从HTML Components的衰落看Web Components的危机

搞前端时间比较长的同学都会知道一个东西，那就是HTC（HTML Components），这个东西名字很现在流行的Web Components很像，但却是不同的两个东西，它们的思路有很多相似点，但是前者已是昨日黄花，后者方兴未艾，是什么造成了它们的这种差距呢？

HTML Components的一些特性

因为主流浏览器里面只有IE支持过HTC，所以很多人潜意识都认为它不标准，但其实它也是有标准文档的，而且到现在还有链接，注意它的时间！

我们来看看它主要能做什么呢？

它可以以两种方式被引入到HTML页面中，一种是作为“行为”被附加到元素，使用CSS引入，一种是作为“组件”，扩展HTML的标签体系。

行为

行为（Behavior）是在IE5中引入的一个概念，主要是为了做文档结构和行为的分离，把行为通过类似样式的方式隔离出去，详细介绍在这里可以看：

行为里可以引入HTC文件，刚才的HTC规范里就有，我们把它摘录出来，能看得清楚一些：

engine.htc

这是一个计算器的例子，我们先大致看一下代码结构，是不是很清晰？再看看现在用jQuery，我们是怎么实现这种东西的：是用选择器选择这些按钮， 然后添加事件处理函数。注意你多了一步选择的过程，而且，整个过程混杂了声明式和命令式两种代码风格。如果按照它这样，你所有的JS基本都丢在了隔离的不 相关的文件中，整个是一个配置的过程，分离得很干净。

除了这种计算器，还有规范文档中举例的改变界面展示，或者添加动画之类，注意它们的切入点，都是相当于附加在特定选中元素上的行为，即使DOM不给JS暴露任何选择器，也毫无影响，因为它们直接就通过CSS的选择器挂到元素上了。

这种在现在看来，意义不算明显，现在广为使用的先选择元素再添加事件，也是不错的展现和行为分离方式。

但另外一种使用方式就不同了。

组件

狭义的HTML5给我们带来了什么？是很多新增的元素标签，比如section，nav，acticle，那这些东西跟原先直接用div实现的，好处在哪里呢？在于语义化。

所谓语义化，就是一个元素能清晰表达自己是干什么的，不会让人有歧义，像div那种，可以类比成是一个Object，它不具体表示什么东西，但可以当成各种东西来用。而nav一写，就知道，它是导航，它就像有class定义的一个实体类，能表达具体含义。

那么，原有的HTML元素显然是不够的，因为实际开发过程中要表达的东西显然远远超出这些

元素，比如日历，这种东西就没有一个元素用来描述它，更不用说在一些企业应用中可能会出现的树之类复杂控件了。

不提供原生元素，对开发造成的困扰是代码写起来麻烦，具体可以看之前我在知乎的一个回复，第三点：

所以，大家都想办法去提供自己的扩充元素的方式，现在我们是知道典型的有angularjs，polymer，但很早的时候也不是没有啊：

看，这就是HTC的添加自定义元素的方式，每个元素可以定义自己对外提供的属性、方法，还有事件，自己内部可以像写一个新页面一样，专注于实现功能。而且你发现没有，它考虑得很长远，提供了命名空间，防止你在一个页面引入两个不同组织提供的同名自定义元素。

这个东西就可以称为组件了，它跟外界是完全隔离的，外界只要把它拿来就可以用，就像用原生元素一样，用选择器选择，设置属性，调用方法，添加事件处理等等，而且，注意到没有，它的属性是带get和set的，这是多么梦寐以求的东西！

正是因为它这么好用，所以在那个时代，我们用它干了很多东西，封装了各种基础控件，比如树，数据表格，日期选择，等等，甚至当时也有人嫌弃浏览器原生select和radio不好看，用这么个东西，里面封装了图片来模拟功能，替换原生的来用。

当时也有人，比如我在04年就想过，能不能把这些扩大化，扩展到除了基础控件之外的地方，把业务的组件也这么搞一下，一切皆组件，多好？

但有些事情我直到后来很久以后才想明白，基于业务的端到端组件虽然写起来很方便，却是有致命缺陷的。

到这里为止，对HTML Components的回顾告一段落，也不讨论它为什么就没了之类，这里面争议太大，我只想谈谈从这里面，能看到Web Components这么个大家寄予厚望的新标准需要面对一些什么问题。

Web Components的挑战

以下逐条列出，挨个说明，有的已经有了，有的差一些，有的没有，不管这么多，总之谈谈我心目中的这个东西应当是怎样的。

自定义元素标签支持命名空间

原因我前面已经说了，可能会有不同组织实现同类功能的组件，存在于同一个页面内，引起命名歧义，所以我想了很久，还是觉得有前缀比较好：

甚至，这里的前缀还可以是个简称别名，比如yours=com.aaa.productA，这可能只有复杂到一定程度才会出现，大家不要以为这太夸张，但总有一天Web体系能构建超大型软件，到那时候你就知道是不是可能了。

