**vue**

1. 原生DOM vs 函数库 vs 框架

原生DOM: 优点: 跨平台好

缺点: 繁琐

函数库jQuery: 优点: 对原生DOM的每个API进行了简化

缺点: 并没有简化开发的步骤

框架Vue,AngularJS,React: 优点: 彻底简化了开发的步骤

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: 可以有选择的逐步使用框架中的组件

"全家桶": 必须全盘使用所有组件

前端js框架: 不需要nodejs，仅靠浏览器就可独立运行

为什么后续需要nodejs？将Vue框架中浏览器不认识的新技术翻译为浏览器认识的ES5的对等标准——已经封装好了

何时: 侧重于以**数据操作为主**的前端项目开发

3. 如何使用Vue:

不支持IE8。

下载: cn.vuejs.org

最新稳定版: 2.6

开发版: 未压缩的，包含完备注释和错误提示信息

优: 可读性好

缺: 体积大，不便于传输

生产版: 代码经过压缩，删除了所有注释和错误提示信息

优: 体积最小化，便于传输

缺: 不便于学习和阅读

使用Vue 2种方式:

(1) 下载独立的vue.js，并引入:

在网页中引入vue.js

编写网页的HTML

强调: 网页的内容必须包含在一个唯一的父元素内。

2件事:

1. 将要显示的数据变量，提前放置在页面的对应位置

2. 顺便绑定事件

编写自定义脚本:

2件事：

1. 创建对象data: 定义页面上需要的所有数据变量

2. 创建Vue对象，将data和网页绑定起来

(2) 用脚手架代码:

**原理: MVVM**——设计模式

旧的前端代码划分:

html: 静态网页的内容，不支持动态变化

css: 静态网页的样式，不支持动态变化

js: 增删改查，事件绑定

MVVM设计模式将前端内容重新划分: （步骤）

界面(View—视图): html+css

为HTML提前添加变量，提前绑定事件

模型数据(Model): 所有页面上需要的/可能发生变化的数据。

集中定义在data={ 数据1:值1, 数据2:值2, ... }

往往模型数据都是ajax从服务端请求来的。

控制器(ViewModel)：将视图View和模型数据(Model)绑定在一起。

绑定: 监控视图和模型，始终保持模型数据与页面自动同步。

包含两大子系统:

(1) 响应系统: 监控模型中每个变量的变化，同时发出通知

只要有变量发生变化，立刻发出通知！

本质: 将data中每个属性都提升为new Vue对象的访问器属性。只要修改new Vue对象的访问器属性，就可修改data中的变量。但是，同时(set)会发出通知给虚拟DOM树！

(2) 虚拟DOM树(virtual DOM tree):

什么是: Vue临时生成的仅保存可能变化的元素和属性的DOM树。

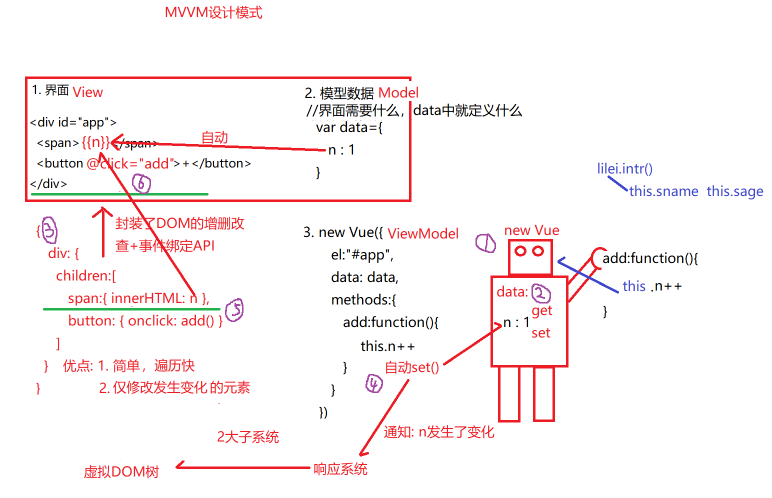
何时生成: new Vue()边扫描受监控的页面元素，边创建虚拟DOM树，仅保留可能发生变化的元素。

何时使用: 响应系统通知某个变量被改变时，告知虚拟DOM树。虚拟DOM树快速遍历自己，找到受影响的DOM元素，仅修改受影响的DOM元素！

总结: 虚拟DOM树:

内容少，遍历极快；

定点更新——仅修改受影响的DOM元素的属性或内容。



**4. 绑定语法**:

{{变量名}} ——学名：Interpolation 插值语法（俗称：大胡子语法）

告知Vue框架，这里需要哪个变量

{{}}中还可写一切正确的有返回值的js表达式:

比如: 运算, 三目, 函数调用，访问数组元素，访问对象属性，模板字符串

问题: 只能绑定内容，不能绑定属性、事件和程序结构(if else for while...)等

**5. 指令**(directive)

什么是: Vue新增的增强HTML功能的特殊**属性**。

为什么: HTML缺少动态原因所需的要素: 变量，运算, 分支，循环

包括:

(1) 专门绑定属性值的指令: v-bind

何时: 绑定属性值

如何: <ANY v-bind:属性名="js表达式">

v-bind: 让普通的属性的""中也可以执行js程序

其实可省略: v-bind

所以: 今后只要绑定属性都用 :属性名="js表达式"

(2) 事件绑定: v-on

如何:

HTML中: <ANY v-on:事件名="处理函数名">

<script>中

new Vue({

el:"#app",

data:{ 模型变量 },

methods:{

处理函数 (){

this.模型变量 –>this指当前vue本身

}

}

})

其实: v-on: 可简写为@，

所以，今后绑定事件都用 @事件名="处理函数"——处理函数建议后面不加（）

(3) v-if: 为HTML添加分支结构

v-if 控制一个元素的显示隐藏：满足条件就显示，否则隐藏！

<ANY v-if="条件表达式"

v-if v-else 可控制两个元素二选一显示：

<ANY v-if="条件表达式"

<ANY v-else

v-if v-else-if v-else 可控制多个元素多选一显示：

<ANY v-if="条件表达式"

<ANY v-else-if="条件表达式"

<ANY v-else

强调：必须连着写，中间不能有其他标签！

(4) v-show: 控制一个元素显示隐藏

用法同v-if: <ANY v-show="条件表达式"

本质: 用display:none隐藏元素——控制一个元素的显示/隐藏

vs v-if的本质: 用空注释节点<!-- -->替代要隐藏的节点，隐藏元素——控制多个元素多选其一显示

(5) v-for: 根据数组反复生成多个相同结构的元素

用法: <ANY v-for="(elem,i) of 数组">

//elem可当做模型变量使用，用作绑定

</ANY>

of会依次读取数组中每个元素，每读取一个元素就创建一个<ANY>;

同时获得元素值和下标，可用作绑定新元素的内容和属性。

在v-for元素最后加上 :key="i"，key是代表按照序列遍历，可清除黄色警告！！

<ANY v-for="i of 数字">

遍历数字，i是从1开始的数字。

(6) 绑定html片段:

问题: {{}}无法绑定HTML片段，它会努力保持HTML片段原样显示

解决: v-text="变量" 不行，v-text等效于{{}}

v-html="变量" 可以，会被浏览器解析为正文

总结: 只要绑定的内容是HTML片段，都用v-html

(7) v-cloak: 在网页加载慢时，短暂隐藏{{}}语法，不让用户看到

如何: 2步

在界面中需要隐藏的元素/父元素上，添加v-cloak自定义属性

<ANY v-cloak>

在页面的<style>中用属性选择器找到所有有v-cloak属性的元素，设置隐藏:

<style>[v-cloak]{ display:none }</style>

结果: 当vue对象加载完成，会自动查找所有v-cloak属性，并移除。所有元素就重新显示出来

vs v-text短暂隐藏：<ANY v-text=“`拼字符串内容`”>

(8) v-pre: 保护内容中碰巧出现的{{}}不被VUE编译，保持原样显示。

<ANY v-pre>

(9) v-once: 控制一个变量只在开始时绑定一次，之后即使模型变量变化，也不再更新。

<ANY v-once>

**day02**

**1. 双向绑定**:

单向绑定: 将内存中模型数据中的值绑定到界面上(Model->View)

问题: 无法绑定表单元素的值

表单元素值的修改，无法自动映射到内存中

解决: 今后只要绑定表单元素，都用双向绑定

什么是双向绑定: 即能将Model中的数据绑定到界面上（Model->View），又能自动将界面上的修改，映射到内存中(View->Model)

原理：为每个v-model的元素自动添加onchange事件，当界面上发生改变时，自动通知vue，更新data中的变量值。

如何: <表单元素 v-model:value="模型变量">

简写: <input type="text"

