Vue.js认识

2022年9月5日　9:11

一、VUE.JS介绍

注：Vue代码提示:需安装Vue 3 Snippets和Veturn

（1）构成:是以**数据驱动和组件化**的思想构建的。

数据驱动：通过js里面对象的数据与HTML所对应的元素进行绑定，**只要js里面的数据发生改变，那么所绑定的HTML对象的内容也会发生改变（不需要通过拿标签去改变值）**

二、第一个vue程序

（1）引入vue.js：<script src="vue"></script>

（2）创建vueModel实例：new vue();

（3）通过el属性（必须与vue框架中的el属性一致）将viewModel绑定标签

（4）通过data属性将Model里面的数据与viewModel进行绑定

（5）在div里面写入插值语句{{vue里面data里面的对象变量名}}

```
<div>{{data}}</div>
 new Vue({//
el:"css选择器（不建议使用标签，不能重复标签）",//绑定网页数据
data:{
data（变量）:",我爱学习Vue框架"//绑定他后面的数据
}
}}
```

注：**在vue中的el属性必须是具有唯一性，不建议使用class和标签**

（6）执行过程

1·将某一个标签挂载至vue的el属性，那么在所挂载的标签内可以任意填写vue的语法

2·创建data属性建立data数据

3.使用插值语法将vuedata中的数据赋值

（7）vue开发与JQuery开发（原生）区别

其一、编程范式的类型不同

vue开发（声明式）：只需要在某一个需要数据的地方声明一下就可以了，不需要了解其内部的过程

原生开发（命令式）：程序执行执行的每一步都必须清晰明了

其二、修改数据极其简便

vue开发修改数据只需在所需的data属性里面的数据进行一定的改变或者在开发者模式中发生改变就会即刻进行变化

原生开发需要改动界面的数据，极其繁琐

三、案例1-输出HelloVue

（1）el属性：挂载标签的属性

el属性类似于标签，只要挂在在某一个标签上那么这个标签的子元素就会具有Vue中可以传递数据的功能（**不可挂载至body与html，要确保挂载的标签的唯一性**）

（2）data属性：创建数据

四、案例2-列表内容输出（水果价格表）

（1）v-for指令：可以将**data属性的数组数据遍历出来**同时也**不需要重复编写相同功能**的代码，并且可以**随便添加数据**

**（2）格式 **

如何在浏览器console中修改数据

单个数据添加：vue对象名.所对应的数据名= “数据”

多个数据添加：vue对象名.所对应的数据名.push("数据")

（3）案例：水果价格表

Fruit_List

四、案例2-计时器

（1）v-on指令：监听事件的发生（事件监听器）进行动作

格式：v-on:click(事件)=""

ES6语法糖编写：@click(事件)=""

（2）简单引入method方法：定义需要操纵数据的方法

（3）this的使用：this可以表示当前作用的范围

Times

MVVM模型

2022年9月19日 15:27

一、MVVM模型

（1）原理图

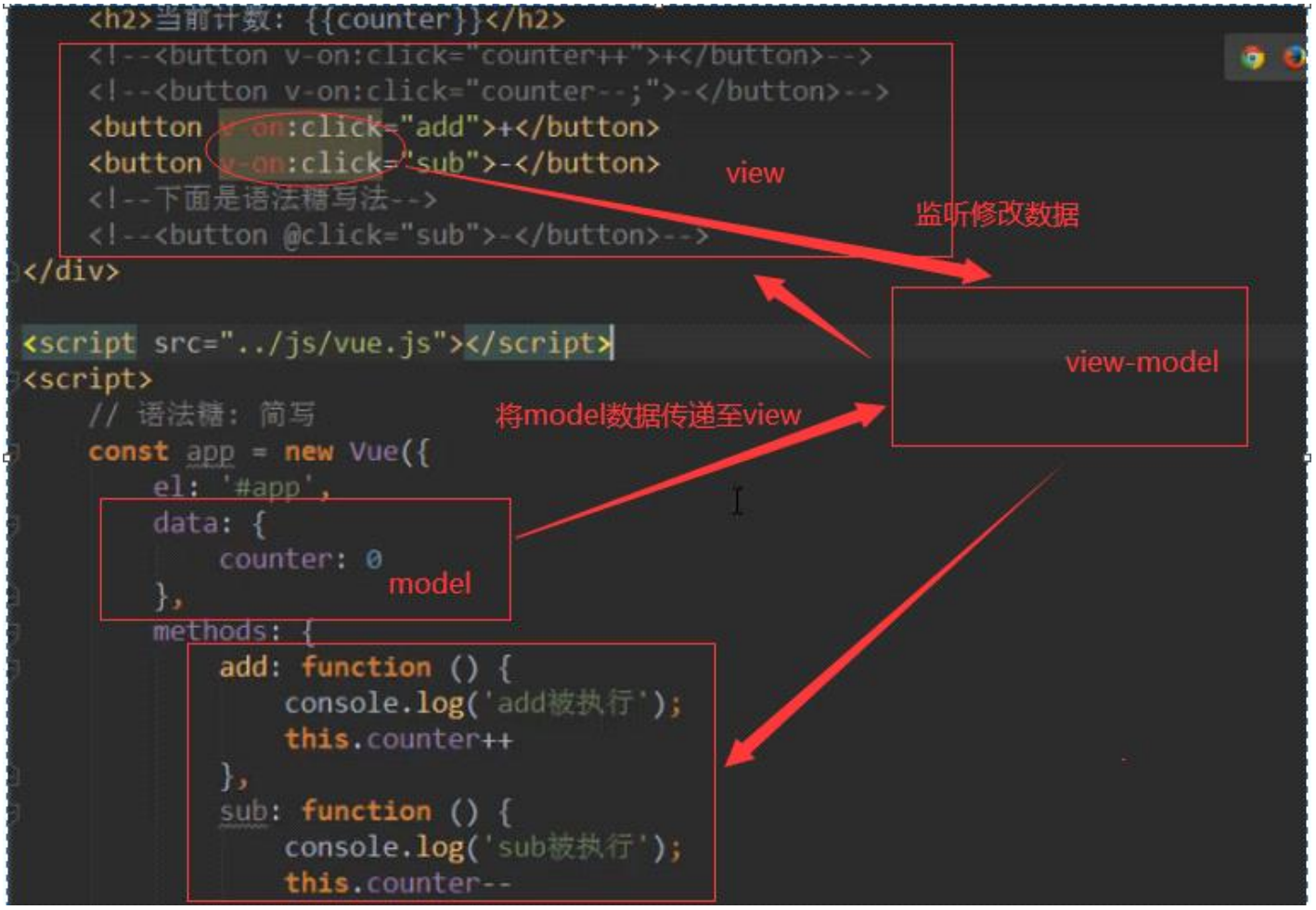
- View层：
  - 视图层
  - 在我们前端开发中，通常就是DOM层。
  - 主要的作用是给用户展示各种信息。
- Model层：
  - 数据层
  - 数据可能是我们固定的死数据，更多的是来自我们服务器，从网络上请求下来的数据。
  - 在我们计数器的案例中，就是后面抽取出来的obj，当然，里面的数据可能没有这么简单。
- VueModel层：
  - 视图模型层
  - 视图模型层是View和Model沟通的桥梁。
  - 一方面它实现了Data Binding，也就是数据绑定，将Model的改变实时的反应到View中
  - 另一方面它实现了DOM Listener，也就是DOM监听，当DOM发生一些事件(点击、滚动、touch等)时，可以监听到，并在需要的情况下改变对应的Data。

（2）工作过程：

由于DOM与model之间无法直接通信，那么此时中间的MVVM模型就起到了决定性的作用，他将数据与model进行绑定并且监听DOM页面的信息。只要DOM或Model其中的一个发生了变化，那么就会对数据进行即时修改。

即监听DOM数据修改model数据，将model最新数据同步至DOM页面

（3）计时器MVVM模型



二、option 选项

（1）官方option选项：API 参考 | Vue.js (vuejs.org)

