# Vue响应式原理

vue响应式实现的方法是object.defineProperty()用于监听劫持vue中的data数据,在该方法中有getter,setter方法,可以对属性值的获取和设置造成影响,每一个属性值都有一个Dep对象,记录所有的订阅者,当属性发生改变时调用Dep.notify()来通知wather调用update发生数据的改变

Vue中编写了一个wather来监听处理数据,在使用getter方法时,会通知wather实例对view进行页面渲染;在使用setter方法时,同时会通知wather实例对view层进行更新

# 开发历程

原生js -->jquery库 -->模板引擎 -->框架（vue.js/Angular.js...）

# 框架和库的区别

框架：是一个完整的解决方案，框架可以完成一整个大型应用开发

vuecli脚手架搭建框架

库：提供一些小功能，是个插件，不能独立完成一个项目，提供便利，完成某些功能

<!-- 当成库使用，直接引用架包 --> <script src=""></script>

当框架使用

vuecli脚手架搭建框架

当库使用

开发版本

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

生产版本 以min.js结尾 压缩版本

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

# vue和jquery的区别

1.写法不一样

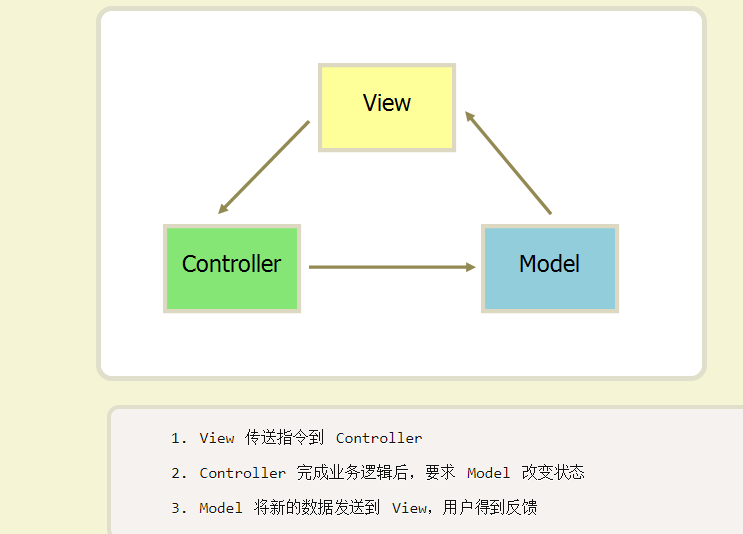
jquery是操作dom，jquery效率偏低

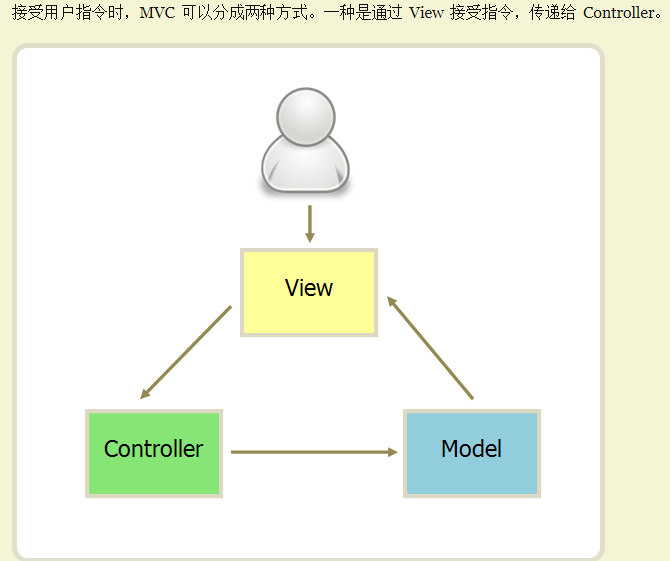
2.思想不一样

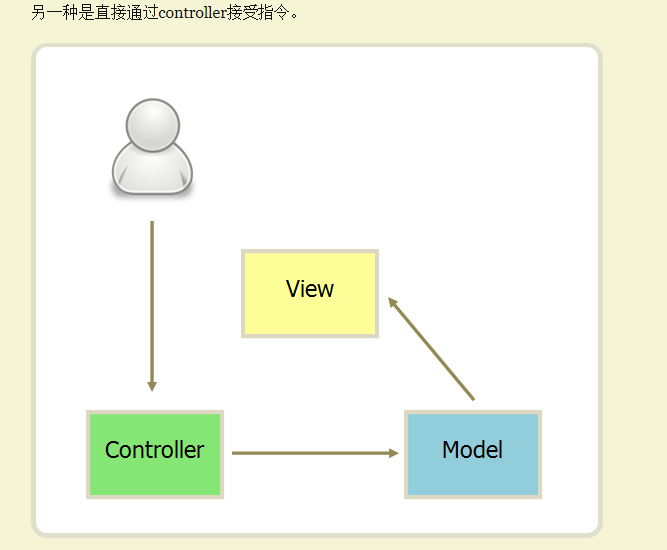
vue思想形成虚拟的dom树（javaScript对象） 通过diff算法，如果发生改变，构建真实的dom，没有改变则不管

# MVC和MVVM和MVP的区别

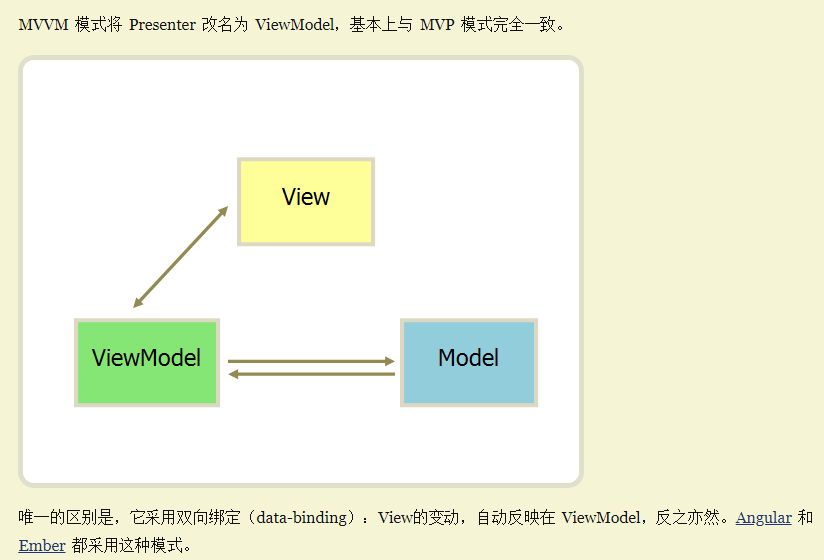
MVC:是后端的分层开发概念 M是modle模板（数据库数据），V是视图view,C是controller控制器 业务逻辑层；V发送指令给C，C完成业务后改变M数据,M将数据反给V 是单向的





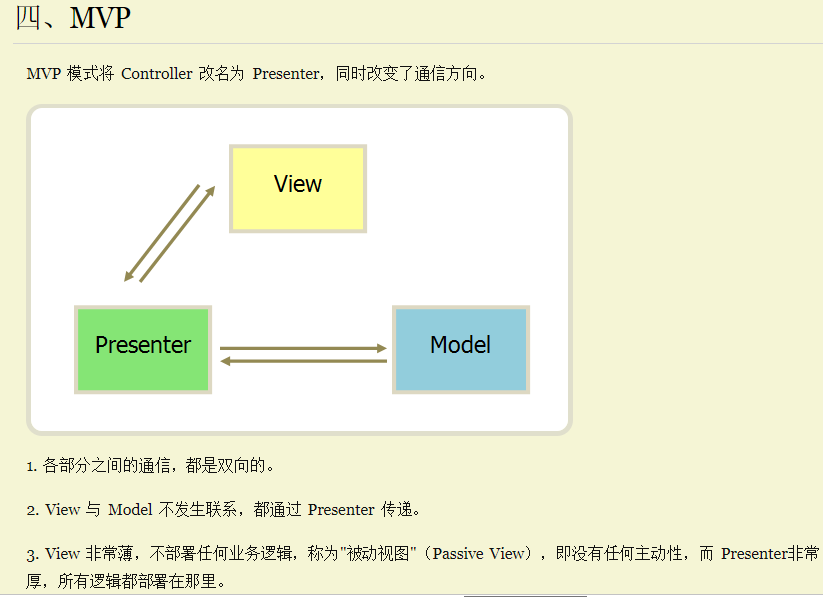


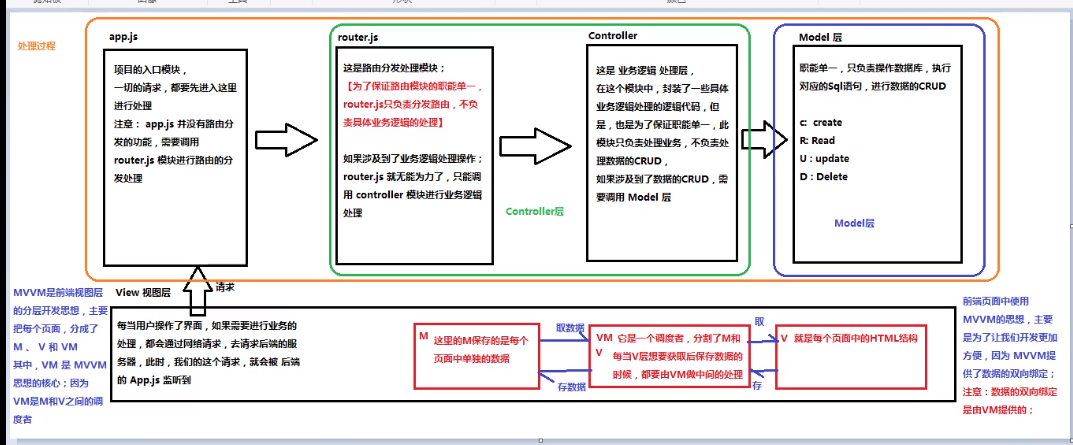
MVVM：是前端视图层概念，主要关注于视图层分离 M是modle模板（data数据），V是视图view,VM是viewModel视图模型，是MVVM的核心，VM是M和V的调度者;MVVM支持双向数据绑定，当修改M层数据时，VM会监测到发生改变，并通知V层进行相应的修改，反之修改了V层数据，M层也会发生改变，也就实现了视图和模型的相互解耦（独立）