<textarea>

<select>

都可简写为: v-model="模型变量"

省略 :value，默认绑定value属性

<input type="checkbox" v-model:checked="模型变量">

可简写为: v-model="模型变量"

但默认绑定的是checked

<input type="radio" value=“1” v-model=“sex”>

可简写为: v-model="模型变量"

但用模型变量和radio的value做比较

如果data中模型变量的值=radio的value时，就选中checked；

如果data中模型变量的值!=radio的value时，就不选中。

watch 监控: 当某个模型变量被修改时，自动触发的函数。

凡是双向绑定的变量，都可添加监听函数，实时监听页面上的变化。

比如: <input v-model:value="kwords"

new Vue({

el:"#app",

data:{ kwords:"" },

methods:{ search(){ ...this.kwords... }},

watch:{ //专门保存所有监控函数的地方

kwords(){ //专门实时监控kwords同名变量的函数，会在kwords变量修改时，自动触发！

this.search() //边输入内容，边查找！

}

}

})

|  |
| --- |
| 示例：模拟百度，边输入，边搜索  <body>  <div id="app">  <input type="text" v-model:value="keywords">  <button @click="search">百度一下</button>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  keywords:""  },  methods:{  search(){  console.log(`搜索${this.keywords}相关信息`);  }  },  watch:{  keywords(){  this.search(); //数据更改，就调用方法中search函数  }  }  })  </script>  </body> |

1. 不同表单元素的双向绑定:

哪个属性会被改变，v-model就自动绑定哪个属性

(1) 文本框和文本域:

<input type="text" v-model[:value]="变量">

<input type="password" v-model[:value]="变量">

<textarea v-model[:value]="变量">

可通过双向绑定，获取文本框中的内容。

(2) 单选按钮:

<input type="radio" name="分组" value="1" v-model="sex">男

<input type="radio" name="分组" value="0" v-model="sex">女

v-model 先用sex的值和value做比较

data:{ sex:0 }

显示时（M->V）: 用data中sex的值和同组内每个radio的value做比较。如果data中sex的值等于当前radio的value，则当前radio自动选中！如果data中sex的值与当前radio的value不相等，则不选中

修改时（V->M）：将当前选中项的value，反向更新会data中的变量中。

(3) 复选框:

单独使用:

<input type="checkbox" v-model="isAgree">同意

显示时（M->V）：如果变量isAgree的值为true，则checkbox的checked属性就为true，checkbox就选中。如果变量isAgree的值为false，则checkbox的checked属性就为false，则checkbox就不选中。

修改时（V->M）：如果选中，当前checkbox的值为true，因为双向绑定，就将checked的值true，反向更新回data中的变量中。如果取消选中，当前checkbox的值就为false，因为双向绑定，就将checked的值false，反向更新回data中的变量中。

|  |
| --- |
| 示例：同意条款，才能提交  <body>  <div id="app">  <input type="checkbox" v-model="isAgree">同意 //双向绑定  <button :disabled="!isAgree">提交</button> //单向绑定  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  isAgree:false  }  })  </script>  </body> |

(4) select元素:

<select v-model[:value]="orderStatus"> value=30就会选中第三个

<option value="10" >未付款</option>

<option value="20" >已付款</option>

<option value="30" >已发货</option>

<option value="40" selected >已签收</option>

data:{ orderStatus:10 } //10会被改为20

显示时（M->V）：将变量orderStatus的值和每个option的value作比较，哪个option的值与变量orderStatus的值相等，就选中。不相等，就不选中。

修改时（V->M）：当前选中的option的value会自动成为整个select元素的value值，又因为双向绑定的原因，自动将select的新value值，更新回data中的变量中。

|  |
| --- |
| 示例：选择城市，展示不同图片  <body>  <div id="app">  <select v-model:value="src">  <option value="img/bj.jpg">北京</option>  <option value="img/hz.jpg">杭州</option>  <option value="img/sh.jpg">上海</option>  </select>  <img :src="src" alt="">  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  src:"img/bj.jpg"  }  })  </script>  </body> |

**2. 绑定class和style**

(1) 绑定style:

其实style也是一个普通的字符串属性，也可以:

<ANY :style="模型变量"

模型变量: "css属性:值; css属性:值; ..."

问题: 字符串拼接很繁琐，不直观

将style看做一个对象绑定:

<ANY :style="对象名"

data:{

对象名:{

css属性:值, //css属性都要去横线变驼峰！

css属性:值

}

}

(2) 绑定class

其实class属性也只是一个字符串属性，也可以:

<ANY :class="拼接字符串"

都用对象方式绑定:

<ANY :class="对象名"

data:{

对象名:{

hide: false,

success: true,

fail:false

}

}

vue在翻译时，如果为true就添加到ANY的class中，false就移除

|  |
| --- |
| 示例：验证手机号  <body>  <div id="app">  <input placeholder="请输入手机号" v-model="phone" type="text">  <span class="msg" :class="my\_class" v-text="my\_msg"></span>  </div>  <script>  var vm=new Vue({  el:"#app",  data:{  my\_class:{  success:false,  fail:false  },  my\_msg:"",  phone:"" //用v-model双向绑定，获取input输入的内容  },  watch:{ //实时监控，input边输入的内容，就边验证  phone(){  var reg=/^1[3-9]\d{9}$/g;  if(reg.test(this.phone)==true){  this.my\_class={success:true,fail:false};  this.my\_msg="输入手机号码格式正确"  }else{  this.my\_class={success:false,fail:true};  this.my\_msg="输入手机号码格式错误"  }  }  }  })  </script>  </body> |

**3. 计算属性**

什么是: 自己不保存值，每次访问都要根据其他属性计算出来。

何时: 要显示的值，需要根据其他属性动态计算才能获得时(对象中的属性之间是不能相互访问的，就要使用computed)。

如何:

定义计算属性:

computed:{

计算属性名(){

//根据其他属性，动态计算出，并返回结果

return 计算属性名;

}

}

绑定时: 和普通属性完全一样，不要加()

vs methods:

methods: 调用时，必须加()，调用几次，就重复计算几次——效率更低

computed: 绑定时，不加()，即使多次绑定，也只计算一次。vue会缓存计算属性的计算结果，只有计算属性依赖的值发生变化，才重新计算。

总结: 如果更关心值/结果，优先选计算属性

如果更关心操作/执行过程，优先选方法

**4. 自定义指令**

如果13种指令不够用，可自定义指令:

如何: 2步

定义指令:

Vue.directive("指令名", { //起名时，一定不要前缀v-

inserted(elem){ //当拥有该指令的元素被添加到DOM树后，自动执行

//elem: 接住当前标有该指令的DOM元素对象

elem. ... //可以用原生的DOM的API，对elem执行各种想要的DOM操作

}

})

强调：写在new vue() 的外面

使用指令:

<ANY v-指令名>

当该元素被加载到DOM树上后，自动执行指令中的inserted

|  |
| --- |
| 示例：百度搜索自动获取焦点（示例接watch下面示例）  <div id="app">  <input type="text" v-model:value="keywords" v-focus>  <button @click="search">百度一下</button>  </div>  <script>  // 自定义指令  Vue.directive("focus",{  inserted(elem){  elem.focus();  }  })  var vm=new vue ... |

**5. 过滤器**

什么是: 对从模型中绑定来的原始值，进行再加工后，再显示

何时: 如果模型中的原始值不能直接使用

如何: 2步

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 改造后的新值;

})

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 }} //下一个过滤器接到的参数，是上一个过滤器的返回值

:属性="原始模型变量 | 过滤器(实参1,...) | 下一个过滤器"

|  |
| --- |
| 示例：过滤性别  <body>  <div id="app">  <h1>性别1：{{sex1 | sexConvert | sexIcon}}</h1>  <h1>性别2：{{sex2 | sexConvert | sexIcon}}</h1>  </div>  <script>  Vue.filter("sexIcon",function(value){  return value=="男" ? value+"♂" : value+"♀";  })  Vue.filter("sexConvert",function(value){  return value==1 ? "男" : "女";  })  var vm=new Vue({  el:"#app",  data:{  sex1:1,  sex2:0  }  })  </script>  </body> |

**6. axios**

什么是: 基于Promise的专门发送http请求的函数库

何时: 今后在vue中发送ajax请求首选axios

为什么:

(1) 手写ajax四步/封装自定义函数——繁琐、功能不够完善

(2) jQuery: $.ajax() —— Vue中没有jQuery

(3) Vue官方曾经提供了专门发送ajax请求的组件：vue-resource——官方不推荐使用

(4) Vue官方推荐axios作为Vue中标准的发送ajax请求的组件

如何:

下载并引入： axios.min.js

内存中，就多出一个axios对象，有两个方法: get() post()

发送get请求:

axios.get("url",{

params:{

uname: "dingding",

upwd:"123456"