（2）目前3大属性：el、data、method

el属性：用于挂载项目（Vue作用域范围（Vue可以管理DOM哪一个范围））

data属性：用于传递数据对象和函数（组件化时data必须是一个函数）

Method 属性：用于定义方法

（3）方法与函数的区别

方法：指的是在**类里面所定义的函数**，与**某个实例相挂钩**的

在VS code中 配置template模板需要到首选项中的用户代码片段

输入vue对已有的内容进行编辑

函数：指的是在**外部的任意一个地方所定义**的函数

三、template 模板的使用

Vs Code配置template

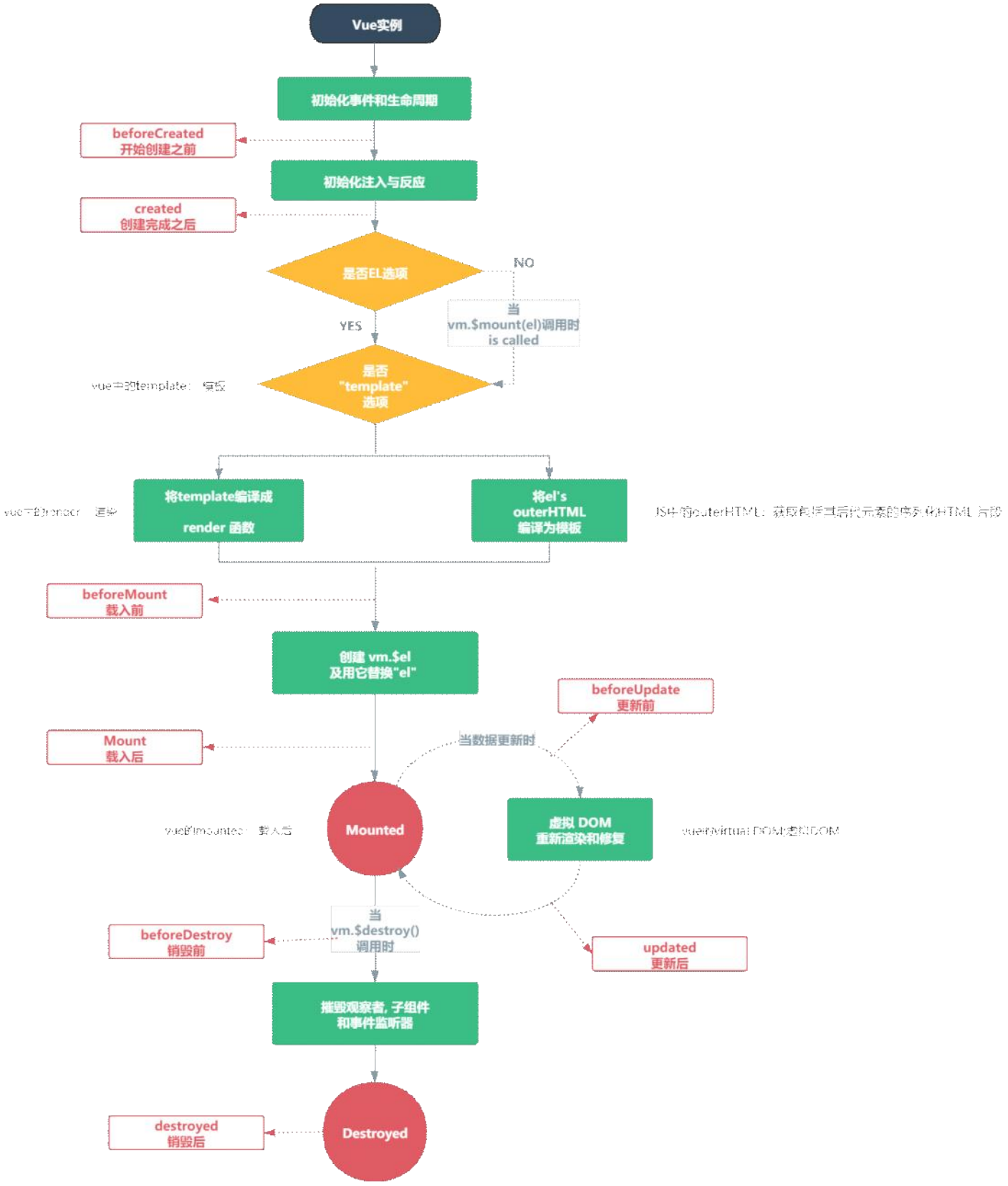
Vue生命周期

2022年9月19日 16:46

一、Vue生命周期

（1）概念：某一个事物从出生至死亡的一个过程

（2）官方生命周期图



过程：创建--->查看是否有挂载--->通过判断是否有模板进行创建模板--->数据渲染--->数据更新--->监听数据是否被摧毁---->数据摧毁

2、生命周期四大组

Create 创建----->在vue初始化数据

Mounted 载入-->渲染数据

Update 更新-->数据更新

Destroy 摧毁--->数据销毁

3、几大函数值

（1）创建：beforeCreate()、Create()

（2）载入：beforeMounted()、Mounted()

（3）更新：beforeUpdate()、Update()

（4）销毁：beforeDestory()、Destory()

生命周期钩子：也称生命周期函数，就是对vue对象所执行的生命周期函数里面的内容

二、template属性（使用snnppets）

（1）生成默认模板：vbase

（2）配置用户模板：

用户代码片段输入vue对代码进行修改后新建vue文件输入vue即可

*插值操作与动态绑定

2022年9月21日 9:44

一、Mustache 语法（双大括号语法）

格式：可以使用单个内容，也可以使用内容拼接，有的时候可以将内容与其他内容进行混合，Mustache语法里面的内容会根据你输入的数据类型进行对应的操作

二、v-once属性作用：当内容进行改动时，那么他所控制的范围就不会被改写（不赋值）

注：动态修改标签的时候不要加上v-once

三、v-html属性作用：将服务器返回的带有html代码的内容直接解析（赋值）

四、v-pre属性作用：将输出的文本原封不动的展示出来（不能不写这个）

五、v-cloak属性作用：防闪烁，需要进行样式设置[v-cloak]

***六、v-bind（动态属性绑定）使用**

1、属性绑定

（1）基础使用：直接在想要动态绑定的属性前加上v-bind属性

（2）v-bind语法糖：属性前面写一个：

```
<img :src="imgHref" alt="">
```

2、class属性的绑定

对象语法：`<div v-bind:class="{类名:值, active:isActive}"></div>`

由于v-blind属性中对象语法每一写入时都会显得及其长，那么我们就可以使用method里面定义一个函数来引用他，需要注意的是在返回值里面它属性所对应的值必须使用this(代指当前函数体内的相同的文件)

注意点：（1）移动到method函数中记得在其值添加this关键字

（2）v-bind属性基本格式:class="{tip:isTip}"

数组语法：在中括号里面添加数据

3、style属性绑定

扩展：*组件化思想：将页面上的元素凑成一个成品，这个成品可以随时随地的被调用类似于template（模板）

对象属性语法

```
<div :style="{color:isColor,fontSize:isSize+`px`}">你好</div>
```

可以使用三元表达式来表示，格式：<div :class="isActive?'类型1':'类型2'">

<div :style="isActive?'类型1':'类型2'">

*计算属性

2022年9月23日 11:20

一、computed属性简单使用

（1）作用：为了避免使用插值语法对可变化的数据进行操作时产生许许多多复杂的问题，因此使用computed属性可以对可变化的数据进行计算

（2）格式：与methods方法格式一致，但不需要加D:\Study Files\HTML Files\04-computed属性的使用\computed入门使用.html上()

（3）案例：拼接姓名（只展示computed）

```
computed:{
  fullname:function(){
    return this.firstname+" "+this.lastname
  }
```

二、ES6语法

（1）方法简写（不写function）

函数名（参数）{

函数体

}

例如

```
PrintSummer() {}
```

（2）for循环遍历

ES5语法（类似javafor循环）：

For(let i=0;i<=this.book.length;i++){

}

ES6语法1（与pythonfor循环相似,不需要赋初值直接拿数组的值）：

For (let i in book){

}

推荐写法：ES6语法2（直接拿取本数组）：

For （let I of book）{

}

二、getter方法与setter方法

（1）computed完整格式：

computed：{

Set:function(){

一般情况下不写的

}

Get:function(){ computed属性：

function(){

----->

} }

}

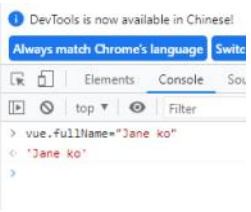
注：**当插值语法中的变量调用computed的时候实际上就是在调用computed属性中的get方法，一般在computed属性中我们并不希望这个数值可以被别人操纵，所以在一般的情况下，我们不写set属性，只写get属性。又因为每次只写get属性，所以就简便成属性名:方法（get属性）**

一般在写computed属性的时候实际上就是在调用computed的get属性

（2）setter属性的使用

当你对属性进行赋值时（所改变的值就是你将要输出的值），那么属性就会传到setter的参数中对之前最终的数据进行改动。

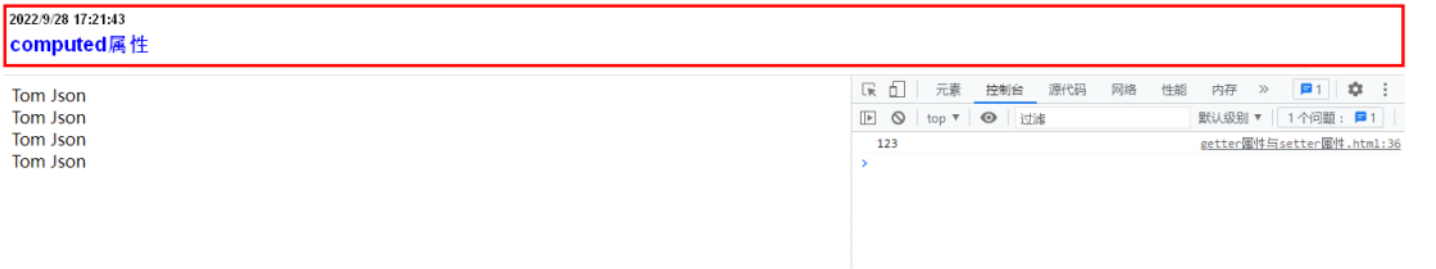
Jane ko



三、计算属性与方法的区别

method属性和computed属性执行效果如下（输出同样内容不同的执行）

getter属性与setter属...



computed属性在内存中有进行过优化，当你输出的内容连续相同时，那么就会将第一个求出来的内容保存下来，之后再输出同样的数据时，就直接调用原来计算的结果就不用反复的去计算了，而methods属性就会像剪切板一样，计算了一次下一次还会在执行。在内存资源的优化上computed属性比methods属性更加好

ES5与ES6的区别

2022年9月28日 17:33

一、let、var、const三者的区别

1、块级作用域（不会被外部内容所篡改）

（1）概念：代码在一定的范围内可以执行代码，超出范围就不能使用

（2）var块级作用域的限制无效所导致的问题

2-1、var所产生的问题：

当我想把某个修改好文件内容发送给别人，但是却不小心把数据重新赋值了一次，那么传输过去的内容就不是原先的内容，而是修改后的内容（主要原因：var在if和for里面没有作用域）

2-2、ES5var属性在使用for循环的时候必须创建一个闭包？

因为当闭包内部的时候我所传递的值是不会发生任何改变的，因为我传进去的值在一定的作用域内，并且在这个作用域内拥有属于自己的参数，不会被外部的值所影响。

综述：总而言之，函数内部操作的数值是我们当时传进去的数而对于外界的数值进行如何的变化时，我们函数内部的数值是不会发生改变的

主要是因为在ES5的作用域中，for和if没有作用域只有函数有作用域。因此每次在执行循环或者判断操作中涉及数值问题都会借助与函数进行相关操作，在ES6中就补上了之前的短板

二、ES5与ES6区别

（1）ES5var作用域的效果

由于var对if语句和for语句是没有作用域的，因此我执行函数但凡里面的数值被篡改，那么整个数据也就会被修改，因此特地引入闭包。

（2）什么是闭包？---- 闭包就是一个函数，只不过是没有function关键字的函数

注：在ES5中只有函数才有作用域

三、const的使用

作用：const关键字是用于存放常量的

注：常量对象的内部属性可以修改但不可以修改常量所指向的对象

四、ES6对象的字面量增强写法

（1）字面量：引用对象不需要声明，直接使用一个{}

（2）对象增强写法

```
//ES5对象赋值时的写法
// let name="张一凡";
// let age=23;
// let address="翻斗花园";
// const app={
//   name:name,
//   age:age,
//   address:address,
// }
//<--这是ES6对象增强写法-->
let name="张一凡";
let age=23;
let address="翻斗花园";
const app={
  name,
  age,
  address,
}
console.log(app);
```

（2）函数增强写法

```
const fun={
  //这是ES5函数的写法
  look:function(){

  },
  //这是ES6函数的写法
  run(){

  }
}
console.log(app);
```

v-on、v-if、v-show、v-for的使用

2022年9月29日 10:30

1、**v-on基本使用**

一、事件监听

（1）作用：可以对用户所进行的交互进行一些操作上的反馈

二、v-on属性基本使用

（1）v-on格式：

v-on:事件=“函数名”

@事件=“函数”

这些函数可以写在computed属性里

（2）v-on的参数传递

情况1：当v-on不用传递参数时，那么括号不用写

情况2：当需要传参时，却没有传参数那么他就会默认将event对象传递过来

情况3：手动传递event对象就使用$event

Event.target可以获取到当前点击的标签

（3）v-on修饰符

修饰符	作用
Stop	阻止事件冒泡
prevent	取消html自带的事件
键码/键名	执行某个键时所做出的响应
Native	监听自定义组件
Once	事件只执行一次

