Vue全家桶

常用语法及 技巧

- id：使用 npm i nanoid 使用时 {nanoid}
- 组件传递数据
  - 父与子　正常用：传递
  - 子与父　父给子 传函数　子用 props 接受　方法中 使用函数　函数回调输入数据
  - 父与孙　　main.js 创建 全局总线　　父挂载中 $bus.$on('名字'，回调)　传数据　$bus.$emit('父的名字'，数据) 传输
- 单向数据绑定：v-bind：　双向绑定 v-model

vue核心

初识Vue：

- 介绍：动态构建用户界面的渐进式 JavaScript 框架　　尤雨溪
- 特点：遵循mvvm模式

Hello案例

- ```
  <!-- 准备好一个容器 -->

    <h1>Hello, {{name.toUpperCase()}}, {{address}}</h1>

   <script type="text/javascript">
    Vue.config.productionTip = false; //阻止 vue 在启动时生成生产提示。
    //创建Vue实例
    new Vue({
      el: "#demo", //el用于指定当前Vue实例为哪个容器服务，值通常为css选择器字符串。
      data: {
        //data中用于存储数据，数据供el所指定的容器去使用，值我们暂时先写成一个对象。
        name: "atguigu",
        address: "北京",
      },
    });
  </script>
  ```

- 注意：
- 1.想让Vue工作，就必须创建一个Vue实例，且要传入一个配置对象；
- 2.root容器里的代码依然符合html规范，只不过混入了一些特殊的Vue语法
- 3.root容器里的代码被称为【Vue模板】；
- 4.Vue实例和容器是一一对应的；
- 5.真实开发中只有一个Vue实例，并且会配合着组件一起使用；
- 6.{{xxx}}中的xxx要写js表达式，且xxx可以自动读取到data中的所有属性；
  ```

- 7.一旦data中的数据发生改变，那么页面中用到该数据的地方也会自动更新；

模板语法

- 1.插值语法：
  - 功能：用于解析标签体内容。
  - 写法：{{xxx}}，xxx是js表达式，且可以直接读取到data中的所有属性。
- 2.指令语法：
  - 功能：用于解析标签（包括：标签属性、标签体内容、绑定事件.....）。
  - 举例：v-bind:href="xxx" 或 简写为 :href="xxx"，xxx同样要写js表达式，且可以直接读取到data中的所有属性。
  - 备注：Vue中有很多的指令，且形式都是：v-????，此处我们只是拿v-bind举个例子。

```
<div id="root">
            <h1>插值语法</h1>
            <h3>你好，{{name}}</h3>
            <hr/>
            <h1>指令语法</h1>
            <a  v-bind:href="school.url.toUpperCase()" x="hello">点我去{{school.name}}学习1</a>
            //缩写
            <a :href="school.url" x="hello">点我去{{school.name}}学习2</a>
        </div>
    new Vue({
            el:'#root',
            data:{
                name:'jack',
                school:{
                    name:'尚硅谷',
                    url:'http://www.atguigu.com',
                }
            }
        })
```

数据绑定

- Vue中有2种数据绑定的方式：
  - 1.单向绑定(v-bind)：数据只能从data流向页面。
  - 2.双向绑定(v-model)：数据不仅能从data流向页面，还可以从页面流向data。\
    - 备注：
    - 1.双向绑定一般都应用在表单类元素上（如：input、select等）
    - 2.v-model:value 可以简写为 v-model，因为v-model默认收集的就是value值。

```
单向数据绑定：<input type="text" :value="name"><br/>
双向数据绑定：<input type="text" v-model="name"><br/>
        new Vue({
            el:'#root',
            data:{
                name:'尚硅谷'
            }
        })
```

data和el的两种写法

- 1.el有2种写法
  - (1).new Vue时候配置el属性。
  - (2).先创建Vue实例，随后再通过vm.$mount('#root')指定el的值。
- 2.data有2种写法
  - (1).对象式
  - 2).函数式
  - 如何选择：目前哪种写法都可以，以后学习到组件时，data必须使用函数式，否则会报错。
- 3.一个重要的原则：
  - 由Vue管理的函数，一定不要写箭头函数，一旦写了箭头函数，this就不再是Vue实例了。

