---

link: null
title: 珠峰架构师成长计划
description: Vue (读音 /vju:/，类似于 view) 是一套用于构建用户界面的渐进式框架。特点: 易用，灵活，高效
渐进式框架
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=117 sentences=132, words=1211

什么是vue?

Vue (读音 /vju:/，类似于 view) 是一套用于构建用户界面的渐进式框架。
特点: 易用，灵活，高效 渐进式框架

逐一递增 vue + components + vue-router + vuex + vue-cli

什么是库，什么是框架?

- 库是将代码集合成一个产品,库是我们调用库中的方法实现自己的功能。
- 框架则是为解决一类问题而开发的产品,框架是我们在指定的位置编写好代码，框架帮我们调用。

框架是库的升级版

初始使用

```
new Vue({
    el:'#app',
    template:'<div>&#x6211;&#x662F;&#x59DC;&#x6587;&#xFF5E;&#xFF5E;</div>', // &#x4F18;&#x5148;&#x4F7F;&#x7528;template
    data:{}
});
```

mvc && mvvm

在传统的mvc中除了model和view以外的逻辑都放在了controller中，导致controller逻辑复杂难以维护,在mvvm中view和model没有直接的关系，全部通过viewModel进行交互

声明式和命令式

- 自己写for循环就是命令式 (命令其按照自己的方式得到结果)
- 声明式就是利用数组的方法forEach (我们想要的是循环，内部帮我们去做)

模板语法 mustache

允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。在使用数据前需要先声明

- 编写三元表达式
- 获取返回值
- JavaScript 表达式

```
<div id="app">
  {{ 1+1 }}
  {{ msg == 'hello'?'yes':'no' }}
  {{ {name:1} }}
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script>
  [illegible]
</script>
```

观察数据变化

```
function notify(){
    console.log('&#x89C6;&#x56FE;&#x66F4;&#x65B0;')
}
let data = {
    name:'jw',
    age:18,
    arr:[]
}
// &#x91CD;&#x5199;&#x6570;&#x7EC4;&#x7684;&#x65B9;&#x6CD5;
let oldProtoMehtods = Array.prototype;
let proto = Object.create(oldProtoMehtods);
['push','pop','shift','unshift'].forEach(method=>{
    proto[method] = function(){
        notify();
        oldProtoMehtods[method].call(this,...arguments)
    }
})
function observer(obj){
    if(Array.isArray(obj)){
        obj.__proto__ = proto
        return;
    }
    if(typeof obj === 'object'){
        for(let key in obj){
            defineReactive(obj,key,obj[key]);
        }
    }
}
function defineReactive(obj,key,value){
    observer(value); // &#x518D;&#x4E00;&#x6B21;&#x5FAA;&#x73AF;value
    Object.defineProperty(obj,key,{ // &#x4E0D;&#x652F;&#x6301;&#x6570;&#x7EC4;
        get(){
            return value;
        },
        set(val){
            notify();
            observer(val);
            value = val;
        }
    });
}
observer(data);
data.arr.push(1);
```

使用proxy实现响应式变化

```
let obj = {
    name:{name:'jw'},
    arr:['&#x5403;','&#x559D;','&#x73A9;']
}
let handler = {
    get(target,key,receiver){
        if(typeof target[key] === 'object' &&  target[key] !== null){
            return new Proxy(target[key],handler);
        }
        return Reflect.get(target,key,receiver);
    },
    set(target,key,value,receiver){
        if(key === 'length') return true;
        console.log('update')
        return Reflect.set(target,key,value,receiver);
    }
}
let proxy = new Proxy(obj,handler);
proxy.name.name = 'zf';
```