2、**v-if、v-else、v-else-if作用**：根据用户需求控制内容是否显示，为true显示，为false不显示

格式：<... v-if="条件判断">

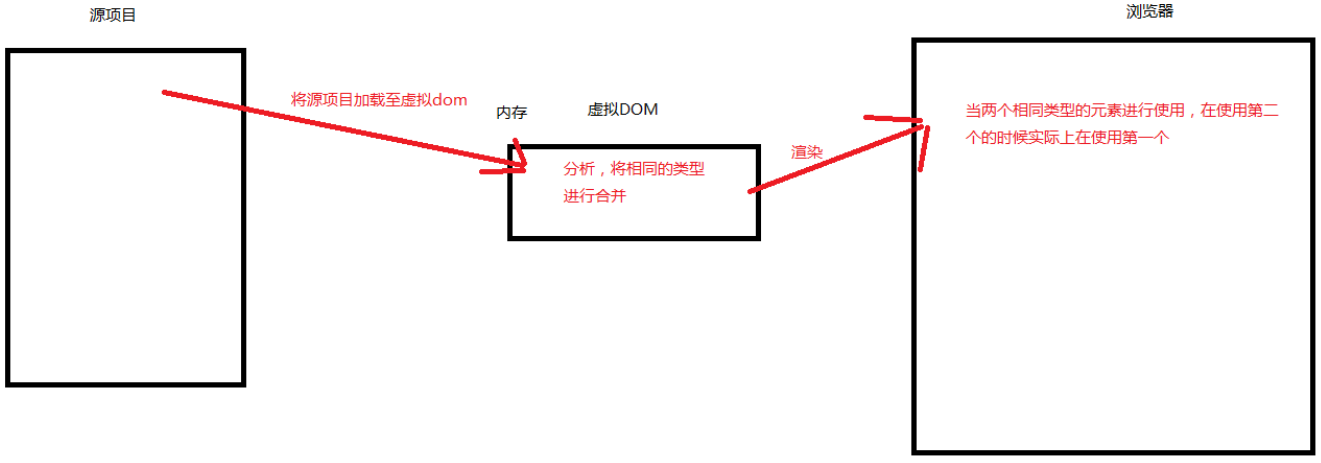
适用于一些小判断

账号切换案例产生问题（文本复用）

问题：切换时文本框内的内容不会清除？

原理：

当案例加载到浏览器之前会到内存建的虚拟DOM里面进行分析，如果里面的格式一样，那么只会把固定的内容改掉，不会替换之前的输入的内容。我们看到不一样的效果实际上只是把里面的内容替换了，本质上不变（input复用）



解决：在输入框属性设置key属性，确保两者不相同，那么不会被复用

3、**v-show与v-if显示问题**

v-if在条件为false时，所在v-if元素就不存在，那么v-if所做的操

作实际上就是在对标签进行添加删除操作来回进行

v-show在条件为false是，所在v-show元素存在，只是样式添加了一个隐藏效果,那么v-show操作实际上就是对标签进行样式设置

如何选择v-if与v-show

	切换程度	渲染程度
v-if操作	偶尔	大
v-show操作	频繁	小

v-show控制的是节点的display属性 v-if是将节点删除了 如*果节点需要频繁显示隐藏 使用v-show性能更佳！*

4、v-for的使用

1、遍历数组

（1）不包含下标

v-for="变量 in 数组"

（2）包含下标

v-for="(变量,下标) in 数组"

2、遍历对象

（1）不包含键值

同遍历数组中不包含下标一致

（2）包含键值

（value,key）in 对象

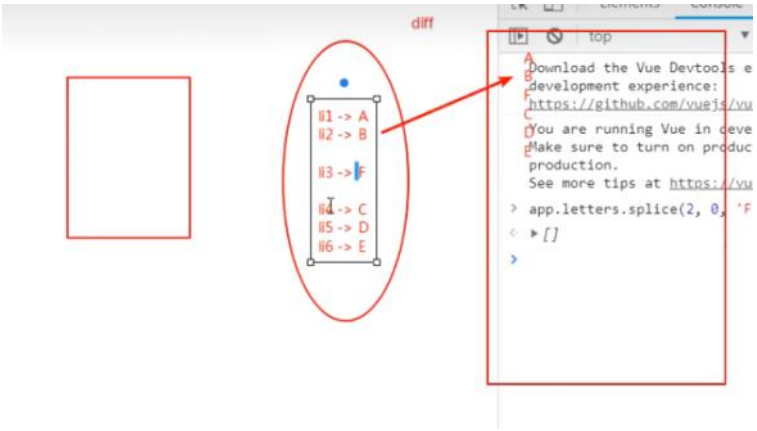
（3）包含下标

（value,key,index）in 对象

3、键值是否绑定问题

（1）在正常顺序中间插入数值

做法：将插入的数值直接替换你要插入的地方，之前的元数据就覆盖后面的数据，以此类推。原来最后一个数据此时就会创建一个新的空间用来存放它，然后将虚拟dom内的内容投放到浏览器上面



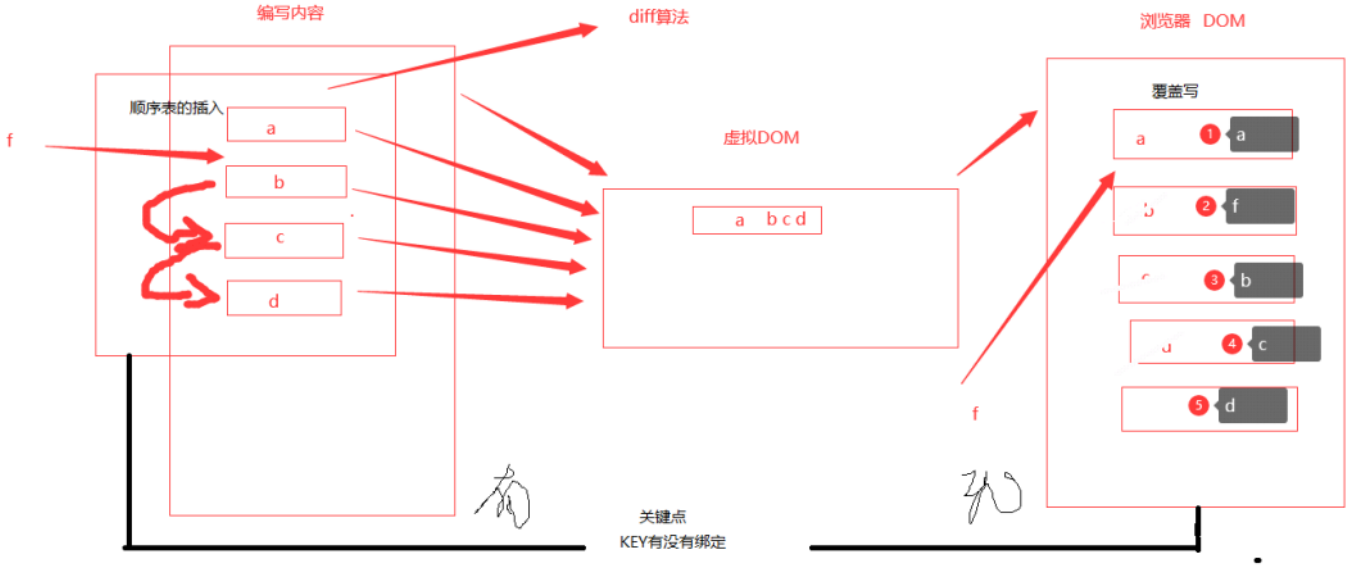
diff算法：**key相同的会进行对比如果相同，则复用，不同，在创建新的元素**

在Vue中插入一个元素必须设置一个key数据才不会被覆盖

建议使用v-for加入一个key（key绑定内容（前提：内容不为1））

过程如下

a
b
c
d



4、数组的响应式方法

（1）复习：何为响应式

当数据发生改变的时候，那么虚拟DOM就会检测到数据的变化

从而进行改变

（2）数据响应式方法

1、push 其数组尾部添加一个内容

2、pop 其数组尾部删除一个元素

3、shift 其数组头部删除一个元素

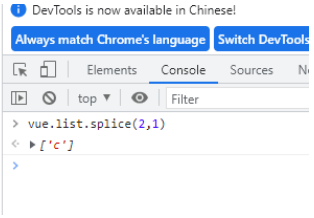
3、unshift 其数组头部添加一个元素

4、splice对数组进行插删改操作（默认为删除操作）

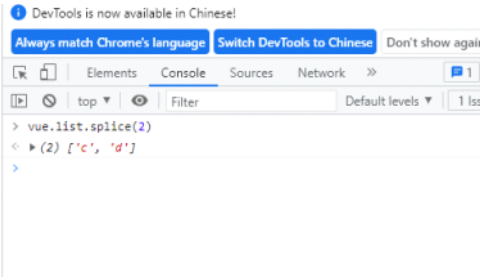
删除操作：splice(起始位置,"删除多少个元素（不写默认删除除

当前以外的全部元素）")

- a
- b
- d



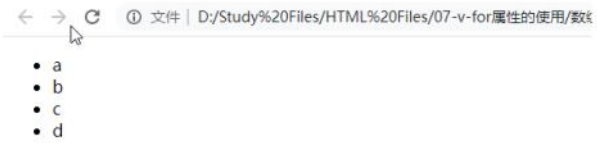
- a
- b

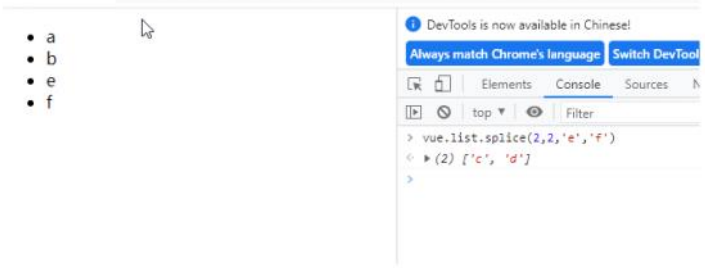


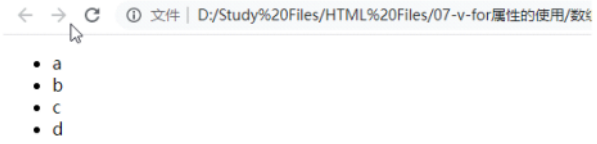
替换操作：splice(起始位置,"替换多少个元素","字符串集")