```
  //el的两种写法
    el:'#root', //第一种写法
             v.$mount('#root') //第二种写法 */
    data:{          name:'尚硅谷'          }
             data(){
            return{
                name:'尚硅谷'
            }
```

Vue中MVVM

- MVVM模型
- 1. M：模型(Model)： data中的数据

  2. V：视图(View)： 模板代码

  3. VM：视图模型(ViewModel)：Vue实例

  4. 观察发现：

     1.data中所有的属性，最后都出现在了vm身上。

     2.vm身上所有的属性 及 Vue原型上所有属性，在Vue模板中都可以直接使用。

数据代理

- 回顾Object.defineproperty方法
  - ```
    <script type="text/javascript" >
    ```
    ```

```
            let number = 18
            let person = {
                name:'张三',
                sex:'男',
            }
            Object.defineProperty(person,'age',{
                // value:18,
                enumerable:true, //控制属性是否可以枚举，默认值是false
                writable:true, //控制属性是否可以被修改，默认值是false
                configurable:true //控制属性是否可以被删除，默认值是false
                //当有人读取person的age属性时，get函数(getter)就会被调用，且返回值就是age的值
                    get(){
                    console.log('有人读取age属性了')
                    return number
                },
                //当有人修改person的age属性时，set函数(setter)就会被调用，且会收到修改的具体值
                set(value){
                    console.log('有人修改了age属性，且值是',value)
                    number = value
                }
            })
            // console.log(Object.keys(person))
            console.log(person)
        </script>
```

vue中的数据代理

- 1.Vue中的数据代理：
  - 通过vm对象来代理data对象中属性的操作（读/写）
- 2.Vue中数据代理的好处：
  - 更加方便的操作data中的数据
- 3.基本原理：
  - 通过Object.defineProperty()把data对象中所有属性添加到vm上。
  - 为每一个添加到vm上的属性，都指定一个getter/setter。
  - 在getter/setter内部去操作（读/写）data中对应的属性。

事件处理

- 1.使用v-on:xxx 或 @xxx绑定事件，其中xxx是事件名；
- 2.事件的回调需要配置在methods对象中，最终会在vm上；
- 3.methods中配置的函数，不要用箭头函数！否则this就不是vm了；
- 4.methods中配置的函数，都是被Vue所管理的函数，this的指向是vm 或 组件实例对象；
- 5.@click="demo" 和 @click="demo($event)" 效果一致，但后者可以传参；

```
<div id="root">
            <h2>欢迎来到{{name}}学习</h2>

            <!-- <button v-on:click="showInfo">点我提示信息</button> -->
```

```
            <button @click="showInfo1">点我提示信息1（不传参）</button>
            <button @click="showInfo2($event,66)">点我提示信息2（传参）</button>
</div>
const vm = new Vue({
            el:'#root',
            data:{
                name:'尚硅谷',
            },
            methods:{
                showInfo1(event){
                    // console.log(event.target.innerText)
                    // console.log(this) //此处的this是vm
                    alert('同学你好！')
                },
                showInfo2(event,number){
                    console.log(event,number)
                    // console.log(event.target.innerText)
                    // console.log(this) //此处的this是vm
                    alert('同学你好！！')
                }
            }
        })
```

- 事件修饰符
  - Vue中的事件修饰符： 前三个常用
    - 1.prevent：阻止默认事件（常用）；
    - 2.stop：阻止事件冒泡（常用）；
    - 3.once：事件只触发一次（常用）；
    - 4.capture：使用事件的捕获模式；
    - 5.self：只有event.target是当前操作的元素时才触发事件；
    - 6.passive：事件的默认行为立即执行，无需等待事件回调执行完毕；
- 键盘事件
  - 1.Vue中常用的按键别名：
    - 回车 => enter
    - 删除 => delete (捕获“删除”和“退格”键)
    - 退出 => esc
    - 空格 => space
    - 换行 => tab (特殊，必须配合keydown去使用)
    - 上 => up
    - 下 => down
    - 左 => left
    - 右 => right
  - 2.Vue未提供别名的按键，可以使用按键原始的key值去绑定，但注意要转为kebab-case（短横线命名）
  - 3.系统修饰键（用法特殊）：ctrl、alt、shift、meta
    - (1).配合keyup使用：按下修饰键的同时，再按下其他键，随后释放其他键，事件才被触发。
    - (2).配合keydown使用：正常触发事件。
  - 4.也可以使用keyCode去指定具体的按键（不推荐）

- 5.Vue.config.keyCodes.自定义键名 = 键码，可以去定制按键别名
- 6.console.log(e.keyCode,e.key); 使用代码查询 按键对应名称 两个单词组合 全小写 + -
  - 如CapsLock 对应名字： caps-lock