}

}).then(function(res){ //res是经过axios再封装的响应对象，不仅是result

res**.data**...//相当于从前的result

}).catch((err) => { //报错返回的数据

reject(error)

}).finally(() => { // then或 catch执行完都会经过此过程

temp.close();

})

发送post请求:

axios.post("url","uname=dingding&upwd=123456")

**7. 组件**

什么是: 拥有专属的HTML，CSS，js和数据的独立页面区域

何时: 只要一个页面功能，需要反复使用时。

如何:

定义组件的HTML和CSS片段

<template id="tp"> //相当于其他元素设置：display: none

<父元素> //HTML模板片段必须包裹在一个，template下的唯一父元素中

模板片段

</父元素>

</template>

定义组件:

Vue.component("组件名",{ //不要用驼峰命名，而用"单词-单词"，以标签来使用，HTML不区分大小写

template:"ID(#tp)", //找到要反复使用的HTML片段——模板

data(){ //省略 :function，每次重复创建组件时，都自动调用data函数，为本次组件创建一个专属的模型对象

return { 模型数据 } //类似于以前的data，比如：{n:1};

},

methods:{ 方法/事件处理函数 },

computed:{ 计算属性 },

watch:{ 监听函数 }

})

HTML片段都要保存在<template>元素中并其一个唯一的id

使用组件:

其实vue中的一个组件，就是一个可重复使用的自定义标签而已。组件名其实就是一个标签名:

在<div id="app">内部中使用：

<组件名></组件名>

原理：定义组件的HTML片段会被template的HTML片段代替！

|  |
| --- |
| 示例：加减数量  <body>  <div id="app">  <my-counter></my-counter>  <my-counter></my-counter>  <my-counter></my-counter>  </div>  <!-- <div id="tplCounter" style="display: none"> 相当于用template-->  <!-- HTML5中：template专门用于为框架/组件保存一段隐藏的模板片段 -->  <template id="tplCounter">  <div>  <button @click="subtract">-</button>  <span>{{n}}</span>  <button @click="add">+</button>  </div>  </template>  <!-- </div> -->  <script>  Vue.component("my-counter",{  template:"#tplCounter",  data:function(){  return data={n:1};  },  methods: {  add(){  this.n++;  },  subtract(){  if(this.n>1){  this.n--  }  }  },  })  var vm=new Vue({  el:"#app",  data:{    }  })  </script>  </body> |

**day03**

**1. 组件化开发**

什么是组件化开发: 今后一个网页都是由组件组成的

拿到网页，先划分组件，再分组件开发，最后再拼接成一个网页

为什么: 1. 团队协作开发

2. 松耦合

如何:

拿到网页，先划分组件

定义组件

将组件拼回主页面中

组件分类:

(1) 根组件: 一个页面中只有一个的组件：

new Vue({

el:"#app",

data:{ ... }

})

(2) 全局组件: 可出现在任意位置的组件

Vue.component("组件名",{

template:"#tplxxx",

data(){ return { ... } }

})

(3) 子组件: 只能出现在规定父组件内的组件

如何: 2步

将Vue.component降级为普通的js对象

//比如：Vue.component("todo-add", data(){ return { ... }} })

=>var todoAdd={ //对象变量名要用组件名，驼峰命名，js不能用-

template:"#tplTodoAdd",

data(){ return { ... } }

}

在父组件的{}中的components下添加子组件对象：

Vue.component("todo",{

template:"#tplTodo",

data(){ return { ... } },

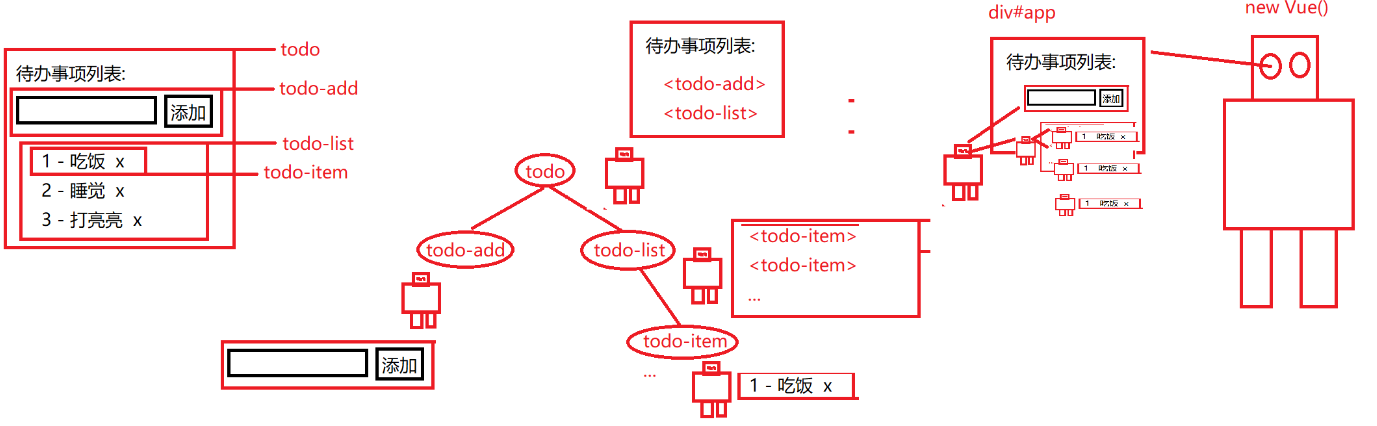
components:{

todoAdd //vue会自动将todoAdd -自动转为-> <todo-add>

}

})

强调：子组件必须定义在父组件之上。



组件间**传参**:

父->子:

问题: 父组件的成员，子组件无权直接使用

解决: 2步

在子组件对象中添加props属性:

var 子组件={

template:"#tpl子组件",

props:**[ "** 自定义属性名 **"** **]** //等效于data:{ ... }

data(){ return { ... } }

}

后果: 2个:

a. 为子组件标签添加了一个自定义扩展属性，用于接收父组件的值。

<子组件 自定义属性="">

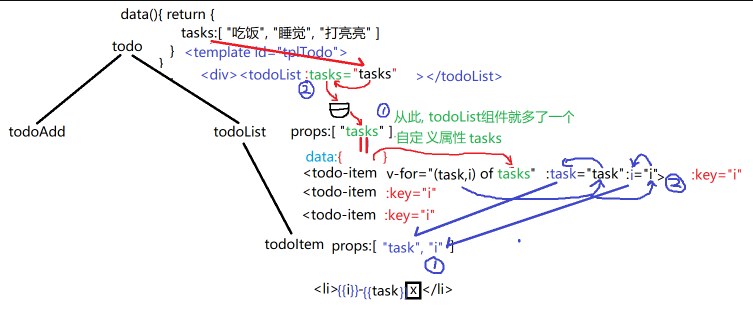
b. 子组件对象中多处一个同名的模型变量，可用于获取外部传入的属性值，在内部执行操作。

在父组件的<template>中

<子组件 **:**自定义属性="父组件的数据"></子组件>

将父组件data中的数据通过绑定方式，传递给子组件的自定义扩展属性。

其实，父子组件用的还是同一个数据对象，任何一方修改，另一方都会受影响。



2. SPA——单页面应用

Single Page Application

什么是: 整个应用程序只有一个完整的HTML页面文件。

单页面应用中所谓的“页面”，其实就是组件；

由一个路由器根据URL地址中的锚点地址不同，选择不同的组件替换。

vs 多页面应用:

单页面应用的优点:

请求次数: 多页面应用每更换一次页面就需要重新发送请求。

单页面应用每更换一次页面，只需要从本地加载组件即可，不用重复发请求。

资源共用: 多页面应用每更换一次页面，都需要重复请求相同的资源。

单页面应用仅在首次加载时请求一次资源文件，之后更换页面，因为不是更换整个html文件，所以共用资源无需重复下载。

效率: 多页面应用，每次更换页面需要重建整棵DOM树，效率低。

单页面应用，每次更换页面无需重建DOM树，仅是替换部分子树片段即可。效率高。

页面过渡效果: 多页面应用不可能实现页面过渡效果。

单页面应用可以实现页面过渡效果。

单页面应用的缺点：

首屏加载慢，要下载全部资源；

已经解决: 在路由中设置延迟加载。

不便于搜索引擎优化。

* 实现单页面应用:

先创建一个唯一的完整的index.html

引入vue-router.js

在div#app中添加一个<router-view>标签，用于为将来要加载的"页面"组件占位

分别创建多个“页面”组件对象和模板

“页面”组件对象必须以子组件形式创建

<template id="tplIndex">

<div>

...