响应式变化

- 数组的变异方法(不能通过通过长度，索引改变数组)

```
<div id="app">
    {{hobbies}}
</div>
<script src="node_modules/vue/dist/vue.js"></script>
<script>
    let vm = new Vue({
        el:'#app',
        data:{
            hobbies:['[illegible]','[illegible]','[illegible]']
        }
    });
    vm.hobbies[0] = '[illegible]';// [illegible]
    vm.hobbies.length--;// [illegible]
</script>
```

```
vm.hobbies = ['&#x559D;&#x6C34;']; // &#x66FF;&#x6362;&#x7684;&#x65B9;&#x5F0F;
vm.hobbies.push('&#x5403;&#x996D;'); // push slice pop ...&#x53D8;&#x5F02;&#x65B9;&#x6CD5;
```

- 不能给对象新增属性

```
<div id="app">
{{state.a}}
</div>
<script src="node_modules/vue/dist/vue.js"></script>
<script>
    let vm = new Vue({
        el:'#app',
        data:{
            state:{count:0}
        }
    });
    //vm.state.a = 100; // 新增熟悉不会响应到视图上
</script>
```

- 使用vm.$set方法强制添加响应式数据

```
function $set(data,key,val){
if(Array.isArray(data)){
    return data.splice(key,1,val);
}
defineReactive(data,key,val);
}
$set(data.arr,0,1);

vm.$set(vm.state,'a','100');
```

vue实例上常见属性和方法

- vm.$set();

```
vm.$set(vm.state,'a','100');
```

- vm.$watch();

```
vm.$watch('state.count',function(newValue,oldValue){
    console.log(newValue,oldValue);
});
```

- vm.$mount();

```
let vm = new Vue({
    data:{state:{count:0}}
});
vm.$mount('#app');
```

- vm.$nextTick();

```
vm.state.count = 100; // &#x66F4;&#x9AD8;&#x6570;&#x636E;&#x540E;&#x4F1A;&#x5C06;&#x66F4;&#x6539;&#x7684;&#x5185;&#x5BB9;&#x7F13;&#x5B58;&#x8D77;&#x6765;
// &#x5728;&#x4E0B;&#x4E00;&#x4E2A;&#x4E8B;&#x4EF6;&#x5FAA;&#x73AF;tick&#x4E2D; &#x5237;&#x65B0;&#x961F;&#x5217;
vm.$nextTick(function(){
    console.log(vm.$el.innerHTML);
});
```

- vm.$data
- vm.$el

vue中的指令

在vue中 指令 (Directives) 是带有 v- 前缀的特殊特性,主要的功能就是操作DOM

- v-once

```
<div v-once>{{state.count}} </div>
```

- v-html （不要对用户输入使用v-html显示）

```

```

https://developer.mozilla.org/zh-CN/docs/Web/API/Element/innerHTML#%E5%AE%89%E5%85%A8%E9%97%AE%E9%A2%98 (https://developer.mozilla.org/zh-CN/docs/Web/API/Element/innerHTML#%E5%AE%89%E5%85%A8%E9%97%AE%E9%A2%98)

- v-text
- v-if/v-else使用

v-for使用

- v-for遍历数组

```
fruits:['&#x9999;&#x8549;','&#x82F9;&#x679C;','&#x6843;&#x5B50;']
<div v-for="(fruit,index) in fruits" :key="index">
    {{fruit}}
</div>
```

- v-for遍历对象

```
info:{name:'jiang',location:'&#x56DE;&#x9F99;&#x89C2;',phone:18310349227}
<div v-for="(item,key) in info" :key="key">
    {{item}} {{key}}
</div>
```

- template的使用

```
<template v-for="(item,key) in fruits">
    <span> hello</span>
    <span v-else>world</span>
</template>
```

- key属性的应用

```
<div v-if="flag">
    <span>&#x73E0;&#x5CF0;</span>
    <input key="2">
</div>
<div v-else>
    <span>&#x67B6;&#x6784;</span>
    <input key="1">
</div>
```

- key尽量不要使用索引

```
 <ul>
    <li key="0">&#x1F34C;</li>
    <li key="1">&#x1F34E;</li>
    <li key="2">&#x1F34A;
</li></ul>
<ul>
    <li key="0">&#x1F34A;</li>
    <li key="1">&#x1F34E;</li>
    <li key="2">&#x1F34C;</li>
</ul>
```