计算属性

- 1.定义：要用的属性不存在，要通过已有属性计算得来。
- 2.原理：底层借助了Objcet.defineproperty方法提供的getter和setter。
- 3.get函数什么时候执行?
  - (1).初次读取时会执行一次。
  - 2).当依赖的数据发生改变时会被再次调用。
- 4.优势：与methods实现相比，内部有缓存机制（复用），效率更高，调试方便。
- 5.备注：
  - 1.计算属性最终会出现在vm上，直接读取使用即可。
  - 2.如果计算属性要被修改，那必须写set函数去响应修改，且set中要引起计算时依赖的数据发生改变。
- 6. 简写 当只有get() 时 可以直接写出函数形式

```
<script type="text/javascript">
        Vue.config.productionTip = false //阻止 vue 在启动时生成生产提示。

        const vm = new Vue({
            el:'#root',
            data:{
                firstName:'张',
                lastName:'三',
                x:'你好'
            },
            },

  computed:{
        fullName:{
        //get有什么作用? 当有人读取fullName时，get就会被调用，且返回值就作为fullName的值
        //get什么时候调用? 1.初次读取fullName时。2.所依赖的数据发生变化时。
        get(){
        console.log('get被调用了')
        return this.firstName + '-' + this.lastName
        },
        //set什么时候调用? 当fullName被修改时。
        set(value){
        console.log('set',value)
        const arr = value.split('-')
        this.firstName = arr[0]
        this.lastName = arr[1]
                    }
                }
            }
        })
            //简写
                fullName(){
```

```
                    console.log('get被调用了')
                    return this.firstName + '-' + this.lastName
                }
    </script>
```

监视属性

- 监视属性watch:
- 1.当被监视的属性变化时, 回调函数自动调用, 进行相关操作
- 2.监视的属性必须存在，才能进行监视!
- 3.监视的两种写法:
  - (1).new Vue时传入watch配置
  - (2).通过vm.$watch监视
- 深度监视:
  - (1).Vue中的watch默认不监测对象内部值的改变（一层）。
  - (2).配置deep:true可以监测对象内部值改变（多层）。
  - 备注:
    - (1).Vue自身可以监测对象内部值的改变，但Vue提供的watch默认不可以!
    - (2).使用watch时根据数据的具体结构，决定是否采用深度监视。
  - (3).// immediate:true, // deep:true,// 两个属性没有时可以简写
-

```
const vm = new Vue({
            el:'#root',
            data:{
                isHot:true,
            },
            computed:{
                info(){
                    return this.isHot ? '炎热' : '凉爽'
                }
            },
            methods: {
                changeWeather(){
                    this.isHot = !this.isHot
                }
            },
            watch:{
                isHot:{
                    immediate:true, //初始化时让handler调用一下
                    //handler什么时候调用? 当isHot发生改变时。
                    handler(newValue,oldValue){
                        console.log('isHot被修改了',newValue,oldValue)
                    }
                }
            }
        })

//.通过vm.$watch监视
```

```
        vm.$watch('isHot',{
            immediate:true, //初始化时让handler调用一下
            //handler什么时候调用？当isHot发生改变时。
            handler(newValue,oldValue){
                console.log('isHot被修改了',newValue,oldValue)
            }
        })
```

- 与computed区别： computed能完成的功能，watch都可以完成。 watch可以执行异步操作
- 所有不被Vue所管理的函数（定时器的回调函数、ajax的回调函数等、Promise的回调函数），最好写成箭头函数， 这样this的指向才是vm 或 组件实例对象。

绑定样式

- class样式
  - 写法:class="xxx" xxx可以是字符串、对象、数组。
  - 字符串写法适用于：类名不确定，要动态获取。
  - 对象写法适用于：要绑定多个样式，个数不确定，名字也不确定。
  - 数组写法适用于：要绑定多个样式，个数确定，名字也确定，但不确定用不用。
- style样式
  - :style="{fontSize: xxx}"其中xxx是动态值。
  - :style="[a,b]"其中a、b是样式对象。
-

```
<div id="root">
            <!-- 绑定class样式--字符串写法，适用于：样式的类名不确定，需要动态指定 -->
            <div class="basic" :class="mood" @click="changeMood">{{name}}</div> <br/>
<br/>

            <!-- 绑定class样式--数组写法，适用于：要绑定的样式个数不确定、名字也不确定 -->
            <div class="basic" :class="classArr">{{name}}</div> <br/><br/>

            <!-- 绑定class样式--对象写法，适用于：要绑定的样式个数确定、名字也确定，但要动态决定用不用 -->
            <div class="basic" :class="classObj">{{name}}</div> <br/><br/>

            <!-- 绑定style样式--对象写法 -->
            <div class="basic" :style="styleObj">{{name}}</div> <br/><br/>
            <!-- 绑定style样式--数组写法 -->
            <div class="basic" :style="styleArr">{{name}}</div>
        </div>
<script type="text/javascript">
        Vue.config.productionTip = false

        const vm = new Vue({
            el:'#root',
            data:{
                name:'尚硅谷',
                mood:'normal',   //字符串
                classArr:['atguigu1','atguigu2','atguigu3'], //数组
                classObj:{          //对象
```

```
                    atguigu1:false,
                    atguigu2:false,
                },
                styleObj:{          //style对象样式
                    fontSize: '40px',
                    color:'red',
                },
                styleObj2:{
                    backgroundColor:'orange'
                },
                styleArr:[            //style数组样式
                    {
                        fontSize: '40px',
                        color:'blue',
                    },
                    {
                        backgroundColor:'gray'
                    }
                ]
            },
            methods: {
                changeMood(){
                    const arr = ['happy','sad','normal']
                    const index = Math.floor(Math.random()*3)
                    this.mood = arr[index]
                }
            },
        })
    </script>
```

