我为大家准备了: 微信小程序——“WEB问题速查”——每天必须跟着看

1. 历届师兄师姐的问题清单: 小程序->首页->VUE->day01 点击右侧列表中的问题，就可弹出窗口，查看问题的详情

2. 当天重点视频片段: 小程序->在线->VUE -> day01

3. 当天的单词列表: 小程序->单词-> VUE -> day01

如果tmooc上的视频有问题看不了，可以直接看小程序中的视频

小程序中内容到鄙视面试前看完就行！不用现在都看完！

关于提问: (剩5天)直接私信我是最快的解决途径——谁问的问题越多，谁的培训费交的越值

1. 上课时哪个问题没听明白，立刻问

2. 笔记中哪句话/哪个图不明白，立刻问

3. 小程序中问题清单里哪个问题看不懂，立刻问

4. 如果个人项目中功能不会做，立刻问

5. 刷笔试题时，遇到不会的笔试题，立刻问

约定: 每个重要的案例，至少抄三遍，直到自己可以写出注释才算自己会。（个别代码写不出来，都没关系！）

正课:

1. 什么是VUE

2. 如何使用VUE

3. MVVM

4. 绑定语法

一. 什么是Vue:

1. 什么是Vue: 第三方开发的基于MVVM设计模式的渐进式的纯前端js框架

(1). 第三方开发: 下载 cn.vuejs.org

(2). 基于MVVM设计模式? 待定

(3). 渐进式: 可以逐步在项目中引入vue的功能。不必整个项目通篇使用vue。

(4). 纯前端js: 无需任何后端知识和技术，vue就可在客户端浏览器中独立使用和运行！

(5). 框架: 已经包含核心功能的半成品代码

2. 为什么框架: 简单——避免大量重复的代码！

3. 何时: 只要以数据操作（增删改查）为主的前端项目，都可用框架开发！

二. 如何使用vue: 我的第一个vue程序

1. 下载: 2种:

(1). 只下载一个独立的vue.js文件，引入网页中使用——只适合于初学者(2天半)