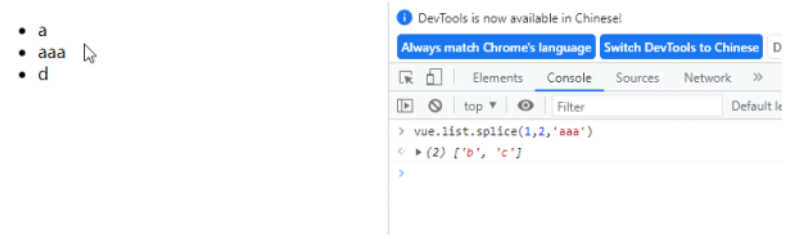
当你的字符集不符合他替换的元素，他就会把多出来的元素删除

掉

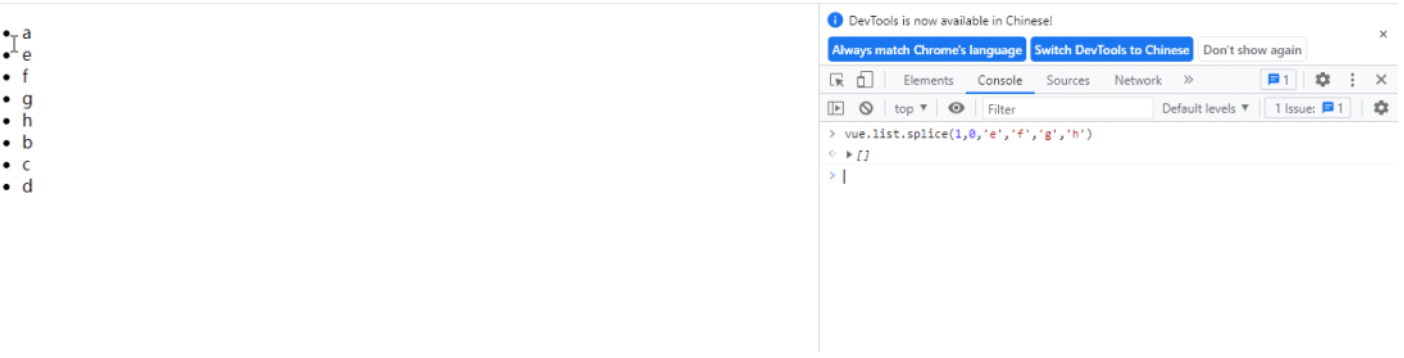






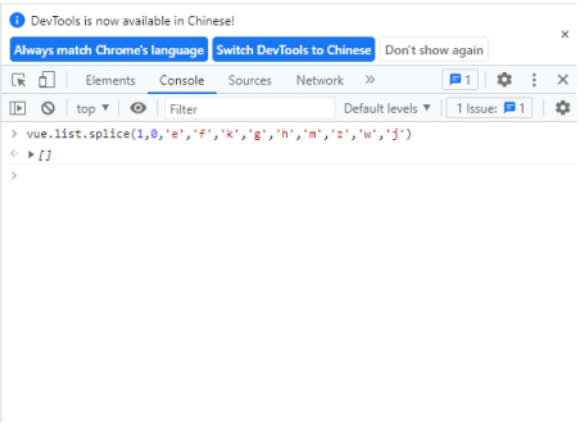


插入操作：splice(起始位置,0,"字符串集")

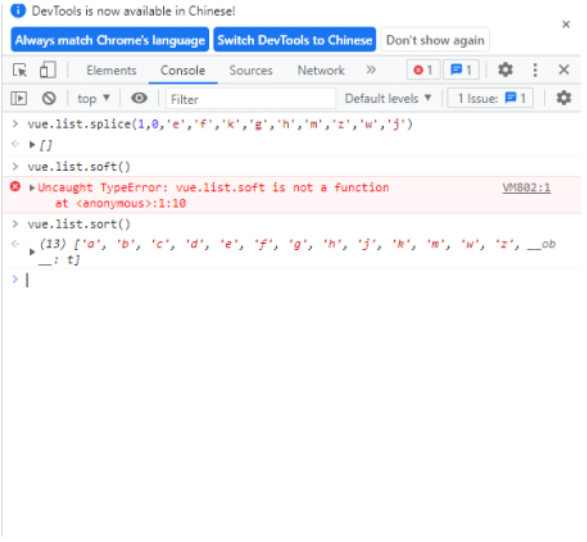


5、soft对数组进行排序操作

- a
- e
- f
- k
- g
- h
- m
- z
- w
- j
- b
- c
- d



- a
- b
- c
- d
- e
- f
- g
- h
- j
- k
- m
- w
- z



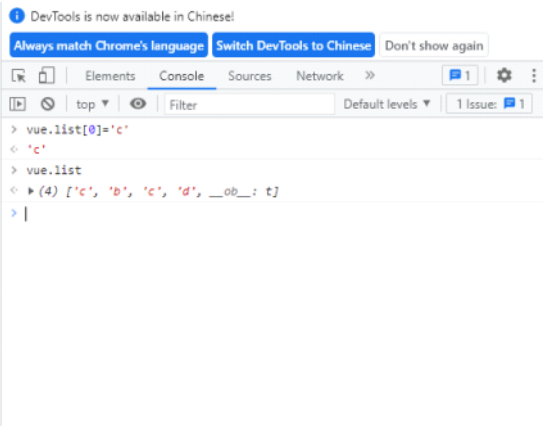
6、reverse对数组进行反排序操作

- a
- b
- c
- d
- e
- f
- g
- h
- j
- k
- m
- w
- z

- z
- w
- m
- k
- j
- h
- g
- f
- e
- d
- c
- b
- a

直接使用索引改变数组的值与splice区别

- a
- b
- c
- d



直接使用索引改变数组的值只会改变列表的值，但虚拟dom没有将改变后的内容实时修改到位，建议使用splice

To fixed（保留位数）可以保留几位小数

7、JS高阶函数使用

（1）js高阶函数：filter函数、map函数、reduce函数

（2）何为高阶函数：不仅仅可以传递普通的值，还可以传递函数

（3）filter、map函数递归调用的要求是满足条件创建一个数组将满足条件的值装到数组中，不满足条件过滤掉（数值类型Boolean）

我们只需要创建一个变量用于存放这个高阶函数，在返回时写入条件即可进行过滤

（4）reduce函数的功能对所有的内容进行汇总

首先要传递两个值，一个是函数内部的值，还有一个是初始化的值，然后将初始化的值设置成0，对函数内部的值进行遍历

filter 对数值进行一定的判断

map 进行一定的数值操作

reduce 数值汇总可以是任意的运算

（5）案例

对一组数据进行筛选：数值小于20，在进行数值操作，最后进行一个汇总

```
// filter对li的数据进行过滤
//map对数值进行操作*2
//reduce对数值进行累加操作
const li=[1,2,3,4,5,6,7,8,9]
let result=li.filter(function(将数组内的值依次传递至filter所在的数组中进行筛选){
    return li<5
}).map(
    function(将数组内的值依次传递至map所在的数组中进行数值操作){
      return n*2
    }
).reduce(function(前,现){
    return n+pre
},0)
```

通过筛选、操纵、汇总对数值（数据必须是初始化完的数据）进行操作

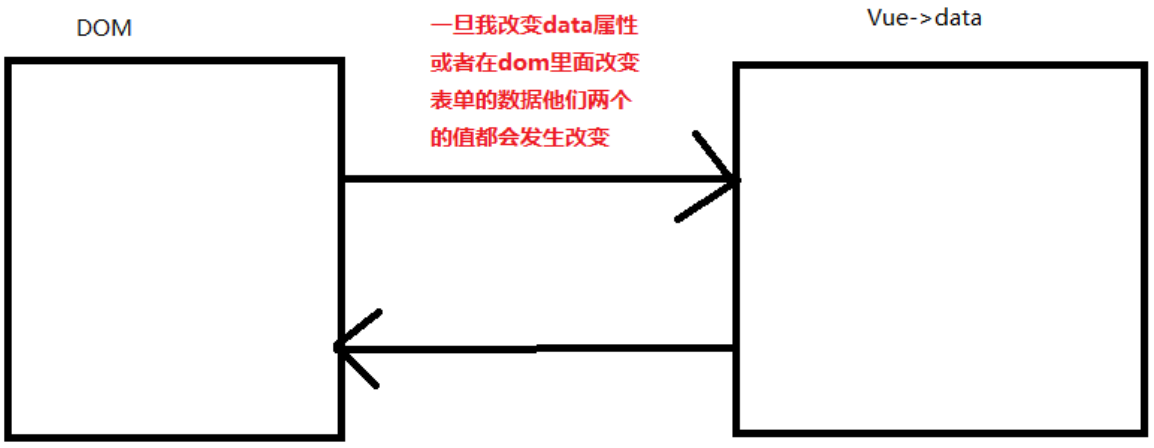
v-model双向数据绑定

2022年10月8日 18:44

1、表单绑定

（1）可以使用v-model来实现双向数据和表单元素绑定

（2）v-model与普通插值语法绑定的区别



插值语法：他可以通过data属性修改我所显示的内容，而不能从dom里面修改data属性

123 原来的你好呀变成输出123

```
> document.querySelector("#app").innerText="123"
< '123'
> console.log(vue.message)
  你好呀                                    VM395:1
< undefined
>
```

原来是你好呀改成123但vue.data属性的数据没有发生变化

（3）双向数据绑定基本使用

```
  <input type="text" name="" id="" v-model="message">
  <br> data中的数据：{{message}}
</div>
body>
cript src="../JavaScript/vue.min.js"></script>
cript>
const vue=new Vue({
  el:"#app",
  data:{
    message:"你好呀"
```

补充：

1、不使用v-model进行双向数据绑定

@input事件 在表单中监听用户输入数据的事件

利用之前的知识实现双向绑定（v-model的原理（拿数据（v-bind）和修改数据（v-on:input）这两个操作））
使用v-bind将数据先从data属性绑定至value，
再利用v-on中input属性获取我输入的值放到data属性中
2、在单选框中如果想要两个选择互斥，除了可以添加name属性之外还可以添加v-model，只要他们两个的值不相同即可（必须加上value属性）
3、按钮中可以使用disabled控制按钮是不是选中状态
4、在chickbox中，单选框所传递的一般是布尔值，而多选则是将其参数传递过去,并且单选时是一个变量，而多选框是一个数组
5、select多选框使用mutiple，v-model一般绑定在select上面

```
<!-- 这是select 单选情况下 -->
<select name="" id="" v-model="fruit">
  <option value="葡萄">葡萄</option>
  <option value="香蕉">香蕉</option>
  <option value="苹果">苹果</option>
  <option value="梨子">梨子</option>
</select>
<h2>你所选择的水果是{{fruit}}</h2>
<!-- 这是select 多选情况下 -->
<select name="" id="" v-model="Fruit" multiple>
  <option value="葡萄">葡萄</option>
  <option value="香蕉">香蕉</option>
  <option value="苹果">苹果</option>
  <option value="梨子">梨子</option>
</select>
```

2、值绑定（动态绑定改变属性值）：内容可以随机灵活的进行变化，并且我们所绑定的内容也可以动态绑定，总而言之所谓的值绑定就是在利用v-bind绑定内容在使用v-model将我当前选中的内容传递到另外一个数组中（利用v-for动态改变数据内容存放在新的数组中）。
3、**v-model修饰符**
lazy修饰符：不需要实时将data数据与我输入的数据同时同步，只需要在某个事件时才将数据存放至data中
number修饰符：将v-model所绑定的字符类型强制转化成数值类型
trim修饰符：将用户输入多余的左右两边的空格去除掉
4、watch监听器
（1）格式：watch：{
监听对象：function(新值，旧值){
}

