Vue面试整理

vue-cli脚手架创建项目

基本使用

指令、插值

- 插值、表达式
- 指令、动态属性
- v-html：有xss风险，会覆盖子组件

```
// 指令、插值
<template>
<div>
    <p>文本插值：{{message}}</p>
    <p>js表达式：{{flag?'yes':'no'}}（只能js表达式，不能是执行语句）</p>
    <p :id='dynamicId'>动态属性id,v-bind的语法糖</p>
    <p v-html='rawHtml'>
        <span>有xss风险</span>
        <span>使用v-html后会覆盖子元素</span>
    </p>
</div>
</template>

<script>
export default {
    data(){
        return {
            message:'hello',
            flag:true,
            rawHtml:'指令-原始html<b>加粗</b><i>斜体</i>',
            dynamicId:`${Date.now()}-id`
        }
    }
}
</script>
```

computed和watch

- computed有缓存，data不变则不会重新计算
- watch如何深度监听?
- watch监听引用类型，拿不到oldVal

```
// computed
<template>
    <div>
        <p>num {{num}}</p>
        <p>double1 {{double1}}</p>
        <input v-model="double2"/>
    </div>
```

```
</template>

<script>
export default {
    data() {
        return {
            num: 20
        }
    },
    computed: {
        double1() {
            return this.num * 2
        },
        double2: {
            get() {
                return this.num * 2
            },
            set(val) {
                this.num = val/2
            }
        }
    }
}
</script>
```

```
// watch深度监听
<template>
    <div>
        <input v-model="name"/>
        <input v-model="info.city"/>
    </div>
</template>

<script>
export default {
    data() {
        return {
            name: '双越',
            info: {
                city: '北京'
            }
        }
    },
    watch: {
        name(oldVal, val) {
            // eslint-disable-next-line
            console.log('watch name', oldVal, val) // 值类型，可正常拿到 oldVal 和
val
        },
        info: {
            handler(oldVal, val) {
                // eslint-disable-next-line
                console.log('watch info', oldVal, val) // 引用类型，拿不到 oldVal
。因为指针相同，此时已经指向了新的 val
            },
            deep: true // 深度监听
        }
```

```
    }
}
</script>
```

class和style

```
<template>
    <div>
        <p :class="{ black: isBlack, yellow: isYellow }">使用 class</p>
        <p :class="[black, yellow]">使用 class （数组）</p>
        <p :style="styleData">使用 style</p>
    </div>
</template>

<script>
export default {
    data() {
        return {
            isBlack: true,
            isYellow: true,

            black: 'black',
            yellow: 'yellow',

            styleData: {
                fontSize: '40px', // 转换为驼峰式
                color: 'red',
                backgroundColor: '#ccc' // 转换为驼峰式
            }
        }
    }
}
</script>

<style scoped>
    .black {
        background-color: #999;
    }
    .yellow {
        color: yellow;
    }
</style>
```

条件渲染

- v-if、v-else的用法，可使用变量，也可使用===表达式
- v-if和v-show的区别?
- v-if和v-show的使用场景?

区别：`v-if`不会渲染所有情况对应的组件，`v-show`会渲染所有情况对应的组件，否定向情况会将`display`置为`none`。

使用场景：若切换更新频繁，则使用`v-show`；若更新不频繁，则使用`v-if`。

```
<template>
    <div>
```

```
        <p v-if="type === 'a'">A</p>
        <p v-else-if="type === 'b'">B</p>
        <p v-else>other</p>

        <p v-show="type === 'a'">A by v-show</p>
        <p v-show="type === 'b'">B by v-show</p>
    </div>
</template>

<script>
export default {
    data() {
        return {
            type: 'a'
        }
    }
}
</script>
```