最高版本: 2.6

开发版: 具有完备的注释、代码格式和见名知义的变量名

带有极其友好的错误提示！便于调试！

可读性好，用于开发和学习之用

体积大，下载慢，不便于生产环境(项目开发测试完，上线之后)使用

生产版: 删除一切注释和代码格式，极简化了变量名

删除了错误提示，不便于调试！

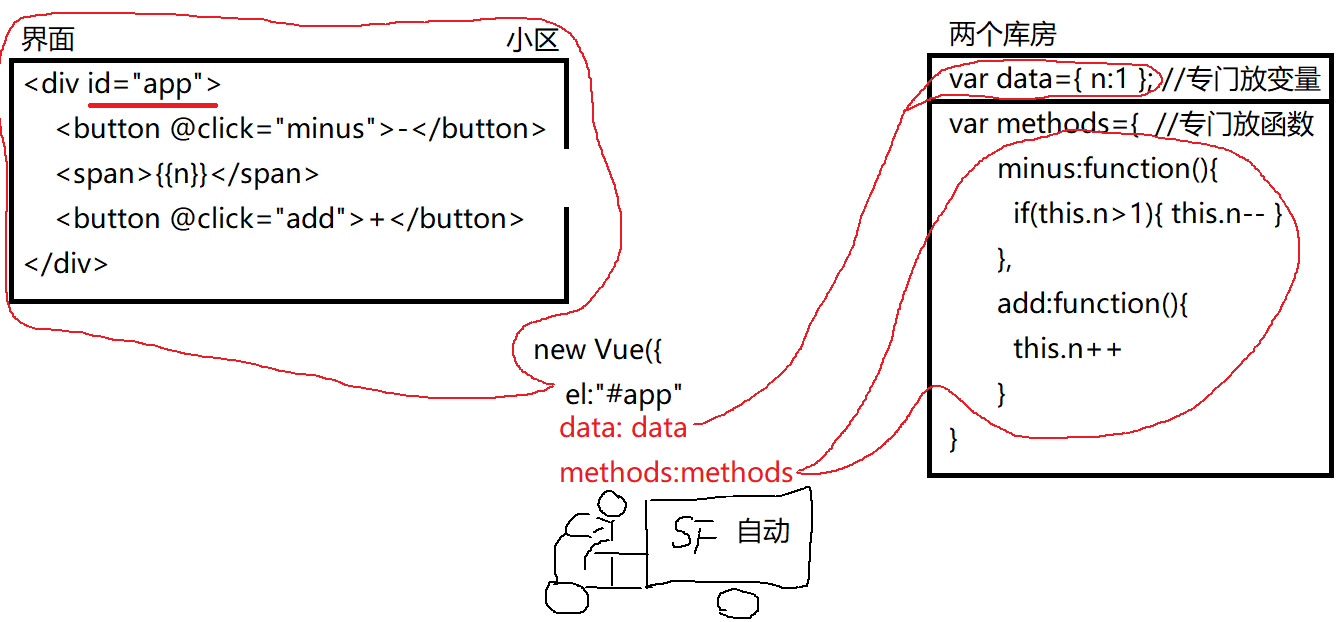
体积小，下载快，用于生产环境中

可读性差，不适合于开发和学习之用

(2). 下载vue项目的脚手架代码——企业中使用(2天)

最高版本: 4.x

2. 我的第一个vue程序: 3步



(1). 定义界面: 3个要求:

a. 整个界面必须包裹在唯一的一个父元素内，习惯上唯一父元素命名为id="app"

b. 找到整个界面中可能发生变化的位置，用{{变量名}}标记出来！

本例中: 因为span的内容在点击时可能发生数量变化，所以: span的内用改为{{n}}

c. 找到触发事件的元素，在元素开始标签中绑定事件处理函数: @事件名="函数名"

(2). 定义data对象和methods对象分别保存界面所需的所有变量和所有事件处理函数

var data={ 页面所需的变量: 初始值 }

var methods={

事件处理函数名(){

//要想操作data中的变量值，必须this.变量名才行！

}

}

问题: data中的变量和methods中的方法不会自己长腿跑到页面上指定位置去

(3). 创建一个new Vue()对象，将要操作的界面和data和methods包裹在一起。

结果: new Vue()会自动将data中的变量和methods中的方法送到界面中指定的位置上。

(4). 示例:

1\_first.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <script src="js/jquery-1.11.3.js"></script>  </head>  <body>    <button>-</button><span>1</span><button>+</button>    <script>      //DOM 4步      //1. 查找触发事件的元素      $("button:contains(+)")      //2. 绑定事件处理函数      .click(function(){        //3. 查找要修改的元素        //本例中: 改span        var $span=$(this).prev();        //4. 修改元素        //先取出span的内容，转为数字        var n=parseInt($span.html());        //数字+1        n++;        //再将数字放回$span内容中        $span.html(n)      });      //1. 查找触发事件的元素      $("button:contains(-)")      //2. 绑定事件处理函数      .click(function(){        //3. 查找要修改的元素        //本例中: 改span        var $span=$(this).next();        //4. 修改元素        //先取出span的内容，转为数字        var n=parseInt($span.html());        //如果n>1，才能-1        if(n>1){ n-- }        //再将数字放回$span内容中        $span.html(n)      })    </script>  </body>  </html>  运行效果： |

2\_vue.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <script src="js/vue.js"></script>  </head>  <body>    <!--1. 先定义界面，要求:    1.1整个界面必须包裹在唯一的一个父元素内，习惯上唯一父元素命名为id="app"    1.2找到整个界面中可能发生变化的位置，用{{变量名}}标记出来！   因为span的内容在点击时可能发生数量变化，所以:    1.3找到触发事件的元素，在元素开始标签中绑定事件处理函数:      @事件名="函数名"-->    <div id="app">      <button @click="minus">-</button><span>{{n}}</span><button @click="add">+</button>    </div>    <script>    /\*2. 定义专门保存页面所需的变量和事件处理函数的对象      //本例中: 页面上一共需要一个变量n和两个事件处理函数(minus和add)，所以需要:      2.1 创建一个对象"data"，集中保存变量n以及n的初始值      2.2 再创建一个对象"methods", 集中保存页面所需的所有事件处理函数    \*/      //不要改名！      var data={ n:1 };      var methods={        //methods中的方法要想操作data中的变量，必须加this.!        //methods中的方法只要专心操作变量值即可！不用关心页面如何增删改查！        minus:function(){          if(this.n>1){ this.n-- }        },        add:function(){          this.n++        }      }    /\*创建一个快递员对象,告知快递员库房在哪儿，告知快递员专门负责哪个小区的住户！\*/      new Vue({        el:"#app", //只负责id为app的小区，及其子内容        data:data, //小区住户们所需的变量存在哪儿        methods:methods //住户们所需的事件处理函数存在哪儿      })    </script>  </body>  </html>  运行效果: |

