# 正课: 尤雨溪

1. DOM vs 函数库 vs 框架

2. 什么是Vue

3. 如何使用Vue

4. Vue的原理

5. 绑定语法

6. 指令

7.双向绑定

### 1. DOM vs 函数库 vs 框架

DOM: API繁琐

函数库: jQuery 对DOM的每个步骤的API进行一对一的简化，但并没有改变DOM做事的步骤和方法。

框架: 一个包含部分已经实现的功能的半成品项目，只是暂时缺少数据和个性化页面。

优点: 彻底简化了开发的步骤，且彻底避免了大量重复的劳动

缺点: 需要转变观念

### 2. 什么是Vue:

Vue是基于MVVM设计模式的渐进式的前端js框架

渐进式: 可以有选择的，逐渐使用框架中的功能

vs "全家桶": 必须完整使用所有功能，不能掺杂其它技术

前端js: Vue是纯前端的框架，无需node.js，就可独立运行。但是，需要node.js在运行之前将vue编译为浏览器认识的普通的HTML、css和js文件。

何时: 适合于以**数据操作为主**的项目

### 3. 如何使用Vue:

官网: https://cn.vuejs.org/

2种:

1. 下载并引入vue.js文件

最新版本:2.6.x

开发版: 未压缩，带错误提示

生产版: 压缩，去掉错误提示

2. 安装脚手架工具

### 4. Vue的原理:

旧式的前端划分:

html: 定义网页静态内容

css: 定义网页静态样式

js: 为网页添加动态行为

MVVM设计模式:

新的前端划分:

1. 将一个.html文件，拆成多个组件文件: .vue/.component

2. 每个组件由细分为3部分:

1. 界面(View): HTML和CSS

为HTML添加了动态功能：变量、if else 、for

2. 数据模型(Model): 在内存中自己创建的或从服务端请求来的保存页面需 要的所有数据的对象或数组: ajax中的res，集中定义在data={数据1：值 1，数据2：值2，....}

3. 控制器(ViewModel/controller): 自动将数据模型中的数据绑定/同步到页面需要的位置——不再需要程序员反复手工查找并修改元素。

控制器的原理: 2大子系统:

1. 响应系统: 监视模型数据的变化，并通知框架修改页面的机制。

其实是将数据模型中每个属性变为一个访问器属性

每次修改模型中的数据时，都会自动触发set

set会发出通知: "xx变量发生改变"

2. 虚拟DOM树: 扫描原始DOM而产生的一棵仅包含可能变化的元素的新的简版DOM树

优点:

1. 仅包含可能变化的元素的可能变化的个别属性——简单——遍历快

2. 修改时，可仅找到发生变化的元素，仅修改发生变化的元素——效率高

vue对象创建时，首次扫描并创建虚拟DOM树

响应系统发出通知时: 虚拟DOM树收到通知，并查找要修改的个别元素，实际修改DOM树。

### 5. 绑定语法:

什么是: 在页面中使用特殊的符号{{}}指定所需的变量名

学名: interpolation 插值语法

何时: 只要页面的某个位置需要动态生成值时

如何:

{{模型变量名}}

{{表达式}}

表达式包括: 算术计算，关系运算/逻辑运算，方法调用，访问数组元素，访问对象的属性，三目运算

强调: 不能写程序结构:

if else if else

for while do while

### 6. 指令(directive):

什么是: 为HTML提供新功能的特殊属性

为什么: {{}}只能绑定元素的内容(innerHTML)，不能绑定属性/事件或执行程序结构

何时: 不能使用{{}}绑定的位置，又想动态生成值，都要用各种指令来增强

如何:

**1. v-bind:** 专门用于动态绑定属性值

何时: 只要一个属性的值需要动态生成时

如何: 完整: <ANY ~~v-bind~~:属性="表达式"

可省略v-bind

**2. v-for:** 专门根据一个数组中的元素，动态反复生成多段相同的页面内容

何时: 只要根据一个数组中的元素，动态绑定页面内容时

如何: <ANY v-for="模型变量 of 数组类型模型变量"

根据of后数组中的元素个数，反复生成多个<ANY>元素

of前的模型变量是临时的，获得数组中的一个当前元素

of前的模型变量在<ANY内部可用于绑定！

3. 控制元素的显示隐藏:

**v-if:** 专门根据一个条件表达式来决定当前元素是否显示

只有条件表达式为true时，当前元素才显示

何时: 只要根据一个条件控制元素的显示隐藏时

如何:

<ANY v-if="条件表达式1"

<ANY v-else-if="条件表达式2"

<ANY v-else-if="..."