</div>

</template>

<script>

var index={ //创建“页面”组件对象

template:"#tplIndex"

};

</script>

定义路由字典:

路由字典:保存一组路径与组件对象对应关系的数组（运行相当于if，从上往下，依次执行）

var routes=**[**

{ path:" / ", component: "页面"组件对象 } //默认路径

{ path:"/相对路径名", component: "页面"组件对象,

//路由嵌套

children:[

{path:'inbox',component:\*\*\*},

{path:'outbox',component:\*\*\*},

...

]

}

{ path:" \* ", component: "页面"组件对象 } //没有找到路径，显示的网页

**]**

定义路由器对象，将路由字典放入路由器中:

才是真正负责导航/替换页面内容的对象

var router=new VueRouter({ routes })

将路由器对象，加入当前页面的根组件new Vue中

new Vue({

...,

router,

...

})

* 嵌套路由:

何时: 多个页面中包含相同的区域(页头,页脚,导航)时

如何: 2步:

将多个页面共用的部分，保存在一个新创建的组件中

新组件同时包含<router-view>

定义路由字典:

//home: 是保存相同部分的组件

{path:"/",component:home,children:[

{path:"/",component:index}, // ‘/’默认路径

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

{path:"/details/lid",component:details,props:true} // /details props:true—自动将数据放入props中

]},

//login和之前页面没有相同部分

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

加载时:

#/ /=="/" 先加载home页头

/=="/" 再加载index替换home中的<router-view>

#/details /=="/" 先加载home页头

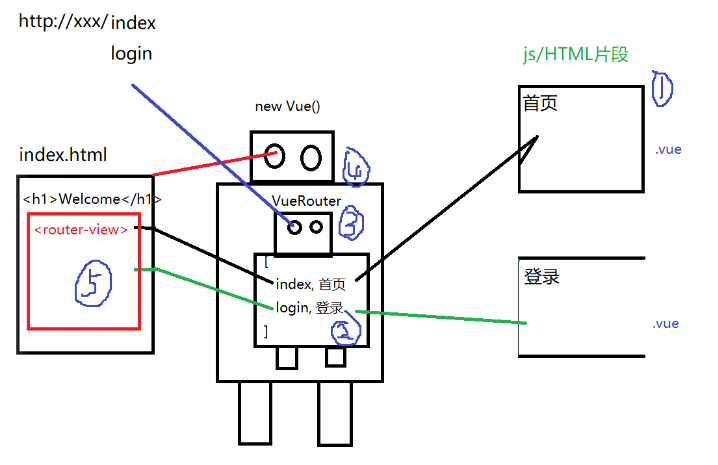
/details="/details" 再加载details替换home中的<router-view>

#/login /=="/" 先加载home页头

没有/login

撤掉home页头

/login="/login" 只加载login



* 路由跳转:

HTML中: 所有a换成<router-link to="路径"></router-link>

运行时，自动翻译为<a href="#/路径"></a>

js中: this.$router.push("路径")

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

要执行跳转这个动作，只能借助$router

凡是以$开头的，都是Vue内置的对象

* 路由传参:

2步:

在路由字典中配置支持参数的相对路径:

{ path:"/details/:lid",component:Details,props:true }

/:lid: 将来请求details页面时必须携带一个lid参数

比如: /details/5

在details页面中获取参数:

this.$route.params.lid

如果定义了props:true，lid参数会自动传递给props中同名的属性: props:["lid"]

this.lid=5

跳转时: <router-link to="/details/5"

3. 组件**生命周期**:

什么是: 一个组件创建的过程

包括: 4个阶段:

create: 创建阶段

创建组件对象，以及模型数据对象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

created(){

this.axios.get(“url”,

{ //this->当前组件对象

params:{ 参数 }

}

).then(res=>{ //箭头函数，内外this通用

this.模型变量=res.data.属性名

this.products=res.data;

})

}

没有虚拟DOM，暂时不能操作DOM元素，比如让元素获得焦点。

mount: 挂载阶段

扫描页面，生成虚拟DOM树，绑定数据到真实的DOM树上。

beforeMount() 挂载阶段开始前自动触发

mounted() 挂载后自动触发

有data 可以axios发送ajax请求

有虚拟DOM，可执行DOM操作

mounted(){ //即有data，又有DOM树

... //也可以写ajax请求

... //可以执行DOM操作

}

update: 更新阶段

当组件的模型数据被更新时自动触发。

beforeUpdate()

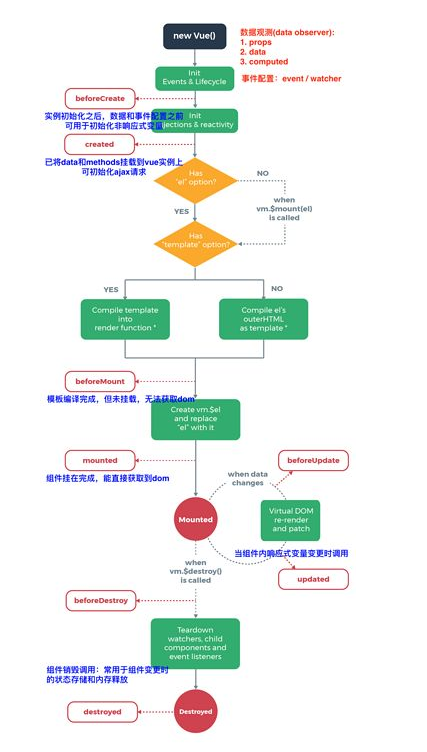
updated()

destroy: 销毁阶段

除非主动调用专门的destroy函数，才能触发destroy阶段。

beforeDestroy()

destoryed()



1. 脚手架:

什么是: 前人总结的包含优秀开发经验的半成品项目

为什么: 避免重复劳动

何时: 今后只要使用框架开发，都用脚手架代码

如何: 2步:

安装命令行工具——用于生成脚手架代码的工具

npm i -g @vue/cli

用命令行工具生成脚手架代码——创建项目

vue create 项目名

babel: js代码翻译工具

将浏览器不认识的新标准的js代码，翻译为浏览器认识的ES5的等效代码，比如

ES6中也定义了模块：

抛出模块: export default { ... }

等效于node中的module.exports={ ... }

引入模块: import from "@/.../模块名"——@指的是src文档

等效于node中的require("模块名")

浏览器不认识，只能请babel翻译为ES5的等效代码。

结果: 在当前运行命令的目录创建了一个项目文件夹，其中包含了脚手架代码

测试: cd 项目名 进入项目文件夹  
 npm run serve

1. 启动临时开发服务器（development Server），宿主编译后的静态页面

2. 编译(compile / build)脚手架代码中浏览器不认识的内容为认识的内容

2. 在脚手架代码中填写个性化的内容

问题: 脚手架代码不包含axios

解决: 安装和配置axios

1. 在项目本地安装axios: npm i --save axios

2. 将axios对象放入vue的原型对象中:

main.js是整个vue项目的唯一根组件new Vue

所有需要的模块或对象，都要在new Vue之前配置好

坑: axios默认不带钥匙到服务器，也就无法打开自己的柜子，访问自己的信息。

解决: 在main.js中要求axios必须携带钥匙

axios.defaults.withCredentials=true; //让axios的请求携带验证信息到服务端

Vue.prototype.axios=axios; //在vue原型对象中添加axios对象（包含get()和post()两个方法）

3. <style scoped>组件样式</style>

scoped：自定随机在样式前面添加前缀，防止样式重名产生冲突。

4. 共用组件的使用

//1. 引入header.vue组件对象，并起名

import myHeader from "@/components/header.vue"

export default {

//2. 设置myHeader组件对象为当前组件的子组件

components:{

myHeader //Vue会自动翻译为<my-header>

}

 }

**Vue-2节**

1：组件库

1.1：vue ui 移动端组件库

-VUE 前端开发框架

-VUE-CLI 脚手架用于开发VUE项目

-VUE-CLI 脚手架可用于开发PC项目也开发移动项目

-MINT-UI 第三方移动端组件库

-MUI 第三方移动端组件库

-以上两个组件混用

1.2：软件开发有针对性

--PC端项目 --移动端项目

-屏幕宽度:970PX < w 970PX > w

-操作方式:鼠标和键盘 手指

-触发事件:单击 tap

1.3: vue ui

vue 开始针对PC端框架

(1)饿了么团队开发

移动组件库 mint-ui

PC组件库 element ui

(2)mui移动库组件库

开源团队:创建原生手机相同移动端组件库

#下载vue cli 脚手架

ftp: vue\_app\_00.zip

1.4：vue ui 组件库下载和使用

-练习或者测试环境: 在mint-ui官网下载 js/css

在项目中使用即可

-在脚手架中使用组件库:

npm i mint-ui -S #下载mint-ui

/node\_modules/mint-ui/...非常多js

#组件库特殊情特殊: mui 独立组件库与vue关系不是很紧密.