MVP

****View：****主要负责界面的显示及跟数据无关的逻辑，比如设置控件的点击事件等  
****Presenter:****主要负责View与Model的交互  
****Model:****数据部分





# Vue基本使用

## 插值表达式{{ }}

1.可以放入字符串

2.number类型

3.boolean类型

4.数组array类型

5.object对象 前后的空格不可以省略

6.简单运算 1+1 'a'+'b'=ab 拼接

7.表达式

8.undefined和null在插值表达式上不会显示

注意：

1.对象中：数据必须先存在才可以发生改变，如果不存在，不会发生改变（数据发生了变化，但是没有渲染视图，界面没有变化）

可以把对象重新赋值，可以渲染：

vm.ojb={s:2,d:4,abc:1}

vm.ojb={...vm.ojb,abc:1}es6引用语法

vm.$set(vm.ojb,'abc',4)

2.数组中：不能通过数组索引的方式改变渲染视图（数据改变了，但是界面中没有改变），通过length的方法修改数组长度也是不能渲染视图的，

数组是不一样的，原生的方法其实不可以办到，在vue中重写了这些方法叫数组变异，

变异方法：

push(尾部插入)

pop(删除尾部插入)

unshif(头部插入)

shif(删除头部插入)

sort(排序)

reverse(翻转数组)

splice(添加新元素在数组中) splice(从何处添加/删除元素，应该删除多少元素，要添加到数组的新元素)

vue中重写了这7个方法

## vm其他的属性

vm.$el 当前组件的根实例

vm.$nextTick 渲染完成以后执行

vm.$mount 等同于实例中的el属性 必须搭配脚手架使用

## vue指令

* v-pre 阻止当前vue对当前元素内的所有内容进行渲染，将内容原封不动的输出

首先是页面自己渲染，vue接管后会重新对vue模板部分进行渲 染

* v-cloak 可以配合css将没有渲染的vue模块进行隐藏 (在网速慢的时候，没有渲染之前不会再出现{{}}了，进行了隐藏)

[v-cloak] {display:none}

* v-once 只能让vue进行一次渲染
* v-html 渲染html
* v-text 渲染成text文本，连标签一起输出
* v-if if后面可以写条件
* v-else-if
* v-else
* v-show

v-if和v-show的区别

1.v-if是控制元素是否存在，如果为false完全移除该元素，

而v-show是控制元素的display属性（隐藏起来）

1. v-if可以支持模板的，v-show是不可以的 （template模块，隐 藏不了）

如果需要频繁的切换就使用v-show(较高的初始渲染消耗) ，如果创建出来永远不会被用户看到就使用v-if （较高的切换性能消耗）

* v-bind 绑定属性 简写为 ：

<img v-bind:src=”VuedataSrc”/>

* v-on 事件绑定 简写为 @

<input type="button" value="点击" @click="clicks">

* v-for 遍历循环

必须加上key属性，这样当成库使用的时候不会出问题，

key值要保持唯一

可以循环数组，对象，数字，字符串

 v-for循环时key属性只能使用number/string

        key在使用时必须使用v-bind属性绑定的形式指定key的值

        在组件中，使用v-for循环时，或者进行一些特殊情况时，如果v-for有问题必须在使用

        v-for的同时指定唯一的字符串/数字类型key值

<!-- 对象形式 遍历对象除了val,key还有index索引 -->

            <p v-for="(val,key,index) in puser">

                值：{{val}}  键：{{key}} 索引:{{index}  }

            </p>

            <!-- in后面还可以使用数字，从1开始 -->

            <p v-for="cont in 10">{{cont}}</p>

* v-model 双向数据绑定（只用于input中） 只要修改input的value值data数据也会发生改
* v-model.lazy 封装了change （输入文字后鼠标点击输入框外才发生改变）
* v-model.number 将字符串转换为数字类型 (输入值显示数字类型)
* v-model.trim 去除首尾的空格

修改数据: Vue.set(要修改对象,String|number,修改后的值)

当修改数值就是number,当修改对象就是string

//vue内部提供的修改方法set数组值

Vue.set(this.arr , 0,'bbb')

### v-on事件

#### 修饰符

##### .stop阻止冒泡

 <!-- .stop阻止冒泡（冒泡：由内而外进行）

            //如果没有阻止冒泡会触发div被点击

         -->

        <input type="button" value="点击" @click.stop="clicks">

##### .prevent阻止默认行为

 <!-- .prevent阻止默认行为 （a链接的默认行为是跳转到指定页面）  -->

         <a href="https://www.baidu.com" @click.prevent.stop="alick">百度</a>

##### .cupture 实现捕获触发事件机制 由外到内触发

<!-- .capture 实现捕获触发事件的机制 由外到内捕获触发 点击按钮先触发外层事件 -->

         <div id="div2" @click.capture="div2click">

            <input type="button" value="点击2" @click.stop="clicks2">

         </div>

##### .self 只有自己本身才能触发

<!-- .self只有自己本身才可以触发 冒泡和捕获都不行  -->

         <div id="div3" @click.self="div3click">

            <input type="button" value="点击3" @click.stop="clicks3">

         </div>

##### .once只触发一次

<!-- .once每个事件只会触发一次，当点击第一次时会阻止默认行为和冒泡，但是第二次就不会 使用时于.的顺序无关 -->

         <a href="https://www.baidu.com" @click.prevent.stop.once="a2lick">百度</a>

##### .stop和.self的区别

 <!--

            .stop和.self的区别

            .stop阻止所有的冒泡

            .self:只阻止自己本身，不会阻止别人的冒泡

        -->

#### v-on事件传参数

1.写方法是可以省略小括号，但是方法本身是需要一个参数，这时vue会默认将浏览器产生的event事件对象作为参数传入到方法中

@click=”add”

2.传入1个参数

@click=”add(1)”

1. 传入一个参数和event

在调用方法时手动获取浏览器参数的event对象：$event

@click=”add(123,$event)”

#### 键盘修饰符

###### Keyup键盘修饰符

按键别名

        .enter

        .tab

        .delete 捕获'删除'和'退格'键

        .esc

        .space  '空格'键

        .up     '上'键

        .down   '下'

        left    '左'

        .ringht '右'

<input type="text" @keyup.enter="alerts"> //按下空格触发alerts方法

###### 自定义修饰符

<input type="text" @keyup.A="alerts">

//自定义全局按键修饰符 keyCodes

        //config配置

        Vue.config.keyCodes.A = 65

### V-model数据双向绑定

 <!--数据双向绑定v-model 只能用于表单元素 绑定value的值-->

            <h3>text绑定</h3>

            <input type="text" v-model="names" id="in"/> <br/>

            <!--v-model.lazy 修饰符 输入文字后鼠标点击输入框外才或回车发生改变-->

            <h3>-v-model.lazy 修饰符</h3>

            <input type="text" v-model.lazy="names" /> <br/>

            名字：{{names}}<br />

            <!--v-model.trim 修饰符 输入去除首尾空格-->

            <h3>-v-model.trim 修饰符</h3>

            <input type="text" v-model.trim="names" /> <br/>

            <!--v-model.number 修饰符 将输入的内容当数字处理-->

            <h3>-v-model.number 修饰符 {{typeof age}}</h3>

            <input type="number" v-model.number="age" /> <br/>

            <!-- 原理 其实是指令 v-bind:value和 input事件完成的 （input是文本框数据发生改变时触发）-->

            <input type="text" v-bind:value="names" @input="names = $event.target.value">

            <br/>

            <!-- 单选框 -->

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

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

            <h2>你选择了:{{sexs}}</h2>

            <br/>

            <!-- 多选框 -->

            <input type="checkbox" v-model="xuan">同意协议

            <button :disabled="!xuan">下一步</button>

            <br/>

            <input type="checkbox" v-model="hobbies" value="跳绳">跳绳

            <input type="checkbox" v-model="hobbies" value="跳舞">跳舞

            <input type="checkbox" v-model="hobbies" value="跑步">跑步

            <input type="checkbox" v-model="hobbies" value="篮球">篮球

            <h2>你的爱好：

                <span v-for="item in hobbies">{{item}}&nbsp;</span>

            </h2>

            <!-- 方法2 值绑定 -->

            <label :for="item" v-for="item in checkboxs">

                <input type="checkbox" :value="item" :id="item" v-model="hobbies">{{item}}

            </label>

            <h2>你的爱好：

                <span v-for="item in hobbies">{{item}}&nbsp;</span>

            </h2>

            <!-- select 下拉框 单个-->

            <select v-model="shuo">

                <option value="香蕉">香蕉</option>

                <option value="苹果">苹果</option>

                <option value="葡萄">葡萄</option>

                <option value="丑橘">丑橘</option>

            </select>

            <h2>你选择的是：{{shuo}}</h2>

            <!-- select 下拉框 多个 multiple 按住ctrl选多个 -->

            <select v-model="shuo2" multiple>

                <option value="香蕉">香蕉</option>

                <option value="苹果">苹果</option>

                <option value="葡萄">葡萄</option>

                <option value="丑橘">丑橘</option>

            </select>

            <h2>你选择的是：{{shuo2}}</h2>

    </div>

    <script>

        const vm = new Vue({

            el:"#app",

            data:{

                n1:0,

                n2:0,

                opt:"+",

                res : 0

            },

            methods:{

                calc:function(){//计算等号运算

                    switch (this.opt) {

                        case "+":

                            this.res = parseInt(this.n1)+parseInt(this.n2)

                            break;

                        case "-":

                            this.res = parseInt(this.n1)-parseInt(this.n2)

                            break;

                        case "\*":

                            this.res = parseInt(this.n1)\*parseInt(this.n2)

                            break;

                        case "/":

                            this.res = parseInt(this.n1)/parseInt(this.n2)

                            break;

                    }

                }

            }

        })