}

（2）作用：监听事件所发生的对象

组件化的使用

2022年10月8日　16:10

一、组件化思想

1、组件化思想（模块化思想）：在一个页面上所进行的操作过程细分成许多个小的功能块，每个功能都有自己的作用，以方便管理和维护。

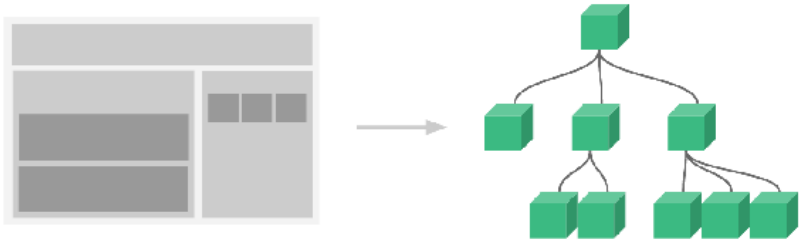
大 不利维护/更新 <---> 小 有利于维护/更新

2、Vue组件化思想

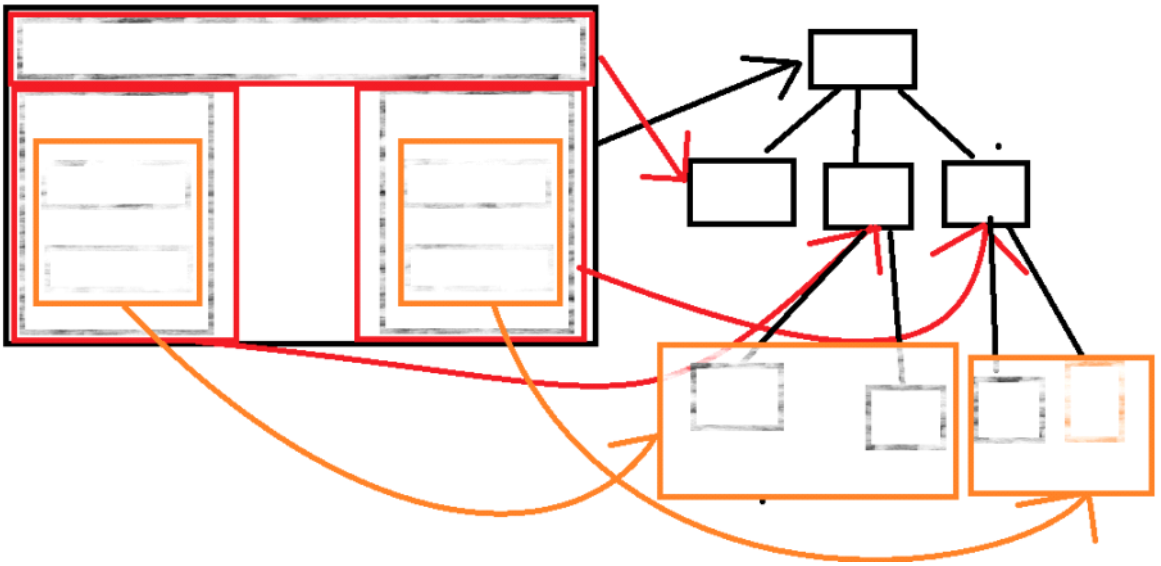
（1）将一个复杂众多的内容划分成一个个小模块，每个模块都具有独立的功能

（2）任何的应用都可能抽象成一个组件树（例如一个大页面相对于另外一个大页面而言，那么这个大页面就是组件）

官方图



内部图



二、组件化的基本使用

1、组件的使用过程

（1）创建组件构造器（extend方法）

（2）注册组件（component 方法）

（3）使用组件（必须在vue的使用范围内）

2、组件使用格式：

```
Vue.extend(
    模板：`想组件化的代码`
)
Vue.component ({
"构造组件后想使用的标签","构造组件的模板"
})
```

```
// const created=Vue.extend({
//    template:`
//     <div>
//       <h1>正在使用组件化创建标题</h1>
//       <p>正在使用组件化创建段落</p>
//     </div>
//   `
// }
// )
// const cpn=Vue.component(
//   'cpn', created
// )
Vue.component(
  'cpn',{
    template:`
    <div>
      <h1>正在使用组件化创建标题</h1>
      <p>正在使用组件化创建段落</p>
    </div>
    `
```

注：创建component 与extend方法必须创建在new对象之前

三、全局组件与局部组件（作用域的不同）

（1）全局组件：默认在vue中创建的组件就是一个全局组件，可以在多个vue对象中使用

（2）局部组件：新建vue对象设置component属性

```
0   const vue=new Vue({
1     el:"#app_all"
2   })
3   new Vue({
4     el:"#app_part",
5     components:{
6       pds:{
7         template:`
8         <h1>这是局部组件</h1>
9         `
0       }
1     }
```

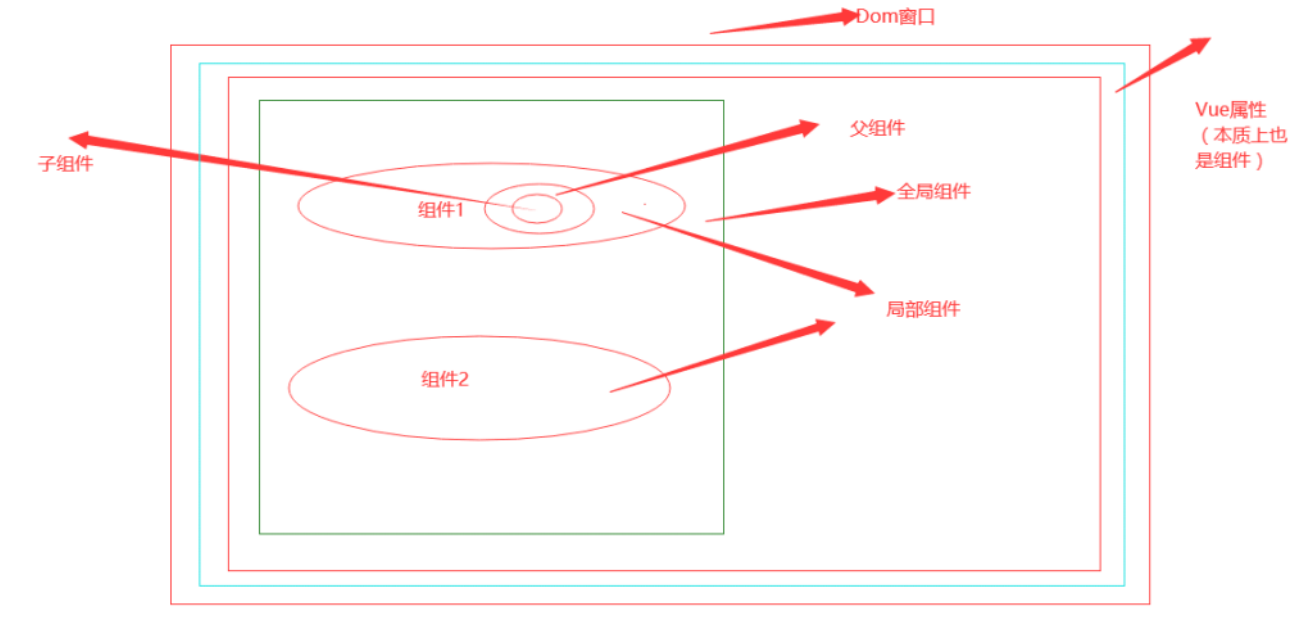
四、父组件与子组件的区分

（1）概论：

父组件：在一个vue对象内创建components属性

子组件：相对于父组件而言，子组件注册组件的地方就是其父组件

（2）层级关系



组件的排序：Vue属性>某个组件>某个组件下的子组件

注意点：组件套用其他组件时，请不要在组件的构造器中使用{}分开，并且在template属性中所使用``的区间内只能有一层代码不能含有多层，如果有很多内容那么就必须使用嵌套将内容包围起来

（3）注册组件语法糖写法：

```
components:{
  'asps':{
    template:`
    <div>      You, 14小时前 • chang
      <h1>asp标签</h1>
    </div>
    `
  }
```

（4）组件抽离写法(必须加上id，否则无法使用)

```
  <!-- 易错点：忘记使用组件 -->
  <xtpl></xtpl>
  <tpl></tpl>
</div>
<script type="x-template" id="xtpl">
  <h1>
  你好呀,这是局部组件下使用x-template抽离组件      You, 45分钟前 • 已更新至组件模
  </h1>
</script>
<template id="tpl">
  <h1>
    你好呀,这是局部组件下使用template标签抽离组件
  </h1>
</template>
</body>
<script src="../../JavaScript/vue.js"></script>
<script>
  //挂载注册组件
  const vue = new Vue({
    el: "#app",
    components: {
      xtpl: {
        template: "#xtpl"
      },
      tpl: {
        template: "#tpl"
      }
    }
  })
</script>

</html>
```

（5）组件实现动态数据变化

1、组件访问动态内容

问：组件可以直接访问Vue实例中的数据？

答：不可以，即便可以访问，那么Vue实例中的data属性会越变越乱。那么组件中的数据应该放置哪里呢？应该放置在属于自己组件下的data中，并且这个data属性必须是一个函数，返回一个键值对类型

模板：

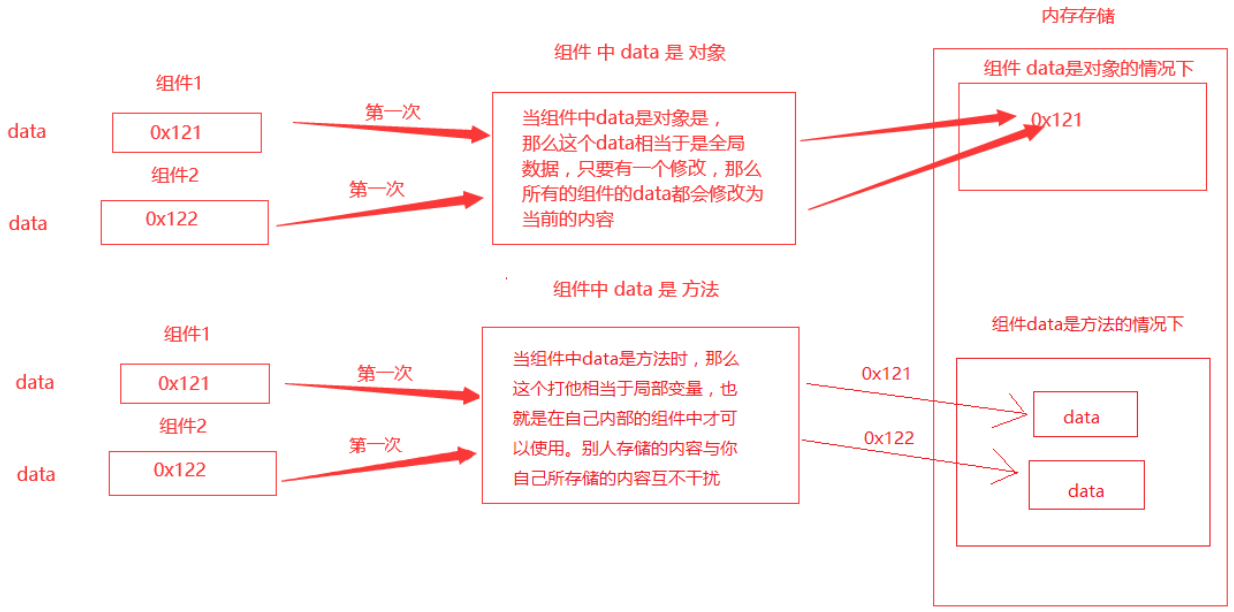
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <!--
    注意点：
    1、注意使用标签
    2、注意使用组件时，不能使用Vue实例对象的数据，
    必须在组件内建立一个data方法返回一个键值对内容
    3、组件使用动态数据必须要有插值语法
  -->
  <div id="app">
    <apps></apps>
  </div>
  <template id="tpl">
    <div>
      <h1>这是组件的水（静态）标签</h1>
      <h1>{{content}}</h1>
    </div>