条件渲染

- 1.v-if
- 写法：(1).v-if="表达式"
  - (2).v-else-if="表达式"
  - (3).v-else="表达式"
  - 适用于：切换频率较低的场景。
  - 特点：不展示的DOM元素直接被移除。
  - 注意：v-if可以和:v-else-if、v-else一起使用，但要求结构不能被“打断”。
- 2.v-show
  - 写法：v-show="表达式"
  - 适用于：切换频率较高的场景。
  - 特点：不展示的DOM元素未被移除，仅仅是使用样式隐藏掉
- 3.备注：使用v-if的时，元素可能无法获取到，而使用v-show一定可以获取到。
-

```
<div id="root">
            <h2>当前的n值是:{{n}}</h2>
            <button @click="n++">点我n+1</button>
            //<!-- 使用v-show做条件渲染 -->
```

```
            <h2 v-show="false">欢迎来到{{name}}</h2> -->
            <h2 v-show="1 === 1">欢迎来到{{name}}</h2> -->
            //使用v-if做条件渲染 -->
            <h2 v-if="false">欢迎来到{{name}}</h2> -->
            <h2 v-if="1 === 1">欢迎来到{{name}}</h2> -->
            //<!-- v-else和v-else-if -->
                <div v-if="n === 1">Angular</div>
            <div v-else-if="n === 2">React</div>
            <div v-else-if="n === 3">Vue</div>
            <div v-else>哈哈</div> -->
            //<!-- v-if与template的配合使用 -->
            <template v-if="n === 1">
                <h2>你好</h2>
                <h2>尚硅谷</h2>
                <h2>北京</h2>
            </template>
        </div>
```

列表渲染

- v-for指令:
- 1.用于展示列表数据
- 2.语法：v-for="(item, index) in xxx" :key="使用唯一标识" 如：id phone 等
- 3.可遍历：数组、对象、字符串（用的很少）、指定次数（用的很少）

key的原理

- 面试题：react、vue中的key有什么作用？（key的内部原理）
- 1、虚拟DOM中key的作用：
  - key是虚拟DOM对象的标识，当数据发生变化时，Vue会根据【新数据】生成【新的虚拟DOM】，随后Vue进行【新虚拟DOM】与【旧虚拟DOM】的差异比较，比较规则如下
- 2、对比规则：
  - (1).旧虚拟DOM中找到了与新虚拟DOM相同的key：
    - ①.若虚拟DOM中内容没变, 直接使用之前的真实DOM！
    - ②.若虚拟DOM中内容变了, 则生成新的真实DOM，随后替换掉页面中之前的真实DOM。
  - (2).旧虚拟DOM中未找到与新虚拟DOM相同的key
    - 创建新的真实DOM，随后渲染到到页面。
- 3、用index作为key可能会引发的问题：
  - .若对数据进行：逆序添加、逆序删除等破坏顺序操作:
    - 会产生没有必要的真实DOM更新 ==> 界面效果没问题, 但效率低。
  - 如果结构中还包含输入类的DOM：
    - 会产生错误DOM更新 ==> 界面有问题。
- 4、开发中如何选择key?
  - 1.最好使用每条数据的唯一标识作为key, 比如id、手机号、身份证号、学号等唯一值。
  - 2.如果不存在对数据的逆序添加、逆序删除等破坏顺序操作，仅用于渲染列表用于展示，使用index作为key是没有问题的。

列表过滤

- 用到 filter方法 和indexof方法
- indexof方法：是否包含字符

```
用computed实现
        new Vue({
            el:'#root',
            data:{
                keyWord:'',
                persons:[
                    {id:'001',name:'马冬梅',age:19,sex:'女'},
                    {id:'002',name:'周冬雨',age:20,sex:'女'},
                    {id:'003',name:'周杰伦',age:21,sex:'男'},
                    {id:'004',name:'温兆伦',age:22,sex:'男'}
                ]
            },
            computed:{
                filPerons(){
                    return this.persons.filter((p)=>{
                        return p.name.indexOf(this.keyWord) !== -1
                    })
                }
            }
        })
```