<ANY v-else//不用写条件

强调: v-if和v-else-if和v-else之间不能插入其它元素

**v-show:**专门根据一个条件表达式来决定当前元素是否显示

只有条件表达式为true时，当前元素才显示

采用display:block/none的方式显示隐藏

何时: 如果仅控制一个元素的显示和隐藏时，首选v-show

vs v-if**:** 采用添加删除元素的方式，控制显示隐藏

何时: 多个元素选其一显示时，首选v-if v-else-if v-else

**4. 绑定事件:v-on**

界面View中:

<ANY **v-on:**事件名="处理函数()"

简写: **v-on:可替换为@**

new Vue({})中:

new Vue({

el:"选择器",

data:{ 模型变量 },

methods:{//专门保存控制器所需所有自定义方法的属性

处理函数(){//处理函数:function(){

//this->当前Vue对象->vm

this.模型变量 //访问模型变量:

}

}

})

问题1: 阻止默认行为:

解决: 2步:

1. 界面: 当事件发生时，自动调用js语句，并传入$event作为参数

<ANY @click="处理函数($event)"

$event就是DOM中的e，是vue封装的事件对象

2. new Vue()中:

methods:{

//定义形参，准备接事件对象

处理函数(e){

e.preventDefault();

}

}

简写: 事件修饰符，对DOM中事件API的简写:

2个:

阻止默认行为: @click.prevent="处理函数()"

取消冒泡: @click.stop="处理函数()" =>e.stopPropagation()

**5. 绑定内容: v-html**

原始HTML片段:

问题: 不能用{{}}绑定，因为{{}}会保持HTML片段原样显示

解决: 用v-html="模型变量" 代替{{}}，可让绑定的HTML片段被浏览器解析为正文

总结: 只要绑定HTML片段，都用v-html

纯文本内容: 可使用{{}}或v-text

表单元素的值

**6. v-cloak:**

何时: 如果希望当new Vue加载慢时，也不让用户看到{{}}语法，就用v-cloak临时隐藏页面元素

如何: 2步:

1. 在页面的<style>中，定义属性选择器:

[v-cloak]{ display:none }

2. 在要临时隐藏的元素上添加v-cloak指令

原理: new Vue加载完之后，会自动查找v-cloak属性，并移除该属性，使临时隐藏的元素，显示出来

**7. v-pre:** 保持内容中的{{}}不被Vue编译

何时: 如果内容中刚好有{{}}作为正文，则需要用v-pre保护

**8. v-once:** 尽在首次Vue加载时，绑定一次，之后即使变量改变，也不更新页面

如何使用: <ANY v-once>绑定语法</ANY>

原理: 在首次绑定后，就将v-once所在的元素从虚拟DOM树中移除

#### **2. 双向绑定:**

问题: 普通的绑定":属性"，只能讲内存中的模型变量单向绑定到界面，无法将界面的更改，自动更新回模型变量中

解决: **v-model代替:value**

什么是双向绑定: 即可以将内存中的数据绑定到界面上，同时又能将界面的修改，反向更新回模型变量中

(M-V, V-M)

何时: 如果希望绑定表单元素的值时，都用双向绑定

如何: <表单元素 v-model:value="模型变量"

:value可省略

原理: Vue用一个死循环，反复扫描虚拟DOM树中的每个节点——监视

如果希望只要界面发生变化，就立刻执行操作时:

new Vue({

...,

data:{ kwords:"" },

...,

watch:{

//在kwords发生改变时，自动执行

kwords(){ }

}

})

