day03

课程复习

vue的基本指令
v-html 用来解析html标签
v-text 用来解析文本，解决{{}}闪屏问题
v-bind: 动态绑定，简写成：
v-on: 事件绑定，简写@
v-if v-show 用于条件判断
v-else 它必须要和v-if使用
v-for 循环遍历，可以循环数组或者对象
v-cloak 用于全局解决{{}}闪屏问题
<div id='app' v-cloak> ,在样式中[v-cloak] {display:none}
v-model 双向数据绑定，用于表单标签

动态样式之class
三种用法： ①变量 :class='变量名'
② :class='[条件?"类名A":"类名B"]'
③ :class='{类名A:条件A,类名B:条件B,类名C:条件C,类名D:条件D,....}'

常用的事件修饰符
.prevent 阻止默认事件
.stop 阻止冒泡事件
.once 只执行一次
.self 只执行自己
.capture 捕获

常用的键盘修饰符

```
.enter 13
.left 37
.right 39
.up 38
.down 40
```

表单的一些双向数据绑定方法
普通文本 input v-model='属性'
单选框radio v-model='' 设置value，初识data一般为后端需要要必传参数，比如sex:0/1 代表男/女
复选框：多个复选框初始值要用[],一个复选框，我们要用true/false
下拉框，v-model要定义在 select,取值用的option的value

表单修饰符

```
.lazy 延迟加载,当失去焦点的时候再把input内容渲染到模板种
.trim 去掉收尾空格
.number 只显示数值,前提是必须以数字开头，那么最终渲染结果只渲染数值，如果以非数值开头，渲染内容原样渲染
```

生命周期

概念

从出生到死亡，从创建到销毁的一个过程
在整个实例创建到销毁的过程种有很多函数的出现，那么我们管这些函数称为生命周期函数

整个生命周期中都有哪些函数？
八大生命周期函数
beforeCreate 创建之前
created 创建完成
beforeMount 挂载之前
mounted 挂载
beforeUpdate 更新之前
updated 更新
beforeDestroy 销毁之前
destroyed 销毁

页面一加载触发哪些生命周期函数

触发了前四个生命周期函数
beforeCreate 创建之前
created 创建完成
beforeMount 挂载之前
mounted 挂载

如何理解生命周期图示/你怎么理解生命周期

创建一个vue实例，在创建之前是一个空的vue对象，只有默认事件个生命周期函数
生命周期在创建之前，没有el元素，没有data属性。
当生命周期执行到创建的时候，依然没有el元素，但是这时候出现了data属性。
生命周期执行到挂载之前，首先我们要判断是否含有el属性，如果没有，我们看一下时候含有$mount()这个手动挂载方法，如果都没有，生命周期结束。如果有$mount('挂载点')生命周期继续执行。出现el属性，我们要先判断是否含有template这个属性，如果有，就直接编译解析成模板，如果有render()这个函数，优先渲染render()函数的内容，如果都没有就把外部的html(outerHTML)作为模板去编译解析。由此可以看出渲染的优先级是render>tempalte属性>outerHTML但是这个时候只是虚拟占位，真实数据并没有渲染，data属性依然存在。
当生命周期挂载完成，视图更新，被data属性所渲染，data数据依然不变。
当数据发生变化，就会触发beforeUpdate和updated
当调用destroy()这个函数方法的时候，执行beforeDestroy销毁之前，destroyed 销毁这两个函数方法。

你经常使用的生命周期函数有哪些？哪些函数中会经常调用ajax

在created的时候已经含有了data和methods，你可以在created中调用ajax。
我个人习惯在mounted中调用ajax调用，这个时候，虚拟的数据已经替换了真实数据，你调用接口结束之后，可能会操作dom，所以在mounted中调用。
在mounted中我们可以进行大量的异步操作，比如ajax，比如setTimenOut

---

watch（侦听器，监听）

```
watch监听，监听数据的变化
一、浅监听
它可以监听字符串、普通数组的变化
watch:{
    val(newVal,oldVal){
        //实时监控数据的变化
    }
}
二、深度监听
可以监听到对象的变化，还可以复杂数组
watch:{
    对象：{
        deep:true,//深度
        handler(新值newVal){
            //可以实时监听到新值newVal的变化
        }
    }
}
```

浅监听

```
<input type="text" v-model='food[1]'>
{{food[1]}}

                watch:{
                    food(newVal,oldVal){

console.log(newVal,'new')

console.log(oldVal,'old')
                    },
                    }
```

深度监听

```
        <input type="text" v-model='obj.name'>
{{obj.name}}
        <hr>
        <input type="text" v-
model='songList[0].name'>{{songList[0].name}}
        watch:{
                        obj:{
                    deep:true,//deep 深度
                    handler(newVal){

console.log(newVal,'new')

console.log(newVal.name,'new')
                    }
                },
                songList:{
                   deep:true,
                   handler(newVal){

console.log(newVal[0].name,'new')
                   }
                },
            }
```