属性绑定：（v-bind）

Class 与 Style 绑定

- 数组的绑定

```

```

- 对象类型的绑定

```

```

绑定事件 @ （v-on）

- 事件的绑定 v-on绑定事件
- 事件修饰符 (.stop .prevent) .capture .self .once .passive

vue的双向绑定 (v-model)

```
  <input type="text" :value="value" @input="input">
  <input type="text" v-model="value">
```

- input,textarea
- select

```
<select v-model="select">
    <option v-for="fruit in fruits" :value="fruit">
            {{fruit}}
    </option>
</select>
```

- radio

```
<input type="radio" v-model="value" value="&#x7537;">
<input type="radio" v-model="value" value="&#x5973;">
```

- checkbox

```
<input type="checkbox" v-model="checks" value="&#x6E38;&#x6CF3;">
<input type="checkbox" v-model="checks" value="&#x5065;&#x8EAB;">
```

- 修饰符应用 .number .lazy .trim

```
<input type="text" v-model.number="value">
<input type="text" v-model.trim="value">
```

鼠标 键盘事件

- 按键、鼠标修饰符 Vue.config.keyCodes

```
Vue.config.keyCodes = {
  'f1':112
}
```

watch & computed

- 计算属性和watch的区别 （异步）

```
let vm = new Vue({
    el:'#app',
    data:{
        firstName:'&#x59DC;',
        lastName:'&#x6587;',
        fullName:''
    },
    mounted(){
        this.getFullName();
    },
    methods:{
        getFullName(){
            this.fullName = this.firstName + this.lastName
        }
    },
    watch:{
        firstName(){
            setTimeout(()=>{
                this.getFullName();
            },1000)
        },
        lastName(){
                this.getFullName();
        }
    }
    // &#x8BA1;&#x7B97;&#x5C5E;&#x6027;&#x4E0D;&#x652F;&#x6301;&#x5F02;&#x6B65;
    // computed:{
    //    fullName(){
    //        return this.firstName + this.lastName;
    //    }
    // }
});
```

- 计算属性和 method的区别 （缓存）

条件渲染

- v-if和v-show区别
- v-if/v-else-if/v-else
- v-show

过滤器的应用 (过滤器中的this都是window)

- 全局过滤器 和 局部过滤器
- 编写一个过滤器

```
<div>{{'hello' | capitalize(3)}}</div>
Vue.filter('capitalize',(value,count=1)=>{
  return value.slice(0,count).toUpperCase() + value.slice(count);
});
```

指令的编写

- 全局指令和 局部指令
- 编写一个自定义指令
    - 钩子函数bind，inserted，update

```
    <input type="text" v-focus.color="'red'">
    Vue.directive('focus',{
        inserted:(el,bindings)=>{
            let color = bindings.modifiers.color;
            if(color){
                console.log('color')
                el.style.boxShadow = `1px 1px 2px ${bindings.value}`
            }
            el.focus();
        }
    });
```

- clickoutside指令

```
<div v-click-outside="change">
  <input type="text" @focus="flag=true">
  <div v-show="flag">
      contenter
  </div>
</div>
let vm = new Vue({
  el:'#app',
  data:{
      flag:false
  },
  methods:{
      change(){
          this.flag = false
      }
  },
  directives:{
      'click-outside'(el,bindings,vnode){
          document.addEventListener('click',(e)=>{
              if(!el.contains(e.target,vnode)){
                  let eventName = bindings.expression;
                  vnode.context[eventName]()
              }
          })
      }
  }
})
```

vue中的生命周期#

- beforeCreate 在实例初始化之后，数据观测(data observer) 和 event/watcher 事件配置之前被调用。
- created 实例已经创建完成之后被调用。在这一步，实例已完成以下的配置：数据观测(data observer)，属性和方法的运算， watch/event 事件回调。这里没有$el
- beforeMount 在挂载开始之前被调用：相关的 render 函数首次被调用。
- mounted el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。
- beforeUpdate 数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁之前。
- updated 由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。
- beforeDestroy 实例销毁之前调用。在这一步，实例仍然完全可用。
- destroyed Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。 该钩子在服务器端渲染期间不被调用。

