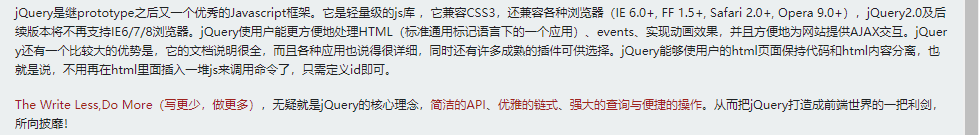
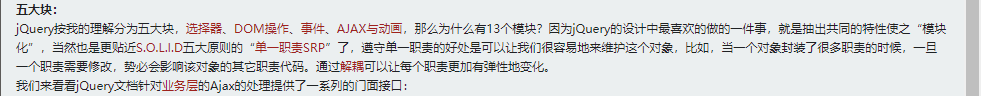
# jQuery源码解析（架构与依赖模块）

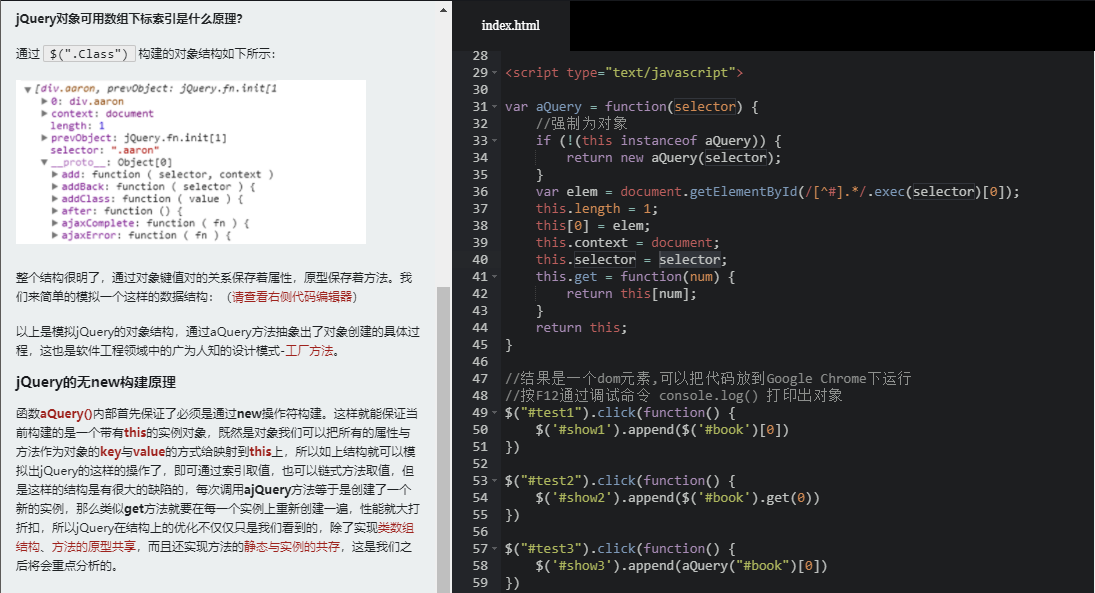
1. jQuery源码解析（架构与依赖模块）：<https://www.imooc.com/learn/172>；
2. 设计理念-写更少做更多