深度监听可能会造成页面卡顿，非必要尽量少的去使用

百度搜索的案例

```
<!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="./node_modules/vue/dist/vue.js"></script>
    <style>
      .select {
        background: #ccc;
      }
    </style>
  </head>
  <!--
    百度搜索数据思路：
    一、我们要监听到用户实时输入的数据同时我们要实时调取
接口
    二、把调取接口之后的返回值渲染到页面中
    调取接口面临着跨域问题：调取ajax的时候
    解决跨域问题的方法？？？
    ①谁去解决跨域问题？
    首先要分环境，在生产环境下出现跨域问题，前端无法解
决，必须由后端去解决，在Nignx上解决（Nginx (engine
x) 是一个高性能的HTTP和反向代理web服务器）
    ②在开发环境下，出现跨域问题，前端解决
    jsonp可以解决跨域问题，但是在真正的项目很少用
    webpack可以解决跨域，这个方法是我们在做项目框架的过
程中经常的用的方法，只做代码配置
    三、当前百度案例调取接口，我们选择没有跨域问题的方式
    jsonp
    四、jsonp的原理是什么？    src href这些属性没有跨
域问题
    ①创建一个script标签
```

```
      let script =
document.createElement('script')
      ② 创建src属性
      script.src = 'url地址'
      ③把创建好的标签插入body中
      document.body.appendChild(script)
-->


  <body>
    <div id="app">
      <div>
        请输入搜索内容:
        <input
          type="text"
          v-model="val"
          @keydown.down="down"
          @keydown.up="up"
          @keydown.enter="enter"
        />
      </div>
      <!-- 循环遍历搜索结果 -->
      <ul>
        <!-- i <4  0 1 2 3  -->
        <li :class="[i==num?'select':'']" v-
for="(item,i) in arr" v-if="i<4">
          {{item}}
        </li>
      </ul>
    </div>
    <script>
      let vm = new Vue({
        el: "#app",
        data: {
```

```
            msg: "今天是周一",
            val: "",
            arr: [],
            num: -1,
          },
          methods: {
            //键盘下箭头事件
            down() {
              console.log(this.num, "初始");
              if (this.num >= 4) {
                this.num = 0;
                return;
              }
              this.num++;
              console.log(this.num, "更新");
            },
            //键盘上箭头事件
            up() {
              if (this.num < 0) {
                this.num = 3;
                return;
              }
              this.num--;
            },
            //回车事件，当用enter的时候跳转到搜索结果
            enter() {
              //跳转链接
              //搜索列表：http://www.baidu.com/s?
wd=
              // window.open() 在新页面打开链接
              //
window.open('http://www.baidu.com/s?
wd='+this.arr[this.num])
```

```
            //window.location.href属性在当前页面打开链接
            window.location.href =
              "http://www.baidu.com/s?wd=" + this.arr[this.num];
          },
        },
        watch: {
          //axios 它是基于node.js开发的http库
          val(newVal, oldVal) {
            //剔除空值
            if (newVal == "") {
              this.arr = [];
              return;
            }
            console.log(newVal, "新值");
            //调用jsonp接口
            //创建script标签
            let script = document.createElement("script");
            //添加src属性
            script.src =
"http://suggestion.baidu.com/su?cb=aa&wd=" + newVal;
            //把创建好的标签插入到body中
            document.body.appendChild(script);
          },
        },
      });
      //创建一个函数aa
      function aa(res) {
        console.log(res, "调取接口之后返回的结果");
        vm.arr = res.s;
```

```
        }
    </script>
  </body>
</html>
```

---

computed（计算属性）

> 作者希望模板内容{{}}简单易于维护，但是有时候我们会书写大量的逻辑，这样作业就开发了computed

计算属性的用法

```
 <!-- 字符串翻转 -->
{{val.split('').reverse().join('')}}
computed: {//计算属性 大量的逻辑操作可以放在计算属性中
                reverse(){
                    return
this.val.split('').reverse().join('')
                },
            }
```

计算属性和methods的案例

```
<!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="./node_modules/vue/dist/vue.js"></script>
</head>
<body>
    <!--
        计算属性和methods之前的区别
        相同点：都是通过return返回数据且都是相关依赖发生变化，值也会跟着变化
        不同点： 计算属性是有缓存作用的，如果依赖没有发生变化，它只执行一次，而方法是调用几次执行几次。计算属性极大的提高渲染速率
        使用场景：有交互事件的时候，我们可以选择methods.如果有大量的数据交互，而且调用多遍我们可以选取计算属性computed
    -->
    <div id="app">
        {{msg}}
        <button @click='num=num+1'>点击我有惊喜</button>
        <hr>
        <!-- 方法 -->
        {{allSum()}}
        {{allSum()}}
        {{allSum()}}
        <hr>
        <!-- 在计算属性 -->
        {{allPrice}}
        {{allPrice}}
        {{allPrice}}
    </div>
    <script>
        new Vue({
            el:'#app',
```

```
            data:{
                msg:'今天是周一',
                num:100
            },
            methods: {
                allSum(){
                    console.log('我是方法')
                    return this.num
                }
            },
            computed: {
                allPrice(){
                    console.log('我是计算属性')
                    return this.num
                }
            },
        })
    </script>
</body>
</html>
```