    </script>

    <script>

        const vm2 = new Vue({

            el:"#app2",

            data:{

                names:'笑笑', //文本框

                age:"12",

                sexs:"", //单选框

                xuan:true, //一个多选框

                checkboxs:['跳绳','跑步','跳舞','逛街'],

                hobbies:[], //多个多选框

                shuo: '香蕉',

                shuo2:''

            }

        })

    </script>

### 自定义全局指令

全局自定义指令，局部自定义指令

定义指令有两种方法：

#### 1.函数的形式

//函数的方式

        //当代码只需要写到bind和update就使用函数形式

        //定义全局自定义指令 Vue.directive('指令名称',函数)

        //定义时不用加v-，但是调用时要使用v-

        // Vue.directive('slice',(el,bindings,vnode)=>{ //函数等于于把代码写到bind和update中去

        //     console.log(el,bindings,vnode);

        //     var val = bindings.value.slice(0,5);//只可以输入五位数

        //     vnode.context.content = val;

        // })

#### 2.对象的形式

//对象的方式 Vue.directive('指令名称',{})

         //定义时不用加v-，但是调用时要使用v-

         //参数1：指令名称

         //参数2：是一个对象，这个对象身上有一些相关的钩子函数，这些函数可以在特定的阶段，执行相关的操作

            //钩子函数 bind, 只调用一次，指令第一次绑定到元素时调用，这个钩子函数可以定义一个在绑定时执行一次的初始化动作

            //        inserted, 插入到DOM元素的时候触发，触发一次

            //        update,   更新时触发 可能触发多次

            //        componentUpdated,

            //        unbind

            //函数参数：el, 被绑定指令的那个元素 el参数是一个原生js对象

            //         bindings, 一个对象，包含（name指令名,value指令绑定值，expression绑定值的字符串形式，oldValue指令绑定的前一个值 只能在update和componentUpdated函数中使用）

            //         vnode,

            //         oldVonde

#### 实际应用

<input type="text" v-model='content' v-slice='content' v-color="'blue'"/>

Vue.directive('slice',{

            //样式：只要通过指令绑定了元素，不管这个元素有没有被插入到页面中去，这个元素肯定有这个内联样式

            //      将来元素肯定会显示到页面中，这个时候，浏览器的渲染引擎必然会解析样式，应用给这个元素

            bind(el,bindings,vnode){//每当指令绑定元素上去的时候触发 v-slice='content'，只执行一次

                console.log('bind')

                // el.style.color = 'red' //指令绑定的元素文本颜色为红色

            },

            update(el,bindings,vnode){//更新时触发 可能触发多次

                console.log('update');

            },

            //行为: 和行为有个的在inserted这个里面去执行，防止js没有效果

            inserted(el){//插入到DOM元素的时候触发，触发一次

                el.focus();//插入到DOM中，才能获取焦点

                console.log('inserted')

            }

        })

### 自定义局部指令

<div id="app">

        <input type="text" v-model='content'  v-color="'blue'"/>

        {{ content }}

        <p v-fontsize = "'60px'">2020-2-2</p>

        <!-- <p v-fontsize = "60">2020-2-2</p> -->

</div>

 //定义局部自定义指令

            //就近原则，先执行局部再全局

            directives:{

                //对象形式

                'color':{

                    bind:function(el,bindings){

                        el.style.color = bindings.value

                    },

                    inserted:function(el){

                        el.focus()

                    }

                },

                //函数形式

                //当代码只需要写到bind和update就使用函数形式

                'fontsize':function(el,bindings){ //函数等于于把代码写到bind和update中去

                    el.style.fontSize = parseInt(bindings.value)+'px'

                }

            }

## Vue样式

### 内联样式

<!-- 值加‘’就是字符串，没有加就是个变量去下面引用 -->

        <!-- 内联样式 -->

        <div id="app">

            <!-- 对象方式  {als:alser}（键名(类名)：true/false-->

            <p v-bind:class="{als:alser}">小明</p>

            <!--texts颜色覆盖了als-->

            <p v-bind:class="{als:alser,'texts':alser2}">小红</p>

            <!--直接绑定数据里的一个对象，进行引用-->

            <p v-bind:class="classcolor">ccc</p>

            <!--数组方式-->

            <p  v-bind:class="[alsClass, textClass]">小小</p>

            <p  v-bind:class="['als', 'texts']">小小2</p> <!--有引号就是style样式名，没有引号就是data数据名-->

            <!--三元素(iscolor是否为true,为true值为textcolor样式为texts;为false值为alscolor样式为als)-->

            <p v-bind:class="[alscolor,iscolor?textcolor:'']">笑笑</p>

            <!-- 数组中写对象 -->

            <p v-bind:class="['als',{'texts':iscolor}]">笑笑</p>

列表颜色切换

            <ul>

                <li v-for="(mm,index) in movies" v-bind:class="{binds:getindex===index}" @click="bian(index)">

                    {{index}}-{{mm}}

                </li>

            </ul>

        </div>

vue实例

new Vue({

                el:'#app',

                data:{

                    //覆盖

                    alser:true,

                    alser2:true,

                    //直接绑定数据里的一个对象

                    classcolor:{

                        als:true

                    },

                    //数组方式

                    alsClass: 'als',

                    textClass: 'texts',

                    //三元素

                    iscolor:true,

                    alscolor:'als',

                    textcolor:'texts',

                    movies:["xxx","ccc","vvv","bbb","nnn"],

                    getindex:0

                },

                methods:{

                    bian:function(index){ //列表切换

                        this.getindex = index

                    }

                }

            })

### 内嵌样式

<!--内嵌样式-->

        <div id="app2">

            <!-- 对象 无序键值对绑定 键(css属性名)：值 -->

            <p v-bind:style="{color:stycolor,fontSize:stysize+'px'}">内嵌样式</p>

            <!--直接绑定数据里的一个对象-->

            <p v-bind:style="stylecolor">对象</p>

            <p v-bind:style="stylecolor2()">对象2</p>

            <!--数组-->

            <p v-bind:style="[colors,sizes]">数组</p>

        </div>

Vue实例

<script type="text/javascript">

            new Vue({

                el:'#app2',

                data:{

                    stycolor:'green',

                    stysize:40,

                    stylecolor:{

                        color:'blue',

                        fontSize:'30px'

                    },

                    //数组

                    colors:{

                        color:'blueviolet'

                    },

                    sizes:{

                        fontSize:'40px',

                        'font-weight':'bold'

                    }

                },

                methods:{

                    stylecolor2:function(){

                        return {color:'blue',fontSize:'30px'}

                    }

                }

            })

## 自定义过滤器

全局，局部

{{content | 过滤器}}

### 全局

过滤器调用格式 {{msg | msgForat}}

        对msg进行过滤可以调用msgForat在msgForat中处理

        调用不带参数就是 msgForat 就是过滤器的返回值

        如果不想固定过滤的内容可以在调用是带参数msgForat(参数1,参数2)

<p>{{msg | msgForat('嘿','123') | msgForat2}}</p>

//过滤器 可以传多个参数 也可多次调用

        // Vue.filter('过滤器名称',function(要过滤的传来数据,调用的参数1,...){})

        // Vue.filter('nameope',function(data){

        //     return data+123

        // })

        Vue.filter('msgForat',function(msg,arg,arg2){

            return msg.replace(/嘻/g,arg+arg2) //将全部嘻变为嘿再加上123

        })

  Vue.filter('msgForat2',function(msg){

            return msg+'===========' //msg后面加=======

        })

### 局部

<!-- 结果为过滤器的返回值 money为过滤器的第一个参数value，3是第二个参数times -->

            {{ money | toMoney(3)}}

//定义私有过滤器

            //条件：过滤器名称 ，处理函数

            //调用过程采用就近原则，如果私有过滤器和全局过滤器名称一致。先调用私有过滤器

            filters:{

                toMoney:(value,times) =>{

//value是money的值，times是传入的参数3

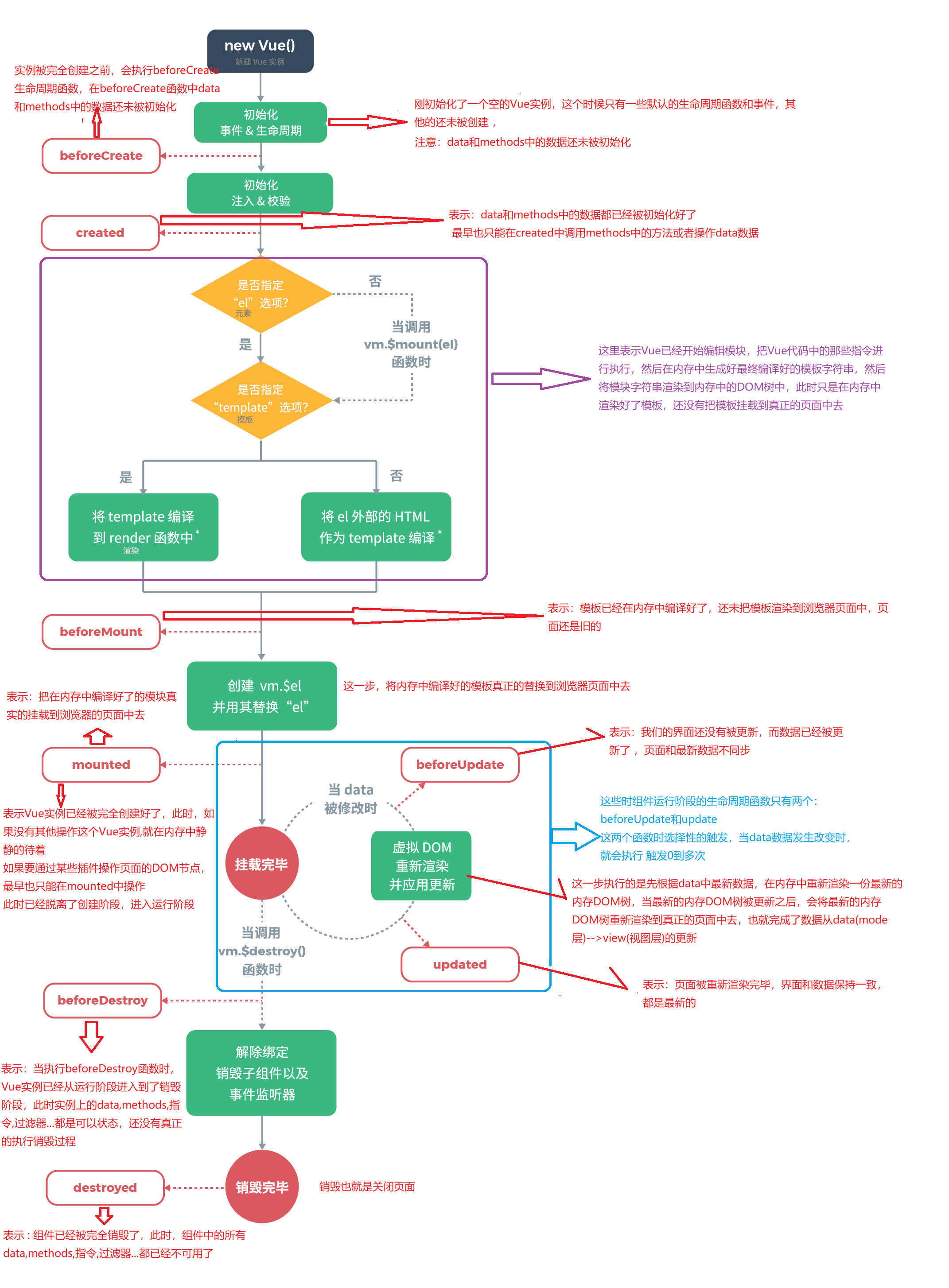
                    console.log(value,times);