1. 整体架构



1. 设计模式 - 工厂方法 & jQuery的类数组对象结构



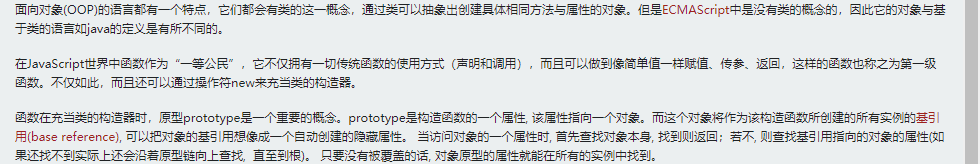
1. 匿名函数 & 自执行



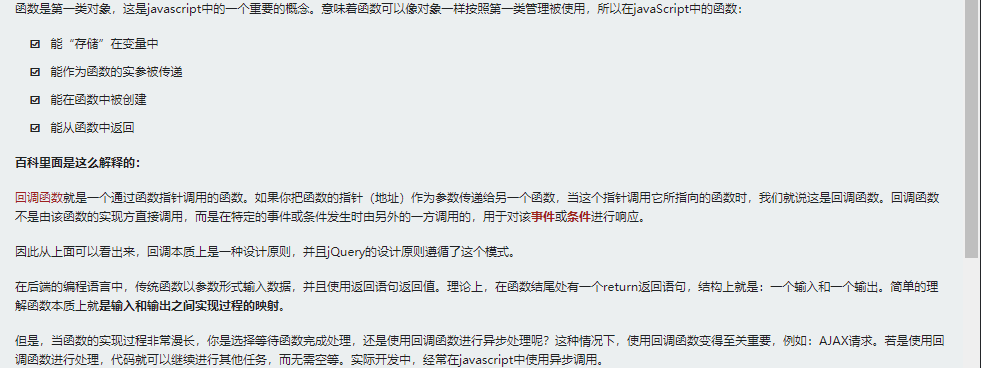
1. jQuery中ready与load事件



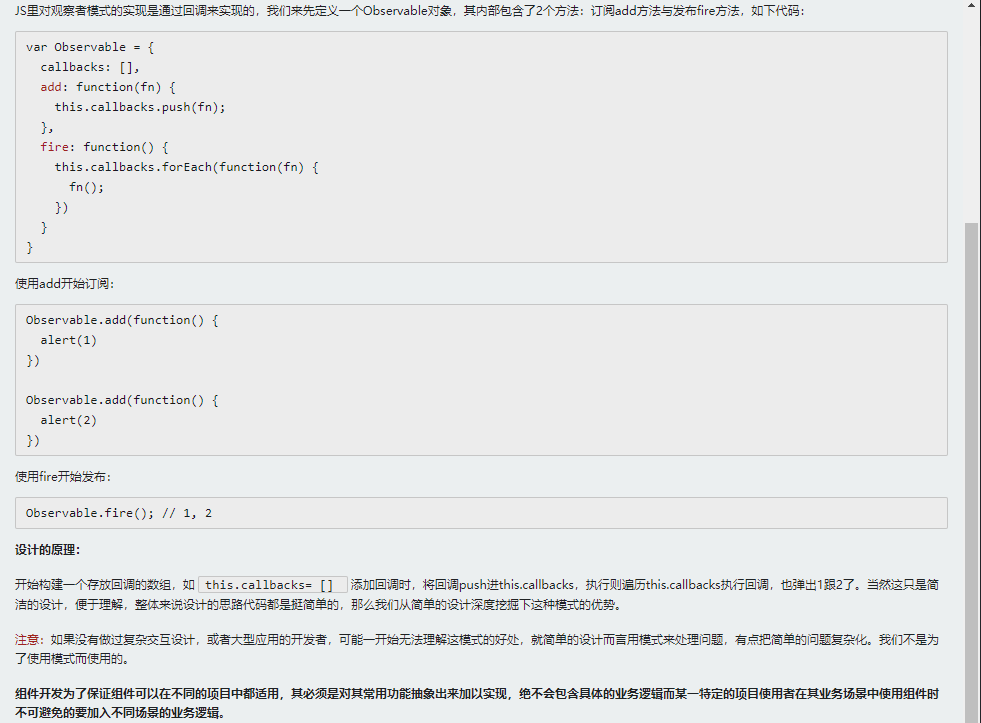
1. 对象构建

1. 回调函数

1. 设计模式 - 观察者模式



· 这种模式感觉就是说我准备了一百种菜给你，但不知道你想吃哪几种，我给个盘子给你看着夹吧，而不是全部菜都塞给你吃；

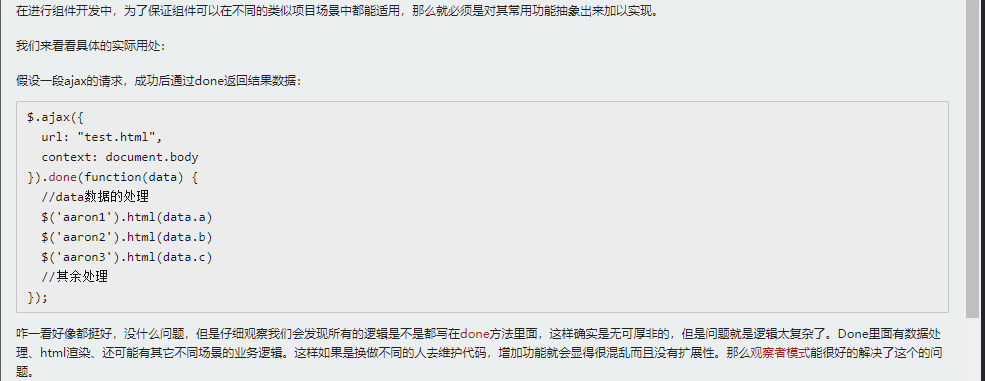
· Observable是被观察者，里面的callbacks数组中的每一个都是一个观察者，每一个元素都是函数形式；

· 将观察者都搞在一个集合中，当被观察者发生变化，就直接找到这个集合，并运行其中的函数；

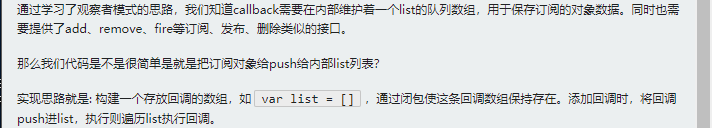
· 跟多个函数放到一个函数中一次执行有什么不同，不明白？到时是对的，但是在调用的时候，只用放一次，而且放到一个函数中这个过程是通过方法来添加的，而不是直接在代码中添加。所谓的“组件”的思想。而且，最重要的是，注意这句话：当一个对象的改变需要同时改变其它对象，并且它不知道具体有多少对象需要改变的时候，就应该考虑使用观察者模式。 如果一次把多个函数放到一个函数中，不仅仅可能造成这个函数过于庞大（比如要同时执行几十上百个函数），而且每次十分不灵巧，函数变得不够“单纯”，换句话说，太low了。。。；