各种表单元素的双向绑定:

1. 文本框: <input type="text"> 和文本域: <textarea>、select

都是绑定value

<ANY v-model="模型变量"> v-model自动绑定value属性

2. 单选按钮:

<input type="radio" name="分组名" value="值" v-model="模型变量">

分组使用:

M-V: 如果**模型变量的值等于value**，则当前radio选中，否则不选中

V-M: 将当前选中的radio的value值自动更新回模型变量

3. 多选框:

<input type="checkbox" v-model="模型变量"

单用: 自动绑定的是checked属性

4. select:

<select v-model="模型变量">  
 <option value="值1">

<option value="值2">

... ...

M-V: 用模型变量的值和每个option的value做比较。如果某个option的value等于模型变量的值，则该option被选中

V-M: 当选中项发生改变时，自动将选中项的value，更新到模型变量中

#### 3. 绑定class和style:

绑定style: 2种:

1. 将style当做普通字符串属性绑定: 不直观，繁琐

2. 可用对象方式绑定style:

<ANY :style="模型变量"

new Vue({

... ,

data:{

模型变量: {

css属性:值,

... : ...

}

}

})

问题: 用style修改多个css属性，代码会很繁琐

解决:

绑定class: 2种:

1. 将class当做普通字符串属性绑定:

<ANY class="不变的class列表" :class="模型变量"

new Vue({

...,

data:{

模型变量:"可能变化的class列表"

}

})

2. 将class当做对象绑定:

<ANY class="不变的class列表" :class="模型变量"

new Vue({

...,

data:{

模型变量:{

class名: true/false,

class名:true/false

}

}

})

#### 4. 计算属性:

什么是: 数据库中或程序中没有保存的，但需要动态计算出来的值

何时: 如果一个值，没有保存，需要每次动态计算获得

比如: 购物车的总价

如何:

new Vue({

...,

computed:{

属性名(){

...

return 属性值

}

}

})

界面中: <ANY>{{属性名}}</ANY>

其实: methods:{

方法(){ return 计算结果 }

}

界面中:<ANY>{{函数名()}}</ANY>

也行

computed vs methods:

相同: 依赖的变量发生变化，都能自动更新

不同: 1. methods要使用函数名()的调用语法

computed使用不加()的属性名即可

2. 如果需要显示多次时，

**methods会重复计算多次**

**computed会缓存第一次计算的值，避免重复计算**

#### 5. 过滤器:

什么是: 接收原始值，经过再加工，再显示出来

何时: 只要模型变量的原始值，不能直接使用时

比如: 性别:0、1

日期/时间: ms

状态: 存数字，显示状态名

如何:

1.定义过滤器

Vue.filter(“过滤器名”,function(value，形参1...){

//value 会接住要处理的原始值

Return 改造后的值

})

2.使用过滤器

在绑定时，{{原始模型变量 | 过滤器(实参1...) | 下一个过滤器}}

：属性=”原始模型变量 | 过滤器”

#### 6. **自定义**指令:

如何:

1. 创建指令

Vue.directive("指令名",{

inserted(elem){//指令所在的元素被加载到DOM树上后自动执行指令

//elem会自动获得指令所在的当前元素

对elem执行DOM操作！

}

})

2. 使用指令: <ANY v-指令名>

### 7. Axios

什么是: 基于promise专门发送http请求的函数库

为什么: ajax

1. 自己封装函数:

2. jQuery: $.ajax() 小题大做

3. Vue有一个组件resource，已不推荐使用

4. Axios 官方推荐

何时: 只要在vue中发送ajax请求都用axios

如何:

1. 引入: axios.min.js

2. 发送2种请求:

get请求:

axios.get("url",{

params:{

参数:值

}

}).then(res=>{

Res是经过axios封装后的对象

获得服务端返回的结果: res.data

})

post请求:

axios.post("url","变量=值&变量=值&...").then(res=>{...})