                    return value+"私有~~~~";

                }

            }

# 生命周期

Vue生命周期 = 生命周期钩子 = 生命周期事件 （只是名字不同，是一个意思）

生命周期分为三类：创建，运行，销毁

### 创建期间的生命周期函数

beforeCreate  ，created ， beforeMount ， mounted

beforeCreate  刚初始化

beforeCreate 表示：刚初始化了一个空的Vue实例，这个时候只有一些默认的生命周期函数和事件，其他的还未被创建 ，

//实例被完全创建之前，会执行beforeCreate生命周期函数

//注意：在beforeCreate函数中data和methods中的数据还未被初始化

created 初始化完成

 // created  表示：data和methods中的数据都已经被初始化好了 ，

 // 最早也只能在created中调用methods中的方法或者操作data数据

beforeMount 模板在内存中编译好了

 // beforeMount 表示：模板已经在内存中编译好了，还未把模板渲染到浏览器页面中，页面还是旧的

mounted 将模板渲染到页面上

// mounted 表示：把在内存中编译好了的模块真实的挂载到浏览器的页面中去

// 注意：mounted生命周期函数是创建生命周期的最后一个函数了，当执行完了mounted之后,就表示Vue实例已经被完全创建好了，

// 此时，如果没有其他操作这个Vue实例,就在内存中静静的待着

 // 如果要通过某些插件操作页面的DOM节点，最早也只能在mounted中操作

 // 此时已经脱离了创建阶段，进入运行阶段

### 运行期间的生命周期函数

beforeUpdate ， updated

//这两个函数，当data上的数据被修改了,就会执行 触发0--多次

// beforeUpdate 表示：我们的界面还没有被更新，而数据已经被更新了 ，页面和最新数据不同步

// updated  表示：页面被重新渲染完毕，界面和数据保持一致，都是最新的

### 销毁期间的生命周期函数 （关闭页面）

beforeDestroy ，destroyed

// beforeDestroy 表示：当执行beforeDestroy函数时，Vue实例已经从运行阶段进入到了销毁阶段，此时实例上的data,methods,指令,过滤器...都是可以状态，还没有真正的执行销毁过程

// destroyed  表示 : 组件已经被完全销毁了，此时，组件中的所有data,methods,指令,过滤器...都已经不可用了

# AJAX异步请求

## vue-resource

注意：Vue-resource依赖于vue，所有先后顺序要注意

this.$http.jsonp

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

<script src="node\_modules/vue-resource/dist/vue-resource.min.js"></script>

 vue-resource 提供了 7 种请求 API(REST 风格)：

 <!--

            vue-resource 提供了 7 种请求 API(REST 风格)：

            get(url, [options])

            head(url, [options])

            delete(url, [options])

            jsonp(url, [options])

            post(url, [body], [options])

            put(url, [body], [options])

            patch(url, [body], [options])

         -->

<input type="button" value="get请求" @click="getInfo">

<input type="button" value="post请求" @click="postInfo">

<input type="button" value="jsonp请求" @click="jsonpInfo">

 const vm = new Vue({

            el:"#app",

            data:{},

            methods:{

                getInfo:function(){//发送get请求

                    // 通过.then获取服务器发送的数据 ，来设置成功的回调函数

                    // this.$http.get('url地址',[options]).then(成功回调，失败回调)

                    this.$http.get('http://route.showapi.com/852-2').then(function(res){

                       //res.body拿到服务器返回成功的数据

                        console.log(res.body)

                    },function(error){

                        console.log("请求失败")

                    })

                },

                postInfo:function(){//post请求

                    // this.$http.post('url地址',[数据对象],[options]).then(成功回调，失败回调)

                    //第一个参数 ：路径

                    //第二个参数 ：要提交给服务器的数据，以对象的形式

                    //第三个参数：配置对象，要以那种表单数据类型提交 {emulateJSON:true}普通表单数据提交

                    //手动发起的post请求是默认没有表单格式，所有有的服务器处理不了

                    //通过post方法的第三个参数 {emulateJSON:true}。

                    // emulateJSON 的作用： 如果Web服务器无法处理编码为 application/json

                    // 的请求（类型改为普通表单数据），你可以启用 emulateJSON 选项。

                    this.$http.post('http://route.showapi.com/852-2',{},{emulateJSON:true}).then(res => {

                        console.log(res.body)

                    },error =>{

                        console.log("请求失败")

                    })

                },

                jsonpInfo:function(){//jsonp请求

                    // jsonp请求：其实是动态创建script，借助script发送跨域请求，必须有服务端的支持

                    // this.$http.jsonp('url地址',[options]).then(成功回调，失败回调)

                     this.$http.jsonp('http://route.showapi.com/852-2').then(res => {

                        console.log(res.body)

                    },error =>{

                        console.log("请求失败")

                    })

                }

            }

        })

## axios

axios本身并不支持发送跨域的请求， 没有提供相应的API，所以只能使用第三方库，如jQuery。

let vm = new Vue({

            el:'#app',

            data(){

                return {

                    msg:null

                }

            },

            //生命周期函数运行期 已经将模板放入到页面中

            mounted(){

                //get请求 axios.get(url).then().catch

                axios

                .get("http://route.showapi.com/852-2")

                .then((response) =>{this.msg =response}) //成功 response请求到的数据

                .catch(function(error){ //失败

                    console.log(error)

                })

                //post请求

                axios.post('http://route.showapi.com/852-2', {

                    firstName: 'Fred',        // 参数 firstName

                    lastName: 'Flintstone'    // 参数 lastName

                })

                .then(function (response) {

                    console.log(response);

                })

                .catch(function (error) {

                    console.log(error);

                });

             }

        })

# 动画

## 过渡动画transition

开始：

##-enter: 定义进入过渡的开始状态 （还没有进入，元素的起始状态）

##-enter-active: 定义过渡的状态 （从开始到结束的过程这个时间段）

##-enter-to: 2.1.8版及以上 定义进入过渡的结束状态

离开：

##-leave: 定义离开过渡的开始状态

##-leave-active: 定义过渡的状态 （从开始到结束的过程这个时间段）

##-leave-to: 2.1.8版及以上 定义离开过渡的结束状态 （已经离开了，元素的结束状态）

.fade-enter, .fade-leave-to /\* .fade-leave-active, 2.1.8 版本以下 \*/ {

                opacity: 0; /\*透明度 0完全透明 1完全不透明\*/

                transform : translateX(20px) /\*2D到3D的转换\*/

            }

.fade-enter-active, .fade-leave-active {

                transition: all 2s ease; /\*ease由慢变快过渡\*/

            }

<!-- transition这是vue官方提供的标签 -->

            <!-- name="fade" 加上name可以不要使用v-...，可以区分不同的动画 -->

<!--mode="out-in" 先出后进-->

            <transition name="fade"  mode="out-in">

                <p v-show="show" v-bind:style="styles">动画</p>

            </transition>

## 过渡动画组transition-group

 /\* 只有上半场动画开始enter 在透明y轴80向不透明y轴0过渡 \*/

            .v-enter , .v-leave-to{

                opacity: 0;/\*从透明到不透明\*/

                transform: translateY(80px);/\*从y轴80px到原位置\*/

            }

            .v-enter-active , .v-leave-active{

                transition: all 1s ease;

            }

            /\* 删除时的动画 固定写法 使元素渐渐的飘上来 \*/

            .v-move{

                transition: all 1s ease;

            }

            .v-leave-active{

                position: absolute;

            }

<!-- 通过v-for渲染的要使用 transition-group-->

                <!-- v-for设置动画必须要设置key值 -->

                <!-- appear实现入场时的动画效果 （页面刚展示时） -->

                <!-- 默认时是将transition-group渲染为span标签 -->

                <!-- tag指定渲染成什么标签 -->

                <transition-group appear tag="ul">

                    <li v-for="(item,index) in  list" :key="item.id" @click="deletes(index)">

                        {{item.id}} -----{{item.name}}

                    </li>

                </transition-group>

 methods:{

                    add(){

                        this.list.push({id:this.id,name:this.name})

                        this.id = this.name=""

                    },

                    deletes(id){

                        console.log(id)

                        this.list.splice(id,1)

                    }

                }

## 钩子函数过渡 before-enter, enter, after-enter

仿购物车小球进车

<div id="app">

            <input type="button" v-on:click="show=!show" value="点击我"/>

            <transition

                v-on:before-enter="beforeEnter"

                v-on:enter="enter"

                v-on:after-enter="afterEnter"

            >

                <h3 class="qiu" v-show="show"></h3>

            </transition>

        </div>

new Vue({

                el:'#app',

                data:{

                    show:false

                },

                methods:{

                    //el等同于document.getElementById("")找到transition标签

                    beforeEnter(el){ //开始进入动画之前

                        el.style.transform="translate(0,0)"

