HTML_Vue

Vue初体验

一.Vue.js-响应式

1.创建Vue对象的时候,传入了options : {}

a.{}中包含了el属性:该属性决定了这个Vue对象挂载到哪一个元素上,这里挂载到id为app的元素上; b.{}中包含了data属性:该属性中通常会存储一些数据:这些数据可以是我们直接定义出来的;来自网络;从服务器加载.

2.浏览器执行代码的流程

div中的代码显示对应的HTML; 执行到script创建的Vue实例来管理app,Vue实例中包含变量,并且对原HTML进行解析和修改.

3.Vue.js-响应式代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id = "app">
        <h1>{{message}}</h1>
        <h2>{{name}}</h2>
    </div>
    <script src="js/vue.js"></script>
    <script>
        // let(变量);const(常量)
        // 编程范式:声明式编程
        const app = new Vue ({
            el : "#app",//用于挂载管理的元素
            data : {//定义数据
                message : "你终于不卡了,啊啊啊",
                name : "张三"
            }
        })

    </script>
    <!--
        元素js的做法(编程范式:命令式编程)

        1.创建div元素,设置ID的属性;
```

```
        2.定义一个变量叫,message;
        3.将message变量放在前面的div元素中显示;
        4.修改message的数据:"今天天气不错";
        5.将修改后的数据再次替换到div元素

        -->
</body>
</html>
```

注:文件结构

二.Vue.js-List

1.复杂的数据展示:数据列表

-比如我们现在从服务器请求过来一个列表;自定义一个列表 -展示在HTML中

2.HTML代码中,使用v-for命令

1.该命令不需要再JavaScript代码中完成DOM的拼接相关操作 2.Vue的响应式,当我们数组中的数据发生改变时,界面会自动改变 3.打开开发者模式的console,可以进行测试

3.Vue.js-列表展示代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="appList">
        <ul>
            <li v-for="item in movies">{{item}}</li>
        </ul>

    </div>
    <!-- 引入Vue.js -->
    <script src="js/vue.js"></script>
    <script>
        const appList = new Vue({
            el : "#appList",
            data : {
                message : "hello,Vue",
                movies: ["电影1","电影2","电影3","电影4","电影5","电影6",]
            }

        })

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

三.Vue.js-计数器

1.实现一个小的计数器

点击+,计数器+1 点击-,计数器-1

2.使用新的指令和属性

新增属性:methods,该属性用于在Vue对象中定义方法 新的指令:@click,该指令由于监听某个元素的点击事件,并且需要指令当前发生点击,执行的方法 (方法通常在methods中定义的方法)

3.Vue.js-列表展示代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="counter_app">
        <h2>当前计数:{{counter}}</h2>
        <!-- v-on:click v-on监听的click事件,
        当事件变得复杂的时候不使用 -->
        <!-- <button v-on:click="counter++">+</button> -->
        <!-- <button v-on:click="counter--">-</button> -->

        <button v-on:click="add">+</button>
        <button v-on:click="sub">-</button>

        <!-- //increment增量
        <button @click="increment">+</button>
        //decrement减量
        <button @click="decrement">-</button> -->

    </div>
    <script src="js/vue.js"></script>
    <script>
        //语法糖:简写v-on:click==@click
        let counter_app = new Vue({
            el : "#counter_app",
            data : {
                counter : 0
            },

            //方法1:
            methods : {
                add : function(){
```

```
                    this.counter++//this指当前对象
                    console.log("add被执行")
                },
                sub : function(){
                    this.counter--
                    console.log("sub被执行")
                }
            }
            // 方法2:
            // methods : {
            //     increment() {
            //         this.counter++
            //     },
            //     decrement(){
            //         this.counter--
            //     }
            // }

        })
    </script>
    <!--
        1.拿button元素
        2.添加监听事件


    -->


</body>
</html>
```

注:代码测试方法1

```
//方法1:
            methods : {
                add : function(){
                    this.counter++//this指当前对象
                    console.log("add被执行")
                },
                sub : function(){
                    this.counter--
                    console.log("sub被执行")
                }
            }
```

注:代码测试方法2

```
// 方法2:
            methods : {
                 increment() {
                     this.counter++
                 },
                 decrement(){
                     this.counter--
                 }
             }
```

四.Vue.js-MVVM

1.Vue的MVVM

Modem View ViewModem;

ViewModem是Modem和View的桥梁;

View DOM ViewModem:定义指令,监听事件DOM Listences, Data Bindings,Modem Plian javaScript Objects.

2.Vue的View层

1.视图层;

2.在前端开发中指DOM层;

3.作用是给客户提供各种信息.

3.Vue的Modem层

1.数据层;

2.数据可能是我们固定的死数据,更多的是来自服务器,从网络上请求下来的数据;

3.在计数器案例中,就是从后面抽取出来的obj.

4.Vue的VueModem层

1.视图模型层;

2.视图模型层是View和Modem沟通的桥梁;

3.一方面它实现了Data Binding,数据绑定,将Modem的改变实时的反应到View中;

4.一方面它实现了DOM Lisenter,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.

五.Vue.js-Optinons

1.元素element

```
el:
            类型:string|HTMLElement
            作用:决定之后Vue实例会管理哪一个DOM
            el :"#app" == el:document.querySelector("#app")
```

2.数据DATA

```
data:
            类型:Object|Function(组件中data必须是一个函数)
            作用:Vue实例对应的数据对象
```

3.方法method

```
methods:
            类型:{[key:string]:Function};key为函数名
            作用:定义属于Vue的一些方法,可以在其他地方调用,也可以在指令中使用
```

4.其他说明

开发中什么叫方法,什么为函数 方法:method,类里面的东西叫方法,与实例挂钩 函数:function,

代码规范: 1.等号左右两边留一个空格 2.冒号之后留空格 3.缩进,一把是4个空格;前端一般缩进2个空格会更加规范些(流行) 4.CLI脚手架中的文件夹editconfig协助规范2个空格

六.Vue.js-生命周期

1.生命周期的介绍

生命周期:事务从诞生到死亡的整个过程

Vue.js查看生命周期的代码流程:

- 从github获取源码: github的Branch中选择tags中的版本, 稳定版会打个tags(开发过程中,叫debug版本,发布版本是release版本)
- src中是源码,core核心代码,vdom中的index.js入口, instance文件夹下,index.js。

2.Vue.js生命周期内容

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、编译模板、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。通俗说就是Vue实例从创建到销毁的过程，就是生命周期。

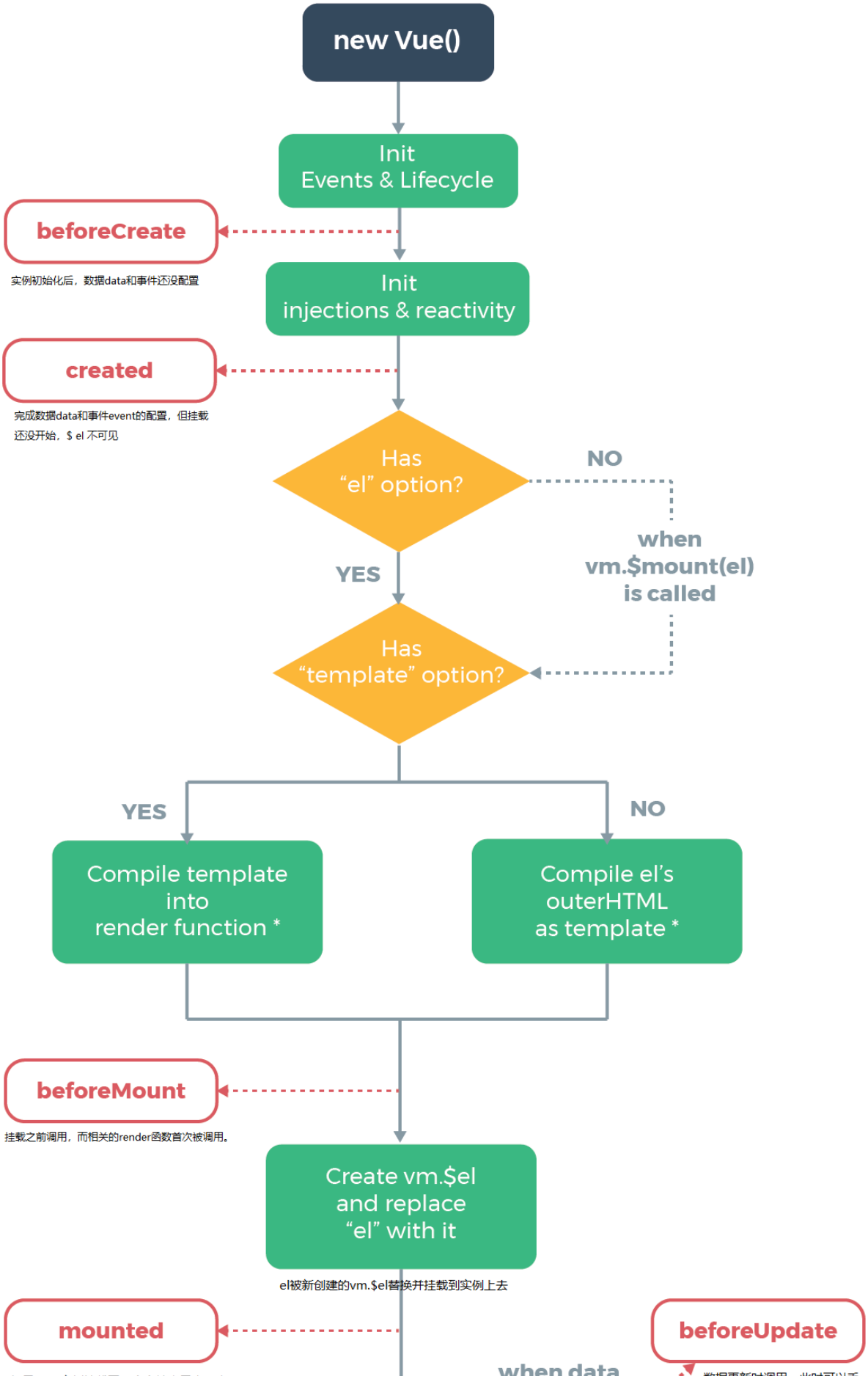
在Vue的整个生命周期中，它提供了一系列的事件，可以让我们在事件触发时注册js方法，可以让我们用自己注册的js方法控制整个大局，在这些事件响应方法中的this直接指向的是vue的实例。