#vue\_zpp\_00/src/lib/mui

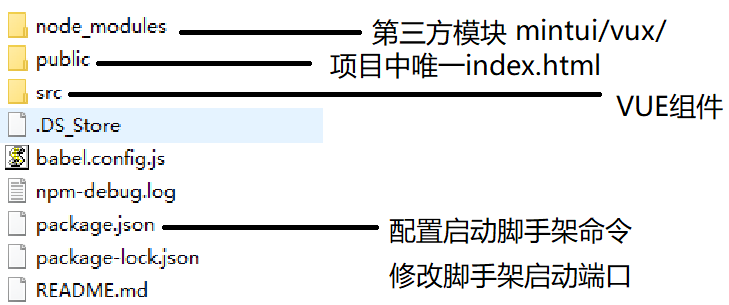
1.5：vue cli 脚手架

-vue项目是否一定要使用脚手架

? vue cli 帮助管理项目中使用js 文件

? vue cli 帮助打包 app.js index.html项目

-vue cli 项目目录结构



(1)node\_modules/mint-ui 移动端组件库

/vuex 全局共享数据

(2)public/index.html 项目中唯一html

-jquery.js public/jquery.js

-index.html

<head>

<script src="jquery.js"></script>

</head>

(3)package.json

修改启动脚手架指令:

scripts下的dev改为 -->serve(启动命令为：**npm run serve**)

修改启动脚手架端口:

port(端口)-->8080

(4)src代码目录结构



1.6：vue cli 脚手架-开发第一个组件

组件= html 模板+style+js (组件：就是一个网页)

创建组件最基本流程

(1)在src/components/Exam01.vue

-模板 html

<template>

<div class="Exam01">

</div>

</template>

-script js

<script>

#export 导出 default 默认{}

#data 数据在模板中使用数据

#return 返回当前data对象模板使用

export default {

data(){

return { }

}

}

</script>

-style css

<style scoped> scoped：作用域，会自动在在样式类前面加上前缀，保证样式不会重名

…

</style>

(2)在src/router.js 为新创建组件指定运行路径

-引入自定义组件

import Exam01 from "./components/Exam01.vue" 路径或者为：@/components/Exam01.vue(@指的是src)

-为组件配置访问路径

{path:'组件访问路径',component:组件对象名称 },

{path:'**/**Exam01',component: Exam01}

(3)如果访问自定义组件

①启动脚手架 vue\_app\_00项目下运行：

npm run serve

查找文件package.json

②浏览器地址栏输入：

http://127.0.0.1:8080/#/Exam01

* 常见错误

1: Failed to compile with 1 errors

\* ./components/Exam011.vue in ./src/router

编译出错:如果在写书程序拼写错误

出错原因 router.js 文件中

1.7：如何使用组件--mint-ui(饿了么团队)

mint-ui移动端组件库与vue关系非常紧密

(1)下载mint-ui 组件

npm i mint-ui -S #下载指令 位置在node\_modules/mint-ui

(2)在src/main.js 引入组件库并且对其配置

-mint-ui 支持二种引入方式

①全部引入

import MintUI from "mint-ui"

注册：

Vue.use(MintUI)

②按需引入:引入组件库某一部分的组件，并放在{ }中

import **{**Header,Button**}** from "mint-ui"

-将Header组件注册Vue实例对象

Vue.component("标签名称",组件对象)

示例：

Vue.component(Header.name,Header); // Header.name –>默认标签名称，等效于"mt-**h**eader"

Vue.component(“mt-button”,Button);

(3)在模板中直接使用

<mt-header title="标题中的文字" fixed></mt-header>

fixed：固定在最顶端

练习:

按需引入Button

(1)按需引入 main.js

import {Header,Button} from "mint-ui"

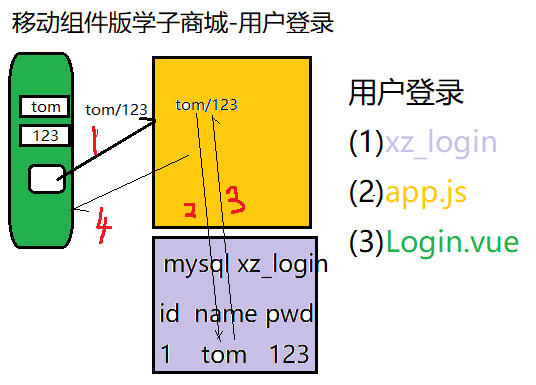
(2)全局注册 main.js

Vue.component(Button.name,Button);

(3)在模板中直接使用

<mt-button size="large" type="primary">登录按钮</mt-button> //尺寸大的，颜色主色

1.8：vue-ui 学子商城---用户登录



(1)数据库 (表xz\_login ) 登录表

id INT PRIMARY KEY AUTO\_INCREMENT

name VARCHAR(25)

pwd VARCHAR(32)

#密码不允许数据库管理人员看到

#密码数据需要加密保存

#加密 原来密码 123

# 加密码 202cb962ac59075b964b07152d234b70

#mysql有一个加密函数 **md5**('123')

#用户登录

-数据库中

id name pwd

1 tom 202cb962ac59075b964b07152d234b70

-用户在表单中输入 tom/123

-服务器端程序

先对密码进行加密，再和数据库已经加密的密码作比较。

md5(123)==数据库密码202cb962ac59075b964b07152d234b70

-两个密码相等代表登录成功

#如果担心md5安全性

解决方案一:多次加密 md5(md5(md5('123')))

解决方案二:用户密码8位以上数字大写字母小写字母

再加一个特殊字符+=\_$

(2)服务器程序 (app.js)

-负责加载服务器模块

mysql/express/cors/express-session

-处理各种不同请求

(3)/login GET 传递二个数据 name=tom&pwd=123

功能分析:获取客户端用户名和密码，去数据查询

是否有匹配记录，如果匹配成功 "登录成功"

如果匹配不成功"用户名或密码有误"

-参数: name='tom'/pwd='121'

-sql语句:

SELECT id FROM xz\_login WHERE name = ? AND pwd = md5(?)

pool.query(sql,['tom','121'],(err,result)=>{

if(err)throw err;

result.length==0 查询返回结果为数组

})

-返回数据格式json:

{code:1,msg:"登录成功"}

{code:-1,msg:"用户名或密码有误"}

-检测服务器是否运行正常

-启动服务器端程序 node app.js

-打开浏览器在地址栏输入程序地址回车

http://127.0.0.1:3000/login?name=tom&pwd=121

(4)脚手架组件 (Login.vue)

-创建空组件 src/components/tabbar/Login.vue

-为组件配置访问路径 router.js /Login

-启动脚手架

http://127.0.0.1:8080/#/Login

**day02**

2.1:学子商城--用户登录

-为登录按钮绑定点击事件 @click="btnClick"

为用户和密码双向绑定

-btnClick

获取用户名和密码

发送ajax

this.axios.get();

2.2:提示用户信息 登录成功 用户名格式不正确

mint-ui Toast("提示消息");

-由于此功能不是每个组件都使用

-在main.js 引入注册全局组件,所有自定义组件都要使用

(1)局部组件使用Toast

①在当前组件(export default抛出之前)

import {Toast} from "mint-ui"

②Toast("消息")

(2)如果是已经引入整个mint-ui组件，toast就属于vue，调用vue的属性#$toast就行

this.$toast("消息")

2.3:发送ajax 使用axios第三方模块

-下载第三方模块 npm i axios

-在main.js 引入模块

import axios from "axios"

-在main.js 配置跨域访问保存session值选项

axios.defaults.withCredentials=true

-在main.js 将axios配置vue实例中

Vue.prototype.axios = axios

-在当前组件库中使用axios

this.axios.get(url).then(result=>{ })

* 常见错误:

- Access-Control-Allow-Origin 跨域访问

脚手架8080 --> node.js 3000 端口不同

解决问题

-node.js下载第三方模块 cors

-app.js 引入模块

const cors = require("cors");

-在服务器配置允许来自8080所有请求

app.use(cors({

origin:["http://127.0.0.1:8080","http://localhost:8080"],

credentials:true

}))

origin:允许访问列表

credentials:每次访问验证

2.4:学子商城--商品列表-数据库表[多表查询]

1:xz\_laptop lid[商品编号]/lname[商品名称]/price[价格]

2:xz\_laptop\_pic laptop\_id ,md[商品中图]

**多表查询**(套路)

-需要查找的列

SELECT l.lid,l.lname,l.price,p.md

-为每张表起一个别名

from xz\_laptop l,xz\_laptop\_pic p

-查找两张表关系(二张表某一列 相等 大于 小于)