                    },

                    enter(el,done){//开始进入动画,这里是结束时的状态

                        el.offsetWidth //只有加这个才可以时动画由过渡效果

                        el.style.transform="translate(200px,400px)"

                        el.style.transition="all 1s ease"

                        done()//只有调用done()才会立即调用afterEnter函数，立刻隐藏小球

                    },

                    afterEnter(el){//进入动画之后，结束后的状态，小球隐藏

                        this.show = !this.show //隐藏

                    }

                }

            })

## 使用第三方动画库animate

<link rel="stylesheet" href="js/node\_modules/animate.css/animate.css">

<div id="app">

            <input type="button" v-on:click="show=!show" value="点击我"/>

            <!-- transition这是vue官方提供的标签 -->

            <!--enter-active-class animate.css提供的 在class中要加入animated,  bounceIn和bounceOut这是样式，想要的效果 -->

            <!-- :duration = "200"统一设置动画出场和离开的时长 -->

            <!-- <transition enter-active-class="animated bounceIn" leave-active-class="animated bounceOut" :duration="100">

                <p v-show="show" v-bind:style="styles">动画</p>

            </transition> -->

            <hr/>

            <!--:duration = "{enter:200,leave:300}"分别设置动画出场和离开的时长   -->

            <transition enter-active-class="animated bounceIn" leave-active-class="animated bounceOut" :duration="{enter:200,leave:300}">

                <p v-show="show" v-bind:style="styles">动画</p>

            </transition>

        </div>

# 组件

## 创建注册方法1

<div id="app">

            <mycomponent></mycomponent>

            <!-- 如果组件名称是myComponent -->

            <!-- <my-component></my-component> -->

        </div>

//Vue.extend创建全局vue组件

            let com = Vue.extend({

                // template属性，指定组件要展示的html内容

                //template属性，只能有一个根元素

                template : "<h2>这是使用Vue.extend创建出来的模板</h2>"

            })

            //Vue.component定义组件

            //Vue.component('组件名',创建出来的模板对象)

            //如果组件名称是myComponent,在html页面使用组件时要my-component

            Vue.component("mycomponent",com)

//方法2

            // Vue.component("mycomponent",Vue.extend({

            //     template : "<h2>这是使用Vue.extend合并一起写创建出来的模板</h2>"

            // }))

## 创建注册方法2

组件模板

<!--1.创建组件 只能有唯一的一个根元素-->

        <template id="account">

            <div>

                <h1>账号组件内容:{{mag}}{{comput}}</h1>

                <a href="#" v-on:click="login">账号事件的绑定</a>

                <a href="">密码</a>

            </div>

        </template>

使用组件

<div id="app">

            <!--3.组件的使用 全局组件可以在任何div中重复使用-->

            <account1></account1>

            <account1></account1>

        </div>

创建注册组件

//2.定义和注册一个组件

            Vue.component('account1',{

                template:'#account',//注册的组件

                //组件使用的数据

                //data是一个function

                //放回值必须是一个对象

                data:function(){

                    return{

                        mag:'hello word',

                        comput:0

                    }

                },

                //组件事件的绑定

                methods:{

                    login:function(){

                        alert("账号");

                        this.comput++

                    }

                }

            })

## 私有组件

<template id="component">

            <div>

                <h2>私有组件</h2>

            </div>

        </template>

<div id="app">

<!--3.组件的使用 私有组件只能app使用-->

    <logins></logins>

</div>

<div id="app2">

     <!-- 不能使用 -->

     <logins></logins>

</div>

new Vue({

                el:'#app',

                data:{},

                components:{

                    logins:{

                        template:'#component'

                    }

                }

            })

            new Vue({

                el:"#app2"

            })

## 两个组件切换

<div id="app">

            <input type="button" value="登录" @click="flag = true">

            <input type="button" value="注册" @click="flag = false">

            <!--3.组件的使用 全局组件可以在任何div中重复使用-->

            <logins v-if="flag"></logins>

            <res v-else="flag"></res>

        </div>

//2.定义和注册一个组件

            //登录

            Vue.component('logins',{

                template:'<h2>登录页面</h2>',//注册的组件

            })

            //注册

            Vue.component('res',{

                template:'<h2>注册页面</2>'

            })

            new Vue({

                el:'#app',

                data:{

                    flag : true

                },

            })

## 多个组件切换

<div id="app">

            <input type="button" value="登录" @click="comName = 'logins'">

            <input type="button" value="注册" @click="comName = 'res'">

            <input type="button" value="邮箱" @click="comName = 'email'">

            <!-- component Vue自己提供的 -->

            <!-- 不带‘’就是一个变量在data数据中，带‘’就是一个字符串 -->

            <component :is="comName"></component>

            <!-- Vue自己提供的标签 component , template , transition , transitionGroup -->

        </div>

//2.定义和注册一个组件

            //登录

            Vue.component('logins',{

                template:'<h2>登录页面</h2>',//注册的组件

            })

            //注册

            Vue.component('res',{

                template:'<h2>注册页面</2>'

            })

            //邮箱

            Vue.component('email',{

                template:'<h2>邮箱页面</2>'

            })

            new Vue({

                el:'#app',

                data:{

                    comName:'logins'

                },

            })

## 结合动画切换

只需要将<component></component>放到transition中去就可以了

<style>

            .v-enter,.v-leave-to{

                opacity: 0;

                transform: translateX(200px);

            }

            .v-enter-active , .v-leace-active{

                transition: all 2s ease;

            }

        </style>

 <!-- component Vue自己提供的 -->

            <!-- mode="out-in" 别人完全出去了再进来 -->

            <!-- 不带‘’就是一个变量在data数据中，带‘’就是一个字符串 -->

            <transition  mode="out-in">

                <component :is="comName"></component>

            </transition>

## 父子组件

<template id="com\_ptemplate">

        <div>

            <h3>hahahahaha子组件</h3>

        </div>

    </template>

    <template id="com\_ptemplate2">

       <div>

           <h2>hahahaha2父组件</h2>

//使用子组件

           <comp></comp>

       </div>

    </template>

<div id="app">

        <comp2></comp2>

    </div>

//创建和注册组件

        //子组件

        let com = Vue.extend({

            template:'#com\_ptemplate'

        })

        //父组件

        let com2 = Vue.extend({

            template:'#com\_ptemplate2',

            components:{//注册子组件

                comp:com

            }

        })

        //根组件 等同于父组件

        let vm = new Vue({

            el:"#app",

            data:{},

            components:{//注册父组件

                comp2:com2

            }

        })

## 父组件向子组件传值 props

1.子组件接收数据 props

Prors:{ names:Array}

2.将父组件的值赋给

<account1 :names='name'></account1>

:names='name' :子组件传递的名称='父组件的data数据'

<!--创建组件-->

        <template id="account">

            <div>

                <ul v-for='item in names'>

                    <li>{{item}}</li>

                </ul>

                <p>{{msg}}</p>

                <p>{{obj}}</p>

            </div>

        </template>

<div id="app">

            <!--使用组件-->

            <!-- 名称为驼峰标识就要使用-隔开 my-name -->

            <!--:names='name' :子组件传递的名称='父组件的data数据'  -->

            <account1 :names='name'></account1>

        </div>

//组件的创建和注册

            Vue.component('account1',{

                template:'#account',

                //子组件接收数据

                // props:['names'] //方法1  names绑定属性的名称

                props:{ //方法2 对象形式

                    names:Array,//数据类型 多类型时可以使用数组

                    msg:{//当没有传值时，使用默认值

                        type:String,

                        default:'222',

                        required:true //必传 :msg='msg'

                    },

                    //类型是对象或数组时必须是一个function

                    obj:{

                        type:Object,

                        default:function(){

                            return {

                                name:'sss',

                                age:18

                            }

                        }

                    }

                }

            })

            new Vue({

                el:'#app',

                //父组件的数据

                data:{

                    msg:'czx',

                    name:[1,2,3,4,5],

                    obj:{

                        name:'czx',

                        age:18

                    }

                }

            })

## 子组件向父组件传值 $emit 自定义事件

1. 子组件通过事件发送数据

<h1 @click="alerts">子组件发送数据</h1>

1. 定义方法 ，发送事件

alerts(){

//发送事件

this.$emit('sends',123)//$emit传值

}

1. 父组件监听子组件发送的事件

<account1 v-on:sends="getdata"></account1>

4.在父组件方法中使用数据

getdata(gets){//gets这个参数默认就是子组件发送事件传的值123

alert(gets);

}

<!--创建子组件-->

        <template id="account">

            <h1 @click="alerts">子组件发送数据</h1>

        </template>

        <!--父组件-->

        <div id="app">

            <!-- 子组件使用 -->

            <!-- 监听子组件发送事件 -->

            <account1 v-on:sends="getdata"></account1>

        </div>

//定义和注册子组件

            Vue.component('account1',{

                template:'#account',

                //定义方法

                methods:{

                    alerts(){

                        //发送事件

                        this.$emit('sends',123)//$emit传值

                    }

                }

            })

            new Vue({

                el:'#app',

                data:{ },

                methods:{

                    getdata(gets){//gets这个参数默认就是子组件发送事件传的值123

                        alert(gets);

                    }

                }

            })

## 父组件访问子组件 $children , $refs(使用比较多)

### $children

this.$children 取子组件数据 取所有子组件 （数组形式）

<button @click="btncllick">点击</button>

 btncllick(){

                    //this.$children取子组件数据 取所有子组件 数组

                    // console.log(this.$children)

                    // for (const key of this.$children) {

                    //     // console.log(key) 所有数据data和methods

                    //     console.log(key.msg)

                    // }

}

### $refs

this.$refs 取指定的子组件 要在子组件标签上加上ref="aaa"

<comp :name="vmname" ref="aaa"></comp>

<button @click="btncllick">点击</button>

btncllick(){

        //this.$refs 取指定的子组件 要在子组件标签上加上ref="aaa"

          console.log(this.$refs.aaa.msg)