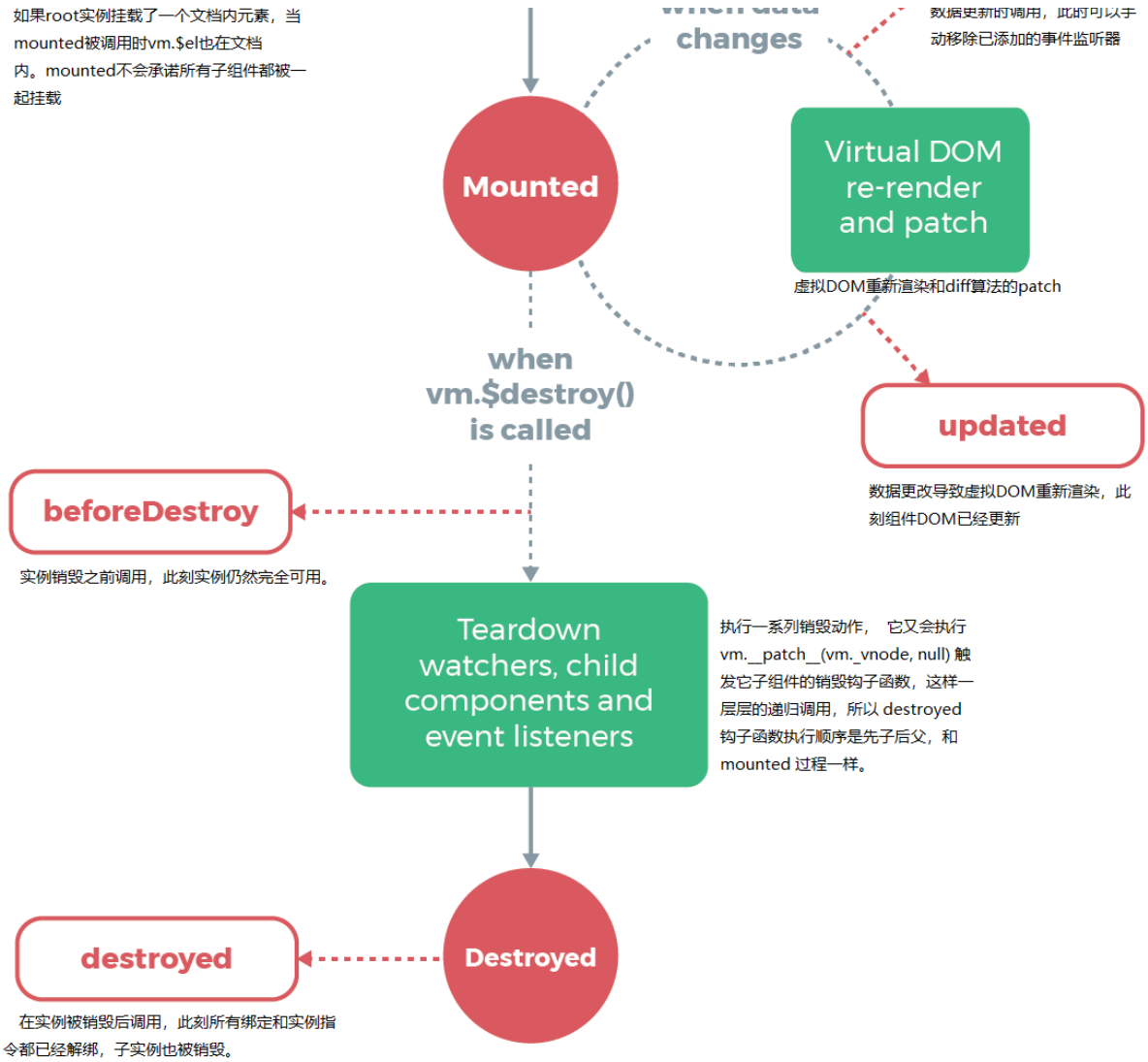
Vue.js 实例生命周期：

- init：在实例开始初始化时同步调用，此刻数据观测和事件等都未初始化。2.0中更名为beforeCreate。
- created：在实例创建之后调用，此刻已完成数据绑定，事件方法。但是尚未开始DOM编译，即未挂载到document中。
- beforeComplie：在DOM编译前调用。2.0废弃了该方法，推荐使用create。
- beforeMount：2.0新增的生命周期钩子，在mounted之前运行。

- complied：在编译结束时调用。此时所有指令已生效，数据变化已能触发DOM更新，但不保证$el 已插入文档。2.0中更名为mounted。
- ready：在编译结束和el第一次插入文档之后调用。2.0废弃了该方法，推荐使用mounted。这个变化其实已经改变ready这个生命周期状态，相当于取消了在el第一次插入文档之后调用。2.0废弃了该方法，推荐使用mounted。这个变化其实已经改变ready这个生命周期状态，相当于取消了在el 首次插入文档后的钩子函数。
- attached：在vm.el插入DOM时调用，ready会在第一次attched后调用，操作el插入DOM时调用，ready会在第一次attched后调用，操作el 必须使用指令或者实例方法，直接操作vm.$el不会触发钩子。2.0废弃了该用法，推荐在其他钩子中自定义检查是否已挂载。
- detached：同attached类似，该钩子在vm.$vm从DOM删除时调用，而且必须是指令或者实例方法。2.0同样废弃了该方法。
- beforeDestroy：在开始销毁实例时调用，此刻实例仍然有效。
- destroyed：在实例被销毁后调用，此刻所有绑定和实例指令都已经解绑，子实例也被销毁。
- beforeUodate：2.0新增的生命周期钩子，在实例挂载之后，再次更新实例时会调用该方法，此时尚未更新DOM结构。
- uodated：2.0新增的生命周期钩子，在实例挂载之后，再次更新实例并更新DOM结构后调用。
- activated：2.0新增的生命周期钩子，需要配合动态组件keep-live属性使用。在动态组件初始化渲染的过程中调用该方法。
- deactivated：　2.0新增的生命周期钩子，需要配合动态组件keep-live属性使用。在动态组件移出的过程中调用该方法。

3.Vue.js生命周期图示


new Vue()
Init
Events & Lifecycle
beforeCreate
实例初始化后，数据data和事件还没配置
Init
injections & reactivity
created
完成数据data和事件event的配置，但挂载
还没开始，$ el 不可见
Has
“el” option?
NO
when
vm.$mount(el)
is called
YES
Has
“template” option?
YES
NO
Compile template
into
render function *
Compile el’s
outerHTML
as template *
beforeMount
挂载之前调用，而相关的render函数首次被调用。
Create vm.$el
and replace
“el” with it
el被新创建的vm.$el替换并挂载到实例上去
mounted
beforeUpdate
when data



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

Vue ES6语法补充

一.Vue.js:const的使用和作用

1.const关键字的使用和作用

const关键字

在很多语言中已经存在,C/C++中,主要的作用是将某个变量修饰为常量;

在JS中也是如此,使用const修饰的标识符为常量,不可以再次赋值

什么时候使用const?

当我们修饰的标识符不会被再次赋值时,就可以使用const来保证数据的安全性

建议:

在ES6开发中,优先使用const,只需要改变某一个标识符的时候才使用let

const的注意

const注意一:

const a=20;

a=30;//错误:不可以修改

const注意二:

const name;//错误:const修饰的标识符必须赋值

const注意三:

常量的含义是指向的对象不能改变,但是可以改变对象内部的属性

name指向字符串对象,保存的对象的内存地址(let是修改了内存地址),属性的修改不会导致内存地址修改

const的内存地址是不可以修改的

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:const的使用和作用</title>
</head>
<body>
    <script>
         const obj={
             name:"张小豆",
             age:2,
             height:0.9
         };

         console.log(obj);

         obj.name="小青年";
         obj.age=11;
         console.log(obj);

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

二.Vue.js:let的块级作用域

1.let的使用

JS中只有函数有作用域

ES5之前因为if和for都没有块级作用域的概念,所以很多时候,我们都必须借助于Function的作用域来解决应用外面变量的问题

ES6中加入了let,let它就是if和for的块级作用域

2.代码示例

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:let的块级作用域</title>
</head>

<body>
    <button>按钮1</button>
    <button>按钮2</button>
    <button>按钮3</button>
    <button>按钮4</button>
    <button>按钮5</button>
    <button>按钮6</button>
    <script src="./js/vue.js"></script>
    <script>


        // 1.变量作用域:变量在什么范围内是可用的
        // {
        //     var name = "张小豆";
        //     console.log(name);
        // }
        // console.log('获取的是张小豆',name);


        // 2.没有块级作用域引起的问题
        // var func;
        // if (true){
        //     var name="吕懒懒";
        //     func=function(){
        //         console.log('获取的是吕懒懒',name)
        //     }
        // }
        // name="二郎神";
        // func()
        // console.log(name)
```