循环（列表）渲染

- 如何遍历对象——可以使用v-for
- key的重要性，key不能乱写（如random或index）
- **v-for和v-if不能一起使用**

```
<template>
    <div>
        <p>遍历数组</p>
        <ul>
            <li v-for="(item, index) in listArr" :key="item.id">
                {{index}} - {{item.id}} - {{item.title}}
            </li>
        </ul>

        <p>遍历对象</p>
        <ul >
            <li v-for="(val, key, index) in listObj" :key="key">
                {{index}} - {{key}} -  {{val.title}}
            </li>
        </ul>
    </div>
</template>

<script>
export default {
    data() {
        return {
            flag: false,
            listArr: [
                { id: 'a', title: '标题1' }, // 数据结构中，最好有 id ，方便使用 key
                { id: 'b', title: '标题2' },
                { id: 'c', title: '标题3' }
            ],
            listObj: {
                a: { title: '标题1' },
                b: { title: '标题2' },
                c: { title: '标题3' },
```

```
            }
        }
    }
}
</script>
```

事件

- event参数，自定义参数
- 事件修饰符，按键修饰符
- 【观察】事件被绑定到哪里?

event是原生的，被挂载在当前元素

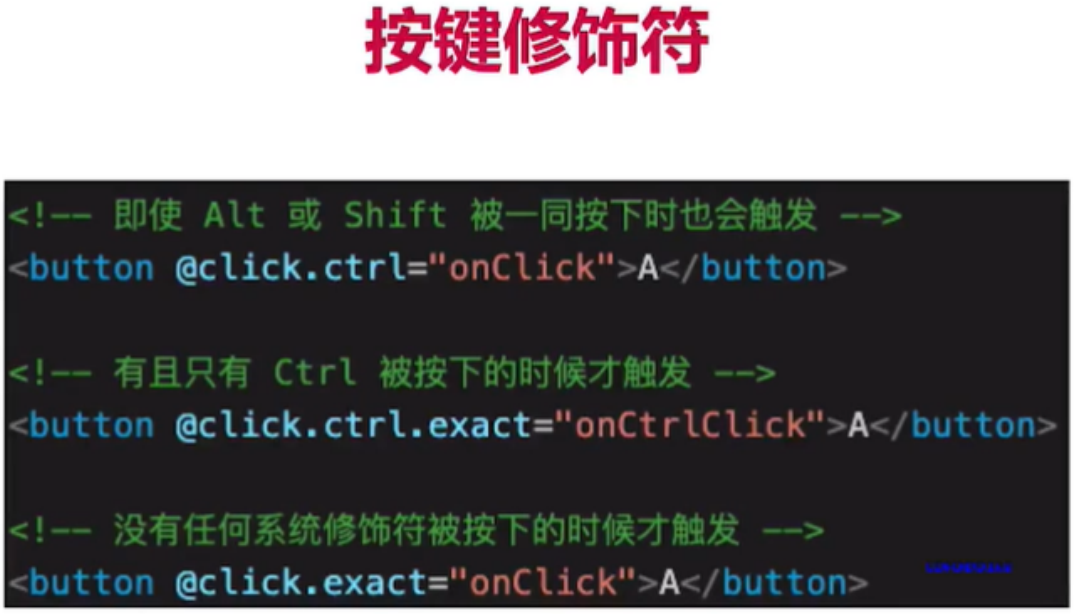
```
<template>
    <div>
        <p>{{num}}</p>
        <button @click="increment1">+1</button>
        <button @click="increment2(2, $event)">+2</button>
    </div>
</template>

<script>
export default {
    data() {
        return {
            num: 0
        }
    },
    methods: {
        increment1(event) {
            console.log('event', event, event.__proto__.constructor) // 是原生的
event 对象
            console.log(event.target)
            console.log(event.currentTarget) // 注意，事件是被注册到当前元素的，和
React 不一样
            this.num++

            // 1. event 是原生的
            // 2. 事件被挂载到当前元素
            // 和 DOM 事件一样
        },
        increment2(val, event) {
            console.log(event.target);
            this.num = this.num + val
        },
        loadHandler() {
            // do some thing
        }
    },
    mounted() {
        window.addEventListener('load', this.loadHandler)
    },
    beforeDestroy() {
        //【注意】用 vue 绑定的事件，组建销毁时会自动被解绑
        // 自己绑定的事件，需要自己销毁！！！
        window.removeEventListener('load', this.loadHandler)
```

```
    }
}
</script>
```