· 跟多个函数放到一个函数中一次执行有什么不同，不明白？如果把多个函数直接放到一个函数中，那如果你需要新增一个函数呢，改动原先的函数？如果那段代码不是你写的，你是不是还要先去搞懂对方写的是什么意思，然后再把函数添加到相应的地方，如果使用观察者模式通过暴露一个接口让你来添加新的函数，这样不是更好；

1. 观察者模式实际应用

1. jQuery回调对象 - jQuery.Callbacks – 观察者模式在jQuery中的应用

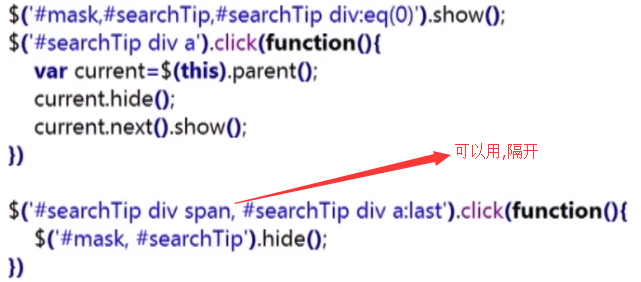


# 如何实现“新手引导”效果

1. 隐藏按钮文字的小技巧：text-indent:-999px;overflow：hidden；
2. 时间加上30天快捷方法



1. jQuery选择器



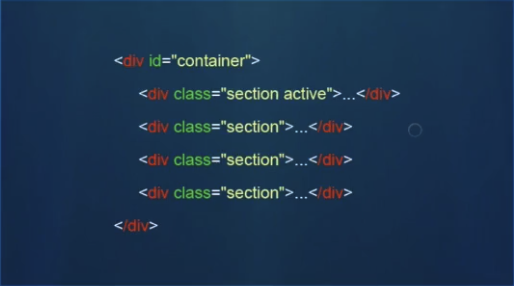
1. 先将新手引导的图片制作完成，可使用绝对定位的方式将图片显示在对应区域；

# 跨域

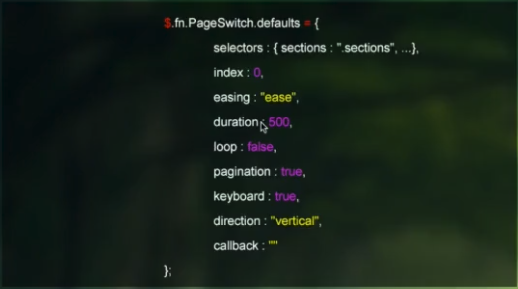
1. 跨域小结(为什么form表单提交没有跨域问题，但ajax提交有跨域问题)：http://blog.csdn.net/wabiaozia/article/details/78771709
2. 1
3. 1
4. 1
5. 1
6. 1
7. 1
8. 1