```
        //3.没有块级作用域引起的问题:for的块级
        // 为什么闭包可以解决问题:函数是一个作用域
        // var btns = document.getElementsByTagName('button');
        // for (var i = 0; i < btns.length; i++) {
        //     btns[i].addEventListener('click', function () {
        //         console.log('第' + i + '个按钮被点击');

        //     })

        // }


        // 闭包的写法for
        // var buttons = document.getElementsByTagName('button');
        // for (var i = 0; i < buttons.length; i++) {
        //     (function (i) {//函数的有自己的参数值
        //         buttons[i].addEventListener('click', function () {
        //             console.log('第' + i + '个按钮被点击');
        //         })
        //     })(i)


        // }


        // 闭包的写法函数有自己的作用域
        // var name="张三"
        // function abc(name){
        //     console.log(name)
        // }
        // name="李四"
        // abc("王五")


        // let的块级作用域
        var buttons = document.getElementsByTagName('button');
        for (let i = 0; i < buttons.length; i++) {
            buttons[i].addEventListener('click', function () {
                console.log('第' + i + '个按钮被点击');

            })

        }
    </script>
</body>

</html>
```

三.Vue.js:ES6对象字面量的增强性写法

1.对象增强写法

ES6中,对对象的字面量进行了很多增强

1.属性的增强型

2.函数的增强型

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:ES6对象字面量的增强性写法</title>
</head>
<body>

     <script>
        //  const obj={
        //     name: "张三",
        //     age:18,
        //     run:function(){
        //         console.log("在休息");
        //     },
        //     eat:function(){
        //         console.log("还要玩");
        //     }

        //  }//对象的字面量
        // 1.属性的增强型
        const name="张小振";
        const age=29;
        const height=1.80;

        // ES5的写法
        // const obj={
        //     name :name,
        //     age:age,
        //     height:height,
        // }
        // ES6的写法
        // const obj={
        //     name,
        //     age,
        //     height,
        // }
        // console.log(obj);

        // 2.函数的增强型
        // ES5的写法
```

```
        //  const obj={
        //     run:function(){
        //         return "还要玩";
        //     },
        //     eat:function(){
        //         console.log("还要吃");
        //     }
        //  };

        // ES6的写法
        const obj={
            run(){
                console.log("在休息");
            },
            eat(){
                console.log("还要吃");
            }
        };
        console.log(obj.run);
    </script>
</body>
</html>
```

Vue插值操作

一.Vue.js-mustache

1.mustache说明

插值操作-Mustache语法;

插值操作mustache(胡子/胡须);

如何将data中的文本数据,插入到HTML中;

可以通过Mustache语法(也就是双大括号),数据是响应式.

2.mustache代码示例

- 可以插入到标签中
- 可以使用了两个Mustache
- 可以使用表达式

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>
```

```
</head>
<body>
    <!--
        插值操作-Mustache语法;
        插值操作mustache(胡子/胡须);
        如何将data中的文本数据,插入到HTML中;
        可以通过Mustache语法(也就是双大括号),数据是响应式.

    -->
    <div id = "app_mustache">
        <!-- 插入到标签中 -->
        <h2>Hello{{message}}</h2>
        <!-- 使用了两个Mustache -->
        <h2>{{firstName}}{{lastName}}</h2>
        <h2>{{firstName + lastName}}</h2>
        <h2>{{firstName + " " + lastName}}</h2>
        <!-- 也可以使用表达式 -->
        <h2>{{counter * 2}}</h2>

    </div>
    <script src="./js/vue.js"></script>
    <script>
        const app = new Vue({
            el : "#app_mustache",
            data : {
            message: "你好",
            name : "VueJS",
            firstName : "lisi",
            lastName : "zhangsan",
            counter : 100

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

二.Vue.js-v-once

1.v-once说明

- v-once:在某些情况下,可能不希望界面随意的跟随改变;
- v-once:该指令后面不需要跟任何表达式(比如之前的v-for后面是由跟表达式)
- v-once: 该指令表示元素和组件只渲染一次,不会随着数据的改变而改变
- v-once: v-once后面无表达式;

2.v-once代码示例

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!--
        v-once
        在某些情况下,可能不希望界面随意的跟随改变
        v-once:
        该指令后面不需要跟任何表达式(比如之前的v-for后面是由跟表达式)
        该指令表示元素和组件只渲染一次,不会随着数据的改变而改变
        v-once后面无表达式;



     -->
    <div id = "app_once">
        <!-- 插入到标签中 -->
        <h2 v-once>{{message}}</h2>
    </div>
    <script src="./js/vue.js"></script>
    <script>
        let vm = new Vue({
            el : "#app_once",
            data : {
            message: "你好",


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

三.Vue.js-其他插值指令使用

1.v-html

服务器请求到数据本身就是一个HTML代码, 直接通过{{}}来输出,会将HTML代码也一起输出;但是我们可能希望的是按照HTML格式进行解析,并且显示对应的内容.

使用v-html指令: 该指令后面往往会跟上一个string类型,会经string的Html解析出来并且进行渲染.

2.v-text

- v-text:一般不使用,不够灵活;会出现覆盖问题
- v-text作用和Mustache比较相似:都是用于将数据显示在界面中

- v-text通常情况下,接受一个string类型

3.v-pre

- v-pre:原封不动的进行显示,不做解析;
- v-pre用于跳过这个元素和其他元素的编译过程,用于显示原本的Mustache语法

4.v-cloak

可以使用在VUE解析之前,div中有一个属性v-cloak; 在VUE解析之前,div中无属性v-cloak;两种方式进行对比.防止在页面卡顿的时候看到代码.

5.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        [v-cloak]{

            display : none;
        }

    </style>
</head>
<body>
    <div id = "app_html" v-cloak>
        <!-- v-html
        服务器请求到数据本身就是一个HTML代码
        如果我们直接通过{{}}来输出,会将HTML代码也一起输出
        但是我们可能希望的是按照HTML格式进行解析,并且显示对应的内容

        解析出HTML展示
        使用v-html指令
            该指令后面往往会跟上一个string类型
            会经string的Html解析出来并且进行渲染

        v-text:一般不使用,不够灵活;会出现覆盖问题
        v-text作用和Mustache比较相似:都是用于将数据显示在界面中
        v-text通常情况下,接受一个string类型

        v-pre:原封不动的进行显示,不做解析
        v-pre用于跳过这个元素和其他元素的编译过程,用于显示原本的Mustache语法

        v-cloak:
        在VUE解析之前,div中有一个属性v-cloak;
        在VUE解析之前,div中无属性v-cloak;



        -->
```

```
        <pre>
            原封不动完全展示出来
        </pre>
        <h2>{{url}}</h2>
        <h2 v-html="url"></h2>
        <h2 v-text="message"></h2>
        <h2 v-pre>{{鼬神}}</h2>
    </div>
    <script src="./js/vue.js"></script>
    <script>
        setTimeout(function(){
        const app = new Vue({
            el : "#app_html",
            data : {
            message: "你好Vue",
            name : "鼬神",
            url : '<a href="http://www.baidu.com">百度一下</a> '


            }
        })
    },1000)
    </script>
</body>
</html>
```

Vue动态绑定

一.Vue.js:v-bind

1.v-bind指令简介

插值操作主要作用是将值插入到模板的内容中,内容需要动态来决定外,某些属性需要动态来绑定;

例:动态绑定a元素的href属性;

动态绑定img元素的src属性;

2.v-bind指令使用说明

- 作用:动态绑定属性
- 缩写: ":"
- 预期:any(with argument) | Object(without argument)(注:参数或者对象)
- 参数:attrOrProp(optional)

  v-bind用于绑定一个或多个属性值,或者向另一个组件传递props值

  需要动态进行绑定的属性:图片的链接SRC,网站的链接href/动态绑定一些类/样式等

3.v-bind语法糖

v-bind有一个对应的语法糖,简写模式 ":"(冒号)

示例:`<img :src="img_url" alt="">`

href:超文本引用

4.代码示例

v-bind的基本使用

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>v-bind的基本使用</title>
</head>
<body>
    <div id = "app_bind">
        <h1>{{message}}</h1>
        <img :src="img_url" alt="">
        <a v-bind:href="a_href">百度一下</a>

    </div>

    <script src="js/vue.js"></script>
    <script>
        // let(变量);const(常量)
        // 编程范式:声明式编程
        const app = new Vue ({
            el : "#app_bind",//用于挂载管理的元素
            data : {//定义数据
                message : "Vue.js",
                img_url :
"https://ss0.bdstatic.com/70cFuHSh_Q1YnxGkpoWK1HF6hhy/it/u=2151058502,193112180&fm=26&gp=0.jpg",
                a_href : "https://www.baidu.com",
            }
        })

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

二.Vue.js:v-bind绑定class对象语法

1.v-bind动态绑定class对象语法

对象语法的含义是:class后面跟的是一个对象

2.v-bind动态绑定class对象语法使用方式

- 方式一:直接通过{}绑定一个类

```
<h2 v-bind:class="{active:isActive}">{{message}}</h2>
```

- 方式二:可以通过判断传入多个值

```
<h2 v-bind:class="{active:isActive,line:isLine}">{{message}}</h2>
```

- 方式三:和普通的类同时存在并不冲突

注:如果isActive和isLine都为true,那么会有title/active/line三个类

```
<h2 class="title",v-bind:class="{active:isActive,line:isLine}">{{message}}</h2>
```

- 方式四:如果过于复杂,可以放在一个methods或者computed中

注:classes是一个计算属性