  </template>
</body>
<script src="../../JavaScript/vue.js"></script>
<script>
  Vue.component(
    'apps',{
      template:"#tpl",
      data(){
        return {content:"这是组件的真实（动态）标签"}      You, 1秒钟前 • Uncommitted ch
      }
    })
  // 这是一个Vue实例对象
  const vue=new Vue({
    el:"#app"
  })
</script>
</html>
```

2、 **“组件设计时其data属性必须是一个函数”的原理**

最主要的目的在于函数进行每一次调用的时候，它所调用的当前的数据永远都不会干扰到其他的数据，永远传递的是你当前传递的值，如果组件中的data是一个对象的话，那么其他组件对表面上看起来是自己组件的data的属性所做的修改那么实际上修改的就是全部data属性的值

原理图



总而言之，组件中的data属性为什么是方法，简单来讲就是为了确保自己的data属性所保存的值不会被其他组件所保存的值所干扰到

五、***父子组件通信（如何将组件中的数据传递到小组件）**

1、概念：一般进行前端网页开发通常都会把每一部分的内容封装成一个组件将大组件获取到的数据传递给小组件，不可能让每一个小组件单独请求数据，那么这样的操作会让服务器的压力变大

2、**父组件数据传递子组件的数据（props属性）**

（1）通过props进行传递（父组件传递子组件）

（2）通过自定义事件（emit）向父组件发送消息（子组件传递父组件）

（3）props属性的使用

1、格式：

（1）数组类型

props：['数据'， ‘数据’]

```
    <!-- 将变量赋父组件的值 -->
    <apps :new_lists="list" :new_title="title"></apps>
  </div>

</body>
<!-- 定义的模板需要放的动态数据是prop所定义的变量 -->
<template id="son">
  <div>
    <h1>{{new_title}}</h1>
    <div>{{new_lists}}</div>
  </div>
</template>

<script src="../../../JavaScript/vue.js"></script>
<script>
//Vue实例本身也可以是一个组件，它是所有的组件的父组件
  const vue=new Vue({
    el:"#app",
    data:{
      list:[
        '这是第一个内容', '这是第二个内容',
        '这是第三个内容', '这是第四个内容',
        '这是第五个内容', '这是第六个内容',
        '这是第七个内容', '这是第八个内容',
        '这是第九个内容', '这是第十个内容',
      ],
      title:"这是一个标题"
    },
    components:{
      'apps':{
        template:"#son",
        data(){
          return { lists :"这是一个列表"}
        },
        props:['new_lists',"new_title"]
      }
    }
  })
</script>
</html>
```

注意：遍历props传递过来的对象必须要写在你写插值语法所在的标签上，其他地方遍历子组件所传递的数值那么这个就是不能用的

（2）对象类型

1、限定父组件传递的对象类型

Props:变量：类型

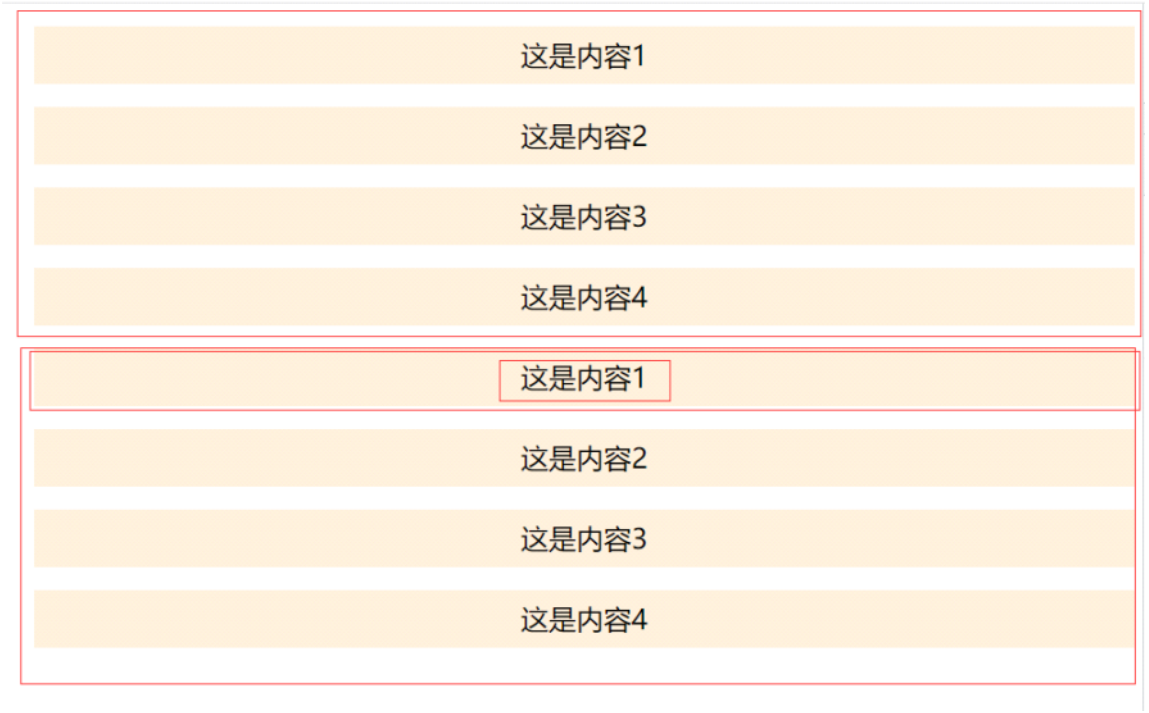
2、赋予父组件的初始值

props：{

Type:"传递的数据类型"，

Default:"设置不传递值时所显示的内容"

}



require属性：代表着这个属性是必传的

3、props属性的类型

```
Vue.component('my-component', {
  props: {
    // 基础的类型检查 (`null` 匹配任何类型)
    propA: Number,
    // 多个可能的类型
    propB: [String, Number],
    // 必填的字符串
    propC: {
      type: String,
      required: true
    },
    // 带有默认值的数字
    propD: {
      type: Number,
      default: 100
    },
    // 带有默认值的对象
    propE: {
      type: Object,
      // 对象或数组默认值必须从一个工厂函数获取
      default: function () {
        return { message: 'hello' }
      }
    },
    // 自定义验证函数
    propF: {
      validator: function (value) {
        // 这个值必须匹配下列字符串中的一个
        return ['success', 'warning', 'danger'].indexOf(value) !== -1
      }
    }
  }
})
```

```
function Person (firstName, lastName) {
  this.firstName = firstName
  this.lastName = lastName
}

Vue.component('blog-post', {
  props: {
    author: Person
  }
})
```

props使用驼峰命名产生的问题

问题：当前你在template中使用含有驼峰标识符会产生错误

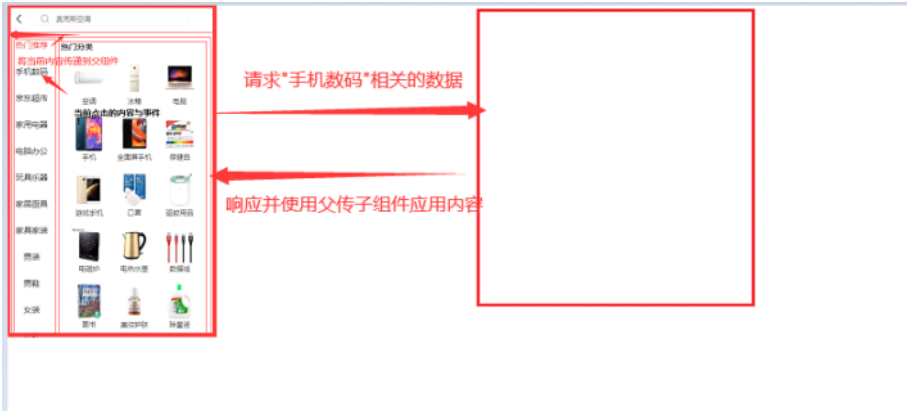
解决方式：将所有的单词中大写的单词转化为“-”进行连接

如下

```
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div id="app">
    <apps :new_-lists="list">
      <!--
        v-bind标签是不支持驼峰标识，因此在v-bind中如果要是使用驼峰标识就必须使用-进行连接转译      You, 1
      -->
    </apps>
  </div>
</body>
<template id="tpl">
  <div>
    <h1>{{new_Lists}}</h1>
  </div>
</template>
<script src="../../../JavaScript/vue.js"></script>
<script>
  const vue=new Vue({
    el:"#app",
    data:{
      list:['列表项1', '列表项2', '列表项3', '列表项4', '列表项5', '列表项6']
    },
    components:{
      'apps':{
        template:"#tpl",
        // 1、props:['new_lists']
        props:{
          new_Lists:{
            type:Array,
            default(){
              return ['None']
            },
          }
        }
      //   3、props:{
      //     new_lists(val){
      //       return ['列表项1'].indexOf(val)!=-1
      //     }
      //   }
      }
    }
  })
</script>
</html>
```

3、**子组件传递父组件emit属性**

（1）作用：当用户操作发生改变时，那么所对应的事件与内容将传递给父组件，父组件将传递数据至最外层的父组件再向服务器发送数据并进行响应



（2）格式：

This.$emit("自定义事件","数据")

父组件就必须使用v-on接入emit传递的值对数据进行修改

注意：在Js代码编写模块时不能使用驼峰标识，而在vue脚手架编写模块时，可以使用驼峰。

六、父子组件之间的访问（如何实现子组件访问父组件的元素或者相反操作）

1、父组件访问子组件（-children/-refs）

（1）概念：通过父组件的某一个对象直接访问到子组件的某一个对象

（2）格式：

（1）通过$children访问子组件里面的值，并且可以进行修改（不建议

原因：需要通过下标去拿组件的值，组件繁多不易拿取

）