样式的局部作用域

这个前一段时间有的浏览器实现过，在组件内部，style上加一个scoped属性，这是正确的

方向。为什么要这么干呢，所谓组件，引入成本越小越好，在无约定的情况下都能引入，不造成问题，那是最佳的结果。

如果你一个组件的样式不是局部的，很可能就跟主界面的冲突了，就算你不跟主界面的冲突，怎么保证不跟主界面中包含的其他组件的样式冲突？靠命名约定是不现实的，看长远一些，等你的系统够大，这就是大问题了。

跟主文档的通讯

一个自定义组件，应当能够跟主文档进行通讯，这个过程包括两个方向，分别可以有多种不同的方式。

从内向外

除了事件，真没有什么好办法可以做这个方向的通讯，但事件也可以有两种定义方式，一种是类似onclick那种，主文档应当能够在它上面直接添加对 应的事件监听函数，就像对原生元素那样，每个事件都能单独使用。另一种是像postMessage那样，只提供一个通道，具体怎么处理，自己去定义消息格 式和处理方式。

这两种实现方式都可行，后者比较偷懒，但也够用了，前者也没有明显优势。

从外向内

这个也可以有两种方式，一种是组件对外暴露属性或者方法，让主文档调用，一种是外部也通过postMessage往里传。前者用起来会比较方便，后者也能凑合用用。

所以，如果特别偷懒，这个组件就变得像一个iframe那样，跟外部基本都通过postMessage交互。

JavaScript

写到这里我是很纠结的，因为终于来到争议最大的地方了。按照很多人的思路，我这里应该也写隔离成局部作用域的JavaScript才对，但真不行，我们可以先假设组件内部的所有JavaScript都跑在局部作用域，它不能访问主文档中的对象。

我这里解释一下之前那个坑，为什么端到端组件是有缺陷的。

先解释什么叫端到端组件。比如说，我有这么一个组件，它封装了对后端某接口的调用，还有自身的一些展示处理，跟外界通过事件通信。它整个是不需要依赖别人的，初始加载数据都是自己内部做，别人要用它也很简单，直接拿来放在页面里就可以了。

照理说，这东西应当非常好才对，使用起来这么方便，到底哪里不对？我来举个场景。

在页面上同时存在这个组件的多个实例，每个组件都去加载了初始数据，假设它们是不带参数的，每个组件加载的数据都一样，这里是不是就有浪费的请求了？有人可能觉得一点点浪费不算问题，那么继续。

假设这个组件就是一个很普通的下拉列表，用于选取人员的职业，初始可能有医生，教师，警察等等，我把这个组件直接放在界面上，它一出现，就自己去加 载了所需的列表信息并且展

示了。有另外一个配置界面，用于配置这些职业信息，这时候我在里面添加了一个护士，并且提交了。假设为了数据一致性，我们把这个 变更推回到页面，麻烦就出现了。

界面只有一个职业下拉列表的时候可能还好办，有多个的时候，这个更新的策略就有问题了。

如果在组件的内部做这个推送的对接，就会出现要推送多份一致的数据给组件的不同实例的问题。如果把这个放在外面，那我们也有两种方式：

订阅发布模式，组件订阅某个数据源，数据源跟服务端对接，当数据变更的时候，发给每个订阅者观察者模式，组件观察某个数据源，当数据变更的时候，去取回来

这两种很类似，不管哪种，都面临一个问题：

数据源放在哪？

很明显不能放在组件内部了，只能放在某个“全局”的地方，但刚才我们假设的是，组件内部的JavaScript代码不能访问外界的对象，所以……

但要是让它能访问，组件的隔离机制等于白搭。最好的方式，也许是两种都支持，默认是局部作用域，另外专门有一个作用域放给JS框架之类的东西用，但浏览器实现的难度可能就大了不少。

可能有人会说，你怎么把问题搞这么复杂，用这么BT的场景来给我们美好的未来出难题。我觉得问题总是要面对的，能在做出来之前就面对问题，结果应该会好一些。

我注意观察了很多朋友对Web Components的态度，大部分都是完全叫好，但其中有一部分，主要是搞前端MV*的同学对它的态度很保守，主要原因应该是我说的这几点。因为这个群体主要都在做单页型的应用，这个里面会遇到的问题是跟传统前端不同的。

那么，比如Angular，或者React，它们跟Web Components的协作点在哪里呢？我个人觉得是把引擎保留下来，上层部分逐步跟Web Components融合，所以它们不是谁吃掉谁的问题，而是怎样去融合。最终就是在前端有两层，一层是数据和业务逻辑层，一层是偏UI的，在那个层里面，可以存在像Web Components那样的垂直切分，这样会很适宜。

最后说说自己对Polymer的意见，我的看法没有@司徒正美 那么粗暴，但我是认同他的观点的，因为Polymer的根本理念就是在做端到端组件，它会面临很多的挑战。虽然它是一个组件化框架，组件化最适宜于解决大 规模协作问题，但是如果是以走向大型单页应用这条路来看，它比Angular和React离目标的距离还远很多。