```
<h2 class="title",v-bind:class="classes">{{message}}</h2>
```

3.代码示例

Vue.js v-bind绑定class对象语法

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js v-bind绑定class对象语法</title>
    <style>
        .active {
            color: red;
        }
    </style>
</head>

<body>
    <div id="app_bind">
         <!-- <h2 class="active">{{message}}</h2> -->
        <!-- <h2 :class="active">{{message}}</h2> -->
        <!-- <h2 v-bind : class="{key1:value1,key2:value2}">{{message}}</h2> -->
        <!-- <h2 v-bind : class="{类型1:true,类型2:boolean}">{{message}}</h2> -->
        <!-- <h2 v-bind:class="{active:isActive,line:isLine}">{{message}}</h2> -->
        <h2 class="title" v-bind:class="get_classes()">{{message}}</h2>
```

```
        <button v-on:click="btn_click">按钮</button>
    </div>
    <script src="js/vue.js"></script>
    <script>
        // let(变量);const(常量)
        // 编程范式:声明式编程
        let app = new Vue({
            el: "#app_bind",//用于挂载管理的元素
            data: {//定义数据
                message: "Vue.js",
                // active: "active",
                isActive: true,
                isLine: true,
            },
            methods: {
                btn_click: function () {
                    this.isActive = !this.isActive
                },
                // 用法4
                get_classes: function () {
                    return { active: this.isActive, line: this.isLine }
                }
            }
        })

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

三.Vue.js:v-bind绑定class数组语法

1.v-bind动态绑定class数组语法使用方式

- 方式一:直接通过{}绑定一个类

```
<h2 :class="[active]">{{message}}</h2>
```

- 方式二: 可以通过判断传入多个值

```
 <h2 :class="[active,line]">{{message}}</h2>
```

- 方式三: 和普通的类同时存在并不冲突

注: 会有title/active/line三个类

```
  <h2 class="title" :class="[active,line]">{{message}}</h2>
```

- 方式四:如果过于复杂,可以放在一个methods或者computed中

  注:classes是一个计算属性

```
<h2 class="title" :class="getClasses()">{{message}}</h2>
```

2.代码示例

Vue.js v-bind绑定class数组语法

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js v-bind绑定class数组语法</title>
    <style>
        .active {
            color: red;
        }
    </style>
</head>

<body>
    <div id="app_bind">
        <h2 class="title" :class="[active,line]">{{message}}</h2>
        <h2 class="title" :class="getClasses()">{{message}}</h2>


    </div>
    <script src="js/vue.js"></script>
    <script>
        // let(变量);const(常量)
        // 编程范式:声明式编程
        let app = new Vue({
            el: "#app_bind",//用于挂载管理的元素
            data: {//定义数据
                message: "Vue.js",
                active: 'active_inner',
                line: "line_inner"
            },
            methods: {
                getClasses: function () {
                    return [this.active, this.line]
                }
            }
        })

    </script>
</body>

</html>
```

四.Vue.js:v-bind动态绑定style对象语法

1.v-bind绑定style简介

利用v-bind:style来绑定一些CSS内联样式

CSS属性名格式:

驼峰式:fontSize;

短横线分隔(kebab-case,需要使用单引号括起来)'font-size'.

2.v-bind动态绑定style对象语法使用方式

使用方式一:对象语法

style后面跟的是一个对象类型,

对象的key是CSS的属性名,

对象的value是具体赋的值,值可以来自于data中的属性.

3.代码示例

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js v-bind动态绑定style对象语法</title>

</head>
<body>
    <div id="app_bindStyle">
        <!-- <h2 :style="{key(CSS属性名):value(CSS属性值)}">{{message}}</h2> -->
        <!-- '50px'必须加上单引号,否则是当做一个变量去解析的 -->
        <!-- <h2 :style="{fontSize:'50px'}">{{message}}</h2> -->
        <!-- finalSize 当成一个变量来解析 -->
        <!-- <h2 :style="{fontSize:finalSize}">{{message}}</h2> -->
        <!-- finalSize1 后面需要加单位 -->
        <h2 :style="{fontSize:finalSize1+'px',color:finalColor}">{{message}}</h2>
        <h2 :style="getStyles()">{{message}}</h2>



    </div>
    <script src="js/vue.js"></script>
    <script>
        // let(变量);const(常量)
        // 编程范式:声明式编程
        let app = new Vue({
```

```
            el: "#app_bindStyle",//用于挂载管理的元素
            data: {//定义数据
                message: "火影忍者",
                finalSize: '100px',
                finalSize1: 100,
                finalColor: 'red',

            },
            methods: {
                getStyles: function () {
                    return { fontSize: this.finalSize1 + 'px', color: this.finalColor }

                }
            }
        })
    </script>
</body>

</html>
```

五.Vue.js:v-bind动态绑定style数组语法

1.v-bind动态绑定style数组语法使用方式

style后面跟的是一个数组类型;

多个值以 ","逗号分割即可.

2.代码示例

v-bind动态绑定style数组语法

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js v-bind动态绑定style数组语法</title>
</head>

<body>
    <div id="app_bindStyle">
        <h2 :style="[baseStyle,baseStyle1]">{{message}}</h2>
    </div>
    <script src="./js/vue.js"></script>
    <script>
        let app = new Vue({
            el : "#app_bindStyle",
            data:{

                message : "鼬神",
```

```
                baseStyle : {backgroundColor : 'red'},
                baseStyle1 : {fontSize : '100px'},
            }
        })
    </script>
</body>
</html>
```

Vue计算属性

一.Vue.js:compute计算属性

1.compute计算属性简介

在模板中可以直接通过插值语法显示一些data中的数据

在某些情况,需要对数据进行一定的转化后再显示,或者我们需要将多个数据结合起来进行显示

示例:有firstName和lastName两个变量,我们需要显示完整的名称

但是如果多个地方都是需要显示完整的名称,我们就需要写多个{{firstName}}{{lastName}}

可以将上面的代码换成计算属性:

计算属性是写在实例的computed选项中的

2.代码示例

compute计算属性代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js compute计算属性</title>
</head>
<body>
    </div>
    <script src="./js/vue.js"></script>
    <script>
        let app =new Vue({
            el: "#app_compute",
            data : {
                firstName : "鼬神",
                lastName : "佐助",
            },
            methods : {
                getFullName : function(){
                    return this.firstName + " " + this.lastName
```

```
                }
            },
            // 计算属性,给函数起的名字
            computed: {
                FullName : function(){
                    return this.firstName + " " + this.lastName
                }
            }
        })
    </script>
</body>
</html>
```

二.Vue.js:compute计算属性复杂操作

1. 代码示例

computed:使用的for循环;

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:compute计算属性复杂操作</title>
</head>

<body>
    <div id="app_computeComplex">
        <h2>总价:{{totalPrice}}</h2>

    </div>
    <script src="./js/vue.js"></script>
    <script>
        const app = new Vue({
            el: "#app_computeComplex",
            data: {
                books: [
                    { id: 100, bookName: "C语言", price: 120 },
                    { id: 101, bookName: "java语言", price: 130 },
                    { id: 102, bookName: "javaScript语言", price: 140 },
                    { id: 103, bookName: "python语言", price: 150 },
                    { id: 104, bookName: "goLong语言", price: 160 },
                ]
            },
            computed: {
                totalPrice: function () {
                    // ES5语法
```

```
                    let result = 0
                    // for (let i = 0; i < this.books.length; i++) {
                    //     result += this.books[i].price
                    // }
                    // ES6的语法一
                    // for (let i in this.books){
                    //     result +=this.books[i].price
                    // }
                    // ES6的语法二
                    for (let book of this.books){
                        result +=book.price
                    }
                    return result
                }
            }
        })
    </script>
</body>
</html>
```

三.Vue.js:计算属性的setter和getter

1. 完整的对象属性:fullName是个对象,对象中有set和get方法
2. 一般没有set方法,仅有只读属性,set属于设置,set方法可以对原有的值进行改变
3. 可以在控制台中检查计算属性的get和set属性
4. 计算属性不能加小括号

1.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js 计算属性的setter和getter</title>
</head>

<body>
    <div id="app_computeComplex">
        <h2>总价:{{totalPrice}}</h2>
        <h2>{{fullName}}</h2>

    </div>
    <script src="./js/vue.js"></script>
    <script>
        const app = new Vue({
            el: "#app_computeComplex",
            data: {
                books: [
                    { id: 100, bookName: "C语言", price: 120 },
```

```
                    { id: 101, bookName: "java语言", price: 130 },
                    { id: 102, bookName: "javaScript语言", price: 140 },
                    { id: 103, bookName: "python语言", price: 150 },
                    { id: 104, bookName: "goLong语言", price: 160 },
                ],
                firstName: "张小豆",
                lastName: "吕懒懒"
            },
            computed: {
                totalPrice: function () {
                    // ES5语法
                    let result = 0
                    // for (let i = 0; i < this.books.length; i++) {
                    //     result += this.books[i].price
                    // }
                    // ES6的语法一
                    // for (let i in this.books){
                    //     result +=this.books[i].price
                    // }
                    // ES6的语法二
                    for (let book of this.books) {
                        result += book.price
                    }
                    return result
                },
                 fullName: {
                    set: function (newValue) {
                        console.log("------",newValue);
                        const names=newValue.split(" ");
                        this.firstName=names[0]
                        this.lastName=names[1]