```
this.$children[0].ChildItems()
for (let i =0;i<this.$children[0].list.length;i++){
  this.$children[0].list[i]=i+2-1/2
}
console.log(this.$children[0].list);
```

（2）通过**$refs**访问子组件的值（不会受到元素的添加而改变数量）

```
    <button @click="refClick()">点击我</button>
  </div>
  <template id="tpl">
    <div>
      <h1>{{content}}</h1>
    </div>
  </template>
</body>
<script src="../../../JavaScript/vue.js"></script>
<script>
  new Vue({
    el:"#app",
    methods: {
      refClick(){
        console.log(this.$refs.first.content);
        this.$refs.first.content="这是被修改后的内容"
        console.log(this.$refs.first.content);
      }
    },
    components:{
      'apps':{
        template:"#tpl",
        data(){
          return{
            content:"这是内容",
            list:["这是第一条", "这是第二条", "这是第三条"]
          }
        }
      }
    }
  })
</script>
</html>
```

注意：如果要通过refs拿子组件的值，那么就必须要在组件标签添加一个ref= “自定义名称”，那么父组件拿子组件就通过这个自定义

名称就可以拿到相对应的内容，不会因为组件的增多而去猜测他所在的下标

2、子组件访问父组件（$parents/$root）

（1）$parents 子组件访问父组件

1、概念：可以通过任意组件来访问其在它上层的父组件（不建议）

原因：降低组件之间的灵活性，提升耦合度

```
<body>
  <div id="app">
    {{content}}
    <apps></apps>
  </div>
  <template id="tpl">
    <div>
      <button @click="btn()">按钮</button>
    </div>
  </template>
</body>
<script src="../../../JavaScript/vue.js"></script>
<script>
  new Vue({
    el:"#app",
    data: {
      content:"这是一个消息",
      list: ['这是内容1', "这是内容2", "这是内容3"]
    },
    components:{
      'apps':{
        template:"#tpl",
        methods: {
          btn(){
            this.$parent.content="这个消息来自子组件的操作"
          }
        },
      }
    }
  })
</script>
</html>
```

（2）$root访问根组件（其实就是访问Vue实例对象）

slot插槽

2022年11月3日 16:29

注意：每次使用具名插槽的时候一定要检查slot属性的设置的位置（不能是使用组件的标签）

一、插槽slot的基本使用

（1）作用：组件变得更加具有扩展性，可以让使用者决定组件内部内容显示什么

（2）格式：

1、传值方式（比较麻烦）

(1)在组件内设置一个插槽

(2)在引用组件的地方设置你想传递的内容即可

2、slot设置默认值（针对少数类型不相同进行设置）

直接在slot标签内直接使用

```
    <div>这是一个span标签</div>
    <span>哈哈哈</span>
  </apps>
  <apps></apps>
  <apps>
    <div>这是一个单选框</div>
    <input type="radio" name="" id="">
  </apps>
</div>
<template id="tpl">
  <div>
    <div>
      <slot>
        <div>这是一个按钮</div>
        <button>按钮</button>
      </slot>
    </div>
  </div>
</template>
</body>
<script src="../../../JavaScript/vue.js"></script>
<script>
  new Vue({
    el:"#app",
    components:{
      'apps':{
        template:"#tpl",

      }
    }
  })
</script>
</html>
```

二、具名插槽

（1）概念：简单来讲就是当你只需替换其中一个内容，可以使用具名插槽对组件内的内容进行替换

（2）使用：

1、slot标签设置name属性

2、在想要替换的插槽位置设置一个slot对象

```
<!--

 使用嵌套传递内容到插槽必须将slot属性设置在父元素上
-->
<body>
  <div id="app">
    <apps>
      <input type="text" name="" id="" placeholder="左边" slot="left">
      <label for="" slot="mid">中间</label>
      <div slot="right">
        <input type="text" name="" id="" placeholder="右边" >
        <input type="checkbox" name="" id="">
      </div>
    </apps>
    <br>
    <apps></apps>
  </div>
  <template id="tpl">
    <div>
      <slot name="left">
        <input type="button" value="左边">
      </slot>
      <slot name="mid">
        <input type="button" value="中间">
      </slot>
      <slot name="right">

        <input type="button" value="右边">
      </slot>
    </div>
  </template>
</body>
<script src="../../../JavaScript/vue.js"></script>
<script>
  new Vue({
    el:"#app",
    components:{
      'apps':{
        template:"#tpl"
      }
    }
  })
</script>
```

注意点：**具名插槽不能直接写在注册组件的标签上必须在其下建立一个标签用于存放**

3、编译作用域：使用某一个元素的时候首先vue会去找他所在的vue的作用域调用相关的内容，实际上就是在某一个组件或者实例中调用的内容就是自己这个范围内所调用的数据

4、**作用域插槽（常用）**

（1）概念：父组件替换插槽里面的标签，但内容是子组件提供的

（2）格式：

1、（已废弃）

首先在插槽内定义一个变量存放子组件的数据

其次在父组件的嵌套上加一个slot-scope把之前插槽的变量存放过来

最后进行遍历

```
      <h1>这是父组件</h1>
      <div slot-scope="data">
        <span v-for="li in data">{{li.join("-")}}</span>
      </div>
    </apps>
  </div>
  <template id="tpl">
    <div>
      <slot :datas="list">
        <ul>
          <li v-for="item in list">{{item}}</li>
        </ul>
      </slot>
    </div>
  </template>

</body>
<script src="../../JavaScript/vue.js"></script>
<script>
  new Vue({
    el:"#app",
    components:{
      'apps':{
        template:"#tpl",
        data(){
          return {
            list:['这是内容1','这是内容2','这是内容3']
          }
        }
      }
    }
  })
</script>
```

2、最新语法：

插槽上给一个name='标记'，绑定：data='变量'，调用时#name='get'，{{get.data}}

模块化开发

2022年11月10日 9:07

一、模块化开发

早期前端使用js所产生的问题

1、变量名冲突问题

2、代码复用性差

解决问题：变量名冲突问题

创建闭包----->产生---->代码不可复用------>建立模块化---->外部引用--->通过模块化进行引用

闭包的创建

所谓的模块化无非就是通过把所有的属性封装起来，可以直接使用这个名称调用内部的属性

3、代码模块化

index文件

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
<script src="./模块化开发（ES5）model_first.js"></script>
<script src="./模块化开发（ES5）model_second.js"></script>
<script src="./模块化开发（ES5）selectA.js"></script>
</html>
```

Model_first JS文件

```
var content =(
  function(){
   // 创建模块化
   var arr=[];
   var name="潇潇";
   var age=20;
   var tip=true
   function doing(name,age) {
     console.log("这是你的名字"+name+"\n"+"这是你的年龄"+age+"\n"+name+"在吃饭");
   }
   if(tip){
     doing(name, age)
   }
   arr.name=name
   arr.doing=doing
   return arr
  }
)()
// 创建闭包：确保有作用域以免变量被修改
// ()(
//   function(){

//   }
// )
// var content = ( //创建模块化
//   function () {
//     var arr = [];
//     var name = "潇潇";
//     var age = 20;
//     var tip = true
//     function doing(name, age) {
//       console.log("这是你的名字" + name + "\n" + "这是你的年龄" + age + "\n" + name + "在吃饭");
//     }
//     if (tip) {
//       doing(name, age)
//     }
//     arr.name = name
//     arr.doing = doing
//     return arr //返回内容保存到模块化名
//   }
// )()
// 别处文件调用使用 模块名.属性进行调用
```

Model_second JS文件

```
(
  function(){
    var name = "这是我的名字";
    var tip = false
    function doing(name) {
      console.log(name+":笑笑");
    }
    if (tip==false) {
      doing(name)
    }
  }
)()
```

SelectA JS文件（调用model_first js文件）

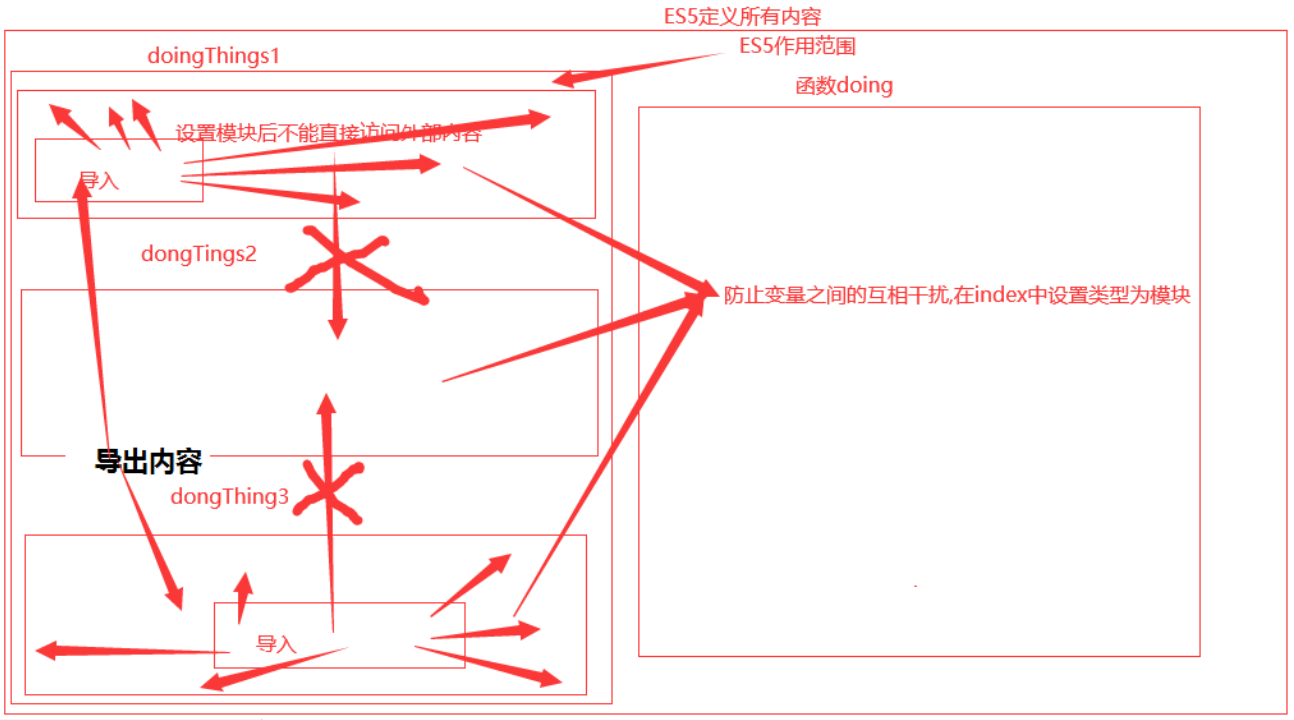
```
content.name="佚名"
content.doing(content.name,12);
```