列表排序

-

```
<script type="text/javascript">
            new Vue({
                el:'#root',
                data:{
                    keyWord:'',
                    sortType:0, //0原顺序 1降序 2升序
                    persons:[
                        {id:'001',name:'马冬梅',age:30,sex:'女'},
                        {id:'002',name:'周冬雨',age:31,sex:'女'},
                        {id:'003',name:'周杰伦',age:18,sex:'男'},
                        {id:'004',name:'温兆伦',age:19,sex:'男'}
                    ]
                },
                computed:{
                    filPerons(){
                        const arr = this.persons.filter((p)=>{
                            return p.name.indexOf(this.keyWord) !== -1
                        })
                        //判断一下是否需要排序
                        if(this.sortType){
                            arr.sort((p1,p2)=>{

                                return this.sortType === 1 ? p2.age-p1.age : p1.age-
```

```
p2.age
                            })
                        }
                        return arr
                    }
                }
            })
        </script>
```

数据监测

- Vue监视数据的原理：
- 1. vue会监视data中所有层次的数据。
- 2.如何监测对象中的数据?
- 通过setter实现监视，且要在new Vue时就传入要监测的数据。
- (1).对象中后追加的属性，Vue默认不做响应式处理
- (2).如需给后添加的属性做响应式，请使用如下API：
- Vue.set(target，propertyName/index，value) 或
- vm.$set(target，propertyName/index，value)
- 3. 如何监测数组中的数据?
- 通过包裹数组更新元素的方法实现，本质就是做了两件事：
- (1).调用原生对应的方法对数组进行更新。
- (2).重新解析模板，进而更新页面。
- 4.在Vue修改数组中的某个元素一定要用如下方法：
- 1.使用这些API:push()、pop()、shift()、unshift()、splice()、sort()、reverse()
- 2.Vue.set() 或 vm.$set()
- 特别注意：Vue.set() 和 vm.$set() 不能给vm 或 vm的根数据对象 添加属性！！
  - vue中无法直接索引直接操作数组元素
-

```
updateHobby(){
                    this.student.hobby.splice(0,1,'开车')
                    Vue.set(this.student.hobby,0,'开车')
                    this.$set(this.student.hobby,0,'开车')
          addFriend(){
                    this.student.friends.unshift({name:'jack',age:70})
                },
                updateFirstFriendName(){
                    this.student.friends[0].name = '张三'
                },
```

收集表单数据

- 收集表单数据：

- 若：<input type="text"/>，则v-model收集的是value值，用户输入的就是value值。
- 若：☐，则v-model收集的是value值，且要给标签配置value值。
- 若：☐
- 1.没有配置input的value属性，那么收集的就是checked（勾选 or 未勾选，是布尔值）
- 2.配置input的value属性:
- (1)v-model的初始值是非数组，那么收集的就是checked（勾选 or 未勾选，是布尔值）
- (2)v-model的初始值是数组，那么收集的的就是value组成的数组
- 注：v-model的三个修饰符：
- lazy：失去焦点再收集数据
- number：输入字符串转为有效的数字
- trim：输入首尾空格过滤
-

```
    <!-- 准备好一个容器-->
    <div id="root">
        <form @submit.prevent="demo">
            账号：<input type="text" v-model.trim="userInfo.account"> <br/><br/>
            密码：<input type="password" v-model="userInfo.password"> <br/><br/>
            年龄：<input type="number" v-model.number="userInfo.age"> <br/><br/>
            性别：
            男<input type="radio" name="sex" v-model="userInfo.sex" value="male">
            女<input type="radio" name="sex" v-model="userInfo.sex" value="female"> <br/><br/>
            爱好：
            学习<input type="checkbox" v-model="userInfo.hobby" value="study">
            打游戏<input type="checkbox" v-model="userInfo.hobby" value="game">
            吃饭<input type="checkbox" v-model="userInfo.hobby" value="eat">
            <br/><br/>
            所属校区
            <select v-model="userInfo.city">
                <option value="">请选择校区</option>
                <option value="beijing">北京</option>
                <option value="shanghai">上海</option>
                <option value="shenzhen">深圳</option>
                <option value="wuhan">武汉</option>
            </select>
            <br/><br/>
            其他信息：
            <textarea v-model.lazy="userInfo.other"></textarea> <br/><br/>
            <input type="checkbox" v-model="userInfo.agree">阅读并接受<a href="http://www.atguigu.com">《用户协议》</a>
            <button>提交</button>
        </form>
    </div>
</body>
```

```
<script type="text/javascript">
    Vue.config.productionTip = false

    new Vue({
        el:'#root',
        data:{
            userInfo:{
                account:'',
                password:'',
                age:18,
                sex:'female',
                hobby:[],
                city:'beijing',
                other:'',
                agree:''
            }
        },
        methods: {
            demo(){
                console.log(JSON.stringify(this.userInfo))
            }
        }
    })
</script>
```