三. MVVM

1. 旧前端代码三大部分划分:

(1)HTML 只负责保存页面的静态内容

(2)CSS 只负责保存页面所需的静态css样式

(3)JS 负责所有页面上的增删改查操作！

2. 问题:

(1). HTML和css因为缺少程序必须的要素(变量、分支、循环)，就好像生活不能自理的小婴儿！

(2). 所有增删改查操作，都要js程序来负责，导致大量重复的代码！

3. MVVM设计模式: 对前端代码重新划分: 还是三部分

(1). 界面(View): 增强版的:

a. 等于HTML+CSS

b. HTML增加了变量、分支、循环等程序功能

(2). 模型对象(Model): 专门保存页面所需的变量和事件处理函数的对象

(3). 视图模型(ViewModel)对象: 专门负责自动将变量和事件处理函数送到界面中指定位置——将界面和模型对象绑定在了一起。

4. MVVM最大的优点: 因为ViewModel可自动维护模型对象内容与界面之间保持一致，所以减少了大量重复的编码！

5. Vue框架正是基于MVVM设计模式实现的！

6. Vue的绑定原理: 其实就是new Vue的原理:

(1). 先创建一个vue类型的子对象

a. 引入data对象:

1). 自动为data对象中每个变量都请了保镖(访问器属性)！

结果: 任何位置访问data中的变量时，其实根本就不是直接访问data中变量，仅仅访问的是为data中的变量请的同名保镖而已

结果: 只要外界试图修改data中的变量时，注定会触发保镖的set()方法。

2). new Vue()还自动在每个变量的set()函数内植入了一个通知函数

结果: 只要外界试图修改data中的变量时，注定会触发保镖的set()方法中的通知函数。整个vue框架就知道一个变量的值被修改了！

b. 引入并打散methods对象: 结果：

1). Methods中所有函数默认直接隶属于new Vue()对象

2). 导致methods中的函数进入new Vue()后和data中变量的保镖(访问器属性)平级

3). 所以，methods中的方法想访问data中的变量，必须加this.变量名

(2). 构建虚拟DOM树:

a. 什么是虚拟DOM树: 通过扫描完整版DOM树创建出的一棵新DOM树

b. 过程:

1). new Vue()会查找到el属性指定的界面上的父元素

2). 扫描找到的父元素及其子元素

3). 碰到"@事件名=函数名"，就自动翻译为on事件名=函数名——自动绑定事件

4). 碰到所有"{{变量名}}"：

i. 先将可能变化的元素对象加入到虚拟DOM树中

ii. 并且首次替换元素内容中{{变量名}}为变量的初始值

(3). 结果: 每次只要data中的变量被修改时:

a. 自动调用变量的set()函数

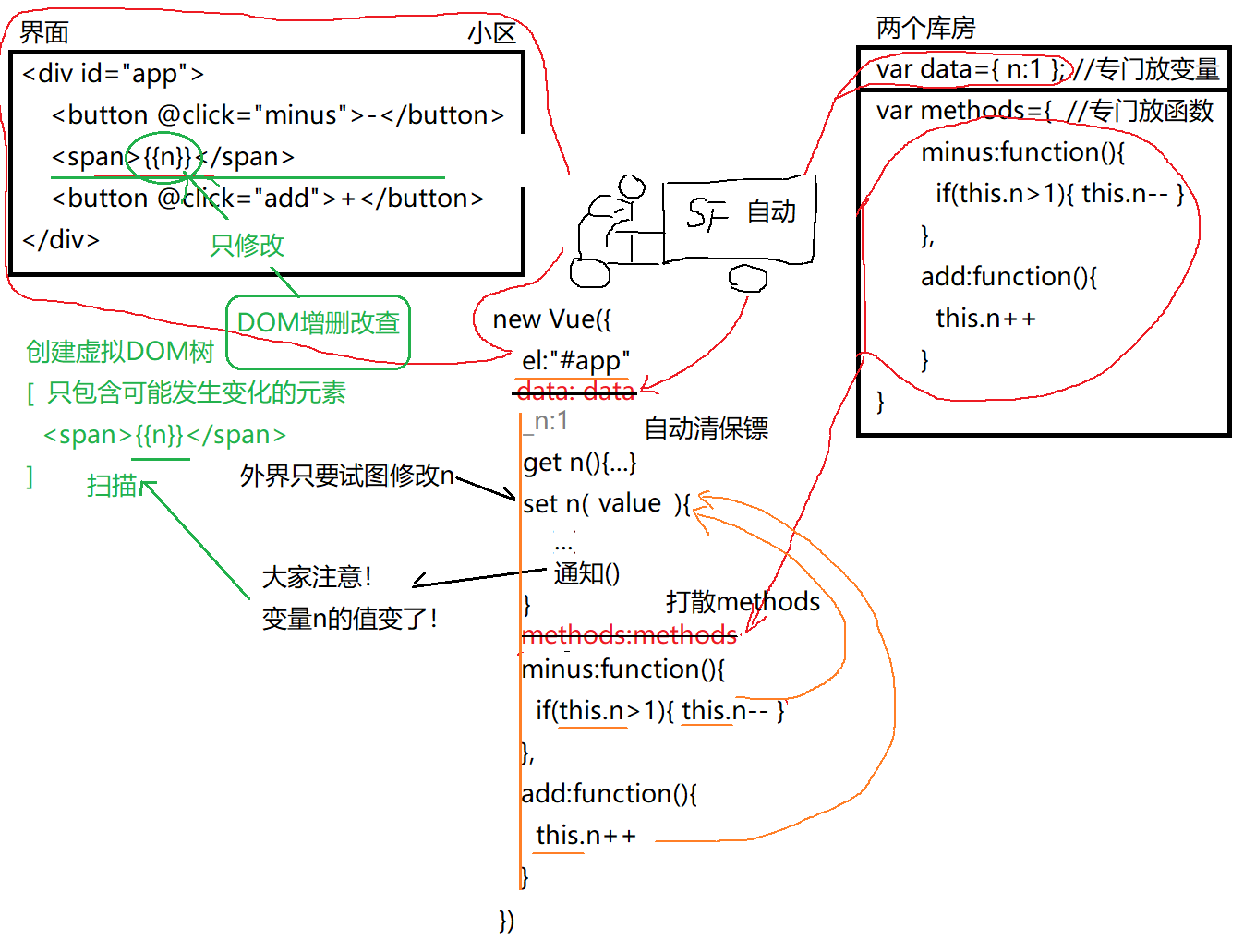
b. set()中内置的通知函数向vue框架发出通知，说xx变量发生了改变

c. 框架会自动扫描虚拟DOM树

d. 查找受本次变量修改影响的所有DOM元素

e. 自动利用已经提前封装好的DOM增删改查操作，只更新页面上受影响的元素。其他不受该变量影响的元素，保持不变！

|  |
| --- |
| 总结:  1. VUE绑定原理: 访问器属性+虚拟DOM树  （学有余力）访问器属性+观察者模式+虚拟DOM  2. 虚拟DOM树的优点: 4点:  (1). 内容少！仅保存可能变化的元素。不变的元素一个都不包含  (2). 遍历快！  (3). 避免大量重复代码！——自动执行DOM增删改查操作  (4). 效率高！每次变量更改时，只修改个别受影响的元素，其他元素保持不变！ |



四. 绑定语法:

1. 什么是绑定语法: Vue中在HTML内标记哪里可能发生变化的特殊语法

2. 何时: 只要HTML中一个元素的内容可能随变量自动变化，就要用绑定语法

3. 如何: <元素>xxx{{变量名}}xxx</元素>

4. 结果:

(1). 在创建虚拟DOM树时，该元素就会被自动加入到虚拟DOM树中

(2). 一旦new Vue()中{{}}中同名的变量发生变化，则new Vue()对象会自动将该变量的新值跟新到HTML中{{变量名}}位置替换——不需要我们做任何增删盖查操作

5. {{}}中: 和ES6的模板字符串${}要求是一样的！

(1)可以放: 变量，算数计算，三目，有返回值的函数调用，创建对象，访问数组元素——所有有返回值的合法的js表达式

(2)不能放: 分支、循环等程序结果，以及其他没有返回值的js表达式！

作业:

1. 复习今日问题清单(VUE->day01)，预习明日问题清单(VUE->day01~day02)

2. （学有余力）看小程序视频学习观察者模式

小程序->在线->vue->day02 高频笔试题：观察者模式