                    },
                    get: function () {
                        return this.firstName + " " + this.lastName
                    }
                }
            }
        })
    </script>
</body>
</html>
```

四.Vue.js:计算属性的缓存

1.计算属性的缓存说明

Vue计算属性和methods的对比

计算属性会进行缓存,如果多次使用时,计算属性只会调用一次

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js计算属性的缓存</title>
</head>
<body>
    <div id="app_computeComplex">
        <!-- 1.直接拼接,语法过于繁琐 -->
        <h2>{{firstName}}{{lastName}}</h2>
        <!-- 2.通过定义methods
            可以在控制台发现getFullName被调用了5次-->
        <h2>getFullName:{{getFullName()}}</h2>
        <h2>getFullName:{{getFullName()}}</h2>
        <h2>getFullName:{{getFullName()}}</h2>
        <h2>getFullName:{{getFullName()}}</h2>
        <!-- 3.通过定义计算属性
         可以在控制台发现fullName被调用了5次-->
        <h2>fullName:{{fullName}}</h2>
        <h2>fullName:{{fullName}}</h2>
        <h2>fullName:{{fullName}}</h2>
        <h2>fullName:{{fullName}}</h2>


    </div>
    <script src="./js/vue.js"></script>
    <script>
        const app = new Vue({
            el: "#app_computeComplex",
            data: {

                firstName: "张小豆",
                lastName: "吕懒懒"
            },
            methods : {
                getFullName : function(){
                    console.log("getFullName");



                    return this.firstName + this.lastName
                }
            },
            computed : {
                fullName: function(){
                    console.log("fullName");

                    return this.firstName + this.lastName
```

```
                }
            }

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

Vue事件监听

一.Vue.js:v-on的基础使用

1.Vue.js:事件监听基本语法简介

前端开发需要经常和用于交互:必须监听用户发生的时间,比如点击/拖拽/键盘事件;

在Vue中如何监听事件,使用v-on指令.

2.Vue.js:事件监听基本语法使用说明

- 作用:绑定事件监听器
- 缩写:@
- 预期:Function | Inline Statement | Object
- 参数:event

3.代码示例

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js事件监听基本语法使用</title>
</head>
<body>
    <div id="app_on">

        <h2>{{counter}}</h2>
        <!-- 单一操作 -->
        <!-- <button v-on:click="counter++">+</button> -->
        <!-- <button v-on:click="counter--">-</button> -->
        <!-- 复杂操作 -->
        <button v-on:click="increment()">+</button>
        <button v-on:click="decrement()">-</button>
        <h2>{{num}}</h2>
```

```
        <button @click="add()">+</button>
        <button @click="del()">-</button>


    </div>

    <script src="../js/vue.js"></script>
    <script>
        const app = new Vue({
            el: "#app_on",
            data: {
                counter:0,
                num:1,

            },
            methods: {
                increment() {
                     this.counter++
                },
                decrement() {
                     this.counter--

                },
                add() {
                     this.num++
                },
                del() {
                     this.num--

                },

            }

        })

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

二.Vue.js:v-on的参数传递问题

1.v-on的参数

v-on参数

当通过methods中定义方法,以供@click调用时,需要注意参数问题:

情况一:

如果该方法不需要参数的时候,那么方法后的()可以不添加

*但是注意:如果方法本身中有一个参数,那么会默认将原生的事件event参数传递进去

情况二:

如果需要同时传入某个参数,同时需要event时,可以通过$event传入事件

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js v-on的参数传递问题</title>
</head>

<body>
    <div id="app_on_para">
    <!-- 事件调用方法没有参数 -->
    <button @click="btn1Click">按钮1</button>
    <button @click="btn1Click">按钮2</button>
    <!-- 在事件定义时,写方法省略了小括号,但是方法本身是需要一个参数的 -->
    <button  @click="btn2Click(123)">按钮3</button>
    <button @click="btn2Click()">按钮4</button>
    <!-- 不会打印undefined 返回MouseEvent;
    Vue会默认将浏览器产生的event事件对象作为参数传入方法

    -->
    <button @click="btn2Click">按钮5</button>
    <!-- 方法定义时,需要event对象,同时需要其他参数 -->
    <!-- 特殊情况 @click="btn3Click"
    abc被解析为MouseEvent
    -->
    <button  @click="btn3Click">按钮6</button>
    <!--
      123解析为123;
      event被解析为undefined
      vue.js:634 [Vue warn]: Property or method "event" is not defined on the instance but
referenced during render. Make sure that this property is reactive, either in the data option,
or for class-based components, by initializing the property.
      vue.js: 634 [Vue警告]: 属性或方法“事件”未在实例上定义，但在渲染期间被引用。 通过初始化属性，确保
在data选项中或对于基于类的组件，此属性都是反应性的。
      event被认为是一个变量

     -->
    <button  @click="btn3Click(123,event)">按钮7</button>
    <!--
      调用方式,如何手动的获取到浏览器参数的event对象:$event
     -->
    <button @click="btn3Click('abc',$event)">按钮8</button>

  </div>
```

```
  <script src="../js/vue.js"></script>
  <script>
    const app = new Vue({
      el:"#app_on_para",
      data:{
        event:"aaa",
      },
      methods:{
        btn1Click(){
          console.log("btn1Click");
        },
        btn2Click(event){
          console.log("btn2Click",event);
        },
        btn3Click(abc,event){
          console.log('+++',abc,event);
        },
      }
    })
    //如果函数需要参数,但是没有传入,那么函数的形参为undefined
    function abc(name) {
      console.log(name);

    }
    abc() //结果是undefined

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

三.Vue.js:v-on的修饰符

1.v-on的修饰符

在某些情况下,我们拿到event的目的可能是进行一些事件处理

Vue提供了修饰符来帮助我们方便处理一些事件:

- .stop -调用event.stopPropagation();
- .prevent -调用event.PreventDefault();
- .{keyCode | keyAlias} 只当事件是从特定键触发时才触发回调;keyCode:键盘的编码,
- .native - 监听组件根元素的原生事件,自定义组件使用
- .once -只触发一次回调

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:v-on的修饰符</title>
</head>
<body>
    <div id = "app_on_modifier">

    <!--
      停止冒泡
    <button @click.stop="doThis"></button>
    阻止默认行为
    <button @click.prevent="doThis"></button>
    阻止默认行为没有表达式
    <form @submit.prevent></form>
    串联修饰符
    <button @click.stop.prevent="doThis"></button>
    键修饰符,键别名
    <input @keyup.enter="onenter">
    键修饰符,键代码
    <input @keyup.13="onEnter">

    点击回调只会触发一次
    <button @click.once="doThis"></button>
    -->

    <div @click ="divClick">
      AAAAAAA
      <button @click="btnClick">按钮1</button>
      <!--
        1..stop修饰符的使用:
        当点击按钮1时,既打印出btnClick,同时打印出divClick出现冒泡;理想状态下是不能打印出"divClick"的,
点击"AAAAAAA",仅可以出现divClick;
        修改见下一步
      -->
      <button @click.stop="btnClick">按钮2</button>
      <!-- 点击按钮2时,仅出现"btnClick" -->



    </div>
    <!--
        2..prevent修饰符的使用:

      -->

    <br>
      <form action="baidu">
        <!-- 自动提交 -->
        <!-- <input type="submit" value="提交"> -->
        <!-- 自动提交同时有打印(一闪而过) -->
        <!-- <input type="submit" value="提交" @click="submitClick"> -->

        <!-- 仅打印(未提交) -->
```

```
        <input type="submit" value="提交" @click.prevent="submitClick">



      </form>
      <!-- 3.监听某个键盘的键帽 -->

      <input type="text" @keyup.enter="keyUp">


      <!-- 4..once修饰符的使用 -->
      <button @click.once="btn3Click">按钮3</button>
  </div>
    <script src="../js/vue.js"></script>
    <script>
      const app = new Vue({
        el:"#app_on_modifier",
        data:{},
        methods:{
          divClick(){
            console.log("divClick")
          },
          btnClick(){
            console.log("btnClick")

          },
          submitClick(){
            console.log("submitClick")
          },
          keyUp(){
            console.log("keyUp")
          },
          btn3Click(){
            console.log("btn3Click")
          },

        },

      })
    </script>

</body>
</html>
```

Vue条件判断

一.Vue.js:条件判断v-if

1.vue条件判断

v-if/v-else-if/v-else

这三个指令与JS的条件语句if、else、else if类似;

Vue的条件指令可以根据表达式在DOM中渲染或销毁元素和组件.

v-if原理:

v-if后面的条件为false，对应的元素以及其子元素不会渲染;

也就是根本没有不会有对应的标签出现在DOM中.

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:条件判断v-if</title>
</head>
<body>
    <div id="app_v_if">
         <!--
            v-if="true"时,message进行显示;
            v-if="false"时,message进行显示;
          -->
        <h2 v-if="isShow">{{message}}</h2>
        <h1 v-else>当is-show为false时,进行显示</h1>

     </div>
     <script src="../js/vue.js"></script>
     <script>
        const app=new Vue({
            el :"#app_v_if",
            data:{
                message:"zhangsan",
                isShow:true,
            }
        })
     </script>
</body>
</html>
```

二.Vue.js:条件判断v-else-if

1.代码示例

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:条件判断v-else-if</title>
</head>
<body>
    <!-- else-if:用的不多 -->
    <div id="app_else_if">
        <p v-if="score>=90">优秀</p>
        <p v-else-if="score>=80">良好</p>
        <p v-else-if="score>=60">及格</p>
        <p v-else>不及格</p>
        <h1>{{result}}</h1>
    </div>
    <script src="../js/vue.js"></script>
    <script>
        const app=new Vue({
            el:"#app_else_if",
            data:{
                score:92
            },
            computed:{
                result() {
                    let showMessage=" ";
                    if (this.score>=90){
                        showMessage="优秀";
                    }
                    else if (this.score>=80){
                        showMessage="良好";
                    }
                    else if (this.score>=60){
                        showMessage="及格";
                    }
                    else{
                        showMessage="不及格";
                    };
                    return showMessage

                }

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

Vue条件渲染案例

一.Vue.js:条件渲染案例

1.代码示例

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:条件渲染案例</title>
</head>
<body>
    <!--
      condition:[kənˈdɪʃn]条件;
      render:['rendə(r)] 渲染;
      用户在登录时,可以切换使用用户账号登录还是邮箱地址登录
      默认显示是用户账号
     -->
    <div id="app_login">
        <span v-if="isUser">
            <label for="username">用户账号</label>
            <input type="text" id ="username" placeholder="用户账号">
        </span>
        <span v-else>
            <label for="email">用户邮箱</label>
            <input type="text" id ="email" placeholder="用户邮箱">
        </span>
        <button @click="isUser=!isUser">切换类型</button>
    </div>
    <script src="../js/vue.js"></script>
    <script>
        const app = new Vue({
            el: "#app_login",
            data: {
                isUser: true,
            },
            computed: {
            },
            methods: {
            },
        })

    </script>
</body>

</html>
```

二.Vue.js:登录切换的input复用问题

1.示例说明

用户账号与用户邮箱切换后,文本框中的内容依旧存在;

问题:

在输入内容的情况下,切换了类型,发现文字依然显示之前输入的内容;

按道理说,应该切换到另一个input元素中了;

另一个input元素中,并没有输入内容.