过滤器 filters:

- 过滤器:
- 定义: 对要显示的数据进行特定格式化后再显示 (适用于一些简单逻辑的处理)
- 语法:
- 1.注册过滤器: Vue.filter(name,callback) 或 new Vue{filters:{}}
- 2.使用过滤器: {{ xxx | 过滤器名}} 或 v-bind:属性 = "xxx | 过滤器名"
- 备注:
- 1.过滤器也可以接收额外参数、多个过滤器也可以串联
- 2.并没有改变原本的数据, 是产生新的对应的数据
-

```
        //局部过滤器
        filters:{
            timeFormater(value,str='YYYY年MM月DD日 HH:mm:ss'){
                // console.log('@',value)
                return dayjs(value).format(str)
            }
        }
```

内置指令

- v-bind :单向绑定解析表达式, 可简写为 :xxx
- v-model :双向数据绑定
- v-for : 遍历数组/对象/字符串
- v-on : 绑定事件监听, 可简写为@
- v-if : 条件渲染 (动态控制节点是否存存在)
- v-else :条件渲染 (动态控制节点是否存存在)
- v-show :条件渲染 (动态控制节点是否展示)
- v-text指令:
- 1.作用: 向其所在的节点中渲染文本内容。
- 2.与插值语法的区别: v-text会替换掉节点中的内容, {{xx}}则不会。
- v-html指令:
- 1.作用: 向指定节点中渲染包含html结构的内容。

- 2.与插值语法的区别：
- (1).v-html会替换掉节点中所有的内容，{{xx}}则不会。
- (2).v-html可以识别html结构。
- 3.严重注意：v-html有安全性问题！！！！
- (1).在网站上动态渲染任意HTML是非常危险的，容易导致XSS攻击。
- (2).一定要在可信的内容上使用v-html，永不要用在用户提交的内容上!
- v-cloak指令（没有值）：
- 1.本质是一个特殊属性，Vue实例创建完毕并接管容器后，会删掉v-cloak属性。
- 2.使用css配合v-cloak可以解决网速慢时页面展示出{{xxx}}的问题。
- v-once指令：
- 1.v-once所在节点在初次动态渲染后，就视为静态内容了。
- 2.以后数据的改变不会引起v-once所在结构的更新，可以用于优化性能。
- v-pre指令：
- 1.跳过其所在节点的编译过程。
- 2.可利用它跳过：没有使用指令语法、没有使用插值语法的节点，会加快编译
-

自定义指令

- 一、定义语法：
- 1).局部指令：在directive里书写 对象和函数两种格式
  - 如果vue把东西渲染到页面时 执行 命令 则 函数形式将 无法使用 推荐 使用对象格式
  - 函数形式 只在绑定成功和重新解析时 调用
- (2).全局指令：Vue.directive 实例对象 之前 配置
- 二、配置对象中常用的3个回调：
- (1).bind：指令与元素成功绑定时调用。
- (2).inserted：指令所在元素被插入页面时调用。
- (3).update：指令所在模板结构被重新解析时调用。
- 三、备注：
- 1.指令定义时不加v-，但使用时要加v-
- 2.指令名如果是多个单词，要使用kebab-case命名方式，不要用camelCase命名。
-

```
//定义全局指令
        Vue.directive('fbind',{
            //指令与元素成功绑定时（一上来）
            bind(element,binding){
                element.value = binding.value
            },
            //指令所在元素被插入页面时
            inserted(element,binding){
                element.focus()
            },
            //指令所在的模板被重新解析时
            update(element,binding){
                element.value = binding.value

            }
```

```
        })

        //局部指令
directives:{                    //函数形式 无法处理
                     'big-number'(element,binding){
                    // console.log('big')
                    element.innerText = binding.value * 10
                },
                big(element,binding){
                    // console.log('big',this) //注意此处的this是window
                    // console.log('big')
                    element.innerText = binding.value * 10
                },
                fbind:{
                    //指令与元素成功绑定时（一上来）
                    bind(element,binding){
                        element.value = binding.value
                    },
                    //指令所在元素被插入页面时
                    inserted(element,binding){
                        element.focus()
                    },
                    //指令所在的模板被重新解析时
                    update(element,binding){
                        element.value = binding.value
                    }
                }
            }
```

