正课:

1. 什么是VUE

2. 如何使用VUE

3. MVVM设计模式

4. 绑定语法

5. 指令

1. 什么是VUE

VUE是渐进式的，基于MVVM设计模式的，纯前端js框架

渐进式: 可以部分使用Vue的技术，与其它技术融合使用。

全家桶: 要用，就必须全套使用这个框架的所有组件。

纯前端: 不需要任何后端技术，就可独立运行。

框架: 已经包含核心功能的半成品代码！程序员只需要添加个性化的部分就可！

原生DOM vs jQuery函数库 vs 框架

原生DOM: 事事都要亲力亲为。且函数名繁琐，步骤繁琐。但是，兼容性好！

jQuery函数库: 虽然对每一步都进行了简化！但是，没有根本简化开发的流程！

框架: 已经包含核心功能，已经封装了重复的劳动！程序员无需反复编写重复的增删改查操作！

但是: 开发人员需要彻底改编程习惯和思路。

为什么: 提高开发效率！

何时: 今后只要以数据为主的项目，都可用vue框架开发。

2. 如何:

官网: cn.vuejs.org

下载: 2种:

1. 下载独立的vue.js文件，引入网页中使用

2个版本:

1. 开发版: 未压缩过的，拥有完备的注释和错误提示的版本。

2. 生产版: 压缩过的，且去掉了错误提示！

2. 安装脚手架代码

在网页中引入vue.js文件：

编写第一个Vue程序的套路:

1. 编写界面:

要求: 界面的所有内容，放在一个唯一的父元素内。父元素起一个id，便于查找。

如果界面中有可能变化的数据: {{变量}}代替

如果界面中有需要绑定事件的地方: @事件名="处理函数名" ——不要加()。

2. 定义数据对象:

要求: 所有界面需要的变量都要定义在名为data的对象中:

var data={ 变量: 值， 变量: 值, ...}

3. 创建Vue对象，将界面和数据绑定起来

new Vue({

el:"选择器", //找到界面中要监控的父元素

data,

methods:{ //集中存放事件处理函数

add(){

this.变量

}

}

})

结果: 只要程序中变量值发生变化，页面上自动更新！

3. MVVM设计模式:

传统前端三大部分:

HTML: 专门定义网页内容

CSS: 专门定义网页内容的样式

JS: 专门为网页添加交互行为

问题: HTML和CSS功能太弱了！一切变化，都需要js编写。导致js需要编写大量重复的增删改查操作！

现代前端三大部分：

1. 界面(View): HTML+CSS

集中精力定义网页要显示的内容和样式

2. 数据模型(Model): var data={ ... }

集中保存界面中所需的所有变量和变量值。

3. 控制器(ViewModel)

专门负责自动将模型中的变量值同步到界面中所需的位置。——封装了以前DOM重复的增删改查操作！

new Vue()的绑定原理:

new Vue()就是创建一个ViewModel对象，自动绑定界面和数据。

new Vue()对象中包含两大子系统:

1. 响应系统:

new Vue()刚创建时，先引入data对象

并自动为data中每个变量添加访问器属性保护起来。

所有访问器属性直接隶属于new Vue()对象。

想修改data中的变量，可通过"this.变量名"方式访问。但是，其实都是访问的变量的set访问器。

而且，new Vue()在每个set访问器中都添加了通知机制。只要修改变量，都会自动调用访问器。只要调用访问器，就会触发通知

2. 虚拟DOM树:

什么是: new Vue()通过扫描真实DOM树，而形成的一个简化版DOM树模型。

仅包含可能变化的元素和可能变化的属性。

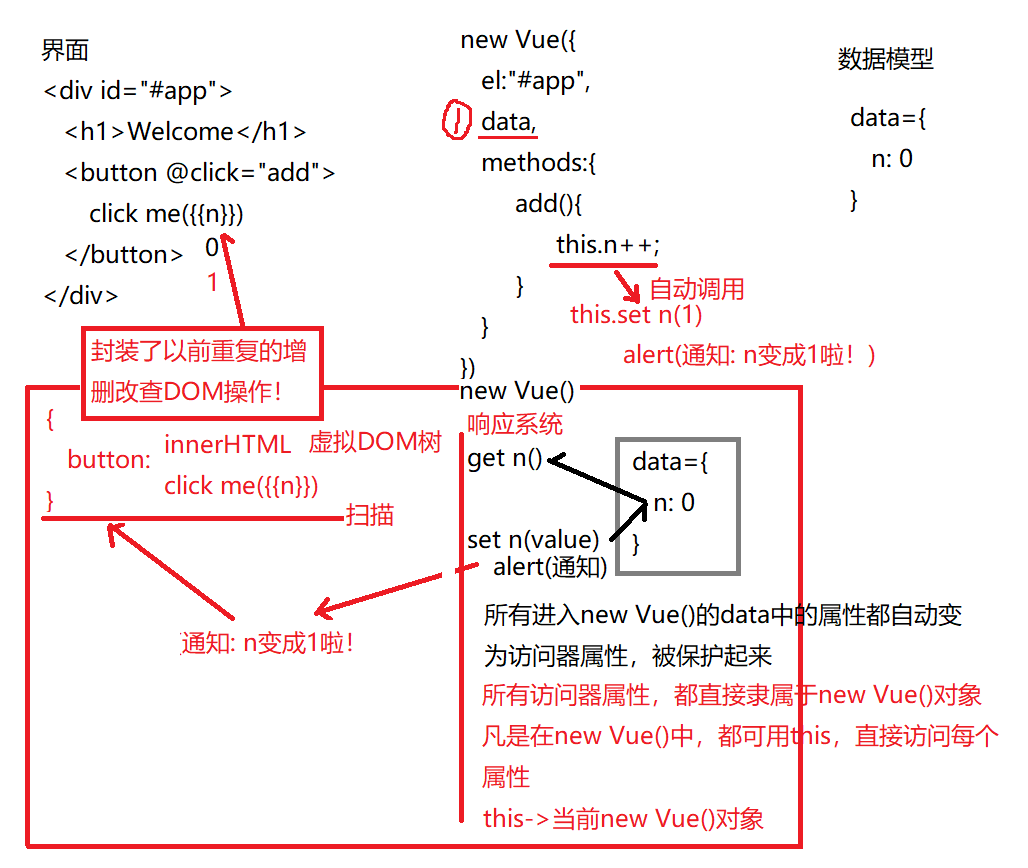
那些固定不变的元素和属性，一点都不包含。

为什么: 因为真实DOM树太大了！包含很多无关的元素和属性。不便于快速查找。

优点: 2个:

1. 小！遍历极快！且只修改变化的元素，影响范围小。

2. 封装了重复的增删改查操作！极大的节约了代码量！



4. 绑定语法:

学名: Interpolation 插值语法

什么是: 专门为HTML中添加可变的变量

为什么: HTML语言本身不支持变量，但是HTML中的个别内容又需要根据程序中的变量自动变化

何时: 只要HTML中任何一个位置的内容，需要根据程序中的变量值自动变化时

如何: {{变量}}

结果:

首次记载时，new Vue()会自动找到这里，用程序中的变量值，替换{{变量}}

当程序中的变量被修改时，再次扫描虚拟DOM树，找到这里，用新值自动替换。

其实: {{可写任意有返回值的正确的js表达式}}

比如: {{n+m}} 算术计算

{{`字符串拼接`}} 字符串拼接

{{fun()}} 函数调用

{{new Date().getLocaleString()}} 创建对象

{{arr[0]}} 访问数组

{{条件?值1:值2}} 三目运算

不能写: 程序结构: 比如if else for while ...

5. 指令: directive

什么是: 为HTML添加新功能的自定义属性

为什么: 因为HTML语言本身太弱了！没有编程语言必须的要素: 变量，if else， for

何时: 只要想在HTML中使用程序的一些特征时: 分支，循环，都要用指令。

包括: 12

1. 绑定属性:

问题: 属性值中不能用{{}}绑定

解决: v-bind或 :

<ANY v-bind:属性="变量或js表达式"

可简写为: <ANY :属性="变量或js表达式"

强调: 用了: 就不要再加{{}}

2. 控制元素显示隐藏:

1. 只控制一个元素的显示隐藏

<ANY v-show="条件表达式"

v-show会判断等号有的条件是否成立。只有判断条件成立时，才显示该元素。如果按断条件不成立，则隐藏该元素。——display:none

比如: <button v-show="pno<pcount"

只有当pno<pcount时，当前button才显示。反之，如果pno>=pcount了，则button不显示！

2. 多个元素根据条件选其一显示

1. 2个元素，二选一

<ANY1 v-if="判断条件">

<ANY2 v-else>

v-if会判断等号后的判断条件。如果条件成立，则显示第一个元素。后一个v-else的元素，不再显示。如果条件不成立，则隐藏第一个元素，选择第二个v-else的元素作为备胎。