注:会抽象出虚拟DOM,放在内存中,在把虚拟DOM渲染到浏览器,虚拟DOM在切换内容时会进行修改

问题解答:

因为Vue在进行DOM渲染时,处于性能考虑,会尽可能的复用已经存在的元素,

而不是重新创建新的元素;

Vue内部会发现原来的input元素不再使用了,直接作为else中的input来使用了

解决方法:

如果希望Vue出现类似重复利用的问题,可以给对应的input添加key;key作为一个标识

并且需要保证key不同.

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:登录切换的input复用问题</title>
</head>
<body>
      <div id="app_login">
        <span v-if="isUser">
            <label for="username">用户账号</label>
            <input type="text" id ="username" placeholder="用户账号" key="username">
        </span>
        <span v-else>
            <label for="email">用户邮箱</label>
            <input type="text" id ="email" placeholder="用户邮箱" key="email">
        </span>
        <button @click="isUser=!isUser">切换类型</button>
    </div>
    <script src="../js/vue.js"></script>
    <script>
        const app = new Vue({
            el: "#app_login",
            data: {
                isUser: true,
```

```
            },
            computed: {
            },
            methods: {
            },
        })

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

Vue循环遍历

一.Vue.js:v-for遍历

1.v-for遍历数组使用方式

- 方式1:`v-for="item in names"`
- 方式2:`v-for="(item,index) in names"`

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js:v-for遍历</title>
    <style>
         li{list-style-type:none;}
    </style>
</head>
<body>
   <div id="app_v_for">
        <ul>
            <li v-for="item in names">{{item}}</li>
            <li>-------分割线-------</li>
            <li v-for="(item,index) in names">{{index+1}}--{{item}}</li>
            <li>-------分割线-------</li>
            <li v-for="(valInner,keyInner) in info">{{keyInner}}--{{valInner}}</li>
        </ul>
    </div>
    <script src="../js/vue.js"></script>
    <script>
        const app = new Vue({
            el:"#app_v_for",

            data:{
```

```
                names:[//数组
                    "张三",
                    "李四",
                    "王五",
                    "麻六",
                ],
                info:{//对象
                    name:"张三",
                    age:100,
                    height:100,
                }

            },
            methods:{
            },
            computed:{
            },
        })
    </script>
</body>
</html>
```

二.Vue.js:v-for遍历组件的key属性

1.组件的key属性

使用v-for时,给对应元素或组件添加上一个:key属性

添加key属性的原因:

其实和Vue的虚拟DOM的diff算法有关系

借用React's diff algorithm中的一张图来简单说明:

当某一层有很多相同的节点时,也就是列表节点时插入一个新的节点,希望可以在B和C

之间加一个f,Diff算法默认执行起来是这样的

即把C更新成F,D更新成C,E更新成D,最后再插入E,是不是很没有效率

所以需要使用key来给每个节点做一个唯一标识

diff算法就可以正确的识别此节点

找到正确的位置区插入新的节点

注:key的作用主要是为了高效的更新虚拟DOM

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:v-for遍历组件的key属性</title>
</head>
<body>
    <!-- 数组删除插入元素:app.names(起始位置,删除几个元素,添加元素内容) -->
   <div id="app_v_for">
    <ul>
        <li v-for="item in names">{{item}}</li>
        <li>-------分割线-------</li>
        <li v-for="(item,index) in names">{{index+1}}--{{item}}</li>
        <li>-------分割线-------</li>
        <li v-for="(valInner,keyInner) in info">{{keyInner}}--{{valInner}}</li>

    </ul>


</div>
<script src="../js/vue.js"></script>
<script>
    const app = new Vue({
        el:"#app_v_for",
        data:{
            names:[//数组
                "张三",
                "李四",
                "王五",
                "麻六",
            ],
            info:{//对象
                name:"张三",
                age:100,
                height:100,
            }

        },

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

Vue阶段案例-1

Shopping Cart Case

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Shopping Cart Case</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <!-- 界面搭建 -->

    <div id = "app_cart">
      <div v-if="books.length">
      <table>
        <thead>
          <tr>
            <th></th>
            <th>书籍名称</th>
            <th>出版日期</th>
            <th>价格</th>
            <th>购买数量</th>
            <th>操作</th>
          </tr>

        </thead>
        <tbody>
          <tr v-for="(item,index) in books">
            <td >{{item.id}}</td>
            <td >{{item.bookName}}</td>
            <td >{{item.date}}</td>
            <td >{{getFinalPrice(item.price)}}</td>
            <!-- <td >{{item.price | showPrice}}</td> -->
            <td >
              <button @click="decrement(index)" v-bind:disabled="item.count <=1">-</button>
              {{item.count}}
              <button @click="increment(index)">+</button>
            </td>
            <td>
              <button @click="removeHandler(index)">
                移除
              </button>
            </td>
          </tr>
        </tbody>
      </table>
      <h2>总价格:{{totalPrice | showPrice}}</h2>

    </div>
    <h2 v-else>购物车为空</h2>
  </div>

  <script src="../js/vue.js"></script>
  <script src="main.js"></script>
```

```
  <!-- <script>
    // 给数字设置保留2位小数
    123.toFixed()
  </script>
   -->
</body>
</html>
```

style.css

```
table{
  border:1px solid #e9e9e9;
  border-collapse: collapse;
  border-spacing: 0;
}


th,td{
  padding: 8px 16px;
  border:1px solid #e9e9e9;
  text-align: left;



}

th{
  background-color: #f7f7f7;
  color:#5c6b77;
  font-weight: 600;
}
```

main.js

```
const app = new Vue({
  el: "#app_cart",
  data: {
    books: [
      {
        id: 1,
        bookName: "JAVA",
        date: '2020-5',
        price: 150.00,
        count: 1
```

```
      },
      {
        id: 2,
        bookName: "Python",
        date: '2020-5',
        price: 200.00,
        count: 1
      },
      {
        id: 3,
        bookName: "AI",
        date: '2020-5',
        price: 80.00,
        count: 1
      },
      {
        id: 4,
        bookName: "人工智障",
        date: '2020-5',
        price: 50.00,
        count: 1
      },
      {
        id: 5,
        bookName: "C语言",
        date: '2020-5',
        price: 90.00,
        count: 1
      },
      {
        id: 6,
        bookName: "C++",
        date: '2020-5',
        price: 90.00,
        count: 1
      },

    ]

  },
  methods: {
    getFinalPrice(price) {
      return "¥" + price.toFixed(2)
    },
    increment(index) {
      console.log("increment,index")
      this.books[index].count++

    },
    decrement(index) {
      console.log("decrement")
      this.books[index].count--
```

```
    },
    removeHandler(index) {
      this.books.splice(index, 1)
    }

  },
  // Vue的过滤器
  filters: {
    showPrice(price) {
      return '¥' + price.toFixed(2)
    }
  },
  computed: {
    totalPrice() {
      // 1.普通的for循环
      // let totalPrice=0
      // for (let i=0; i<this.books.length;i++){
      //   totalPrice+=this.books[i].price*this.books[i].count
      // }
      // return totalPrice

      // 2.for( let i in this.books),i是一个索引值
      // let totalPrice=0
      // for (let i in this.books) {
      //   totalPrice+=this.books[i].price*this.books[i].count
      // }
      // return totalPrice

      // 3.for( let i of this.books),i指的是book
      // let totalPrice = 0
      // for (let item of this.books) {
      //   totalPrice += item.price * item.count
      // }
      // return totalPrice

      // 4.reduce
      return this.books.reduce(function (preValue, book) {
        return preValue += book.price * book.count

      }, 0)

    }

  }
})

// 编程范式:面向对象编程(第一公民:对象)/函数式编程(第一公民:函数)
// 高阶函数:filter/map/reduce:函数式编程
// filter函数的使用
// filter的回调函数有一个要求必须返回一个Boolean值
// true:当返回true时.函数内部会自动将这次回调的n加入到新的数组中
// false:当返回false时,函数内部会过滤掉这次的n

// function可以写成箭头函数
```

```
const nums = [10, 20, 111, 232, 444, 50]

let totalNum = nums.filter(n => n < 100).map(n => n * 2).reduce((pre, n) => pre + n);
console.log(totalNum)
let newNums = nums.filter(function (n) {
  return n < 100
})
console.log(newNums)

// map函数的使用
let new1Nums = newNums.map(function (n) {
  return n * 2
})
console.log(new1Nums)

// reduce函数的使用
// reduce作用是对数组中所有的内容进行汇总
let total = new1Nums.reduce(function (preValue, n) {
  return preValue + n
}, 0)
console.log(total)

// 1.获取所有小于100的数字
// let new1Num=[]
// for (let n in nums){
//   if(n<100){
//     new1Num.push(n)
//   }
// };

// 2.获取所有小于100的数字*2
// let new2Num=[]
// for (let n in nums){

//     new2Num.push(n*2)

// };

// 3.获取所有小于100的数字相加
// let total=0
// for (let n in new2Num){

//   total+=n;

// };
```

Vue表单输入绑定

一.Vue.js:表单绑定v-model:基本使用

1.v-model原理

v-model其实是一个语法糖,它的背后本质上包含两个操作:

1.v-bind绑定一个value属性

2.v-on指令给当前元素绑定input事件

v-model相当于2个指令的结合v-bind:value="message" v-on:input="valueChange"

1.v-bind的语法糖:":"

2.v-on的语法糖:"@"

2.案例示例

表单绑定v-model(双向绑定),注册页面,登录页面

表单控件在实际开发中是非常常见的,特别是对于用户信息的提交,

需要大量的表单

Vue中使用v-model指令来实现表单元素与数据的双向绑定.