# 全屏切换效果

1. Dom结构

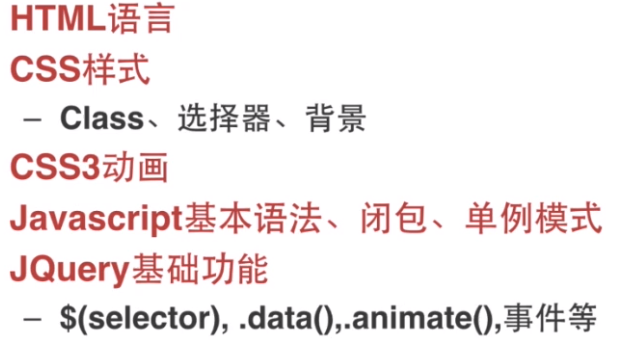


1. 插件引用



1. 预备知识：

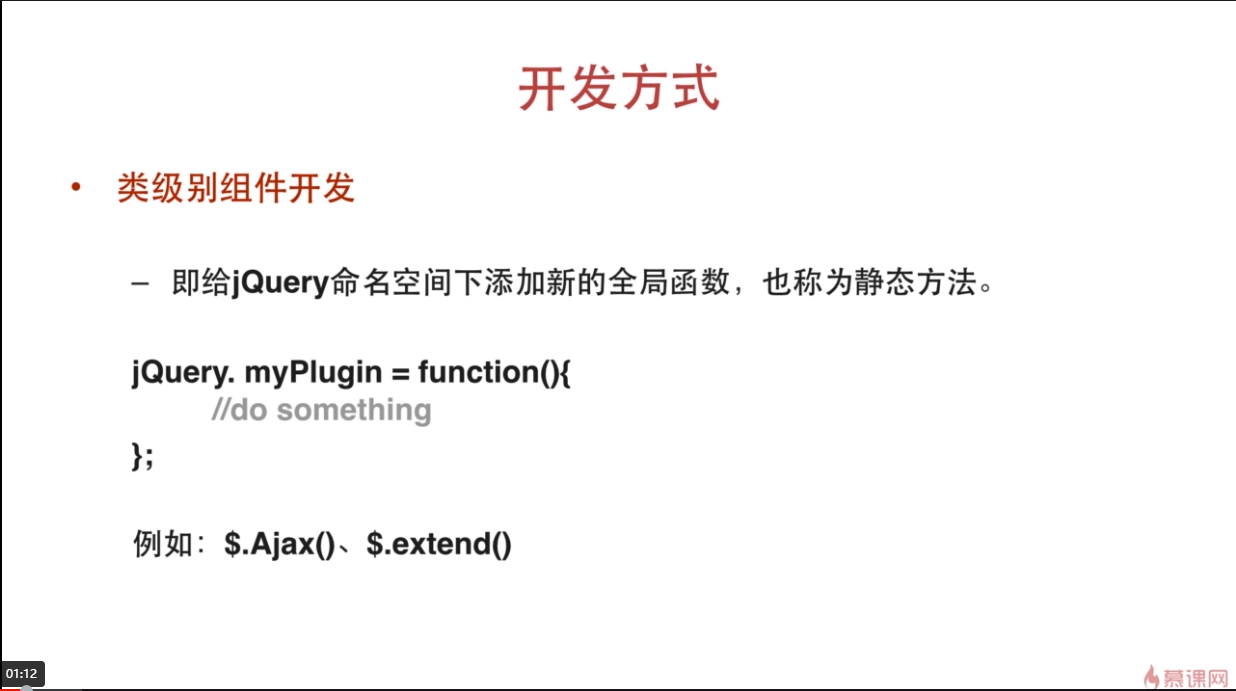
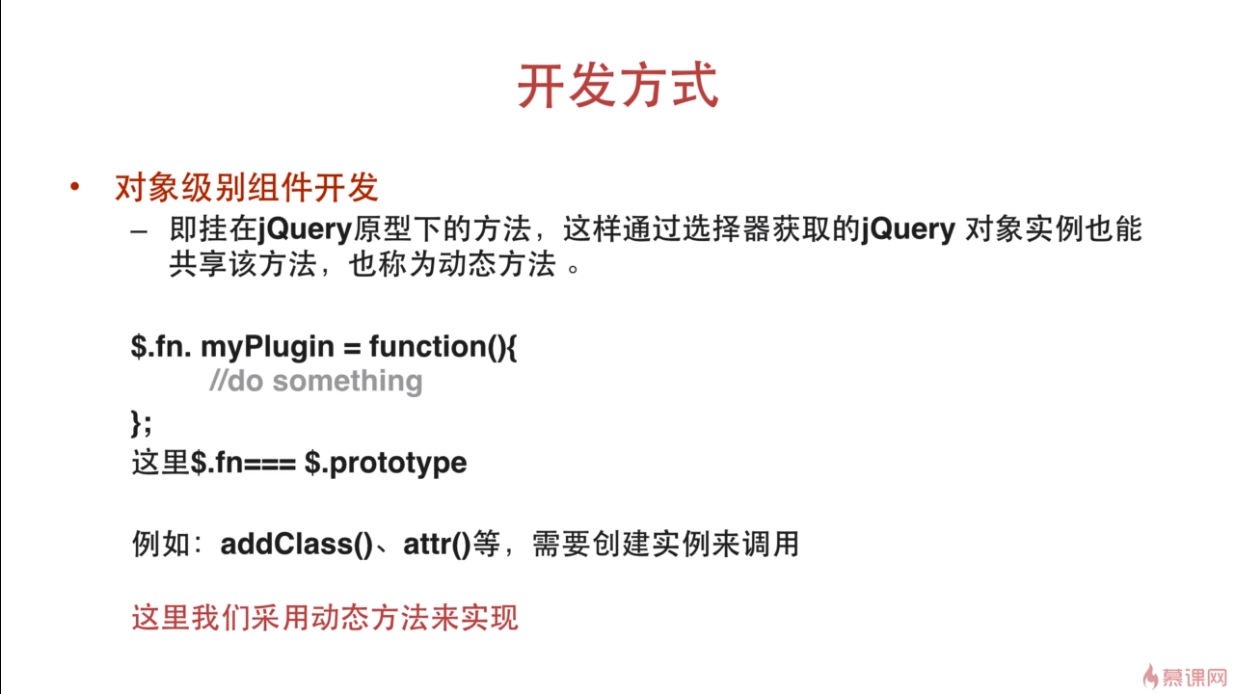
通过CSS使div实现全屏；jQuery插件框架；鼠标滚轮事件；键盘事件；实现滑动动画；