计算属性中的get和set的方法

```
<!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="./node_modules/vue/dist/vue.js"></script>
```

```
</head>
<body>
    <div id="app">
        {{allNum}}
        <hr>
        <h1>{{allSum}}</h1>
    </div>
    <script>
       let vm= new Vue({
            el:'#app',
            data:{
                msg:'今天是周一',
                xName:100
            },
            computed:{
                allNum(){
                    return 100
                },
                allSum:{
                    get(){
                        return this.xName
                    },
                    set(val){
                        console.log(val,'设置的
值')
                        this.xName = val
                    }
                }
            }
        })
        vm.allSum = 500
    </script>
</body>
```

```
</html>

<!-- allNum(){
    return 100
},
这种写法，相当于
allSum:{
    get(){
        return this.xName
    }
} -->
```

购物车案例

购物车：
有两种状态
一、必须要登录才能看见购物车以谁为首，比如天猫
二、无需登录，你就可以加入购物车，并看到购物车列表。以京东（pc端）为案例
登录会相对简单。

作业

一、整理今天笔记

二、整理所有案例，敲最少3遍

三、完善购物车案例以及单选全选

全选和反选的思路
全选 控制单选 ，如果打勾，每一个单选都打勾。如果全选未打勾，单选都不打勾
单选控制全选，所有的单选都打勾，全选打勾。只要有一个单选未打勾，全选为打勾

面试题

Class 与 Style 如何动态绑定?

Class 可以通过对象语法和数组语法进行动态绑定：
对象语法：

```
data: {
  isActive: true,
  hasError: false
}
```

数组语法：

```
data: {
  activeClass: 'active',
  errorClass: 'text-danger'
}
```

Style 也可以通过对象语法和数组语法进行动态绑定：
对象语法：

```
data: {
  activeColor: 'red',
  fontSize: 30
}
```

数组语法：

```
data: {
  styleColor: {
     color: 'red'
   },
  styleSize:{
     fontSize:'23px'
  }
}
```

Vue中常用的修饰符有哪些？并简要说明它们的作用。

Stop：阻止冒泡；
prevent：阻止默认事件；
once：只执行一次；
capture：将事件流设为捕获方式；
self：直对自身触发事件； enter：回车键；Up:上；
down：下；left:左；right：右；

vue初始化页面闪动问题

答：使用vue开发时，在vue初始化之前，由于div是不归vue管的，所以我们写的代码在还没有解析的情况下会容易出现花屏现象，看到类似于{{message}}的字样，虽然一般情况下这个时间很短暂，但是我们还是有必要让解决这个问题的。
首先：在css里加上[v-cloak] {
display: none;
}。
如果没有彻底解决问题，则在根元素加上style="display: none;" :style="{display: 'block'}"

计算属性和methods的区别

两种方式得到的结果是相同的。
不同的是计算属性是基于变量的值进行缓存的，只要在它关联的变量值发生变化时计算属性就会重新执行。而methods没有缓存，所以每次访问都要重新执行。
2、计算属性的特点：
计算属性使数据处理结构清晰；
依赖于数据，如果数据有更新，则计算属性的结果自动更新；
计算属性的结果无须在data中定义就能够在页面中直接使用；
相较于methods，如果依赖的数据不更新，则读取缓存，不会重新计算。