WHERE l.lid = p.laptop\_id

最终结果

SELECT l.lid,l.lname,l.price,p.md(直接写列，使用\*运行更慢)

from xz\_laptop l,xz\_laptop\_pic p

WHERE l.lid = p.laptop\_id

GROUP BY l.lid

LIMIT 0,7;

小结:多表查询方法使用三步套路

多行相同数据只取第一个 GROUP BY

分页 LIMIT((页码-1)\*页大小,页大小)

-node.js 程序学子商城--商品列表

获取脚手架发送二个数据 第几页 一页几行

查询数据库内容并且将结果返回

-参数 pno[第几页] pageSize[一页几行]

默认值 pno=1 pageSize=4

-sql 现成

-json {code:1,data:[..]}

练习1:第一页与第二页追加

data(){list:[]}

list:显示商品列表追加

var rows = this.list.concat(result.data.data);

this.list = rows;

解决使用数组拼接

var a = [1,2,3];//一页

var b = [4,5,6];//二页

var c = a.concat(b); //[1,2,3,4,5,6]

练习2:点击加载一页 1 2 3 4

data(){pno:1} //保存页码

loadMore(){ //加载下一页

this.pno++;

url = "http://127.0.0.1:3000/getGoodList?pno=";

url+=this.pno;

}

|  |
| --- |
| -当需要同时查询商品lid，lname，price，又要查询商品的总个数时？  //7:功能二:商品列表 分页  app.get("/getGoodList",(req,res)=>{  //1:获取二个参数,只要分页功能  // pno 页码  // pageSize 页大小  var pno = req.query.pno;  var pageSize = req.query.pageSize;  //1.1:为页码和页大小参数设置默认值  if(!pno){  pno = 1;  }  if(!pageSize){  pageSize = 4;  }  //1.1:创建变量，保存发送给客户端数据  var obj = {code:1};  //1.2:创建变量保存进度  var progress = 0;  //2:创建一条sql语句  var sql =" SELECT lid,lname,price";  sql+=" FROM xz\_laptop";  sql+=" LIMIT ?,?";  //2.1:创建一个变量offset 起始行数  var offset = (pno-1)\*pageSize;  //2.2:创建一个变量ps 一页数据  var ps = parseInt(pageSize);  //3:执行sql语句  pool.query(sql,[offset,ps],(err,result)=>{  if(err) throw err;  progress+=50;  obj.data = result;  //4:获取数据库返回结果  //5: send：发送数据+不再发送数据  //res.send({code:1,data:result});  if(progress==100){  res.send(obj);  }  })  //6:计算总页数  //6.1:创建sql查询总记录数  var sql =" SELECT count(lid) AS c FROM";  sql+=" xz\_laptop";  pool.query(sql,(err,result)=>{  if(err)throw err;  progress+=50;  // var pc = Math.ceil(result[0].c/pageSize);  var pc =result[0].c;  obj.pc = pc;  if(progress==100){  res.send(obj)  }  });  }); |
| ②使用回调函数，把另一个查询放在肚子里  pool.query(sql,[offset,ps],(err,result)=>{  if(err)throw err;  obj.data = result;  var sql =" SELECT count(lid) AS c FROM";  sql+=" xz\_laptop";  pool.query(sql,(err,result)=>{  if(err)throw err;  var pc =result[0].c;  obj.pc = pc;  res.send(obj)  });  }) |

2.5:学子商城--商品列表-脚手架

-分析设计外观

最外层父元素弹性布局

子元素弹性布局

子元素宽度 49%

-创建空组件 src/componet/tabbar/GoodList.vue

-指定组件访问路径 /GoodList

#项目很多图片:商品图片,广告..

#所有图片都保存服务器/public下 node.js

vue中使用时，通过服务器路径：http://127.0.0.1:3000/1.png

**day03**

1:完成学子商城--首页(mint-ui/mui)

#主要完成功能 脚手架

-创建首页组件 src/componets/tabbar/Home.vue

-指定首页路径 /Home

(1)顶部导航条

(2)轮播图

(3)九宫格

(4)底部导航条tabbar

1.1:完成学子商城--首页-顶部导航条

-mint-ui组件库 header 组件

-main.js 引入组件

import {Header} from "mint-ui"

-注册组件

Vue.component(Header.name,Header)

-在模板中调用

<mt-header></mt-header>

属性:

title="显示文字"

fixed 固定在顶部

1.2:完成学子商城--首页-**轮播图**slider

-mint-ui组件库

-引入组件 main.js

import {Swipe,SwipeItem} from "mint-ui"

-注册组件 main.js

Vue.component(Swipe.name,Swipe);

Vue.component(SwipeItem.name,SwipeItem);

-在模板调用 Home.vue

<mt-swipe>

<mt-swipe-item> 1 img..</mt-swipe-item>

...

</mt-swipe>

#属性

:auto="4000" 自动播放的间隔时间

:auto="0" 取消轮播

:show-indictors=”true/false” 轮播指示器显示/隐藏

#注意: 轮播图默认高度为 0

1.3:vue ui组件库-- Lazy load 图片**懒加载**组件

(1)按需引入组件 main.js

import {Lazyload} from "mint-ui"

(2)全局组件注册 main.js

Vue.use(Lazyload)

(3)在组件中调用

<ul>

<li v-for="item of list">

<img v-lazy="item.img\_url" />

<!--img src="img/1.jpg" lazy="loading" /-->

</li>

</ul>

#元素背景设置暗色

#组件对样式特殊要求

img[lazy=loading]{

width:40px;

height:300px;

margin:auto;

}

#mint-ui 组件有一个BUG 使用lazyLoad

#如果你采用按需要加载不生效 lazy="loaded"

#临时使用完整引入mint-ui库

1.4:vue ui组件库--**MUI组件库**

MUI 开源移动端组件库--外观精美

MUI 独立组件库与VUE没关系，只有三个文件

\*mui.css 样式文件

\*mui.ttf 图标字体

\*mui.js js文件

在脚手架中使用MUI组件

(1)将css文件和字体文件 src/lib/mui/css

(2)在main.js 引入css文件

import './lib/mui/css/mui.css'

import './lib/mui/css/icons-extra.css'

(3)在任意组件都以使用mui组件内容

(4)查找指定mui组件文件examples/buttons.html

<button type="button" class="mui-btn">

默认

</button>

(5)vue ui组件库--MUI组件库-图标

查找指定mui组件文件 mui-master\examples\hello-mui\examples/icons.html

examples/ icons-extra.html

1.5:vue ui组件库

-node.js 效率非常高(非阻塞/异步)

-java 适合开发大型软件项目(阻塞/同步)

-php 中小规模互联网项目(阻塞/同步)

-.net 中小规模互联网项目(阻塞/同步)

同步:有序,效率低

异步:无序,效率高

1.6:vue ui组件库-Home 主组件

(1)**九宫格**(MUI) examples/grid-default.html

-创建空组件Exam05.vue

-为组件绑定路径 /Exam05

-添加九宫格

更换图标，用<img src=“...”>替换span

(2)为九宫格图片添加宽度和高度样式 120\*120

-在移动端元素宽度高度是多少?

-iphone 4 技术标准

-Google Android

-iphone 6/7/8 iphone6 plus

-前端设计师 ps 图片——物理像素

-前端开发工程师: html/css/js图片功能网页——逻辑像素

物理像素120(iphone6) = DPR\*逻辑像素

DPR:iphone 6 7 8 [2]

DPR:iphone6 plus [3]

结果:物理像素 120 = 2 \* 60;

(3)Tabbar(MUI)

**day04**

2.1:**组件之间跳转与数据传递(传参)**

(1)组件之间跳转

方式一:标签

<router-link to="组件路径">文字</router-link>

示例:Exam01 -> Exam02

<router-link to="/Exam02">跳转组件</router-link>

方式二:编程

this.$route**r**.push("组件路径");

示例:

<img src="1.jpg" @click="jump"/>

function jump(){

this.$router.push("组件路径")

}

(2)数据传递

|  |
| --- |
| 组件中传递参数到服务器  methods:{  loadNewsList(){  /\*①url拼接参数  var url=’ http://127.0.0.1:3000/newslist’  url+=` ?pno=this.pno&pageSize=this.pageSize`  \*/  //②直接带参数  this.axios.get("http://127.0.0.1:3000/newslist",  {  params:{  pno:this.pno,  pageSize:this.pageSize  }  }  )  .then(result=>{  this.list=result.data.data;  this.pc=result.data.pc;  })  }, |

①第一种：?后面参数传递

A组件发送数据:

方式一跳转:<router-link **:**to="`/B?nid=${id}`"> ... id为变量

方式二跳转:this.$router.push("/B?nid=5")

B组件接收数据:

this.**$route**.query.nid

#最早接收参数 created(){ }

|  |
| --- |
| export default {  data(){  return {  lid:""  }  },  created(){  // this.lid=this.$route.query.lid  this.lid=this.$route.params.lid  console.log(this.lid);  }  } |