生命周期

- 生命周期：
- 1.又名：生命周期回调函数、生命周期函数、生命周期钩子。
- 2.是什么：Vue在关键时刻帮我们调用的一些特殊名称的函数。
- 3.生命周期函数的名字不可更改，但函数的具体内容是程序员根据需求编写的。
- 4.生命周期函数中的this指向是vm 或 组件实例对象
- 常用的生命周期钩子：
- 1.mounted: 发送ajax请求、启动定时器、绑定自定义事件、订阅消息等【初始化操作】。
- 2.beforeDestroy: 清除定时器、解绑自定义事件、取消订阅消息等【收尾工作】。 $destroy()自毁
- 关于销毁Vue实例
- 1.销毁后借助Vue开发者工具看不到任何信息。
- 2.销毁后自定义事件会失效，但原生DOM事件依然有效。
- 3.一般不会在beforeDestroy操作数据，因为即便操作数据，也不会再触发更新流程了。
-

数据代理创建前　beforeCreate()	数据代理创建完成 Create()
数据挂载前 beforeMount()	数据挂载完毕mounted()
数据更新前beforeUpdate()	数据更新完毕updated()
数据销毁beforeDestory()	数据销毁完毕destoryed()

组件

非单文件 单个组件 需要脚手架

- Vue中使用组件的三大步骤：
  - 一、定义组件(创建组件)
  - 二、注册组件
  - 三、使用组件(写组件标签)
    - 一、如何定义一个组件?
      - 使用Vue.extend(options)创建
    - 二、如何注册组件?
      - 1.局部注册：靠new Vue的时候传入components选项
      - 2.全局注册：靠Vue.component('组件名',组件)
    - 三、编写组件标签：<school></school> 不用使用脚手架时，<school/>会导致后续组件不能渲染。
    - 注意：组件名：一个单词：首字母大写 多个单词：首字母大写 不用与html标签重名
- ```
  //创建组件
  const  school = vue.extend({除了el不能写其他同vue，data使用函数})
  //注册组件  在vue中 添加 components
  components:{
                  student
              }
  //使用组件
  <school></school>
  //组件嵌套  在父组件中   使用components  注册子组件  在结构中 使用子组件标签
  template:`

                      <h2>学校名称：{{name}}</h2>
                      <h2>学校地址：{{address}}</h2>
                      <student></student>

              `,
  ```

脚手架文件结构
  ```

```
├── node_modules
├── public
│   ├── favicon.ico: 页签图标
│   └── index.html: 主页面
├── src
│   ├── assets: 存放静态资源
│   │   └── logo.png
│   ├── component: 存放组件
│   │   └── HelloWorld.vue
│   ├── App.vue: 汇总所有组件
│   ├── main.js: 入口文件
├── .gitignore: git版本管制忽略的配置
├── babel.config.js: babel的配置文件
├── package.json: 应用包配置文件
├── README.md: 应用描述文件
├── package-lock.json：包版本控制文件
```

关于不同版本的Vue

1. vue.js与vue.runtime.xxx.js的区别：
    1. vue.js是完整版的Vue，包含：核心功能 + 模板解析器。
    2. vue.runtime.xxx.js是运行版的Vue，只包含：核心功能；没有模板解析器。
2. 因为vue.runtime.xxx.js没有模板解析器，所以不能使用template这个配置项，需要使用render函数接收到的createElement函数去指定具体内容。

vue.config.js配置文件

1. 使用vue inspect > output.js可以查看到Vue脚手架的默认配置。
2. 使用vue.config.js可以对脚手架进行个性化定制，详情见：https://cli.vuejs.org/zh

ref属性

1. 被用来给元素或子组件注册引用信息（id的替代者）
2. 应用在html标签上获取的是真实DOM元素，应用在组件标签上是组件实例对象（vc）
3. 使用方式：
    1. 打标识：`<h1 ref="xxx">.....</h1>` 或 `<School ref="xxx"></School>`
    2. 获取：`this.$refs.xxx`

props配置项

1. 功能：让组件接收外部传过来的数据
2. 传递数据：`<Demo name="xxx"/>`
3. 接收数据：
    1. 第一种方式（只接收）：`props:['name']`
    2. 第二种方式（限制类型）：`props:{name:String}`

3. 第三种方式（限制类型、限制必要性、指定默认值）：

```
props:{
    name:{
    type:String, //类型
    required:true, //必要性
    default:'老王' //默认值
    }
}
```

> 备注：props是只读的，Vue底层会监测你对props的修改，如果进行了修改，就会发出警告，若业务需求确实需要修改，那么请复制props的内容到data中一份，然后去修改data中的数据。

mixin(混入)

1. 功能：可以把多个组件共用的配置提取成一个混入对象
2. 使用方式：

   第一步定义混合：

```
{
    data(){....},
    methods:{....}
    ....
}
```

   第二步使用混入：

   全局混入：`Vue.mixin(xxx)` 局部混入：`mixins:['xxx']`

插件

1. 功能：用于增强Vue
2. 本质：包含install方法的一个对象，install的第一个参数是Vue，第二个以后的参数是插件使用者传递的数据。
3. 定义插件：

```
对象.install = function (Vue, options) {
    // 1. 添加全局过滤器
    Vue.filter(....)