事件修饰符


```
<!-- 阻止单击事件继续传播 -->
<a v-on:click.stop="doThis"></a>
```

按键修饰符



表单

- v-model
- 常见表单项：textarea,checkbox,radio,select
- 修饰符lazy number trim

```
<template>
    <div>
        <p>输入框: {{name}}</p>
        <input type="text" v-model.trim="name"/>
        <input type="text" v-model.lazy="name"/>
        <input type="text" v-model.number="age"/>

        <p>多行文本: {{desc}}</p>
        <textarea v-model="desc"></textarea>
        <!-- 注意，<textarea>{{desc}}</textarea> 是不允许的！！！ -->

        <p>复选框 {{checked}}</p>
        <input type="checkbox" v-model="checked"/>

        <p>多个复选框 {{checkedNames}}</p>
        <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
        <label for="jack">Jack</label>
        <input type="checkbox" id="john" value="John" v-model="checkedNames">
        <label for="john">John</label>
        <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
        <label for="mike">Mike</label>

        <p>单选 {{gender}}</p>
        <input type="radio" id="male" value="male" v-model="gender"/>
        <label for="male">男</label>
        <input type="radio" id="female" value="female" v-model="gender"/>
        <label for="female">女</label>

        <p>下拉列表选择 {{selected}}</p>
        <select v-model="selected">
            <option disabled value="">请选择</option>
            <option>A</option>
            <option>B</option>
            <option>C</option>
        </select>

        <p>下拉列表选择（多选） {{selectedList}}</p>
        <select v-model="selectedList" multiple>
            <option disabled value="">请选择</option>
            <option>A</option>
            <option>B</option>
            <option>C</option>
        </select>
    </div>
</template>

<script>
export default {
    data() {
        return {
            name: '双越',
```

```
            age: 18,
            desc: '自我介绍',

            checked: true,
            checkedNames: [],

            gender: 'male',

            selected: '',
            selectedList: []
        }
    }
}
</script>
```

组件使用

- props和$emit

  父子组件通讯，父组件给子组件传递props，子组件通过this.$emi调用传递进来的父组件的方法

- 组件间通讯-自定义事件

  绑定自定义事件时，需及时在beforeDestroy解除自定义事件绑定，自定义事件可用于任意组件间通讯（可直接使用vue自实现new Vue().event对象）

- 组件生命周期

  挂载阶段

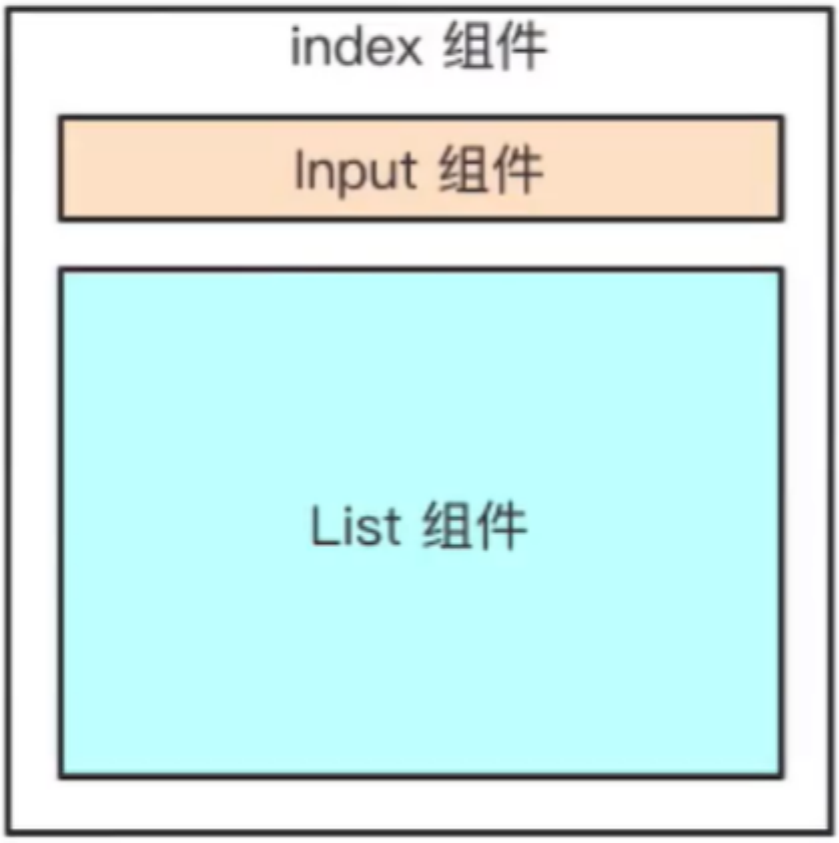
  created和mounted区别：created阶段初始化vue实例未渲染，mounted阶段真正实现dom渲染完毕