②第二种：/后面参数传递

在组件router.js中配置组件的路径是放置需参数

{path:'/Exam06/:lid',component:Exam06,props:true}

在组件xxx.vue中

|  |
| --- |
| export default {  props:["lid"], //接url中的lid参数  data(){  return {...}  },  method:{  load(){  this.axios.get(  "http://localhost:3000/details",  {  params:{ lid:this.lid }  })  .then(res=>{  //this->当前组件对象  console.log(res.data);  })  }  }  } |

2.2:学子商城--**新闻列表**

(1)分析功能:创建新闻列表组件

-创建新组件 src/component/home/NewsList.vue

-为组件指定路径 /NewsList

-修改组件外观 mui/.../examples/media-list.html 83~92行/ul

-组件创建成功后发送第一页请求

-点击组件"加载更多"按钮 发送下一页请求

(2)学子商城--新闻列表-日期格式错误

网页通常日期格式货币格式不满足要求使用过滤器

解决问题

格式:

\*main.js创建全局过滤器其它任意组件都可以直接调用

Vue.filter("过滤器名称",function(原始数据格式value){

①var newValue=new Date(value).toLocaleString(); //对原始数据格式处理

return newValue; //返回字符串

②var now = new Data(val);

//1:获取年份 月份 日期

var y = now.getFullYear(); 获取四位年

var m = now.getMonth()+1; 获取月份[0~11]

var d = now.getDate(); 获取日期

//2:获取小时 分钟 秒

var h = now.getHours();

var mi = now.getMinutes();

var s = now.getSeconds();

return `${y}-${m}-${d} ${h}:${mi}:${s}`

})

\*在模板中直接调用过滤器(元素上加v-cloak，防止代码外漏)

{{item.ctime | 过滤器名称}}

(3)学子商城--新闻详细--评论列表(子组件)

-创建空组件 src/component/home/NewsInfo.vue

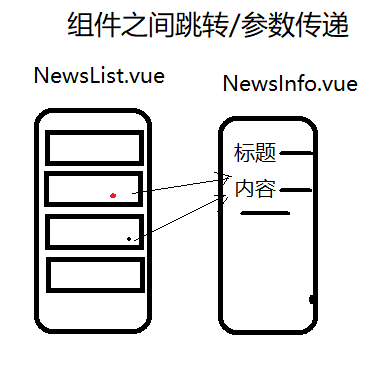
-为此组件绑定路径 /NewsInfo

-显示新闻详细信息 mui/.../examples/card.html

-当用户点击新闻列表中某条新闻

跳转NewsInfo.vue 组件nid=3

-将当前新闻编号传递NewsInfo.vue组件



(4)学子商城--新闻详细

技巧1:SQL

var sql = "INSERT INTO xz\_comment VALUES(null,?,?,now())";

**now()** 当前时间

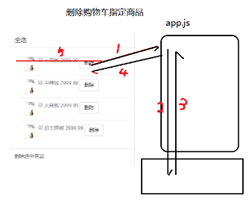
问题：如果1年项目升级,在表xz\_comment添加了3列——可能会报错

解决：var sql = "INSERT INTO xz\_comment(id,nid,content,ctime)";

sql+=" VALUES(null,?,?,now())";

技巧2:post请求 vue 使用ajax库 axios

#原因 axios 处理post请求参数有一个内部问题.

 #第三方式 qs 将js对象转换字符串

①在main.js 引入qs 模块,将模块注册Vue实例

import qs from "qs"

Vue.prototype.qs = qs;

②发送post数据将数据先创建js对象，再用qs转换字符串

var postData = this.qs.stringify({

nid:3,

content:"..."

});

this.axios.post(url,postData).then(...)

**day05**

1.1:vue元素与处理函数之间**数据传递**

(1)数据传递方式一:通过函数参数传递

<div @click="jump(10)">联想笔记本</div>

jump(pid){

console.log(pid);

}

注意:很多框架不支持以上这种方式

(2)数据传递方式二:通过自定义属性传递

<div @click="handle2" :data-pid="10">联想笔记本</div>

handle2(e){

#e 事件对象:当用户点击操作，浏览器创对象

#e.target = div 触发事件对象

# dataset 自定义属性

#e.target.dataset.pid 属性保存所有定义属性值

console.log(e.target.dataset.pid); -->10

}

#小结:以上两种方式建议使用第二种

(1)通用:很多框架都可以使用

(2)传递较多数据

1.2:mui组件库-**卡片**

card.html 很多风格

(1)带页头页脚

(2)带图片

card中可以套用media组件，用来显示详情。

1.3:mui组件库-**数字按钮**

numbox.html 数字按钮

(1)完成数字按钮需要 mui.css 支持

(2)完成数字按钮操作 mui.js 支持

#如果加载lib/mui.js 在脚手架中 报错

#mui.js 很多非严格模式代码,脚手架默认运行严格

式模式

#解决简单方式

(1)将mui.js 复制脚手架 public目录

(2)在 public/index.html 在header添加mui.js

2.1:学子商城--**购物车**(重点\*\*\*\*\*)

购物车:保存用户喜欢商品

(1)将商品添加至购物车

(2)查询购物车列表

(3)删除购物车中商品

2.2:学子商城--购物车-购物车列表

-xz\_cart [id;lname;price;count;uid]

id 购物车编号 INT

lname 商品名称 VARCHAR(255)

price 商品价格 DECIMAL(10,2)

count 商品数量 INT

uid 用户编号 INT

1 华为笔记本 3999.99 1 1

2 华为笔记本 999.99 3 2

注意:计算机语言，小数计算有时候产生误差

2.00-1.99=0.01; 你认为结果

2.00-1.99=0.000000000000001; 结果

解决方案:

(1)第一种高精度浮点类型 DECIMAL(10,2)

(2)第二种将货币单位从元转分保存

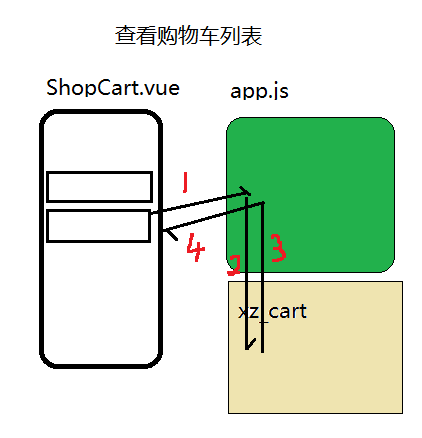
2.00-1.99=?

使用INT 保存 2.00=200分 保存

保用INT 保存 1.99=199分 保存

200-199=1/100=0.01 显示给用户

-app.js



(1)uid=1 保存服务器

(2)sql SELECT id,lname,price,count FROM xz\_cart WHERE uid = ?

(3)json {code:1,data:[]}

-ShopCart.vue

(1)创建空组件 tabbar/ShopCart.vue

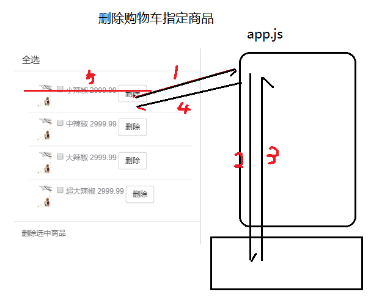
(2)指定路径 /ShopCart

(3)创建标签 mui card.html 页眉页脚30~38

(4) mui media\_list.html 左侧图片商品信息

(5)当组件创建成功后获取购物车列表

2.3:删除购物车中指定商品



服务器端程序:app.js

参数:id

sql:DELETE FROM xz\_cart WHERE id = ?

json:{code:1,msg:"删除成功"}

注意:node.js 执行sql后 result

(1)SELECT ... result=[]

(2)DELETE;UPDATE;INSERT result={ affectedRows :0}

affectedRows:0 执行完成sql影响行数

2.4:购物车[列表/删除指定购物车中商品内容]

分析功能:临时保存用户喜欢商品

将用户商品保存购物车表中

-数据库表 xz\_cart[id,uid,pid,pname,price]

-app.js

\*参数 id 购物中编号

\*sql DELETE FROM xz\_cart WHERE id = ?

\*json {code:1,msg:"删除成功"}

-ShopCart.vue

axios.post(....)