案例解析:

1.在输入框输入内容;

2.因为input中的v-model绑定了message,所以会实时讲输入的内容传递给message,message发生改变

3.当message发生改变时,因为使用Mustache语法,将message的值插入到DOM中,所以DOM会发生响应的改变

4.需要使用v-model用于textarea元素

3.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:表单绑定v-model:基本使用</title>
</head>
<body>
 <div id="app_model">
    <!-- 方式一 :推荐-->
    <input type="text" v-model="message">
    <!-- 方式二 -->
    <input type="text" v-bind:value="message" v-on:input="valueChange">
    <!-- 方式三 -->
    <input type="text" v-bind:value="message" v-on:input="message=$event.target.value">
    <h2>{{message}}</h2>

   </div>
```

```
    <script src="../js/vue.js"></script>
    <script>
      const app=new Vue({
       el:"#app_model",
       data:{
         message:"Vue"
       },
       methods:{
         // 界面出现一个事件,浏览器会默认生成一个event
         valueChange(event){
           this.message=event.target.value;
         }

       },

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

二.Vue.js:v-model结合复选框

1.v-model结合复选框

复选框分为两种情况:单个复选框和多个复选框

单个勾选框:

v-model为布尔值;

此时input的value并不影响v-model的值.

多个勾选框:

当是多个复选框时,因为可以选中多个,所以对应的data中属性是一个数组,

当选中某一个时,就会将input的value添加到数组中.

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:v-model结合复选框</title>
</head>
<body>
    <!--1. checkbox单选框 -->

  <div id="app_model_checkbox">
```

```
    <label for="licence">
      <!--
        同意协议
        协议:protocol['prəʊtəkɒl]
        许可证，执照:licence['laɪsns]
        有Label,可以点文字
      -->
      <input type="checkbox" id="agree" v-model="isAgree">同意协议
    </label>
    <h2>{{isAgree}}</h2>
    <button :disabled="!isAgree">下一步</button>


    <!--2. checkbox多选框 -->
    <br>

      <input type="checkbox" value="篮球" v-model="hobbies">篮球
      <input type="checkbox" value="排球" v-model="hobbies">排球
      <input type="checkbox" value="羽毛球" v-model="hobbies">羽毛球
      <input type="checkbox" value="乒乓球" v-model="hobbies">乒乓球
      <input type="checkbox" value="足球" v-model="hobbies">足球
    <h2>您的爱好是:{{hobbies}}</h2>

  </div>
  <script src="../js/vue.js"></script>
  <script>
    const app=new Vue({
      el:"#app_model_checkbox",
      data:{
        isAgree:false,
        hobbies:[],

      },

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

三.Vue.js:v-model的radio单选框的使用

1.v-model的radio单选框的使用注意事项

如果v-model绑定的是同一个,则可以去掉name="sex",v-model绑定的相同的也是互斥的

必须添加name属性才可以互斥

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:v-model的radio单选框的使用</title>
</head>
<body>
  <div id = "app_model_radio">
      <label for="male">

      <input type="radio" value="男"  id ="male" name="sex" v-model="sex">男
    </label>
    <label for="female">
      <input type="radio" value="女"  id ="female" name="sex" v-model="sex">女
    </label>
    <p>选择:{{sex}}</p>

  </div>
  <script src="../js/vue.js"></script>
  <script>
    const app=new Vue({
      el:"#app_model_radio",
      data:{

        sex :"",
        // 最后可以把sex发送至服务器保存
      },

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

四.Vue.js:v-model的select使用

1.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:v-model的select使用</title>
</head>
<body>
  <!--
    password和text类型相似
   -->
  <div id ="app_model_select">
    <!-- 1.选择一个 -->

    <select name="abc" id="" v-model="fruit">
```

```
      <option value="苹果" >苹果</option>
      <option value="香蕉" >香蕉</option>
      <option value="鸭梨" >鸭梨</option>
      <option value="菠萝" >菠萝</option>
    </select>
    <h2>您选择的水果是:{{fruit}}</h2>


    <!-- 2.选择多个 -->
    <select name="abc" id="" v-model="fruits" multiple>
      <option value="苹果" >苹果</option>
      <option value="香蕉" >香蕉</option>
      <option value="鸭梨" >鸭梨</option>
      <option value="菠萝" >菠萝</option>
    </select>
    <h2>您选择的水果是:{{fruits}}</h2>

  </div>
  <script src="../js/vue.js"></script>
  <script>
    const app =new Vue({
      el:"#app_model_select",
      data:{
        fruit:"香蕉",
        fruits:[]
      },
      methods:{

      },

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

Vue组件化

一.Vue.js:组件化得使用步骤

1.组件的使用分成3个步骤

1.创建组件构造器:调用Vue.extend()(注:extend[ɪkˈstend] 延伸；扩大；推广；伸出；)

2.注册组件(全局注册,局部注册):调用Vue.component()(component[kəmˈpəʊnənt]组成部分；成分；组件，元件)

3.使用组件:在Vue实例的作用范围内使用组件

组价化得优点:代码的服用,可移植性高

constructor[kənˈstrʌktə(r)] 构造函数；构造器；建造者

2.注册组件的步骤解析

1.Vue.extend():

调用Vue.extend()创建的是一个组件构造器

通常创建组件构造器时,传入的template代表自定义组件的模板

该模板就是在使用组件的地方,要显示的HTML代码

实际应用中,这种写法在Vue2.x的文档中几乎看不到了,它会直接使用下面说到的语法糖.

2.Vue.component():

调用Vue.component()是将刚才的组件构造器注册为一个组件并且给它起一个组件的标签名称

需要传入的参数:1.注册组件的标签名;2.组件构造器

3.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:组件化得使用步骤</title>
</head>
<body>
    <div id ="app_register">

    <my-cpn></my-cpn>
   </div>

   <script src="../js/vue.js"></script>
   <script>
     //1.创建组件构造器
    //  ES6:使用 "`"定义字符串
    const cpnConstructor=Vue.extend(
      {
        template:`
        <div>
          <h2>标题</h2>
          <p>内容1</p>
          <p>内容2</p>
          <p>内容3</p>
        </div>`
      }
    )
    //  2.注册组件
    // Vue.component('组件的标签名',组件构造器)
    Vue.component('my-cpn',cpnConstructor)
    const app=new Vue({
      el:"#app_register",
    })

   </script>
```

```
</body>
</html>
```

二.Vue.js:全局组件和局部组件

1.全局组件和局部组件说明

全局组件:可以在多个实例下进行使用

通过调用Vue.component()注册组件时,组件的注册是全局的;

意味着该组件可以在任意Vue示例下进行使用.

局部组件:在Vue实例中添加components属性;

如果注册的组件是挂载在某个实例中,则组件是一个局部组件;

开发中:一般只有一个Vue实例,常使用局部组件.

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:全局组件和局部组件</title>
</head>
<body>
    <div id = "app_whole_part">
    <cpn></cpn>
  </div>
</body>
<script src="../js/vue.js"></script>
<script>
  // 1.创建组件的构造器
  const cpnC=Vue.extend({
    template:`
      <div>
        <h2>标题</h2>
        <p>内容1</p>
        <p>内容2</p>
      </div>
    `
  })
  //2.局部组件
  const app =new Vue({
    el:"#app_whole_part",
    methods:{
```

```
    },
    data:{

    },
    components:{
      cpn:cpnC,
    }
  })

</script>
</html>
```

三.Vue.js:父组件和子组件的区分

1.Vue.js:父组件和子组件注意事项

父组件和子组件

错误用法:当子组件注册到父组件的components时,Vue会编译好父组件的模块

模板的内容决定了父组件将要渲染的HTML(相当于父组件中已经有了子组件中的内容)

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:父组件和子组件的区分</title>
</head>
<body>
    <div id = "app_parentSub">
    <!-- <cpn1></cpn1> -->
    <cpn2></cpn2>
  </div>
  <script src="../js/vue.js"></script>
  <script>
    // 1.创建第一个组件构造器(子组件)
    const cpnC1=Vue.extend({
      template:`
      <div>
        <h2>Title1</h2>
        <p>Inner1</p>
      </div>


      `,

    })
    // 2.创建第二个组件构造器(父组件)
```

```
    const cpnC2=Vue.extend({
      template:`
      <div>
        <h2>Title2</h2>
        <p>Inner2</p>
        <cpn1></cpn1>
      </div>
      `,
      components:{
        cpn1:cpnC1,
      },
    })
// root组件(根组件)
  const app=new Vue({
    el:"#app_parentSub",
    data:{