二、ES6模块化的使用（ES6模块化作用域）

注意：**使用vscode产生跨域请求错误可以使用live server插件进行解决此问题，并且运行时请切换至带有html后缀的文件右击选择open with live server**

1、为什么在script标签中加入type=module就不能使用了

过程见图：



2、常用导入导出方式：

（1）首先在主页面先分别导入要使用的js文件

（2）在所在的页面设计一个js文件

（3）在使用export { 导出的内容 } 格式对使用的变量或函数进行导出

（4）使用import { 将导出内容进行导入 } from "js文件相对路径（注意文件后缀）"

3、导出方式：

（1）、export{任意内容}（推荐）

（2）、export 数据类型 数据=值

（3）、export function 函数名/类名(){}

（4）、export default {

}(只能写一次)

Export default 导出时不用命名，导入时他的名称可以自定义命名

4、导入方式（import）：

格式：

1、直接导入：import {} from “相对路径(**加上文件后缀**)”

2、默认导入：import 自定义命名 from “相对路径(**加上文件后缀**)”（只用于export default)

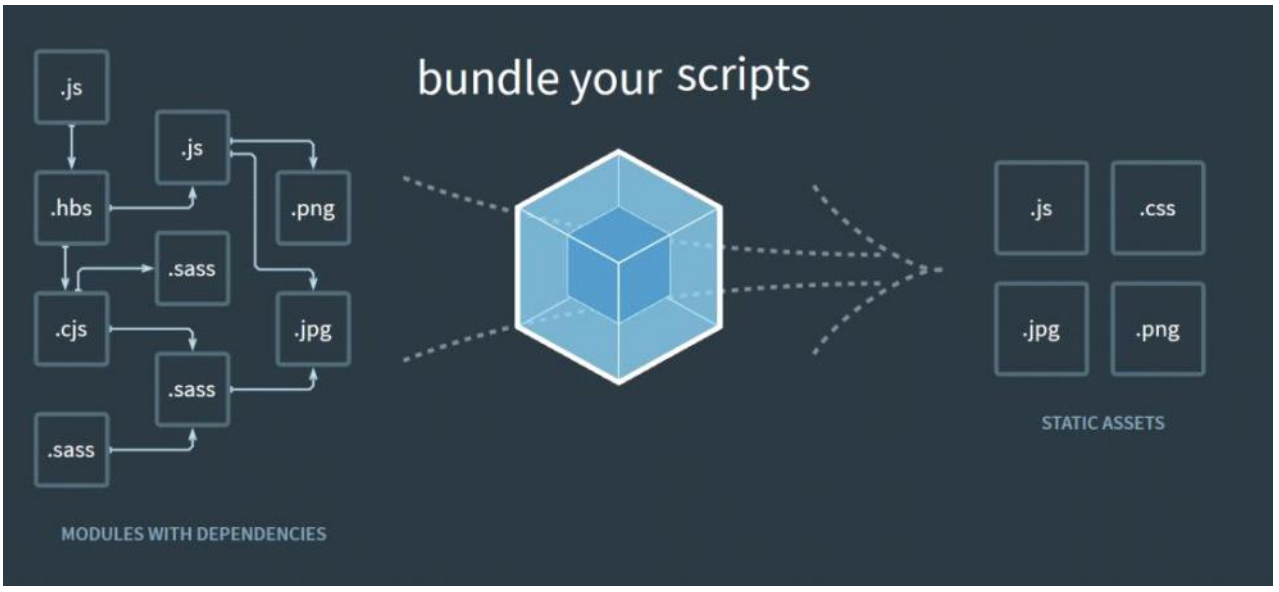
3、全部导入：import * as 自定义命名 from “相对路径(**加上文件后缀**)” 导入所有内容,通过自定义命名.属性的方式拿取内容

webpack模块化开发

2022年11月12日 17:03

一、webpack介绍

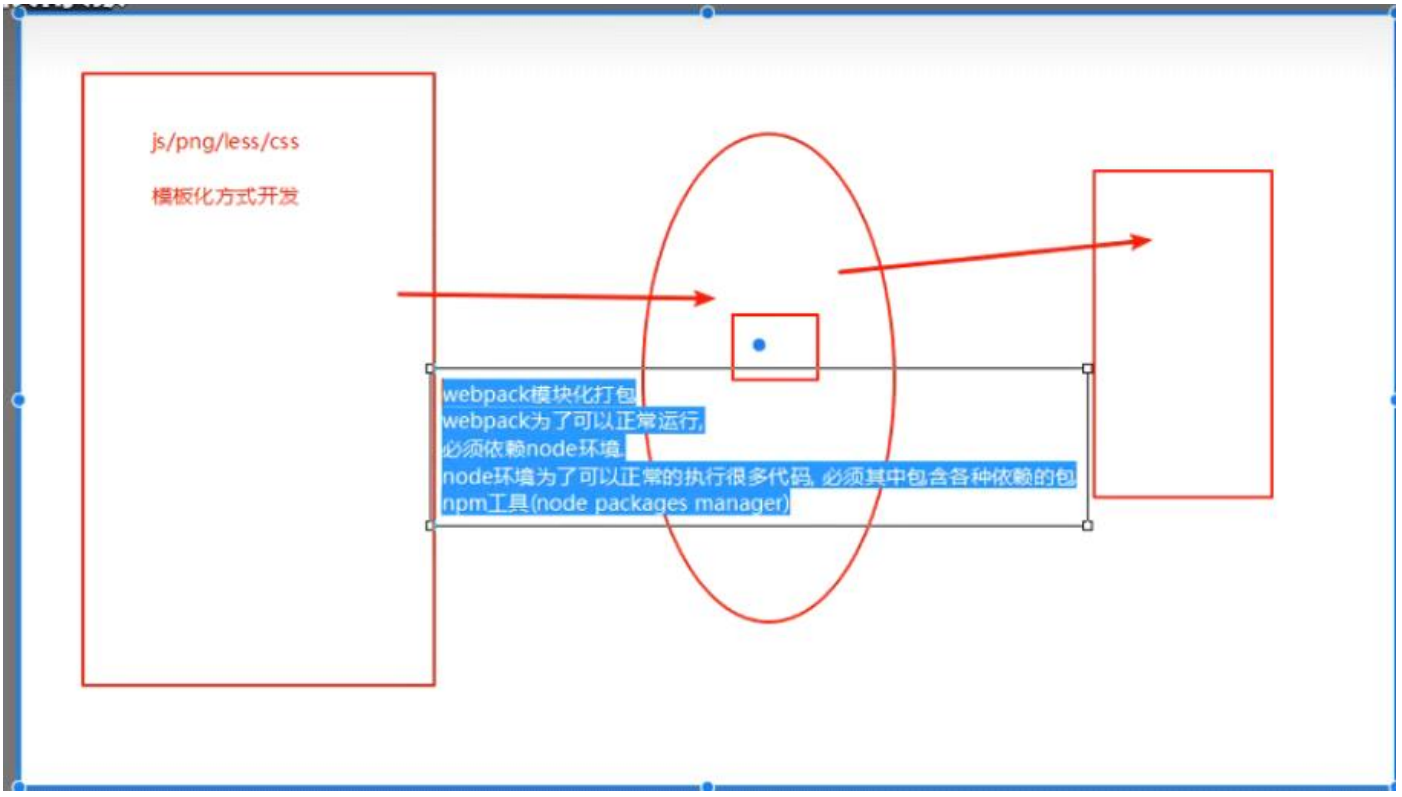
（1）概念：一个可以将项目进行**模块化**以及**打包**的工具



解析：从官方给出的图中我们可以知道，现实生活中所有的数据都会存储在服务器中，而网站中所展示的内容也是如此。但是并**不是所有的类型都可以在web服务器中展示**的，例如sass、cjs、hbs等类型的文件,将他们的通过一个工具进行打包转化放到浏览器上，**变成浏览器所可以识别的类型**。还有一点就是如果你**编写的语法与浏览器的语法不兼容**，那么可以通过打包工具进行转化

简单来讲打包工具的作用就是将我们写的代码转化成浏览器可识别的内容

（2）npm、node与webpack之间的关系



其依赖关系：webpack---依赖--->node环境-----为了管理包---->npm(node 包管理)

二、webpack的使用*

格式：

1、手动webpack编译

终端进入到相应的位置输入

Webpack 编译之前位置 编译之后位置

编译的文件只需要找最后实现的结果即可，其他的文件webpack会通过你引入的路径去找到相关的内容，从而进行编译

2、自动webpack编译（无需配置路径（必须在当前目录下））

（1）创建一个webpack.config.js文件

（2）设置相关的出口和入口

```
module.exports={
  entry:"./src/index.js",//设置文件的入口
  output:{
    path:"./dist",//设置文件的出口路径
    filename:"result.js"//设置文件名称
  }
}
```

产生导出文件不是绝对路径问题，但是不可以直接使用当前文件夹的绝对路径，毕竟你一旦改变其他路径，那么当前的路径也要发生改变，那怎么办？

因此我们需要动态获取当前的绝对路径，就使用node.js中的path模块来动态获取当前的绝对路径，而这个path模块是在node.js中的一个包中，所以我们要对它进行初始化

（3）npm init 将nodejs中的一些基础包初始化到当前文件夹下生成package.json

Package.json 是任何一个项目想要单独的依赖某一个文件就必须使用这个

当package.json中有某一些依赖的项，那么就必须单独使用npm install 再来单独安装某一些依赖项

（4）在output下的path属性中使用resolve函数，将__dirname(当前这个路径的全局变量保存的就是我当前文件的绝对路径)与输出的内容拼接

格式:path.resolve(__dirname,"输出文件夹名")

__dirname 在nodejs中获取当前文件的上下文（绝对路径）

最后定位到当前文件夹下的终端输入webpack即可

3、项目开发最经常用

在以上第二大步完成后,打开由node.js生成的package.json包中，找到script对象再原有的基础之上再一次添加"名称":"webpack"（创建映射名称与web pack之间的映射）

以后只要运行webpack就只要在当前终端输入npm run 名称即可

补充：第三种方法所运行的webpack他会**优先找到本地的webpack，如果当前文件下的webpack找不到的话那么就会去全局webpack里面找**

注意：有时侯在项目中所使用的webpack可能因为考虑到兼容性等问题会单独安装一个webpack，但是此时**本地的webpack与全局的webpack他们之间的版本可能是不一样**的，并且**本地的webpack是会跟项目的进展保持实时同步**的，因此我们除了要安装全局的webpack还要安装本地的

webpack

补充：--save--dev将我们所配置的内容保存到"dev"中

注意：**只要在终端使用的webpack那么就是使用全局的，一旦在package.json中映射一个内容到webpack中，那么这个就是使用局部的webpack**

终端：（本地执行）npm run 内容

终端：（全局）webpack