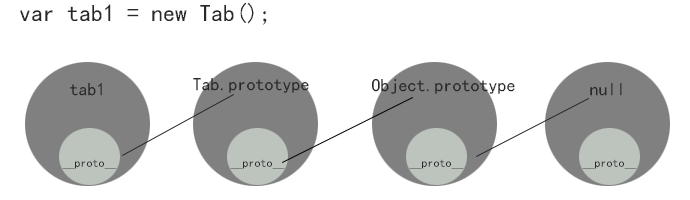
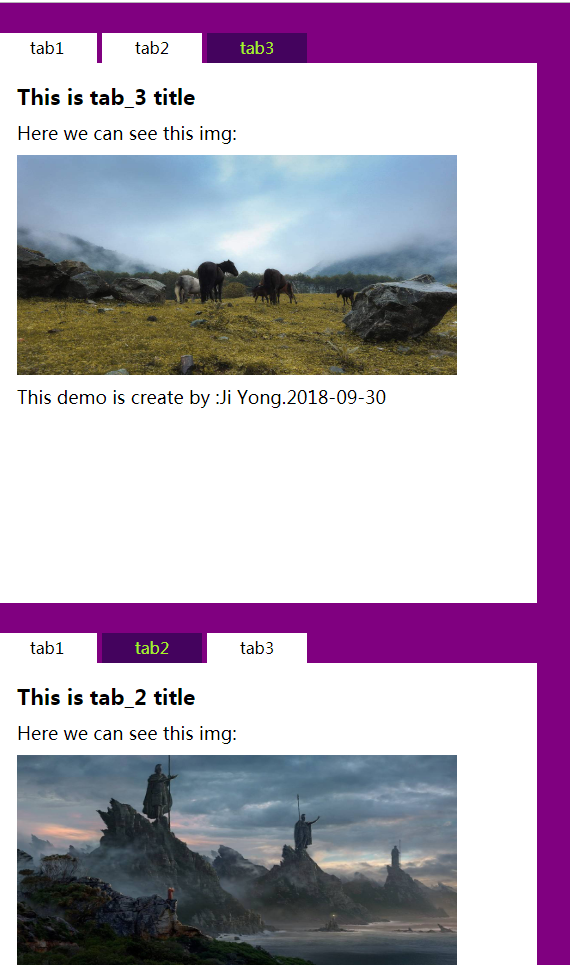
**JS TABS 组件文档说明**

概述：JS TABS DEMO主要使用JS 原型链方法实现。JS原型链上的原型是一个对象，拥有隐藏的\_\_proto\_\_属性，函数（Function）有prototype属性，继承关系如下图所描述。



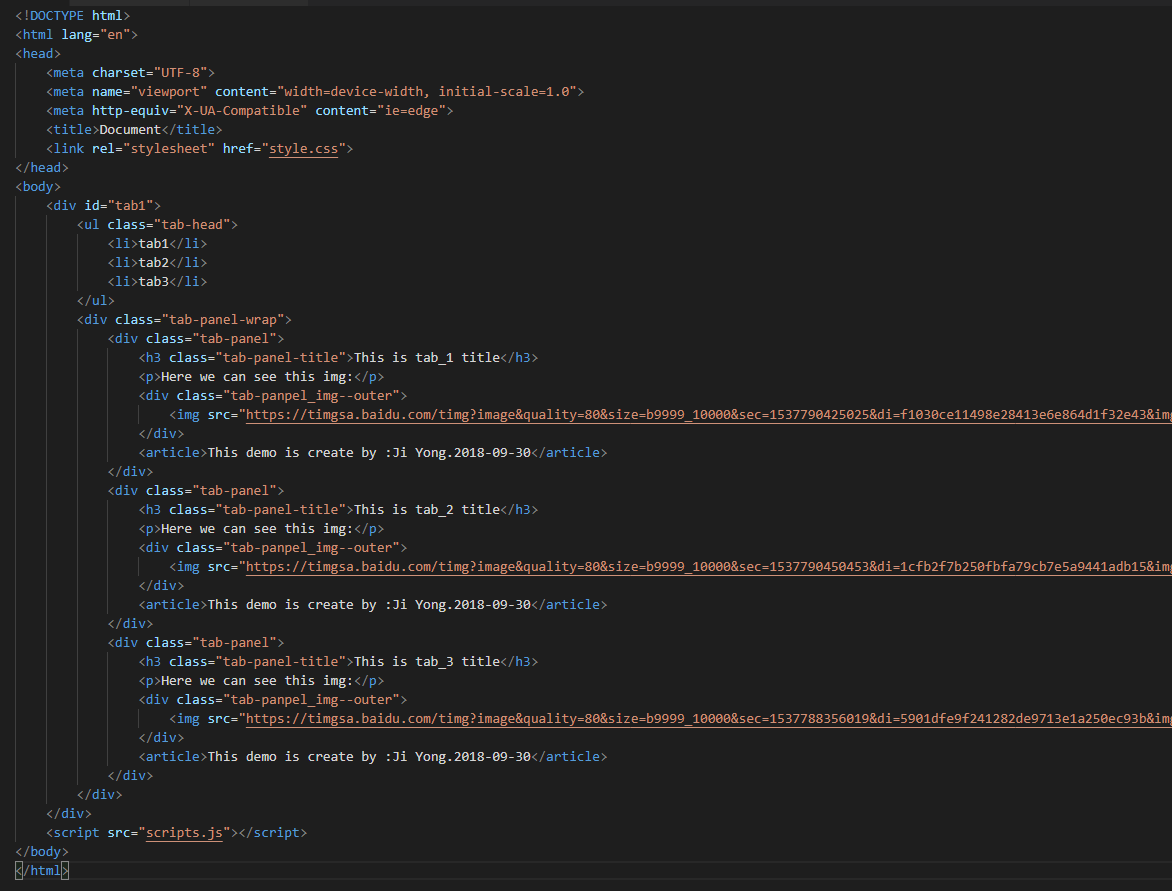
从上图可以看到，对象tab1通过\_\_proto\_\_实现了继承function Tab，并且继承自Object.prototype，最终回到原型链顶层null，形成了一条原型链。而原型链上的函数方法并不会影响外部元素，从而做到了独立性和可复用性。本次JS TABS DEMO就是用这种原型链的方法，实现组件化。