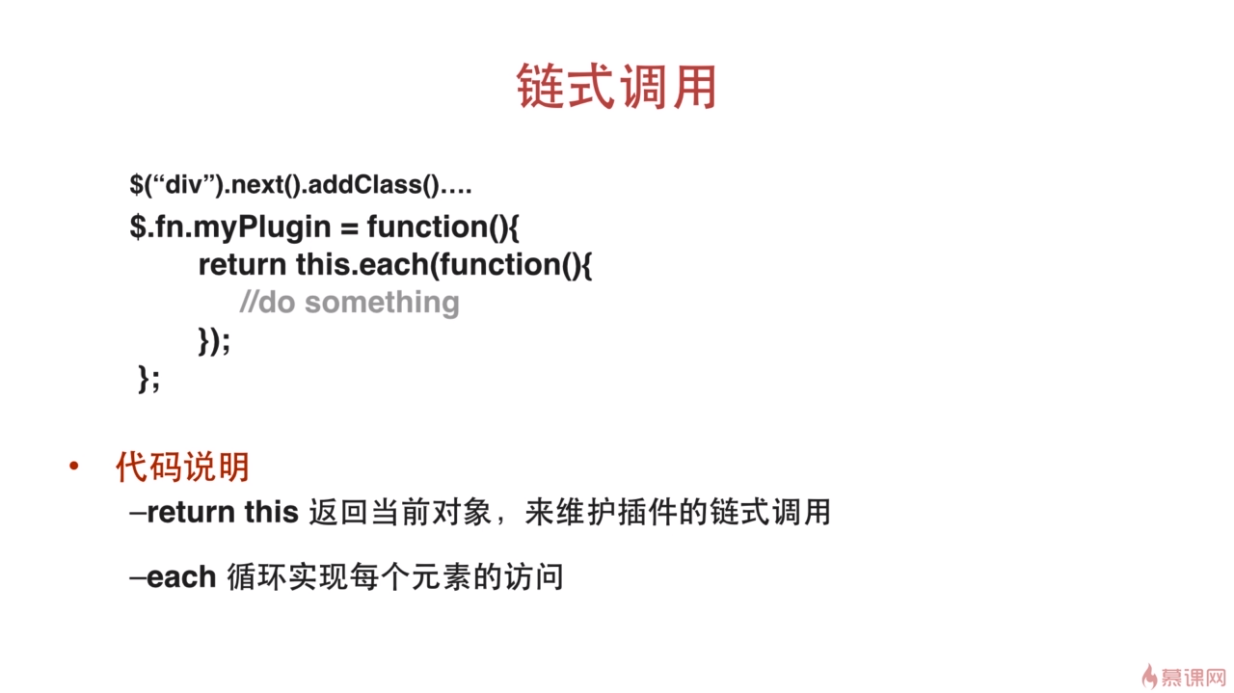


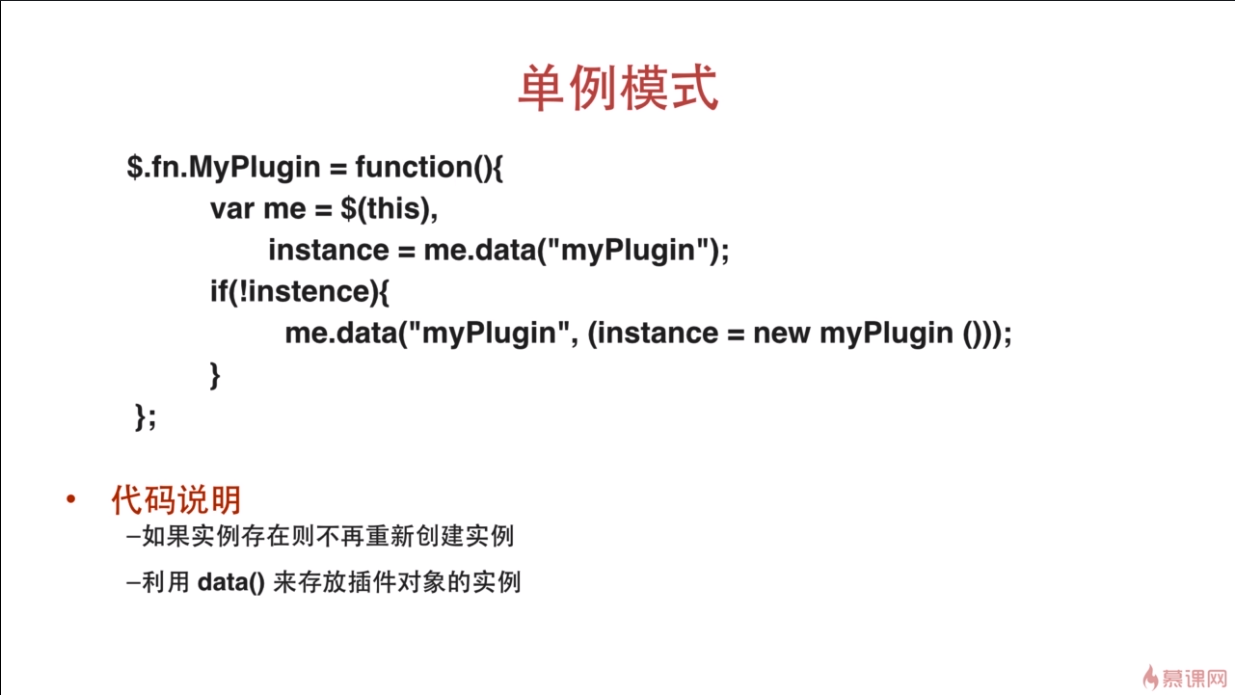
1. 闭包的作用



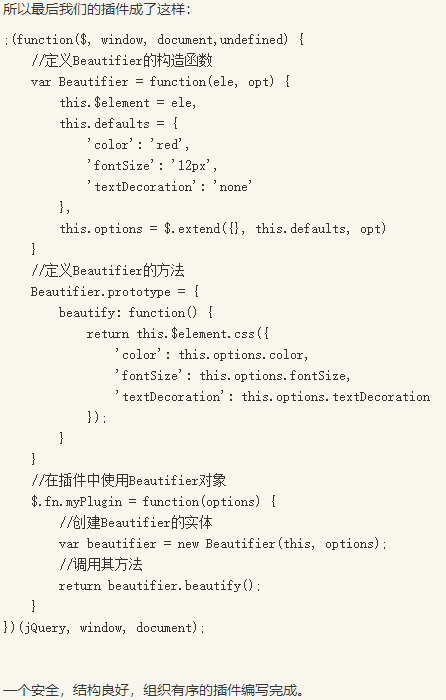
1. jQuery插件开发方式