#结论:如果有数据需要传递事件处理函数

#有数据在当前元素添加自定义属性

#<input type="text" :data-idx="item.id"

# :data-age="item.age"

#@click="btnHandle"

btnHandle(e){

e.target.dataset.age;

e.target.dataset.idx;

}

<input type="button" @click="removeItem"

value="删除"/>

removeItem(e){

e事件对象:浏览器自动创建对象(包含与事件相关)

target 触发事件对象

e.target.dataset.id

}

2.5:学子商城--购物车--删除多个商品

-数据库:sql xz\_cart[id,uid,pid,pname,price]

-node.js app.js

\*参数 ids:"4,6,9"

\*sql DELETE FROM xz\_cart WHERE id IN (ids);

\*json {code:1,msg:"删除成功"}

http://127.0.0.1:3000/removeItem?ids=5,6,8

-ShopCart.vue 传递参数多

<input type="checkbox" :checked="item.cb" :data-id="item.id" />

#分析功能删除选中多个商品

#this.list = [{cb:false,id:1,pname,price},

{cb:true,id:2,pname,price},];

函数结果拼接字符串 "1,3" 发送请求

解决问题一:商品选中状态

-cb当前商品选中状态:false 没选中/true 选中

-商品加载时默认false

-loadMore 获取list 先循环result.data.data

①对象先添加cb属性值false,再保存list

var rows=result.data.data;

// 给每一个对象中添加一个元素

for(var item of rows){

item.cb=false;

}

this.list=rows;

②还可以使用：this.$set(item,’cb’,false)

#顺序(注意)

3.1:小结购物车组件

分析总体功能:

(1)要求用户登录成功后操作购物车

#用户登录成功,服务器保存一个值 uid=1

#购物车保存用户喜欢商品

#用户操作

-将一个商品添加至购物车

-显示购物车中商品列表

-删除购物车中某一个商品

-删除购物车中多个商品

-方便用户:全选按钮

3.2:小结购物车组件-列表

分析:当用户打开购物车页面显示他添加商品列表

app.js

表:xz\_cart[id,uid,pid,pname,price]

参数(没有): uid 保存服务器

sql:SELECT id,uid,pid,pname,price FROM xz\_cart

查询结果-->json:{code:1,data:[]}

ShopCart.vue

data:{ list:[]}

methods:{loadMore(){.... this.list}}

什么时候调用

created(){this.loadMore()}

3.3:小结购物车组件-删除指定1件商品

表:xz\_card[id,uid,pid,pname,price]

app.js

参数:id

sql: DELETE FROM xz\_cart WHERE id = ?

json: {code:1,msg:"删除成功/失败"}

ShopCart.vue

<input type="button" value="删除" :data-id="item.id" @click="removeItem"/>

removeItem(e){

1:获取当前购物车id e.target.dataset.id(获得当前点击对象id)

2:发送ajax请求 this.axios.post(...)

}

3.4:小结购物车组件-删除多个商品

分析:用户选择某一些商品删除 [\*]

点击按钮[删除选中商品]

app.js

表:xz\_cart [id,uid,pid,pname,price]

参数:删除多个商品,获取多个商品id "1,3,5"

var ids = req.query.ids; //1,3,5

sql:"DELETE FROM xz\_cart WHERE id IN ("+ids+")"

json:{code:1,msg:"删除成功"}

ShopCart.vue

修改loadMore 加载购物车列表

loadMore(){

#为购物车列表中对象添加属性 item.cb=false

#当前商品前 复选框状态

}

<input type="checkbox" :checked="item.cb" />

添加modifyItem

modifyItem(e){

#当用户选中某个商品,修改list数组对应对象

#item.cb=true

var i=e.target.dataset.i //当前元素数组下标

}

<input type="checkbox" @click="modifyItem" :data-i="i"/> //每次循环都记录当前下标

添加delItem 删除选中商品按钮

delItem(){

this.list[{id:1,cb:true},{id:2,cb:false},{id:3,cb:true}]

"1,3,4," 拼接的字符串后面有逗号

substring(起始下标,结束下标this.list.length-1);

this.axios(url)

}

3.5:小结购物车组件-全选

--添加 selectAll 全选

<input type="checkbox" @click="selectAll" />全选

selectAll(e){

var rs = e.target.checked; //当前点击对象的checked状态-->true/false

for(var item of this.list){

item.cb = rs;

}

}

--全选按钮添加事件 selectAll 全选/清除判断

<input type="checkbox" @click="selectAll" :checked="allcb" />

data:{

allcb:false //希望全选状态和变量绑定

},

selectAll(e){

var rs = e.target.checked;//复选框状态

allcb = rs; //复选框状态变量同步

},

modifyItem(){

商品前复选框:需要计算操作(如果商品属性cb的true个数等于this.list的个数，就要全选)

1:数组个数 14

2:数组中选中元素个数 cb==true 14

if(size1 == count){

this.allcb = true;

}else{

this.allcb =false;

}

}

-常见错误

(1)node.js NaN id=undefined

原因:数据从组件没有发送node.js

post请求

解决:F12->NETWORK->Header->response

Header标签

FormData post请求发送数据格式与内容

id:3

**day06**

1.1:mint-ui 组件库 MessageBox **操作框[确认框]**

-作用:如果用户执行一个重要操作，再次提醒

-局部引入时使用

import {MessageBox} from "mint-ui"

MessageBox.confirm("确认执行次操作?").then(action=>{ // action用户选择确认返回字符串 "confirm"

如果用户选择确认执行的操作;

}).catch(result=>{ //result 用户选择取消返回字符串 "cancel"

如果用户选择取消执行的操作;

});

注意:哪些操作:支付;删除;获取头像;

以上操作需要用户再次确认

-全部引入时使用

this.$messagebox.confirm(`确定删除选中的商品？`).then(action=>{

如果用户选择确认执行的操作

}).catch(result=>{

如果用户选择取消

})

2.1:**session** 概念

session (会话):一个操作过程

①会话开始：当用户打开某个网站的第一个页面，话开始可以创建session对象，将数据保存session对象中。

②会话中： 用户在此网站内打开多个页面

③会话结束：用户关闭浏览器

session对象(保存在服务器端)

作用:在服务器保存用户信息，跟踪用户操作记录。

每一个用户一个session对象，session对象保存用户编号。

如果用户关闭浏览器：会话结束同时，session对象中保存数据也就丢失，session也失效。

2.2:完成以上操作 sesion

(1)服务器修改

-安装npm i express-session #node\_modules/..

-引入模块 express-session

const session = require("express-session");

-配置模块对象

server.use(session({

secret:"密钥字符串", //为了保证安全session数据，生成128位密码

resave:false, //是否每次请求都修改session数据

//如果添加了路由，resave->true

saveUninitalized:true //是否保存session对象初始化数据

cookie : {

maxAge : 1000 \* 60 \* 3, // 设置 session 的有效时间，单位毫秒

}

}));

(2)脚手架修改

import axios from "axios"

axios.defaults.withCredentials=true //让axios的请求携带验证信息到服务端

Vue.prototype.axios = axios

(3)如何操作

①将数据保存session对象(session自有属性id)

req.session.uid = 5; 登录成功

②从session对象获取数据

var uid = req.session.uid; 删除商品

③提前将 session对象中数据清除: 退出

req.session.uid = null;

示例:

-用户登录成功后将uid保存session 中

app.js

server.get("/login",(req,res)=>{

// 将用户登录成功凭据保存到session对象中

// ①获取uid，请求查询返回的数组中

var uid=result[0].id;

// ②将uid保存到session中

req.session.uid=uid;

})

-用户在操作购物车时，需要验证session uid

app.js

server.get("/getCart",(req,res)=>{

//使用购物车前，从session中拿到uid并进行判断

if(!req.session.uid){

res.send({code:-1,msg:"请登录",data:[]});

return;

}

});

ShopCart.vue

-1 请登录

this.axios.get("http://127.0.0.1:3000/shopcart")

.then(result=>{

//从session中获取用户的uid，并进行判断是否存在

if(result.data.code==-1){

this.$toast("请先登录")

}

//先创建一个新数组保存从服务接收的数据

var rows=result.data.data;

//再给每一个对象中添加一个元素

for(var item of rows){

item.tf=false;

}

//最后将添加好元素的数据给当前组件的模型数据

this.list=rows;

})

3.1:发表评论

分析功能:新闻详情组件显示当前新闻内容

用户可以对新闻发表评论..

-数据库表 xz\_comment[id,nid,content,ctime]

-app.js

app.post("/addcomment",(req,res))

\*参数:nid[新闻编号];content[评论内容]

req.body.nid req.body.content

\*INSERT INTO xz\_comment VALUES(null,?,?,now()); //最好把新增每一项列出

INSERT INTO xz\_comment(id,nid,content,ctime) VALUES(null,?,?,now())

\*判断条件执行是否成功

result.affectedRows 影响行数 0

\*{code:1,msg:"删除成功"}

-NewsInfo.vue axios组件发送post请求问题

main.js 引入qs

import qs from "qs"

Vue.prototype.qs = qs;

var postData = this.qs.stringify({

nid:1,

content:".."

});

this.axios.post(url,postData)...