    },
    methods:{

    },
    computed:{

    },
    components:{
      // cpn1:cpnC1,
      cpn2:cpnC2,
    }
  })
  </script>
</body>
</html>
```

四.Vue.js:组件的语法糖注册方式

1.注册组件语法糖

Vue为了简化流程，提供了注册的语法糖

主要是省去了调用Vue.extend()的步骤,而是可以直接使用一个对象来代替

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:组件的语法糖注册方式</title>
```

```
</head>
<body>
    <div id = "app_grammar">
    <!-- cpn1: 全局组件 -->
    <cpn1></cpn1>
    <!-- cpn2: 局部组件 -->
    <cpn2></cpn2>

  </div>
  <script src="../js/vue.js"></script>
  <script>
    //1.全局组件注册的语法糖
    // 1.创建组件的构造器
    // const cpn1=Vue.extend({
    //   template:
    //   `
    //   <div>
    //     <h2>Title1</h2>
    //     <p>Inner1</p>
    //   </div>
    //   `
    // })

    // 2.注册全局组件的语法糖
    Vue.component("cpn1",
    {
      template:
      `
      <div>
        <h2>Title1</h2>
        <p>Inner1</p>
      </div>
      `
    }
    )


    // 2.注册局部组件的语法糖
    const app=new Vue({
      el:"#app_grammar",
      data:{

      },
      methods:{

      },
      computed:{

      },
      components:{
        'cpn2':{
          template:`

          <div>
```

```
            <h2>Title2</h2>
            <p>Inner2</p>
          </div>

          `
        }

      },
    })

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

五.Vue.js:组件模板抽离的写法

1.模板的分离写法

HTML分离出来写,挂载到对应的组件上,结构会变得清晰

Vue提供了两种方案来定义HTML模块内容

使用script标签(需要添加ID)

使用template标签(需要添加ID)

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:组件模板抽离的写法</title>
</head>
<body>
    <div id="app_template_pullAway">
    <my-cpn1></my-cpn1>
    <my-cpn2></my-cpn2>
  </div>
  <!-- 注册一个全局组件 -->
  <script type="text/x-template" id= "myCpn">
    <div>
      <h2>组件标题</h2>
      <p>组件的内容</p>
    </div>

  </script>

  <template id ="myTem">
```

```
    <div>
      <h2>template标题</h2>
      <p>template内容</p>
    </div>
  </template>

  <script src="../js/vue.js"></script>
  <script>
    // 1.注册一个全局组件
    // Vue.component("cpn1",
    // {
    //   template:
    //   `
    //   <div>
    //     <h2>Title1</h2>
    //     <p>Inner1</p>
    //   </div>
    //   `
    // }
    // )


    const app=new Vue({
      el:"#app_template_pullAway",

      components:{
        "my-cpn1":{
          template:"#myCpn",
        },
        "my-cpn2":{
          template:"#myTem",
        }

      },
    })

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

六.Vue.js:组件的data为函数

1.组件的data必须是函数

组件内部是不能访问Vue实例数据的

组件是一个单独功能模块的封装

这个模块有属于自己的HTML模板,也应该有属于自己的数据data

我们发现不能访问,而且即使访问,如果将所有数据都放在Vue实例中,Vue实例就会变得非常臃肿;

结论:Vue组件应该有自己保存数据的地方

组件数据的存放

组件对象有一个data属性(也可以有methods等属性,)

只是这个data属性必须是一个函数

而且这个函数返回一个对象,对象内部保存着数据.

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:组件的data为函数</title>
</head>
<body>
    <div id = "app_component_func">
    <cpn></cpn>
  </div>

  <template id ="myTem">
    <div>
      <h2>{{title}}</h2>
      <!--
        解析:template标题可以显示
        原因:
        这是因为目前组件访问的是自己当中的data
       -->
      <p>template内容</p>
    </div>
  </template>

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


    const app=new Vue({
      el:"#app_component_func",
      data:{

      },
      methods:{

      },
      computed:{

      },
      components:{
        'cpn':{
```

```
            template:"#myTem",
            data(){
              return{
                title:'template标题',
              }
            }

        },
      },
    })

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

七.Vue.js:组件通信父组件向子组件传递数据

1.父子组件的通信

子组件式不能引用父组件或者Vue实例的数据的;

开发中数据需要从上层传递到下层:

举例:

一个页面中,从服务器请求到了很多的数据

其中一部分数据,并非是我们整个页面的大组件来展示的,而是需要下面的子组件进行展示

并不会让子组件再次发送一个网络请求,而是直接让大组件(父组件)将数据传递给小组件

父子组件间的通信:

Vue官方提到:

通过props向子组件传递数据

子组件通过事件向父组件发送消息

props基本用法

组件中,使用选项props来声明需要从父级收到的数据

props的值有两种方式:

方式一:字符串数组,数组中的字符串就是传递时的名称

方式二:对象,对象可以设置传递时的类型.也可以设置默认值

props数据验证

props选项是使用一个数组

除了数组之外,也可以使用对象,当需要对props进行类型等验证时,就需要对象写法了

验证都支持哪些数据类型

- String
- Number

- Boolean
- Array
- Object
- Date
- Function
- Symbol

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:组件通信父组件向子组件传递数据</title>
</head>
<body>
    <div id = "app_props">
<!--
如果不使用v-bind会把"numberList","message"当做一个字符串来显示
 -->
    <!-- <cpn v-bind:numbers="numberList" :inner="message" ></cpn> -->
    <cpn  :numbers="numberList" :inner="message" ></cpn>



  </div>
  <template id="cpn1">
    <div>
      <h1>标题{{inner}}</h1>

      <ul>
        <li v-for="item in numbers">{{item}}</li>
      </ul>
    </div>
  </template>
  <script src="../js/vue.js"></script>
  <script>
    // 父传子通过props
    const cpn={
      template:'#cpn1',
      // 一.通过数组
      // props:["numbers","inner"],
      props:{
        // 1.类型限制
        // numbers:Array,//给变量指定了类型,
        // inner:string,
        // 2.提供一些默认值,以及必传值
        inner:{

          type:String,
```

```
      default:"aaaaa",
      required:true,//一旦有required是必须传这个属性的
    },
    // 类型是一个对象或者数组时,默认值必须是一个函数

    numbers:{
      type:Array,
      // default:[];会报错
      default(){
        return []
      }
    }
  },
  data(){
    return {}
  },
  methods:{

  }
}



const app=new Vue({
  el:"#app_props",
  data:{
    message:"数字",
    numberList:[1,2,3,4,5]

  },
  methods:{

  },
  computed:{

  },
  components:{
    cpn
    //字面量增强写法中属性的增强写法.相当于
    // cpn:{

    // }
    // 'cpn' :cpn

  },
})

//属性的增强式写法
// const name='abc'
// const odj={
//   name:name,
//   name //属性的增强式写法

// }
```

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

八.Vue.js:驼峰标识问题

1.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:驼峰标识问题</title>
</head>
<body>
  <div id="app_HumpLogo">

  </div>

  <template id="my-cpn">
    <div>
      <h1>标题:</h1>
      <p>内容:</p>
    </div>
  </template>
  <script src="../js/vue.js"></script>
  <script>

    const cpn={
      template:'#my-cpn'
    }

  const app=new Vue({
    el:"#app_HumpLogo",
    data:{
      message:"HumpLogo"

    },
    methods:{

    },
    computed:{
      cpn

    },

    components:{
```

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

九.Vue.js:组件通信子组件向父组件传递数据

1.子父组件的通信

子组件向父组件通信:自定义事件

props用于父组件向子组件传递数据,还有一种比较常见的是子组件传递数据或者事件到父组件中

需要使用自定义事件来完成

当子组件需要向父组件传递数据时,就要用到自定义事件了

之前v-on不仅仅可以用于监听DOM事件,也可以用于组件件的自定义事件

自定义事件流程

在子组件中,通过$emit()来触发事件

在父组件中,通过v-on来监听子组件事件

例如:

之前做过一个两个按钮+和-,点击后修改counter

整个操作过程还是在子组件中完成,但是之后的展示交给父组件

需要将子组件中的counter,传给父组件的某个属性,比如total

2.代码示例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Vue.js:组件通信子组件向父组件传递数据</title>
</head>
<body>
   <!-- 父组件模板 -->
  <div id = "app_props">
    <!-- 默认传参item -->
    <cpn v-on:item-click="cpnClick"></cpn>


  </div>
```

```
<!-- 子组件模板
希望子组件的产生的事件 -->
<template id="cpn1">
  <div>
    <button v-for="item in categories"
     @click="btnClick(item)">
      {{item.name}}
    </button>
  </div>
</template>
<script src="../js/vue.js"></script>
<script>
  // 1.子组件
  const cpn={
    template:'#cpn1',
    data(){
      return {categories:[
        { id :"12",name:"热门推荐"},
        { id :"13",name:"手机数码"},
        { id :"14",name:"家用家电"},
        { id :"15",name:"电脑办公"},
      ]
      }
    },
    methods:{
      btnClick(item){
        // console.log(item.id);
        this.$emit('item-click',item)
        // emit发射
      }
    },



  }


  // 2.父组件
  const app=new Vue({
    el:"#app_props",
    data:{


    },
    methods:{
      cpnClick(item){
        console.log('cpnClick',item)
      }
    },
    computed:{


    },
```

```
      components:{
        cpn
        //字面量增强写法中属性的增强写法.相当于
        // cpn:{
        // }
        // 'cpn' :cpn

      },
    })

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