1. jQuery链式调用
2. 单例模式



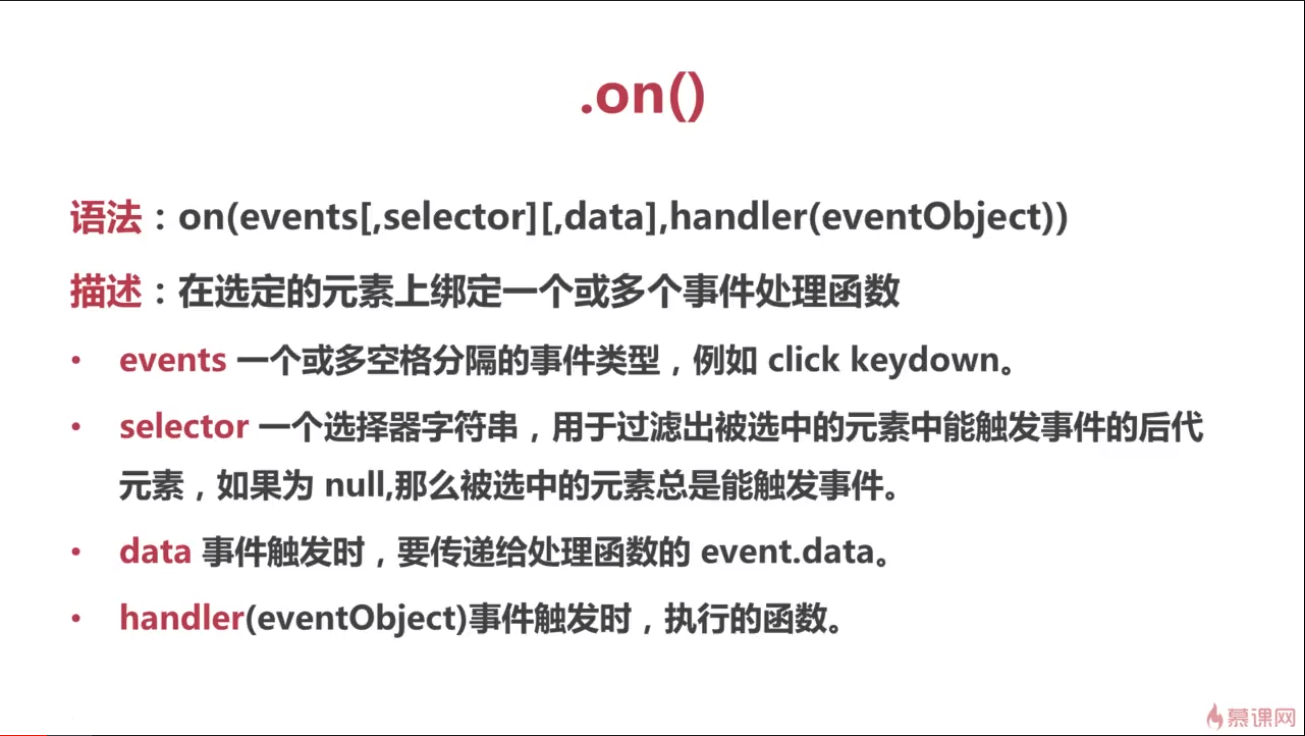
1. jQuery插件基础开发教程：<https://www.cnblogs.com/ajianbeyourself/p/5815689.html>；