效果如下图：



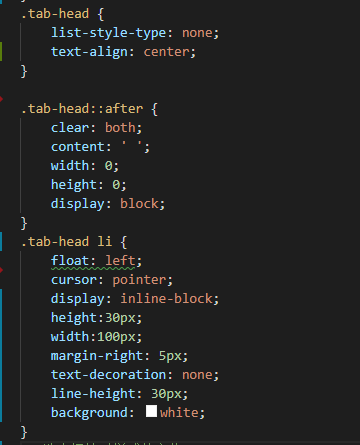
代码解释：

**Html页面结构：**

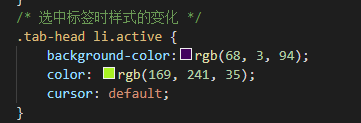


HTML页面并没有复杂的东西存在，一个单纯的JS TABS DIV组件，定义了ID方便后面引用。

**CSS页面：**



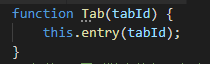
上图是TABS标签部分，采用的是浮动方法排列，并且在父元素清除了浮动，以免产生影响。



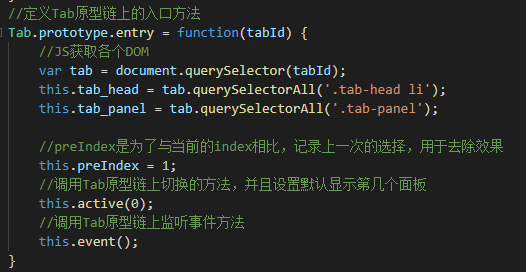


这两个样式是在标签被选中时产生，通过类的增删产生；

**JS 部分：**



这一部分定义了Tab类，并且传入参数tabId，同时向上查找原型链上的entry()方法，并且传入参数tabId。



这一部分定义了Tab原型链上entry的入口函数方法，在这一部分实现JS DOM获取操作，同时设置了preIndex（意义：上次操作记录），为的是实现后续方法的Tab切换效果。然后继续查找原型链山active()和event()方法，并且执行；



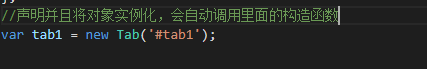
这一部分定义了Tab原型链上的active切换方法，通过判断当index(意义：当前的值)与preIndex不相等时，将Tab上对应index值的标题和面板添加预设好的active类，然后再把preIndex值对应的上次操作的标题和面板删除active类。从而实现了切换效果。

这里之所以使用preIndex这种方法，是为了避免循环遍历时产生的性能问题。以往的做法都是for循环将两个保存了DOM的数组遍历一遍，再逐个删除或者增加类，这样会影响到网页的性能问题。因此在这里改用preIndex方法。



这一部分定义了Tab原型链上的event监听事件方法，用var save = this保存了当前this的指向，是因为在addEventListener中this的指向是addEventListner的对象，因此需要提前保存。

循环块区域用的是let进行声明，因为let可以保证在块级作用域中起作用，并且在其他地方无效。如果用var的话，for循环代码块不具备块级作用域，因此i认为是全局变量，直接放在全局变量中;因此在监听事件执行环境中不存在变量i，所以此时会沿着作用域链向上寻找，即进入了全局作用域中寻找变量i，最终会达到i的最大值，因此实现不了效果。



最后声明并且将对象实例化，实现Tab组件套用。

**总结：**

主要是运用原型链的作用原理，通过继承的方式实现原型链上方法的传递，从而实现组件化。

组件是独立的功能模块，不受其他因素影响，也不会影响其他元素。一个良好的组件应该能够实现在页面上复用。原型链对模块组件化具有很好的功能，因为原型链上的方法只在原型链上传递，并不会影响到其他因素。