 }

## 子组件访问父组件 $parent $root

this.$parent子组件访问父组件的数据

This.$root 直接访问根组件（vue实例）

<!-- app的子组件 -->

    <template id="comp">

        <div>

            <childcomp></childcomp>

        </div>

    </template>

    <!-- comp的子组件 -->

    <template id="ccomp">

        <div>

            <h2>子组件的子组件childcomp</h2>

            <button @click="btncllick">点击</button>

        </div>

    </template>

    <!-- 根组件 -->

    <div id="app">

        <comp :name="vmname"></comp>

        <comp :name="vmname"></comp>

    </div>

const vm = new Vue({

//根组件

            el:"#app",

            data:{

                vmname:"123"

            },

//子组件comp

            components:{

                comp:{

                    template:"#comp",

                    data(){

                        return{

                           msg:"hello world comp"

                        }

                    },

                    props:{

                        name:{

                            type:String,

                            default:"czx"

                        }

                    },

//子子组件 ccomp

                    components:{

                        childcomp:{

                            template:'#ccomp',

                            methods:{

                                btncllick(){

                                    //父组件是comp

                                    //this.$parent子组件访问父组件的数据 对象

                                    console.log(this.$parent)

                                    console.log(this.$parent.msg)

                                    //直接访问根组件（vue实例）

                                    console.log(this.$root)

                                    console.log(this.$root.vmname)

                                }

                            }

                        },

                    }

                }

            }

        })

# 模块化

js模块演化：没有--》CMD（同步） --》AMD(异步) --》语言提供模块支持（同步，异步）

CommonJS是主要为了JS在后端的表现制定的,是后端模块规范，用于服务器 导入导出

AMD是前端，异步模块定义，它采用异步方式加载模块，模块的加载不影响它后面语句的运行，

因为浏览器是异步操作，只能使用异步模块 RequireJS实现了AMD 用于浏览器

CMD和AMD很像，大名远扬的玉伯写了seajs，就是遵循他提出的CMD规范，CMD是同步加载模块

ES6 模块不是对象，而是通过export命令显式指定输出的代码（导出），再通过import命令输入（导入）。

ES6的模块浏览器不支持，要使用webpack编译

# 插槽slot

插槽就是增加复用性 可以多个地方使用 结构一样内容不一样 抽取共性，保留不同

## 基本使用 slot

 <template id="comp">

        <div>

            <h2>子组件</h2>

            <slot><h3>插槽默认值</h3></slot>

        </div>

    </template>

 <div id="app">

        <comp><button>按钮</button></comp> //按钮

        <comp></comp> //插槽默认值

        <comp>

            <!-- 多个一起替换 -->

            <span>hahahah</span>

            <p>xixixi</p>

        </comp>

    </div>

## 具名插槽

插槽

<template id="comp">

        <div>

            <slot name="left"><span>左边</span></slot>

            <slot name="content"><span>中间</span></slot>

            <slot name="right"><span>右边</span></slot>

        </div>

</template>

<div id="app">

        <!-- 替换指定插槽 -->

        <comp>

            <span slot="left">xxx</span>

            <input type="text" slot="content">

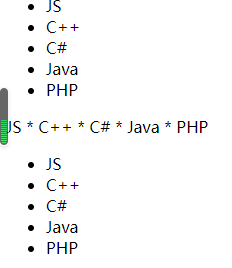
            <span slot="right">返回</span>

        </comp>

    </div>

## 作用域插槽

父组件显示的数据是子组件提供的



<template id="comp">

        <div>

            <!-- 获取子组件的值planguages -->

            <slot :data="planguages">

                <ul>

                    <li v-for="item in planguages">{{item}}</li>

                </ul>

            </slot>

        </div>

    </template>

 <div id="app">

        <!-- 替换指定插槽 -->

        <comp></comp>

        <comp>

            <!-- 将子组件获取的值显示 获取子组件的数据 v-slot -->

           <template v-slot="slots">

                <!-- <span v-for="item in slots.data">{{item}}-</span> -->

                <span>{{slots.data.join(' \* ')}}</span>

           </template>

        </comp>

        <comp></comp>

    </div>

 const vm = new Vue({

            el:"#app",

            data:{},

            components:{

                comp:{

                    template:"#comp",

                    data(){

                        return{

                            planguages:['JS','C++','C#','Java','PHP']

                        }

                    }

                }

            }

        })

# Roter路由

## 什么是路由

后端路由：所有的url地址对应服务器上对应的资源

前端路由：实现不同页面的切换，跳转，借助哈希（hash）实现,

Hash的特点：HTTP请求中不会包含hash相关的内容，所以单页面程序中的页面跳转主要用hash实现 （a链接锚点的实现就是使用hash）,

使用hash跳转页面就是前端路由

## 路由步骤

导入vue-router插件

1.使用router-link组件来导航，to指定链接;

2.使用router-view占位;

3.定义跟组件，导航的组件;

4.定义路由和注册路由;

5.创建vue实例，使用router配置参数注入路由

<script src="js/vue-route.js" type="text/javascript" charset="utf-8"></script>

1.

<!-- 使用 router-link 组件来导航. -->

    <!-- 通过传入 `to` 属性指定链接. -->

    <!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->

    <!-- tag:'span'可以改变为任何标签 -->

    <router-link to="/login/小小">登录</router-link>  <!--传入参数：小小-->

    <router-link to="/zhuce">注册</router-link>

2.

<!-- 路由出口 -->

  <!-- 路由匹配到的组件将渲染在这里 -->

  <router-view></router-view>

3.

//定义登录组件

//注册组件,只能使用对象，不能使用组件引用名称（不能使用Vue.component来注册的）

    var login={

        template:'<h2>登录:{{onmae}}</h2>',

        data:function(){

            return{

                onmae:''

            }

        },

        //接收数据

        created:function(){

            // console.log(this.$route)

            this.onmae=this.$route.params.uname;

        }

    };

4.

//定义路由和注册

    var routers=new VueRouter({//创建router实例

        routes:[//定义路由

            //path:表示监听那个路由地址

            //component: 地址匹配成功，展示component属性中的组件内容

{path:'/',redirect:'/login'}, //redirect重定向

            {path:'/login/:uname',component:login},

            {path:'/zhuce',component:zhuce}

        ]，

//修改默认类 router-link-active 为myclass

         linkActiveClass:'myclass'

    });

5.

new Vue({

        el:'#app',

        router:routers//通过 router 配置参数注入路由

    })

<!-- tag指定渲染为什么标签 -->

      <!-- replace不可以使用网页进行回退 使用replaceState模式 -->

      <!-- active-class 修改默认class名 也可以在router中添加linkActiveClass就不用写active-class-->

      <!-- <router-link to="/" tag="button" replace active-class="activebtn">Home</router-link>  -->

## 路由传参数

<!-- 链接 -->

        <!-- 方法1 直接在链接上？传参 query -->

        <!-- <router-link to="/login?id=10">登录</router-link> -->

        <!-- 在vuerouter实例上的path传参 params-->

        <router-link to="/login/12/小i下">登录</router-link>

        <router-link to="/regs">注册</router-link>

 <!-- 占位符 -->

        <router-view></router-view>

//注册组件,只能使用对象，不能使用组件引用名称（使用Vue.component来注册的）

        let login = {

            template:'<h2>登录组件 ---{{id}}---{{name}}</h2>',

            data(){

                return{

                    id:'',

                    name:''

                }

            },

            //接收数据,生命周期函数，data和methods都已经初始化好了

            created(){

//$route是跳转的路由对象

//使用？传参数是使用this.$route.query

                // this.id = this.$route.query.id

                // console.log(this.$route)

                this.id = this.$route.params.id

                this.name = this.$route.params.name

            }

        }

//定义注册路由

        const routers = new VueRouter({

            //定义路由

            routes :[

                //path:表示监听那个路由地址

                //component: 地址匹配成功，展示component属性中的组件内容

                // {path:'/',component:login}, 根路径为登录页面

///login/:id 动态传入参数

                {path:'/',redirect:'/login'}, //redirect重定向

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

                {path:'/login/:id/:name', component:login},

                {path:'/regs', component:regs}

            ]

        })

        const vm = new Vue({

            el:'#app',

            data:{},

            router : routers //注册路由到vue实例上

        })

## 嵌套路由

{

                    path:'/login',

                    component:login,

                    children:[//子路由

                        {path:'',component:user},

                        {path:'user',component:user},

                        {path:'pasd',component:pasd},

                    ]

},

<!-- tag指定渲染为什么标签 -->

      <!-- replace不可以使用网页进行回退 使用replaceState模式 -->

      <!-- active-class 修改默认class名 也可以在router中添加linkActiveClass就不用写active-class-->

      <!-- <router-link to="/" tag="button" replace active-class="activebtn">Home</router-link>  -->

      <router-link to="/Home" replace>Home</router-link>

      <router-link to="/about" replace>About</router-link>

      <button @click="routerclick">按钮</button>

</div>

## Keep-alive

    <!-- 组件只会创建一次,不会销毁 (keep-alive)-->

    <!-- exclude 排除 重新创建，离开销毁 -->

    <!-- <keep-alive exclude='home'>  home为导出的name-->

    <!-- <keep-alive exclude='home,about'> 排除多个  -->

    <keep-alive>

       <router-view></router-view>

    </keep-alive>

## 导航守卫

{

    path: '/Home',

    name: 'Home',

    component: Home,

    //路由独享守卫

    beforeEnter:(to,from,next) =>{

      console.log('Homer独享守卫')

      next()

    },

    // 元数据（描述数据的数据）

    meta:{

      title:'首页'

    }

  },

//全局守卫

//前置钩子（hook）

//导航守卫(guard) 页面跳转页面的title也会改变

//跳转前回调

router.beforeEach((to,from,next) =>{

  //从from跳到to (从离开的路由跳转到去的路由)

  document.title = to.matched[0].meta.title

  next()

})

//后置钩子

//跳转后回调

router.afterEach((to,from) =>{})

//组件守卫

  beforeRouterEnter(to,from,next){

  },

   beforeRouterUpdate(to,from,next){

  },

  beforeRouterLeave(to,from,next){

  }

## 函数

export default {

  name:'About',

  data(){

    return{

      path:'/about/New'