1. js基础知识复习

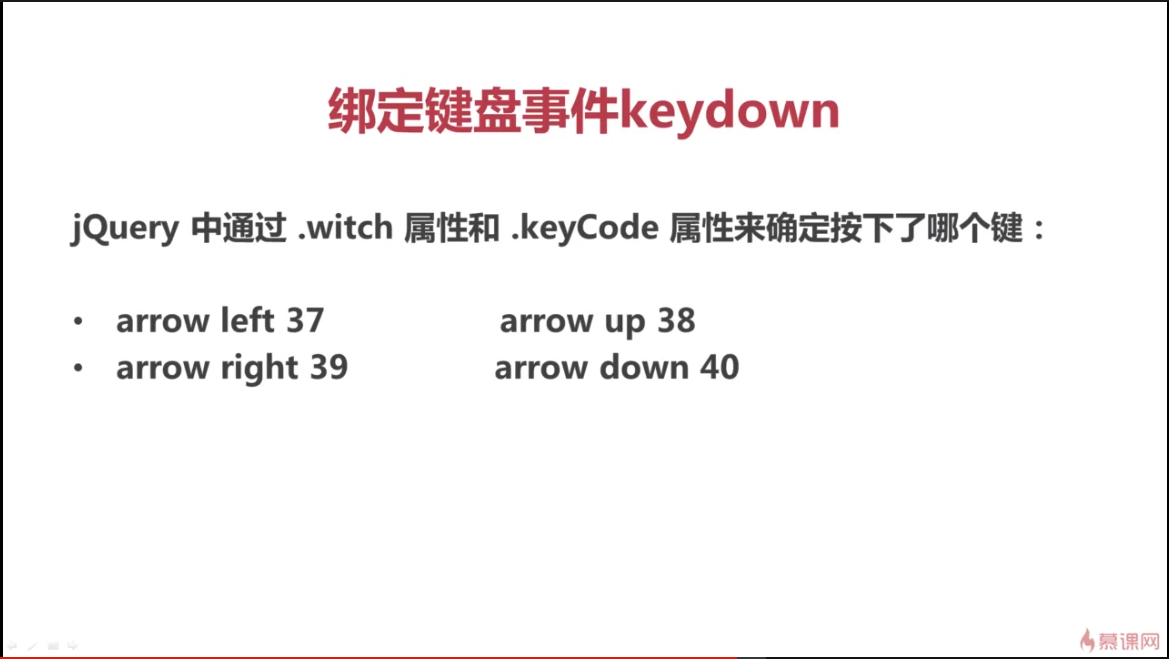
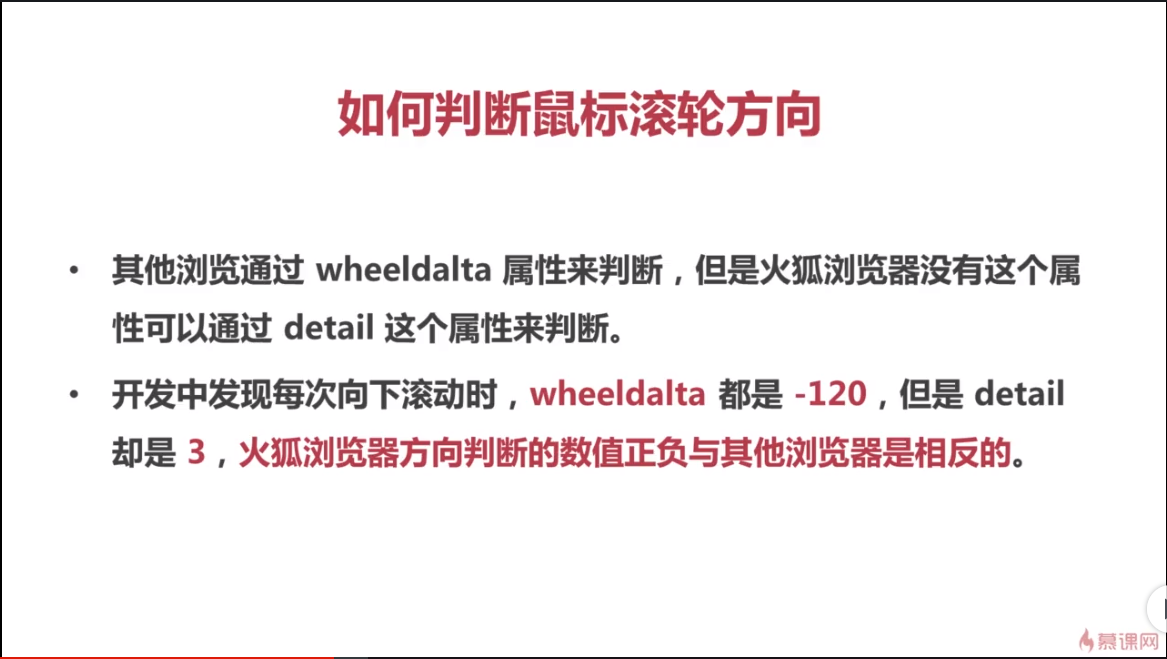


