种方式也有比较麻烦的地方，那就是界面调整比较繁琐。

我们来看看如何把一个业务界面切割成组件。

有这么一个简单场景：一个雇员列表界面包括两个部分，雇员表格和用于填写雇员信息的表单。在这个场景下，存在哪些组件？

对于这个问题，主要存在两种倾向，一种是仅仅把“控件”和比较有通用性的东西封装成组件，另外一种是整个应用都组件化。

对前一种方式来说，这里面只存在数据表格这么一个组件。  
对后一种方式来说，这里面有可能存在：数据表格，雇员表单，甚至还包括雇员列表界面这么一个更大的组件。

这两种方式，就是我们之前所说的“局部组件化”，“全组件化”。

比如Angular里面的这种：

<div ng-include="'aaa/bbb/ccc.html'"></div>

就不给它什么名字，直接include进来，用文件路径来区分。这个片段的作用可以用其目录结构描述，也就是通过物理名而非逻辑名来标识，目录层次充当了一个很好的命名空间。

就像刚才的雇员表单，既然你不从标签的命名上去区分，那一定会在组件上加配置。比如你原来想这样：

<EmployeeForm heading="雇员表单"></EmployeeForm>

然后在组件内部，判断有没有设置heading，如果没有就不显示，如果有，就显示。过了两天，产品问能不能把heading里面的某几个字加粗或者换色，然后码农开始允许这个heading属性传入html。没多久之后，你会惊奇地发现有人用你的组件，没跟你说，就在heading里面传入了折叠按钮的html，并且用选择器给折叠按钮加了事件，点一下之后还能折叠这个表单了……

然后你一想，这个不行，我得给他再加个配置，让他能很

2015前端组件化框架之路(转) - GISER\_U - 博客园.htm

这个问题讨论完了，我们来看看另外一个问题：如果UI组件有业务逻辑，应该如何处理。

比如说，性别选择的下拉框，它是一个非常通用化的功能，照理说是很适合被当做组件来提供的。但是究竟如何封装它，我们就有些犯难了。这个组件里除了界面，还有数据，这些数据应当内置在组件里吗？理论上从组件的封装性来说，是都应当在里面的，于是就这么造了一个组件：

<GenderSelect></GenderSelect>

里的标签，并不只是界面元素，甚至逻辑组件也可以这样，比如这个代码：

<my-panel>

<core-ajax id="ajax" url="http://url" params="{{formdata}}" method="post"></core-ajax>

</my-panel>

注意到这里的core-ajax标签，很明显这已经是纯逻辑的了，在大多数前端框架或者库中，调用ajax肯定不是这样的，但在浏览器端这么干也不是它独创，比如flash里面的WebService，比如早期IE中基于htc实现的webservice.htc等等，都是这么干的。在Polymer中，这类东西称为非可见元素（non-visual-element）。

在Web Components与前端组件化框架的关系上，我觉得是这么个样子：

各种前端组件化框架应当尽可能以Web Components为基石，它致力于组织这些Components与数据模型之间的关系，而不去关注某个具体Component的内部实现，比如说，一个列表组件，它究竟内部使用什么实现，组件化框架其实是不必关心的，它只应当关注这个组件的数据存取接口。

然后，这些组件化框架再去根据自己的理念，进一步对这些标准Web Components进行封装。换句话说，业务开发人员使用某个组件的时候，他是应当感知不到这个组件内部究竟使用了Web Components，还是直接使用传统方式。（这一点有些理想化，可能并不是那么容易做到，因为我们还要管理像import之类的事情）。

### **高内聚**

****又是一个软件工程的高频词！****我们将相关的一些功能组织在一起，把一切封装起来，而在组件的例子中，就可能是相关的功能逻辑和静态资源：JavaScript、HTML、CSS以及图像等。这就是我们所说的内聚。

这种做法将让组件更容易维护，并且这么做之后，组件的可靠性也将提高。同时，它也能让组件的功能明确，增大组件重用的可能性。

### **可组合**

之前也讨论过，基于组件的架构让组件组合成新组件更加容易。这样的设计让组件更加专注，也让其他组件中构建和暴露的功能更好利用。

不论是给程序添加功能，还是用来制作完整的程序，更加复杂的功能也能如法炮制。这就是这种方法的主要好处。

是否有必要把所有的东西转换成组件，事实上取决于你自己。没有任何理由让你的程序由 你自己 的组件组合成你最惊叹的功能 ，乃至 最花哨的功能。而这些组件又反过来构成其他组件。如果你从这个方法中得到了好处，就想方设法地去坚持它。然而要注意的是，不要用同样的方法把事情变得复杂，你并不需要过分关注如何让组件重用。而是要关注呈现程序的功能。

#### **Iframe 容器化**

还记得iframe们吗？我们还在使用它们，是因为他们能确保组件和控件的JavaScript和CSS不会影响页面。 [Shadow DOM](http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/) 也能提供这样的保护，并且没有iframe带来的负担。正式的说法是：

Shadow DOM的设计是在shadow根下隐藏DOM子树从而提供封装机制。它提供了建立和保障DOM树之间的功能界限，以及给这些树提供交互的功能，从而在DOM树上提供了更好的功能封装。

#### **HTML导入**

我们长时间以前就可以导入JavaScript和CSS了。 [HTML导入功能](http://www.html5rocks.com/en/tutorials/webcomponents/imports/)提供了从其他HTML文档中导入和重用HTML文档的能力。这种简单性同时意味着可以很方便地用一些组件构建另一些组件。

最后，这样的格式很理想，适合可重用组件，并且可以用你最喜欢的包管理解决方案发布（例如： [bower](http://bower.io/)、 [npm](https://www.npmjs.org/) 或者 [Component](https://github.com/component/guide)）。

组件化的Web王国 - 博客 - 伯乐在线.htm