    }

  },

//创建

  created(){

    console.log('about created')

  },

//销毁

  destroyed(){

    console.log('about destroyed')

  },

  //该组件为活跃状态时调用（也就是在当前页面）

  activated(){

//跳转到指定路径

//this.$router.push指定路由

    this.$router.push(this.path)

    console.log('activated')

  },

  //该组件为不活跃状态时调用

  deactivated(){

    console.log('deactivated')

  },

  //导航离开该组件对应路由时调用

  beforeRouteLeave(to,from,next){

    console.log(this.$route.path)

    //当离开时，当前路由变成最后点击时的路由

    this.path = this.$route.path

    next()

  }

}

</script>

## $router和$route的区别

$router 实例，全局路由对象包含很多子对象和属性 history

router为VueRouter的实例，相当于一个全局的路由器对象，里面含有很多属性和子对象，

        例如history对象。。。经常用的跳转链接就可以用this.$router.push，和router-link跳转一样

        this.$router.push会往history栈中添加一个新的记录。详细见vue官方文档https://router.vuejs.org/zh/guide/essentials/navigation.html

$route 当前跳转的路由对象

        route相当于当前正在跳转的路由对象。。可以从里面获取name,path,params,query等。。

console.log(this.$router) //vuerouter实例

      console.log(this.$route) //路由对象（可以获取动态路径）

跳转路由 push,replace

      //跳转路由

      // this.$router.push('/about')

      this.$router.replace('/Home')

# webpack打包vue

const VueLoaderPlugin = require('vue-loader/lib/plugin')

 //vue  vue-loader vue-template-compiler

            //在vue-loader@15版本之后要搭配vueloaderplugin使用

            //const VueLoaderPlugin = require('vue-loader/lib/plugin')

            {

                test:/\.vue$/,

                use:['vue-loader']

            }

 plugins:[

        new VueLoaderPlugin()

]

配置vue打包  runtime-only改为compiler-included 就可以使用template 不修改不可以使用template标签

    resolve: {

        extensions:['.js','.css','.vue'], //在导入vue文件没有省略后缀名

        alias: {

            'vue$': 'vue/dist/vue.js'

        }

    }

# Vue-cli脚手架

## 安装步骤

### 安装node.js node.js官网直接下载

node -v查看版本包 有版本号则安装成功

### 安装npm

安装node高版本npm直接包含（下载文件较慢 npm install 下载的架包）

npm -v查看版本号 有版本号则安装成功

也可以使用淘宝镜像把npm的所有内容放到自己的服务器底 下，(可选项)：安装cnpm:

npm i -g cnpm --registry=https://registry.npm.taobao.org

安装了cnpm之后npm的所有命令都可以使用cnpm代替

### 安装脚手架 3

用于构建vue项目 npm install -g @vue/cli

可选 ：安装快速原型开发（以.vue结尾的文件）

npm install -g @vue/cli-service-global

去运行vue文件：vue serve 文件名 （如果文件名为app.vue 则可以直接 vue serve去运行就可以了）

复制Network的路径去浏览器运行

补充：目前的vue-cli是3版本 ，如果以后要使用2版本解决方法：

* 卸载之前的cli,重新安装 npm uninstall vue-cli -g
* 安装一个桥接工具 npm install -g @vue/cli-init

### 安装脚手架2

npm install -g @vue/cli-init

## 创建项目 脚手架3

#### 命令行工具创建

创建：vue create 项目名

1. 如果第一次使用，会问是否使用淘宝镜像 选择Y)
2. 选择所依赖的项，选第二个)
3. 只用babel, 空格选择，取消除了babel前面的所有选项的\*号)
4. 第一个从新创建一个文件config放配置，第二个是再package.json)
5. 存一个预设 y n ,y下次创建就不用再写了）

如果想要删除预设就在C:\Users\Administrator\.vuerc中的presets删除

1. 保存名字输入,下一步就进行创建了)

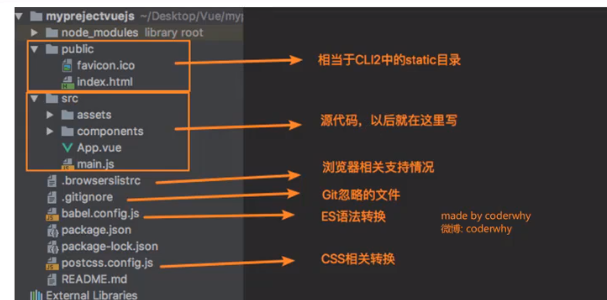
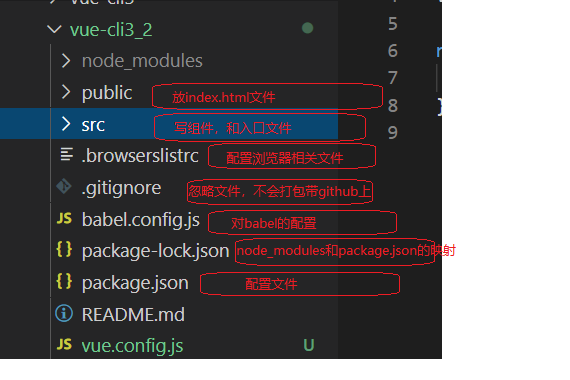
进入文件目录下运行：npm run serve

#### 图形化界面创建

vue ui

和上面的选项是一样的

#### 目录结构



/\* 构建信息，在终端提示 \*/

Vue.config.productionTip = false

new Vue({

  render: h => h(App),

}).$mount('#app')

和

new Vue({

el:’app’

  render: h => h(App),

}).$mount('#app')

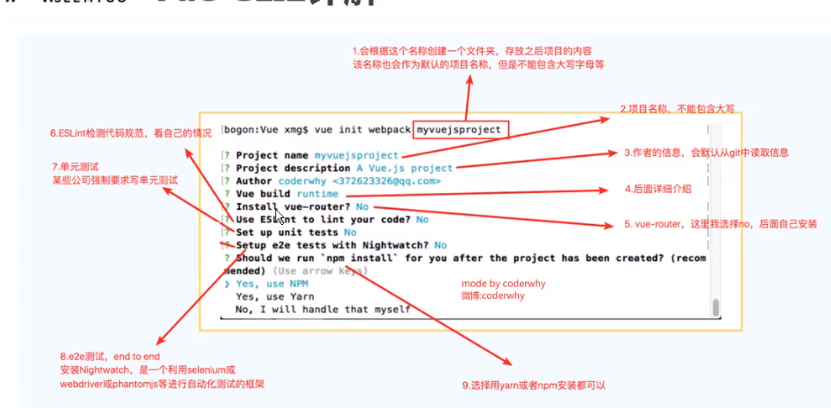
是一样的只是源码内部多做了一层判断，最终都将会被render函数中的组件替换，内部执行$mount(‘#app’)

## 创建项目 脚手架2

创建：vue init webpack 项目名

运行 ： npm run dev

### 创建脚手架步骤:



### runntime-compiler和runntiome-only的区别

主要区别在于main.js

Vue运行进程：template --解析为-->ast(抽象语法树)--编译为-->render函数--翻译为-->virtual DOM(虚拟DOM树) --转为-->真实DOM ,渲染到页面上

runntime-compiler是在vue实例中注册组件（components）,然后使用模板引用组件（template）

template ->ast ->render ->vdom ->ui

runntiome-only是在vue实例中使用render函数

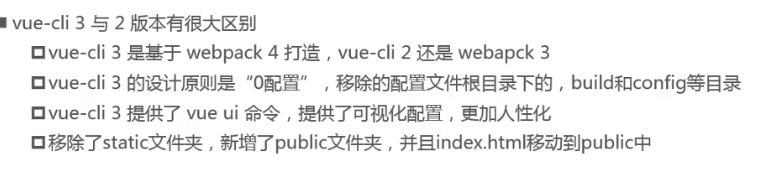
render ->vdom ->ui

runntiome-only性能更高，代码量少

在开发中使用template就选择runntime-compiler ,在开发中使用.vue文件开发，选择runntime-only

### 目录结构

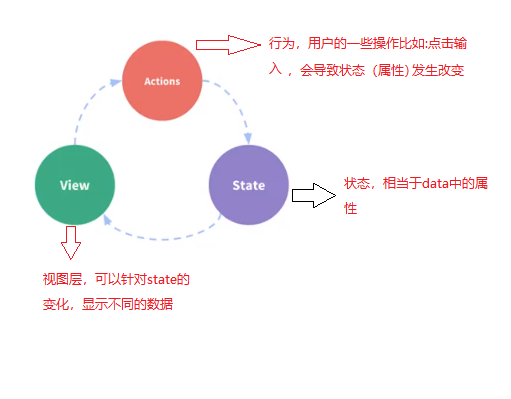


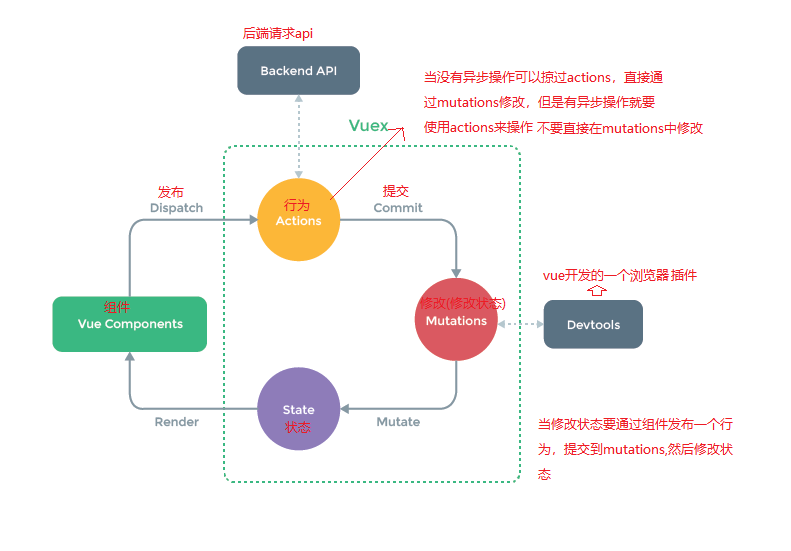


# Vuex

vuex是专门为vue.js提供的状态管理模式，他采取集中式存储管理

状态管理模式：也就是将许多组件共享的变量全部存储在一个对象中，然后将这个对象放在顶层的vue实例中，其他组件就可以使用了，而且这个对象是响应式的，多个组件就可以共享这个对象中的变量了





