课程目标

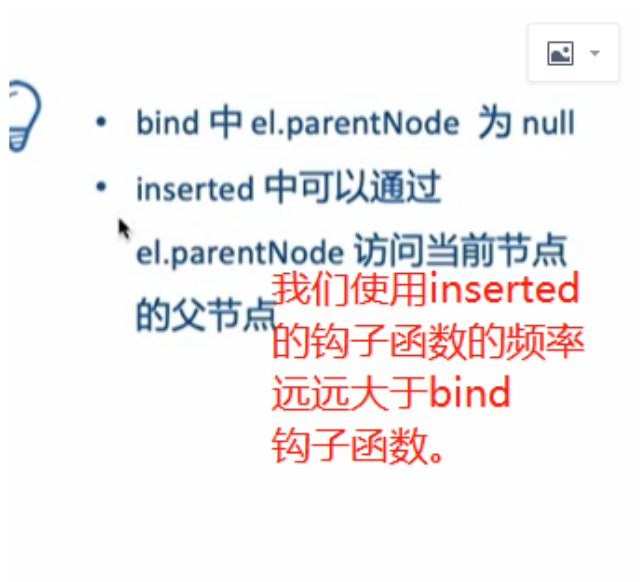
自定义指令

全局注册一个指令

Vue.directive('demo',{}),在任何地方都可以使用这个指令了。

之定义指令如何编写?

```
Vue.directive("demo", {
  // 只调用一次，指令第一次绑定到元素时调用。
  // 在这里可以进行一次性的初始化设置。
  bind: function(el, binding, vnode) {},
  // 被绑定元素插入父节点时调用
  // (仅保证父节点存在，但不一定已被插入文档中)。
  inserted: function(el, binding, vnode) {},
  // 所在组件的 VNode 更新时调用，
  // 但是可能发生在其子 VNode 更新之前。
  // 指令的值可能发生了改变，也可能没有，
  // 但是可以通过比较更新前后的值来忽略不必要的模板更新
  update: function(el, binding, vnode, oldVnode) {},
  // 指令所在组件的 VNode 及其子 VNode 全部更新后调用。
  componentUpdate: function(el, binding,vnode, oldVnode
  // 只调用一次，指令与元素解绑时调用。
  unbind: function(el, binding, vnode) {}
});
```



可以根据比较 oldVnode 和 vnode 之间的差异来判断模板是否需要更新，以减少不必要的模板更新，从而一定程度提高组件性能。

钩子函数参数

```
function(
    // 指令所绑定的元素，可以用来直接操作 DOM
    el,
    // binding一个对象，包含以下属性
    {
      // 指令名，不包括 v- 前缀。
      name,
      // 指令的绑定值，例如：v-my-directive="1 + 1" 中，
      // 绑定值为 2。
      value,
      // 指令绑定的前一个值，
      // 仅在 update 和 componentUpdated 钩子中可用。
      oldValue,
      // 字符串形式的指令表达式。
      // 例如 v-my-directive="1 + 1" 中，表达式为 "1 + 1"。
      expression,
      // 传给指令的参数，可选。
      // 例如 v-my-directive:foo 中，参数为 "foo"。
      arg,
      // 一个包含修饰符的对象。
      // 例如：v-my-directive.foo.bar 中，
      // 修饰符对象为 { foo: true, bar: true }。
      modifiers
    },
    // Vue 编译生成的虚拟节点
    vnode,
    // 上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。
    oldVnode
  )
```

除了 el 之外，其它参数都应该是只读的，切勿进行修改。如果需要在钩子之间共享数据，建议通过元素的 dataset 来进行。

什么时候用自定义指令

当我们的methods中存在操作DOM/BOM的逻辑的时候，就该考虑是否可以抽象成一个自定义指令。请看下面实战，自定义指令，根据窗口的大小，绑定某个elDOM不同操作。

```
//全局main.js 注入全局指令
Vue.directive('resize',{
 inserted(el,binding){
 const callback = binding.value // 这里的value定义的是一个函数
 const direction = binding.arg // 传给指令的参数 例如: v-resize:foo 注意是冒号后面的参数
 const modifiers = binding.modifiers // 修饰符 例如: v-resize:foo.stop 注意是foo.后面的
 const result =()=>{
 return direction ==='vertical'?window.innerHeight:window.innerWidth
 }
 const onResize = ()=>callback(result())
 // 这里需要一个
 window.addEventListener('resize',onResize)
 //前面说的el是共享可以修改的 其他的都是只读的,所以我们可以把数据共享到el上
 el._onResize = onResize
 // .quiet 来控制是否在 指令初始化的时候 响应onResize函数
 if(!modifiers||!modifiers.quiet){
 onResize()
 }
 },
 unbind(el){ //解除resize绑定
 if(!el._onResize) return
 window.removeEventListener('resize',el._onResize)
 //删除相关属性 释放内存
 delete el._onResize
 }


})
```

```
//相关vue组件调用
<template>
 <!-- 1. v-resize 指令， 监听浏览器窗口大小改变的时候， 通过监听函数 onResize 响应-->
 <!-- <div v-resize="onResize">window width is: {{ length }}</div> -->
```

```
  <!-- 2. 可通过 direction，控制监听页面高度 或者 宽度的变化 -->
  <div v-resize.quiet="onResize">window Height is:  {{ length }}</div>
  <!-- 3. 可通过 修饰符 .quiet 来控制是否在 指令初始化的时候 响应onResize函数 -->
  <!-- <div v-resize.quiet="onResize">window width is:  {{ length }}</div> -->
 <!-- <div v-resize:[direction]="onResize">window width is:  {{ length }}</div>-->
 </template>

 <script>
  export default {
  name: "DirectivePage",
  data(){
  return {
  direction:"vertical",
  flag:false,
  length:0
  }
  },
  methods:{
  onResize(length){
  if(length>900){
  this.flag = true
  }
  this.length = length
  }
  }
  }
 </script>
```