毕业论文--关于组件化开发

  相信各位前端的小伙们对于“组件”这个词一定不陌生，平时工作中也经常用到，不管是使用来自UI框架的组件，还是同事/自己编写。那么我们怎么就开始了使用“组件”了呢？关于组件，我们又有多少了解呢？以下是我自己的看法。

### 1.什么是组件化

  就前端来说，组件就是将一段UI样式和其对应的功能作为独立的整体去看待，无论这个整体放在哪里去使用，它都具有一样的功能和样式，从而实现复用，这种整体化的细想就是组件化。不难看出，组件化设计就是为了增加复用性，灵活性，提高系统设计，从而提高开发效率。（形象点理解，我觉得七巧板给我的感觉就和组件很相似）

### 2.组件化的演变

  在jQuery大热的时候，大部分的前端开发其实都是十分过程式的开发：操作DOM，发起ajax 请求，刷新数据，局部更新页面。这样的动作反反复复，甚至在同一个项目里同样的流程也许还要重复，其实jQuery 本身也有有自己模块化的设计。 频繁操作DOM，过程式的开发方式的确不怎么样。

这时开始流行MV\* ，比如MVC，前端开始学习后端的思想，讲业务逻辑，UI，功能，可以按照不同的文件去划分，结构清晰，设计明了，开发起来也不错。在这个基础上，又有了更加不错的MVVM 框架，它的出现，更加简化了前端的操作，并且将前端的UI赋予了真实意义：你所看到的任何UI，应该都对应其相应的ViewModel，即你看到的view 就是真实的数据，并且实现了双向绑定，只要数据改变，所对应的UI也改变。这的确很方便，但大部分的MVVM框架，并没有实现组件化，或者说没有很好的实现组件化，因为MVVM最大的问题就是：

1.执行效率，只要数据改变，它下面所有监测数据上绑定的UI一般都会去更新，效率很低，如果你操作频繁，很可能调了几十万遍（有可能层次太深或者监测了太多的数据变化）。

2.由于MVVM 一般需要严格的ViewModel 的作用域，因此大部分情况不支持多次绑定，或者只允许绑定一个根节点做为顶层DOM渲染，这就给组件化带来了困难（不能独立的去绑定部分UI）。 而后，在此基础上，一些新的前端框架“取其精华，去其糟粕”，开始大力推广前端组件化的开发方式，从这一点来说，React 和Vue 是类似的。 但从框架本身来说，React 和Vue 是完全不同的，前者是单向数据流管理设计的先驱（action->request, reducer->controller ）；而后者则是后起之秀，既吸取了React 的数据流管理方式（Vue 本身也可以用类似React 的方式去开发，但难度比较大而已，不是很Vue ）的设计理念，也实现了MVVM 的双向绑定和数据监控（这应该是Vue 的核心了），所以Vue是比较灵活的，可以按需扩展。

### 3.如何实现组件化（有种设计拼图的感觉）

1. 分析页面布局
2. 当我们拿到页面UI图的时候，我们可以对着UI图进行大体上的布局划分
3. 针对大区域内的内容进行功能区和内容展示区的划分（如果是一个重复结构，只要对一个进行分析划分就可以了），直到划分不了为止
4. 对已经划分出来的内容进行整合，在整体界面中对比是否在自己划分的不同布局中也存在相同的小功能或是内容展示块
5. 初步开发
6. 进行静态界面的开发（数据写死，逻辑操作可以预留方法但是先不实现具体逻辑）
7. 化繁为简
8. 按照划分出来的小块进行抽取分离成独立的功能块/内容块（此时可以开始完成具体的功能以及开始设定）
9. 划分时需要注意，有一些差异化的存在并不能作为将其划分为不同模块的标准，比如说宽度，如果两个模块功能相似，只是所占宽度不同，可以先将其划定为相同的模块，具体差异，可以由attribute/property配置来调整
10. 组件抽象
11. 虽然是拆分了，但是模块的通用性并不完善，这个时候应该更进一步的进行共同点的抽离和拆分以增加通用性
12. 以上其实都是基于vue等框架为基准点的描述，如果完全自己构建的话则还需要考虑这些问题
13. 组件元素的创建和挂载
14. 组件的生命周期（创建、挂载、卸载）
15. 事件（内部事件）
16. attribute/property
17. 方法（暴露给外面使用）
18. 子元素（children/加载进来的组件）
19. 状态（state）
20. config
21. 等等等等（现在想想看，框架是不是帮我们做了不少的事情？）

### 4.写在最后

也许并不需要对每一个组件都进行到这么细致的拆分设定，但任何的设计都是基于一定的复杂度才衍生出来的，其实大部分情况下这种设计都是需要将功能逻辑代码也纳入其中的，并不光只是UI（如antd, element-ui等），我这里举的例子也相对比较简单。

个人认为在一个大型前端项目中，这种组件化的抽象设计是很重要的，不仅增加了复用性提高了工作效率，从某种程度上来说也反应了程序员对业务和产品设计的理解，一旦有问题或者需要功能扩展时，你就会发现之前的设计是多么的有用 （毕竟需求总是在变，说了都是泪啊）。