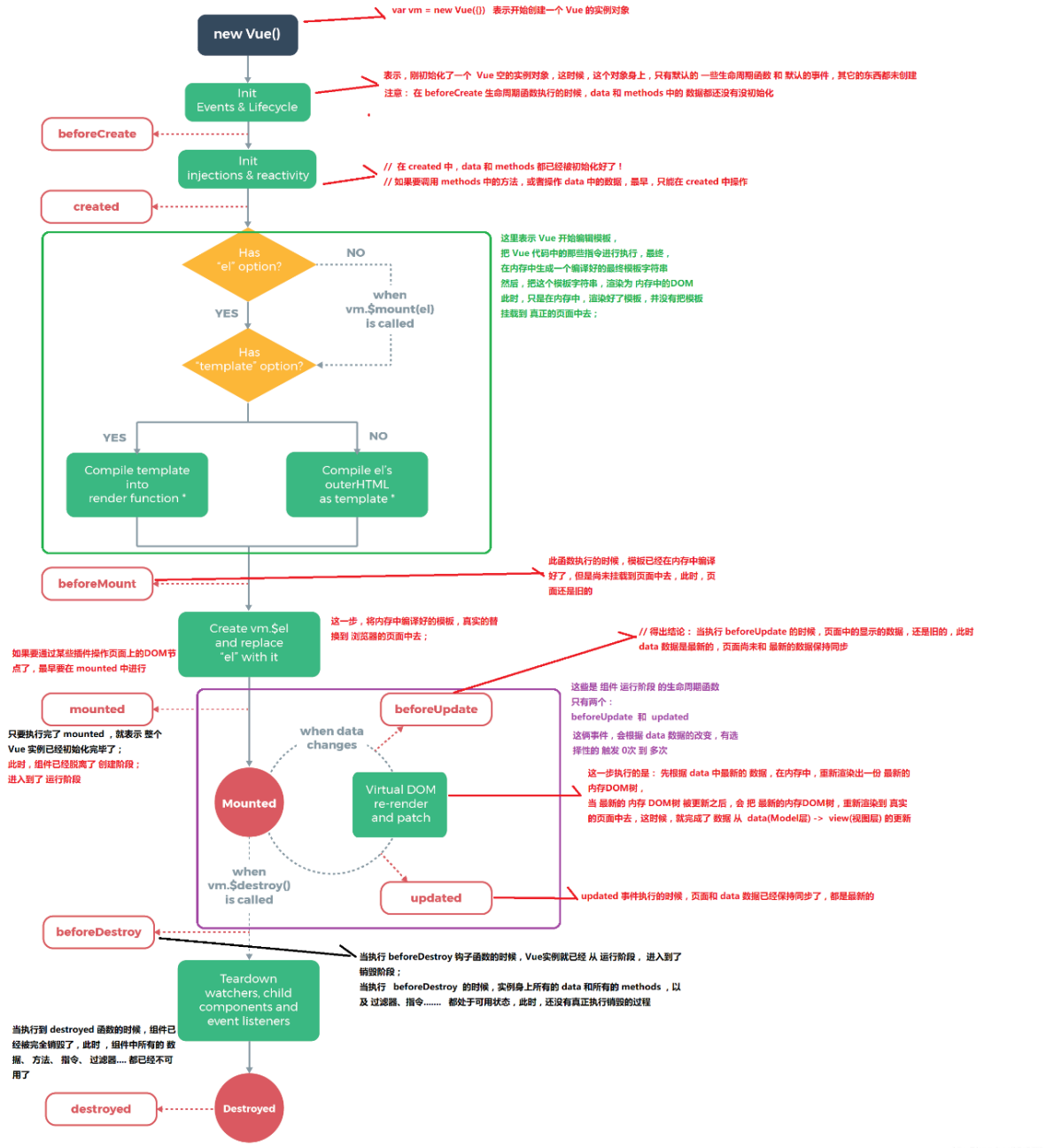
计算属性和watch的区别

共同点：都是以Vue的依赖追踪机制为基础的，都是希望在依赖数据发生改变的时候，被依赖的数据根据预先定义好的函数，发生“自动”的变化。
不同点：watch擅长处理的场景：一个数据影响多个数据；computed擅长处理的场景：一个数据受多个数据影响。虽然计算属性在大多数情况下更合适，但有时也需要一个自定义的侦听器，当需要在数据变化时执行异步或开销较大的操作时，通过侦听器最有用。
总结：
当在模板内使用了复杂逻辑的表达式时，应当使用计算属性。
虽然方法也能实现同样的效果，但是因为计算属性可以基于它们的依赖进行缓存，所以选择计算属性会比方法更优。

当需要在数据变化时执行异步或开销较大的操作时，可以使用 watch。

谈谈你对 Vue 生命周期的理解?

（1）生命周期是什么?
Vue 实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模版、挂载 Dom -> 渲染、更新 -> 渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。
（2）各个生命周期的作用
生命周期的描述
beforeCreate 组件实例被创建之初，组件的属性生效之前
created 组件实例已经完全创建，属性也绑定，但真实 dom 还没有生成，$el 还不可用
beforeMount 在挂载开始之前被调用：相关的 render 函数首次被调用 mounted el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子 beforeUpdate 组件数据更新之前调用，发生在虚拟 DOM 打补丁之前
update 组件数据更新之后
beforeDestory 组件销毁前调用
destoryed 组件销毁后调用



在哪个生命周期内调用异步请求?

可以在钩子函数 created、beforeMount、mounted 中进行调用，因为在这三个钩子函数中，data 已经创建，可以将服务端端返回的数据进行赋值。（本人常放置到mounted中可以操作dom,当然也可以放到created中）