  更新阶段

  卸载阶段

  beforeDestroy阶段用于解除绑定、销毁子组件、事件监听器、清除定时器。

生命周期（父子组件）



父子组件生命周期执行顺序：

```
index created
list created
list mounted
index mounted
▶GET http://192.168.1.101
080/sockjs-node/info?t=157
net::ERR_CONNECTION_TIMED_
on add title abc
index before update
list before update
list updated
index updated
```

挂载阶段：

created:父组件->子组件

mounted:子组件->父组件

更新阶段：

beforeUpdate:父组件->子组件

updated:子组件->父组件

卸载阶段：

beforeDestroy:父组件->子组件

destroyed:子组件->父组件

```
// event.js
import Vue from 'vue'

export default new Vue();
```

```
// Input.vue
<template>
    <div>
        <input type="text" v-model="title"/>
        <button @click="addTitle">add</button>
    </div>
</template>

<script>
    import event from './event'

    export default {
        data() {
            return {
                title: ''
            }
        },
        methods: {
            addTitle() {
                // 调用父组件的事件
                this.$emit('add', this.title)

                // 调用自定义事件
                event.$emit('onAddTitle', this.title)

                this.title = ''
            }
        }
    }
</script>


// List.vue
<template>
    <div>
        <ul>
            <li v-for="item in list" :key="item.id">
                {{item.title}}

                <button @click="deleteItem(item.id)">删除</button>
```

```
            </li>
        </ul>
    </div>
</template>

<script>
    import event from './event'

    export default {
        // props: ['list']
        props: {
            // prop 类型和默认值
            list: {
                type: Array,
                default() {
                    return []
                }
            }
        },
        data() {
            return {}
        },
        methods: {
            deleteItem(id) {
                this.$emit('delete', id)
            },
            addTitleHandler(title) {
                // eslint-disable-next-line
                console.log('on add title', title)
            }
        },
        created() {
            // eslint-disable-next-line
            console.log('list created')
        },
        mounted() {
            // eslint-disable-next-line
            console.log('list mounted')

            // 绑定自定义事件
            event.$on('onAddTitle', this.addTitleHandler)
        },
        beforeUpdate() {
            // eslint-disable-next-line
            console.log('list before update')
        },
        updated() {
            // eslint-disable-next-line
            console.log('list updated')
        },
        beforeDestroy() {
            // eslint-disable-next-line no-console
            console.log(`list before destroy`);
            // 及时销毁，否则可能造成内存泄露
            event.$off('onAddTitle', this.addTitleHandler)
        },
        destroyed() {
            // eslint-disable-next-line no-console
```

```
            console.log(`list destroyed`);
        }
    }
</script>


// index.vue
<template>
    <div>
        <Input @add="addHandler"/>
        <List :list="list" @delete="deleteHandler"/>
    </div>
</template>

<script>
import Input from './Input'
import List from './List'

export default {
    components: {
        Input,
        List
    },
    data() {
        return {
            list: [
                {
                    id: 'id-1',
                    title: '标题1'
                },
                {
                    id: 'id-2',
                    title: '标题2'
                }
            ]
        }
    },
    methods: {
        addHandler(title) {
            this.list.push({
                id: `id-${Date.now()}`,
                title
            })
        },
        deleteHandler(id) {
            this.list = this.list.filter(item => item.id !== id)
        }
    },
    created() {
        // eslint-disable-next-line
        console.log('index created')
    },
    mounted() {
        // eslint-disable-next-line
        console.log('index mounted')
    },
    beforeUpdate() {
        // eslint-disable-next-line
```

```
        console.log('index before update')
    },
    updated() {
        // eslint-disable-next-line
        console.log('index updated')
    },
    beforeDestroy() {
        // eslint-disable-next-line no-console
        console.log(`index before destroy`);
    },
    destroyed() {
        // eslint-disable-next-line no-console
        console.log(`index destroyed`);
    }
}
</script>
```