强调: 1. v-else后不要跟条件

2. 两个元素必须连续写，重点不能插入任何第三个元素！

原理: v-if和v-else是通过添加删除元素的方式，控制元素的显示隐藏

v-show vs v-if

1. v-show: display:none ——效率高

v-if: 添加删除元素 ——效率低

2. 当控制多个元素显示隐藏时:

v-show被迫要把条件反复写多遍

v-if只需要把条件写一遍即可，v-else不需要条件！

2. 多个元素，多选一显示:

<ANY1 v-if="判断条件1">

<ANY2 v-else-if="判断条件2">

<ANYn v-else>

原理:

如果v-if的条件成立，就显示v-if的元素。后边多个元素不再显示

如果v-if的条件不成为，则按顺序判断每个v-else-if的条件，哪个成立，就显示哪个。除当前v-else-if之外的其余v-if, v-else等都不显示

如果所有条件都不满足，在只有v-else显示

3. 事件绑定:

1. HTML中绑定处理函数:

<ANY @事件名="处理函数名"

比如: <button @click="add">

当单击按钮时，自动调用add函数

2. js中要定义处理函数:

处理函数必须定义在methods属性中

new Vue({

el:"#app",

data:{ ... },

methods:{

//处理函数名

add(){  
 ...操作模型变量...

}

}

})

强调: 无论是data中的变量，还是methods中的函数，进入new Vue()后，都直接隶属于new Vue()。所以，在new Vue()直属的方法中，访问new Vue()直属的属性，只要用“this.”

4. 反复生成多个相同结构的元素：

<ANY v-for="(elem,i) of 数组" :key="i" >

v-for会自动遍历数组中每个元素

每遍历一个元素，就自动创建一个<ANY>元素对象。

of前的elem和i，都可用<ANY>内部子元素上的绑定。——虽然没有定义在data中，但是，也可以像data中的变量一样用于绑定！

强调: v-for必须写在要反复生成的元素上。而不要写在父元素上！

为什么必须加:key="i":

如果不加:key="i"，v-for生成的多个元素之间，除了内容之外，没有任何差别。如果数组中某个元素更新，就无法精准的更新某一个元素。而必须将整个v-for重新执行一遍！——效率低

如果给每个生成的元素都添加一个key="i"，其中i是0 1 2 3下标，且不重复！当某一个元素发生改变时，vue只需要根据key属性的值，找到要更新的这一个元素，就可即可。不用把整个v-for重新执行一遍。——效率高！

5. 绑定html片段:

问题: 如果要绑定的内容是一段html片段时，使用{{}}绑定内容，会将html片段原样显示，而不会解析其中的HTML内容。

解决: 今后只要绑定HTML片段，都用专门的v-html指令。

6. 如何防止用户短暂看到{{}}语法:

问题: 当vue加载慢时，用户可能短暂看到{{}}

解决: 2种:

1. 麻烦的办法: v-cloak 斗篷

如何:

1. 给需要临时隐藏的元素上加上v-cloak属性，不要给值！

<h1 v-cloak>用户名:{{uname}}</h1>

2. 手动添加一个属性选择器，隐藏所有带v-cloak的元素

<style>

[v-cloak]{display:none}

</style>

原理:

new Vue()没有加载时，自定义的[v-cloak]{display:none}会隐藏所有带v-cloak属性的元素

但是，当new Vue()加载完，会自动找到所有v-cloak属性，并移除v-cloak属性。原理被v-cloak隐藏的元素，现在显示出来了。

2. 简单的解决方法: v-text指令代替{{}}

v-text是{{}}的等效指令

只不过v-text写在属性中，而{{}}写在内容中。

<h2 v-text="`积分:${score}`"></h2>

<!--js的模板字符串-->

7. 保护内容中的{{}}不被编译:

特殊情况: 内容的正文中刚好包含{{变量}}这种内容。但是，却不希望被vue解析。

8. 只在首次绑定时，绑定一次，今后，即使变量值发生变化，也不会重复绑定！

<ANY v-once>{{变量}}

原理: 在首次绑定完之后，就自动从虚拟DOM树中移除对该元素的监视。

总结:

1. 绑定元素内容: {{}}

2. 绑定属性值: : v-bind

3. 控制一个元素显示隐藏: v-show

4. 控制两个元素二选一显示: v-if v-else

5. 控制多个元素多选一显示: v-if v-else-if v-else

6. 绑定事件: @ v-on

7. 反复生成多个相同结构的元素: v-for

8. 绑定html片段: v-html

9. 防止短暂看到内容中的{{}}: v-cloak 或 v-text

10. 防止内容中的{{}}被vue编译: v-pre

11. 只在首次加载时，绑定一次。只有即使变量变化，也不更新: v-once