1. 事件委托：委托事件不仅可以给未创建的后代元素绑定事件外，当需要监听很多元素的时候，委托事件的开销更小；

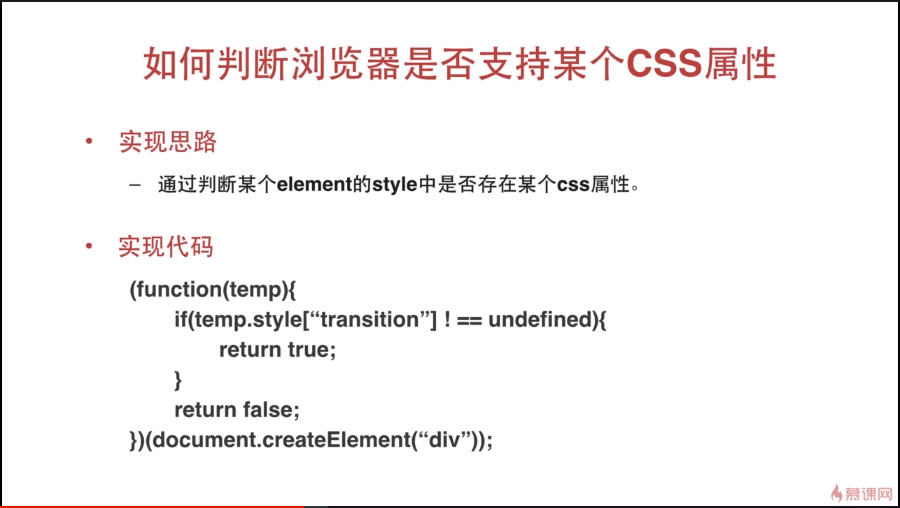


1. 鼠标滚轮事件、键盘事件





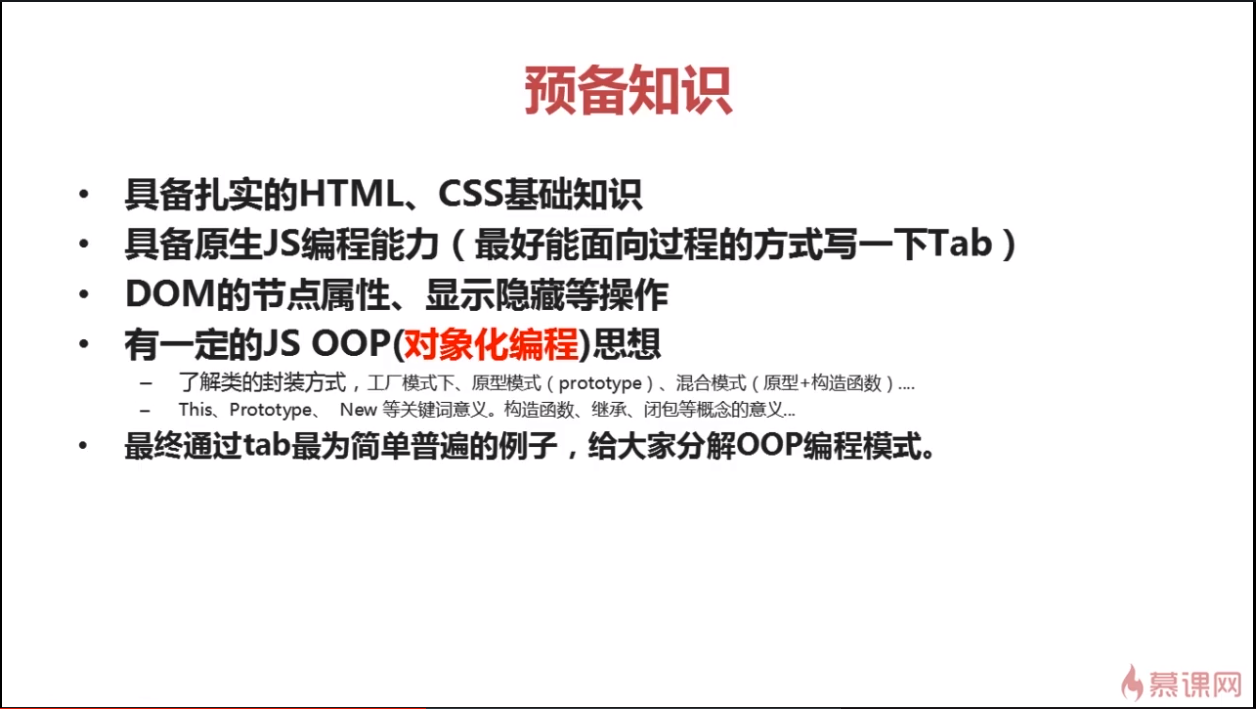
1. 如何判断浏览器是否支持某个CSS属性



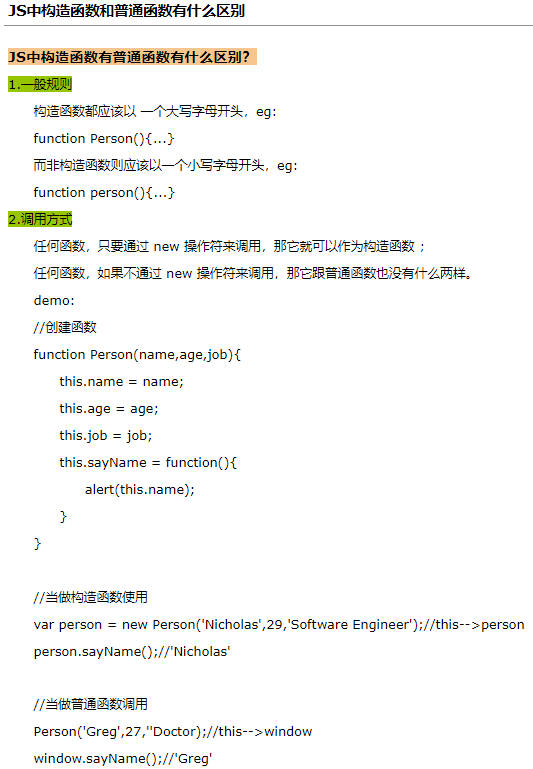
1. 对于不支持transition属性的低版本浏览器，可以通过jQuery的animate方法实现动画效果；

# JS插件开发之-Tab选项卡

1. 预备知识



1. 深入理解js构造函数：<https://www.cnblogs.com/wangyingblog/p/5583825.html>；



1. OOP（对象化编程）（Object Oriented Programming）思想；
2. OOP三大基本要素：对象获取、事件绑定、交互功能；核心思想：封装，继承，多态；
3. JavaScript构造函数及原型对象：<http://blog.csdn.net/a153375250/article/details/51083245>；



1. 1
2. 1