高级特性

自定义v-model

更多查看https://www.jianshu.com/p/22f214bb4294

```
// CustomVModel.vue
<template>
    <!-- 例如：vue 颜色选择 -->
    <input type="text"
        :value="text1"
        @input="$emit('change1', $event.target.value)"
    >
    <!--
        1. 上面的 input 使用了 :value 而不是 v-model
        2. 上面的 change1 和 model.event 要对应起来
        3. text1 属性对应起来
    -->
</template>

<script>
export default {
    model: {
        prop: 'text1', // 对应 props text1
        event: 'change1'
    },
    props: {
        text1: String,
        default() {
            return ''
        }
    }
}
</script>


// index.vue
```

```
<template>
<div>
    <p>{{name}}</p>
    <CustomVModel v-model="name"/>
</div>
</template>

<script>
import CustomVModel from './CustomVModel'

export default {
    components: {
         CustomVModel,
    },
    data() {
        return {
            name: '双越',
        }
    }
}
</script>
```

$nextTick

- vue是异步渲染
- data改变后，Dom不会立刻渲染，data多次改变会整合渲染
- $nextTick会在Dom渲染之后被触发，以获取最新Dom节点

```
// NextTick.vue
<template>
  <div id="app">
    <ul ref="ul1">
        <li v-for="(item, index) in list" :key="index">
            {{item}}
        </li>
    </ul>
    <button @click="addItem">添加一项</button>
  </div>
</template>

<script>
export default {
  name: 'app',
  data() {
      return {
        list: ['a', 'b', 'c']
      }
  },
  methods: {
    addItem() {
        this.list.push(`${Date.now()}`)
        this.list.push(`${Date.now()}`)
        this.list.push(`${Date.now()}`)

         // 获取 DOM 元素
          const ulElem = this.$refs.ul1
```

```
          // eslint-disable-next-line
          console.log( ulElem.childNodes.length ) // 第1次执行结果：3，第2次执行结果：6

        // 1. 异步渲染，$nextTick 待 DOM 渲染完再回调
        // 3. 页面渲染时会将 data 的修改做整合，多次 data 修改只会渲染一次
        this.$nextTick(() => {
          // 获取 DOM 元素
          const ulElem = this.$refs.ul1
          // eslint-disable-next-line
          console.log( ulElem.childNodes.length ) // 第1次执行结果：6，第2次执行结果：9
        })
    }
  }
}
</script>

// index.vue
<template>
<div>
   <NextTick />
</div>
</template>

<script>
import NextTick from './NextTick'

export default {
    components: {
         NextTick,
    }
}
</script>
```

refs

用法见上一条$nextTick用法示例

slot

- 基本使用
- 作用域插槽
- 具名插槽

```
// SlotDemo.vue
<template>
    <a :href="url">
        <slot>
            默认内容，即父组件没设置内容时，这里显示
        </slot>
    </a>
</template>

<script>
```

```
export default {
    props: ['url'],
    data() {
        return {}
    }
}
</script>


// ScopedSlotDemo.vue
<template>
    <a :href="url">
        <slot :slotData="website">
            {{website.subTitle}} <!-- 默认值显示 subTitle ，即父组件不传内容时 -->
        </slot>
    </a>
</template>

<script>
export default {
    props: ['url'],
    data() {
        return {
            website: {
                url: 'http://wangEditor.com/',
                title: 'wangEditor',
                subTitle: '轻量级富文本编辑器'
            }
        }
    }
}
</script>


// index.vue
<template>
<div>
    <!-- 基本使用 -->
    <SlotDemo :url="website.url">
        {{website.title}}
    </SlotDemo>

    <!-- 作用域插槽 -->
     <ScopedSlotDemo :url="website.url">
         <template v-slot="slotProps">
             {{slotProps.slotData.title}}
         </template>
     </ScopedSlotDemo>
</div>
</template>

<script>
import SlotDemo from './SlotDemo'
import ScopedSlotDemo from './ScopedSlotDemo'

export default {
    components: {
         SlotDemo,
```

```
        ScopedSlotDemo,
    },
    data() {
        return {
                website: {
                    url: 'http://imooc.com/',
                    title: 'imooc',
                    subTitle: '程序员的梦工厂'
                },
            }
        }
}
</script>
```

slot – 具名插槽

```
<!-- NamedSlot 组件 -->
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