** 钩子函数中该做的事情 #**

- created 实例已经创建完成，因为它是最早触发的原因可以进行一些数据，资源的请求。
- mounted 实例已经挂载完成，可以进行一些DOM操作
- beforeUpdate 可以在这个钩子中进一步地更改状态，这不会触发附加的重渲染过程。
- updated 可以执行依赖于 DOM 的操作。然而在大多数情况下，你应该避免在此期间更改状态，因为这可能会导致更新无限循环。 该钩子在服务器端渲染期间不被调用。
- destroyed 可以执行一些优化操作,清空定时器，解除绑定事件

** vue中的动画 #**

vue中的动画就是从无到有或者从有到无产生的。有以下几个状态 transition组件的应用

```
.v-enter-active,.v-leave-active {
    transition: opacity 0.25s ease-out;
}
.v-enter, .v-leave-to {
    opacity: 0;
}
```

切换isShow的显示或者隐藏就显示出效果啦~

```
<button @click="toggle">toggle</button>
<transition>
    <span v-show="isShow">&#x73E0;&#x5CF0;&#x67B6;&#x6784;</span>
</transition>
```

> 默认的name是以v-开头，当然你可以自己指定name属性来修改前缀
> ** 使用animate.css设置动画 #**

```
.v-enter-active {
    animation:zoomIn  2s linear
}
.v-leave-avitve{
    animation:zoomOut 2s linear
}
```

直接修改激活时的样式

```
<transition enter-active-class="zoomIn" leave-active-class="zoomOut">
    <span class="animated" v-show="isShow">&#x73E0;&#x5CF0;&#x67B6;&#x6784;</span>
</transition>
```

** vue中js动画 #**

```
<transition @before-enter="beforeEnter" @enter="enter" @after-enter="afterEnter">
    <span class="animated" v-show="isShow">&#x73E0;&#x5CF0;&#x67B6;&#x6784;</span>
</transition>
```

对应的钩子有before-leave,leave,after-leave钩子函数,函数的参数为当前元素

```
beforeEnter(el){
    el.style.color="red"
},
enter(el,done){
    setTimeout(()=>{
        el.style.color = 'green'
    },1000);
    setTimeout(() => {
        done();
    }, 2000);
},
afterEnter(el){
    el.style.color = 'blue';
}
```

** 使用js动画库 #**

https://github.com/julianshapiro/velocity (https://github.com/julianshapiro/velocity)

```
<script src="node_modules/velocity-animate/velocity.js"></script>
beforeEnter(el){
    el.style.opacity = 0;
},
enter(el,done){
    Velocity(el, {opacity: 1}, {duration: 2000, complete: done})
},
afterEnter(el){
    el.style.color = 'blue';
},
leave(el,done){
    Velocity(el, {opacity: 0}, {duration: 2000, complete: done})
}
```

** 筛选动画 #**

```
<div id="app">
    <input type="text" v-model="filterData">
    <transition-group enter-active-class="zoomInLeft" leave-active-class="zoomOutRight">
        <div v-for="(l,index) in computedData" :key="l.title" class="animated">
            {{l.title}}
        </div>
    </transition-group>
</div>
<script src="./node_modules/vue/dist/vue.js"></script>
<script>
    let vm = new Vue({
        el:'#app',
        data:{
            filterData:'',
            datalist:[
                {title:'精选1'},
                {title:'精选2'},
                {title:'精选3'},
                {title:'精选4'}
            ]
        },
        computed:{
            computedData(){
                return this.datalist.filter(item=>{
                    return item.title.includes(this.filterData);
                })
            }
        },
    })
</script>
```

- {{

slot v-for i in 5