### 8. 组件化开发:

什么是组件: 拥有专属的HTML，CSS，JS和数据的页面独立区域

什么是组件化: 今后所有页面都是由多个组件组成的

每定义一个网页，都要先划分组件结构，再分别编写每个组件

为什么:

1. 重用

2. 便于大项目的分工协作

3. 松散耦合

如何:

#### 1. 创建组件:

每个组件包含三部分:

HTML: 独立的HTML片段:

<template id="tplxxx">

<div> //所有子元素都必须包含在唯一的父元素里

</div>

</template>

<template>是HTML5中新增的专门保存一段隐藏的HTML片段的元素——专门为框架保存HTML模板

JS: <script>

Vue.component("组件名",{

//和new Vue()极其相似

template:"#tplxxx", //代替了el，每创建一个组件的副本，就自动复制一次模板中的html片段

data:function(){//每创建一个组件的副本，就调用一次data()，为当前组件创建专属的模型数据data  
 return {

count:1

}

},

//其余和new Vue()完全一样

})

CSS: <style> 暂时不关心

#### 2. 使用组件:

组件名其实就是一个标签名

只要在主内容中，添加一个<组件名></组件名>，运行时Vue就会用组件的template代替<组件名>标签所在位置

组件分类:

1. new Vue({ ——根组件

el:"#app",

data:{}

})

2. Vue.component("组件名",{——全局组件: 可在任何位置随意使用的组件

template:"#tplxxx",

data:function(){ return {} }

})

3. 子组件: 仅限于在某个父组件内才能使用的组件

如何: 3步:

1. 只要将Vue.component降级为普通对象

//Vue.component("todo-add",

var todoAdd={

template: "#tplTodoAdd",

data() {

return { }

}

}

强调: 变量名最好是将来组件标签名的驼峰方式

2. 在父组件中添加components属性:

Vue.component("todo",{

template:"#tplTodo",

data(){

return{ }

},

components:{todoAdd}

})

{

子组件变量名, 子组件变量名, //ES6简写

}

Vue会将驼峰命名的组件名，转为小写，并用-分隔

比如: todoAdd -> todo-add

todoList -> todo-list

#### 组件间传递数据:

父子间:

1. 父->子:

为什么: Vue中父组件的数据模型是专属的，子组件默认不能使用父组件的数据

解决: 2步:

1. 子组件:

var 子组件={

template:"xxx",

props:["变量", ... ] //声明一个内外两用的变量

//对外: 父元素可为其绑定数据

//对内: 相当于data:{ 变量 }

}

<template id="tpl子组件">

{{变量}} <ANY :属性="变量"

</template>

2. 父组件:

<template id="tpl父组件">

<子组件 :变量="模型变量"></子组件>

父组件将自己的模型变量的值传递给子组件的变量属性，子组件就获得了父组件的数据

2. 子->父:

1. 父组件: 在子组件身上提前定义自定义事件和处理函数:

<template id="tpl父组件">

<子组件 @自定义事件="父的处理函数"//不要加()

//当有人触发自定义事件时，自动执行父组件中的处理函数，修改父组件自己的数据

父组件{

...

methods:{

处理函数(形参) {//形参会收到子传来的参数  
 //操作父组件中的数据

}

}

}

2. 子组件:

任何时候都可以:

this.$emit("父给子定义的事件名", 实参)

意为触发父给子提前定义的事件，并自动引发执行父的处理函数。

# 正课:

1. SPA

2. Vue-router组件

3. 脚手架

## 1. SPA

: Single Page Application

什么是SPA: 整个应用程序只有一个完整的HTML页面和多个保存不同"页面"内容的组件片段。

SPA应用侦测地址栏中相对路径的变化，选择匹配的组件片段替换唯一HTML文件中的相应区域。

vs 多页面:

请求次数:

单页面应用只在首次加载时请求一次，将一个完整的HTML页面和多个片段都获取到客户端保存。每次跳转时，不再需要向服务端发送请求。——只有一次

多页面应用，每跳转一次页面，都重新向服务器发送请求——多次

页面跳转:

单页面应用，不需要替换整棵DOM树，只需要替换DOM树中局部节点即可——效率高

多页面应用，每次跳转，需要删除并重建整棵DOM树——效率低

相同资源的请求: Bootstrap

单页面应用因为页面没有改变，所以无需重复请求资源

多页面应用，每加载一次页面，都重新请求所需的资源

## 2. Vue-router组件:

什么是: 在Vue中专门实现单页面应用的组件

何时: 如果用Vue框架实现单页面引用时

如何:

1. 下载并引入vue.js和vue-router.js

2. 定义基本的根组件

<div id="app">

</div>

<script>

new Vue({

el:"#app",

data:{}

})

3. 在根组件内容中，预留可能被其他子组件替换的区域

<router-view/> 为其他子组件的片段占位——路由插槽

何时: 只要页面中可能变化的片段区域，都要用router-view占位

4. 定义多个子组件, 每个组件2部分组成:<template>和<script>

5. 定义路由: 3步:

1. 定义路由字典:

路由字典: 包含相对路径和组件间对应关系的集合

var routes=[

{path:"/", component: Index},

{path:"/index", component: Index },

{path:"/login", component: Login },

{path:"/\*", component: NotFount}

];

2. 创建路由器对象，装载路由字典

var router=new VueRouter({ routes })

3. 将路由器装入根组件new Vue中

new Vue({

...,

router

})

### 嵌套路由:

什么是: 在路由字典中的一个路由对象中，又包含子路由对象

何时: 多个"页面"组件中包含部分相同的区域时:header

如何:

1. 相同部分的组件，提取成全局组件

2. 再额外提取出一个父组件，包含:

<相同部分的全局组件>

<router-view>

3. 配置路由字典:

{path: /父路径 , component:父组件, children:[

{path:"/父路径/子路径", component: 子组件},

...

]}

运行时: 地址栏中: #/父路径/子路径

/父路径 先用父组件代替div#app中的<router-view>

/子组件 再用子组件代替父组件中的<router-view>

路由跳转: 2种:

1. HTML: <router-link to="/相对路径">文本</router-link>

加载时: <a href="/相对路径">文本</router-link>

自动判断是否需要#

2. js: this.$router.push("/目标地址")

$router 就是var router=new VueRouter()路由器对象

凡是以$开头的都是Vue内置的对象

### 路由传参:

1. 在路由字典中，就提前配置好:

{path:"/details/:lid", ...

2. 跳转时: /details/9

3. 组件中接收参数值: [this.]$route.params.lid

location search

简化: {path:"/details/:lid", ..., props:true}

var 组件={

... ,

props:["lid"]

}

绑定时，可直接使用lid变量

## 3. 脚手架:

什么是脚手架代码: 半成品的项目源代码，只需添加个性化的数据和界面

什么是脚手架命令行工具(CLI): 反复创建脚手架源代码

如何:

1. 用npm安装脚手架命令行工具:

npm i -g @vue/cli

结构: 可在**命令行**中使用vue命令反复创建Vue脚手架项目源代码

2. 用Vue脚手架工具命令创建项目源代码:

vue create 项目名-> Manually select features->选择需要的,回车

3. 运行项目:

vs code中，打开项目文件夹，并在终端中打开，输入:

npm run serve

编译成功后，打开浏览器，访问: http://localhost:5500

在vs code中修改vue项目内容后，npm run serve会自动编译重启。

4.组件的生命周期

什么是一个组件创建的过程

包括：4个阶段

1.create：创建阶段

创建组件对象，以及模型对象data

2个钩子函数：

beforeCreate() 创建对象前触发

created() 创建对象和data数据后触发

有data 可以axios发送ajax请求

没有虚拟DOM 暂时不能操作DOM元素

2.mount：挂载阶段

生成虚拟DOM树，绑定数据到真实的DOM树上

3.update：更新阶段

当组件的模型数据被更新时自动触发

4.destroy：销毁阶段

除非调用专门的destroy函数，才能触发销毁阶段