插件 <https://chromecj.com/web-development/2018-01/886/download.html>

## 单页面状态管理

下载vuex: npm install vuex -S

Store/index.js

import Vue from 'vue'

import Vuex from 'vuex'

//安装插件

Vue.use(Vuex)

//创建对象

const store = new Vuex.Store({

    //状态（属性）

    state:{

        comp:10

    },

    mutations:{},

    actions:{},

    getters:{},

})

export default store

Main.js

import Vue from 'vue'

import App from './App.vue'

import router from './router'

import store from './store'

Vue.config.productionTip = false

// Vue.prototype.$store = store

new Vue({

  router,

  store,

 render: h => h(App)

}).$mount('#app')

App.vue

<!-- $store就是创建的vuex中的store对象 -->

      <!-- const store = new Vuex.Store({}) -->

      <!-- Vue.prototype.$store = store   prototype原型-->

<!--$store.state.comp使用store中是数据-->

      <h2>{{ $store.state.comp }}</h2>

下载浏览器插件vue.js devtools：

<https://chromecj.com/web-development/2018-01/886/download.html>

## Vuex-store

Store/index.js

//创建对象

const store = new Vuex.Store({

    //状态（属性）

    state:{

        comp:10

    },

    //修改，变化（修改状态  方法）

    //不可以直接在view中修改state,要在mutations中修改

    mutations:{

        add(state){

            state.comp++

        },

        jian(state){

            state.comp--

        },

    },

    actions:{},

    getters:{},

})

App.vue

 <!-- actions行为 -->

      <button @click='adds'>++</button>

      <button @click='jians'>--</button>

 methods:{

        adds(){

            //提交在store中的mutations中的方法

            this.$store.commit('add')

        },

        jians(){

            this.$store.commit('jian')

        }

    }

## State单一状态树

一个项目中最好只创建一个store对象

vuex.Store({

State:{} ,//共享状态（属性）

//修改，变化（修改状态 方法）必须是同步函数

//不可以直接在view中修改state,要在mutations中修改

mutations{} ,

//异步操作 行为

actions:{},

//等同于计算属性computed，获取state数据变化后的值,例如获取comp的平方显示在页面上

getters:{},

//模块，解决store变得臃肿的问题，可以分多个模块，每个模块都有state,mutations,getters,action...等等

modules:{ a:modulesA }

})

//模块对象

const modulesA={

state:{

name:'zhangsan'

},

mutations:{

updatename(state,playload){

state.name = playload

}

},

actions:{

//context上下文

//模块中的只能调用模块中的muations

aupdatename(context,playload){

setTimeout(() => {

console.log(context)

context.commit('updatename',playload)

}, 1000);

}

},

getters:{

fullname(state){

return state.name + '名字'

},

fullname2(state,getters){

return getters.fullname + '名字2'

},

//第一个参数是state状态

//第二个参数getters对象

//第三个参数store实例中的state状态

fullname3(state,getters,rootstate){

return getters.fullname2 + rootstate.comp

}

}

}

## 例子

import Vue from 'vue'

import Vuex from 'vuex'

import{Add} from './mutations.type.js'

//安装插件

Vue.use(Vuex)

//模块对象

const modulesA={

 state:{

     name:'zhangsan'

 },

 mutations:{

     updatename(state,playload){

         state.name = playload

     }

 },

 actions:{

     //context上下文

     //模块中的只能调用模块中的muations

     aupdatename(context,playload){

        setTimeout(() => {

            console.log(context)

            context.commit('updatename',playload)

        }, 1000);

     }

 },

 getters:{

    fullname(state){

         return state.name + '名字'

    },

    fullname2(state,getters){

        return getters.fullname + '名字2'

    },

    //第一个参数是state状态

    //第二个参数getters对象

    //第三个参数store实例中的state状态

    fullname3(state,getters,rootstate){

        return getters.fullname2 + rootstate.comp

    }

 }

}

//创建对象

const store = new Vuex.Store({

    //共享状态（属性）

    state:{

        comp:10,

        students:[

            {id:1,name:'小米',age:18},

            {id:2,name:'小红',age:28},

            {id:3,name:'小黑',age:30},

            {id:4,name:'小米白',age:12}

        ],

        info:{

            name:'小i下',

            age:18,

            sex:'女'

        }

    },

    //修改，变化（修改状态  方法）必须是同步函数

    //不可以直接在view中修改state,要在mutations中修改

    mutations:{

        //方法的参数就是状态中的数据

        //在mutations.type.js中定义了Add常量

        [Add](state){

            state.comp++

        },

        // add(state){

        //     state.comp++

        // },

        jian(state){

            state.comp--

        },

        //动态传参 特殊提交风格 playload就是对象，普通就是字符串或数值

        numcont(state,playload){

            console.log(playload) //对象 { type:'numcont',count}

            // state.comp += count

            state.comp += playload.count

        },

        //添加数组对象

        addstu(state,stu){

            state.students.push(stu)

        },

        //修改对象，只有在state初始化了的才可以进行响应式,

        //如果向对象中直接添加属性，是不会发生响应式的，只有使用Vue.set才可以添加

       //state中有的属性可以做到响应式，但是后面加的不会做到响应式

        updateinfo(state){

            //state中初始化了属性name,可以响应式修改

            state.info.name ='czx'

            //虽然添加到了对象中，但是不是响应式的页面中没有显示，state中没有这个属性，没有初始化

            // state.info['addres'] = '中国s'

            //添加属性addres

            // Vue.set(state.info,'addres','中国')

            //删除属性 ，delete也不是响应式的

            // delete state.info.sex

            // Vue.delete(state.info , 'sex')

        }

    },

    //异步操作 行为

    actions:{

        //context上下文 等同于store对象

        //playload传递的参数

        // aupdateinfo(context,playload){

        //     setTimeout(() => {

        //         //先提交mutations中的方法，在去页面发布（dispatch）actions中的方法

        //         context.commit('updateinfo')

        //         // console.log(context)

        //         console.log(playload.magess)

        //         playload.success()

        //     }, 1000);

        // }

        //使用proimse传递参数

        aupdateinfo(context,playload){

            return new Promise((reslove,reject)=>{

                setTimeout(() => {

                    //先提交mutations中的方法，在去页面发布（dispatch）actions中的方法

                    context.commit('updateinfo')

                    // console.log(context)

                    console.log(playload)

                    reslove('xxxx')

                }, 1000);

            })

        }

    },

    //等同于计算属性computed，获取state数据变化后的值,例如获取comp的平方显示在页面上

    getters:{

        //计算comp的平方

        getcomp(state){

            return state.comp \* state.comp

        },

        //计算学生的年龄小于20岁的

        getage(state){

            return state.students.filter(s =>{

                return s.age >=20

            })

        },

        //第二个参数就是getters这个对象

        getagelength(state,getters){

            return getters.getage.length

        },

        //计算学生年龄小于用户传的参数

        getAllage(state){

            return age =>{

                return state.students.filter(s => s.age >age)

            }

        }

    },

    //模块，解决store变得臃肿的问题，可以分多个模块，每个模块都有state,mutations,getters,action...等等

    modules:{

        a:modulesA

    }

})

export default store

# 脚手架axios

Network/request\_axios.js

import axios from 'axios'

//方法1传入函数，然后使用回调函数

export function request(config,success,shibai){

    //创建axios实例

    const instances1 = axios.create({

        baseURL:'http://route.showapi.com',

        timeout:1000

    })

    //axios拦截器

    // axios.interceptors 全局拦截

    //请求拦截

    instances1.interceptors.request.use(config =>{//发送请求成功

        // console.log(config)

        //什么时候使用拦截器

        //1.当请求的数据不符合服务器的要求，要改变就要使用拦截

        //2.比如在网络请求时想显示一个请求图标

        //3.某些网络请求（登录（token）），必须携带一些特殊信息

        return config //拦截了要返回值，别人才可以拿到

    }, err =>{//发送请求失败

        // console.log(err)

    })

    //响应拦截

    instances1.interceptors.response.use(res =>{

        // console.log(res)

        return res

    },err =>{

        console.log(err)

    })

    //发生真正网络请求

    instances1(config)

    .then(res =>{

        success(res)

    }).catch(err =>{

        shibai(err)

    })

}

//方法2 使用回调地狱promise

// export function request(config){

//     return new Promise( (reslove,reject) =>{

//         //创建axios实例

//         const instance2 = axios.create({

//             baseURL:'http://route.showapi.com',

//             timeout:2000

//         })

//         instance2(config)

//         .then(res =>{

//             reslove(res)

//         })

//         .catch(err =>{

//             reject(err)

//         })

//     })

// }

//方法3 直接返回参数

// export function request(config){

//     //创建axios实例

//     const instances1 = axios.create({

//         baseURL:'http://route.showapi.com',

//         timeout:1000

//     })

//     return instances1(config)

// }

//方法4

// export function request(config){

//     //创建axios实例

//     const instances1 = axios.create({

//         baseURL:'http://route.showapi.com',

//         timeout:1000

//     })

//     instances1(config.base)

//     .then(res =>{

//         config.success(res)

//     })

//     .catch(err =>{

//         config.shibai(err)

//     })

// }

Main.js

//封装方法1

request({

  url:'/852-2'

},res =>{

  console.log(res)

},err =>{

  console.log(err)

})

//封装方法2,promise

// request({

//   url:'/852-2'

// }).then(res =>{

//   console.log(res)

// }).catch(err =>{

//   console.log(err)

// })

//方法3,直接返回

// request({

//   url:'/852-2'

// }).then(res =>{

//   console.log(res)

// }).catch(err =>{

//   console.log(err)

// })

//方法4

// request({

//   base:'/852-2',

//   success(res){

//     console.log(res)

//   },

//   shibai(err){

//     console.log(err)

//   }

// })