```
<NamedSlot>
  <!-- 缩写 <template #header> -->
  <template v-slot:header>
    <h1>将插入 header slot 中</h1>
  </template>

  <p>将插入到 main slot 中，即未命名的 slot</p>

  <template v-slot:footer>
    <p>将插入到 footer slot 中</p>
  </template>
</NamedSlot>
```

动态、异步组件

```
// index.vue
<template>
<div>
     <!-- 动态组件 -->
     <component :is="NextTickName"/>

     <!-- 异步组件 -->
     <FormDemo v-if="showFormDemo"/>
     <button @click="showFormDemo = true">show form demo</button>
</div>
</template>

<script>
import NextTick from './NextTick'

export default {
    components: {
         FormDemo: () => import('../BaseUse/FormDemo'),// 异步加载
    },
    data() {
        return {
               NextTickName:"NextTick",
```

```
                showFormDemo: false
            }
        }
}
</script>
```

keep-alive

- 实际是一种缓存组件
- 出现场景：频繁切换，不需要重复渲染,，如tab标签页切换
- 会出现在Vue常见性能优化类问题中

```
// KeepAliveStageA.vue
<template>
    <p>state A</p>
</template>

<script>
export default {
    mounted() {
        // eslint-disable-next-line
        console.log('A mounted')
    },
    destroyed() {
        // eslint-disable-next-line
        console.log('A destroyed')
    }
}
</script>


// KeepAliveStageB.vue
<template>
    <p>state B</p>
</template>

<script>
export default {
    mounted() {
        // eslint-disable-next-line
        console.log('B mounted')
    },
    destroyed() {
        // eslint-disable-next-line
        console.log('B destroyed')
    }
}
</script>


// KeepAliveStageC.vue
<template>
    <p>state C</p>
</template>

<script>
```

```
export default {
    mounted() {
        // eslint-disable-next-line
        console.log('C mounted')
    },
    destroyed() {
        // eslint-disable-next-line
        console.log('C destroyed')
    }
}
</script>


// index.vue
<template>
    <div>
        <button @click="changeState('A')">A</button>
        <button @click="changeState('B')">B</button>
        <button @click="changeState('C')">C</button>

        <keep-alive> <!-- tab 切换 -->
            <KeepAliveStageA v-if="state === 'A'"/>
            <KeepAliveStageB v-if="state === 'B'"/>
            <KeepAliveStageC v-if="state === 'C'"/>
        </keep-alive>
    </div>
</template>

<script>
import KeepAliveStageA from './KeepAliveStateA'
import KeepAliveStageB from './KeepAliveStateB'
import KeepAliveStageC from './KeepAliveStateC'

export default {
    components: {
        KeepAliveStageA,
        KeepAliveStageB,
        KeepAliveStageC
    },
    data() {
        return {
            state: 'A'
        }
    },
    methods: {
        changeState(state) {
            this.state = state
        }
    }
}
</script>
```

mixin

- 多个组件有相同逻辑，抽离出来
- mixin并不是完美的解决方案，有问题

    - 变量来源不明确，不利于阅读，代码可读性差
    - 多mixin可能会造成命名冲突
    - mixin和组件可能出现多对多的关系，复杂度较高
- vue3提出的composition API旨在解决上面的问题

```
// mixin.js
export default {
    data() {
        return {
            city: '北京'
        }
    },
    methods: {
        showName() {
            // eslint-disable-next-line
            console.log(this.name)
        }
    },
    mounted() {
        // eslint-disable-next-line
        console.log('mixin mounted', this.name)
    }
}
```

```
// index.vue
<template>
    <div>
        <p>{{name}} {{major}} {{city}}</p>
        <button @click="showName">显示姓名</button>
    </div>
</template>

<script>
import myMixin from './mixin'

export default {
    mixins: [myMixin], // 可以添加多个，会自动合并起来
    data() {
        return {
            name: '双越',
            major: 'web 前端'
        }
    },
    methods: {
    },
    mounted() {
        // eslint-disable-next-line
        console.log('component mounted', this.name)
    }
}
</script>
```