    // 2. 添加全局指令
    Vue.directive(....)

    // 3. 配置全局混入(合)
    Vue.mixin(....)

    // 4. 添加实例方法

    Vue.prototype.$myMethod = function () {...}
```

```
    Vue.prototype.$myProperty = xxxx
}
```

4. 使用插件：`Vue.use()`

scoped样式

1. 作用：让样式在局部生效，防止冲突。
2. 写法：`<style scoped>`

总结TodoList案例

1. 组件化编码流程：

   (1).拆分静态组件：组件要按照功能点拆分，命名不要与html元素冲突。

   (2).实现动态组件：考虑好数据的存放位置，数据是一个组件在用，还是一些组件在用：

   1).一个组件在用：放在组件自身即可。

   2). 一些组件在用：放在他们共同的父组件上（状态提升）。

   (3).实现交互：从绑定事件开始。

2. props适用于：

   (1).父组件 ==> 子组件 通信

   (2).子组件 ==> 父组件 通信（要求父先给子一个函数）

3. 使用v-model时要切记：v-model绑定的值不能是props传过来的值，因为props是不可以修改的!

4. props传过来的若是对象类型的值，修改对象中的属性时Vue不会报错，但不推荐这样做。

webStorage

1. 存储内容大小一般支持5MB左右（不同浏览器可能还不一样）
2. 浏览器端通过 Window.sessionStorage 和 Window.localStorage 属性来实现本地存储机制。
3. 相关API：
   1. `xxxxxStorage.setItem('key', 'value');` 该方法接受一个键和值作为参数，会把键值对添加到存储中，如果键名存在，则更新其对应的值。
   2. `xxxxxStorage.getItem('person');`

      该方法接受一个键名作为参数，返回键名对应的值。
   3. `xxxxxStorage.removeItem('key');`

      该方法接受一个键名作为参数，并把该键名从存储中删除。
   4. `xxxxxStorage.clear()`

      该方法会清空存储中的所有数据。
4. 备注：
   1. SessionStorage存储的内容会随着浏览器窗口关闭而消失。
   2. LocalStorage存储的内容，需要手动清除才会消失。
   3. `xxxxxStorage.getItem(xxx)` 如果xxx对应的value获取不到，那么getItem的返回值是null。

4. `JSON.parse(null)`的结果依然是null。

组件的自定义事件

1. 一种组件间通信的方式，适用于：**子组件 ===> 父组件**
2. 使用场景：A是父组件，B是子组件，B想给A传数据，那么就要在A中给B绑定自定义事件（事件的回调在A中）。
3. 绑定自定义事件：
   1. 第一种方式，在父组件中：`<Demo @atguigu="test"/>` 或 `<Demo v-on:atguigu="test"/>`
   2. 第二种方式，在父组件中：

```
<Demo ref="demo"/>
......
mounted(){
   this.$refs.xxx.$on('atguigu',this.test)
}
```

   3. 若想让自定义事件只能触发一次，可以使用`once`修饰符，或`$once`方法。

      触发自定义事件：`this.$emit('atguigu',数据)`
4. 解绑自定义事件`this.$off('atguigu')`
5. 组件上也可以绑定原生DOM事件，需要使用`native`修饰符。
6. 注意：通过`this.$refs.xxx.$on('atguigu',回调)`绑定自定义事件时，回调要么配置在methods中，要么用箭头函数，否则this指向会出问题！

全局事件总线（GlobalEventBus）

1. 一种组件间通信的方式，适用于任意组件间通信。
2. 安装全局事件总线：

```
new Vue({
    ......
    beforeCreate() {
        Vue.prototype.$bus = this //安装全局事件总线，$bus就是当前应用的vm
    },
    ......
})
```

3. 使用事件总线：
   1. 接收数据：A组件想接收数据，则在A组件中给$bus绑定自定义事件，事件的回调留在A组件自身。

```
methods(){
  demo(data){......}
}
......
mounted() {
  this.$bus.$on('xxxx',this.demo)
}
```

2. 提供数据：`this.$bus.$emit('xxxx',数据)`

4. 最好在beforeDestroy钩子中，用$off去解绑当前组件所用到的事件。

消息订阅与发布（pubsub）

1. 一种组件间通信的方式，适用于任意组件间通信。
2. 使用步骤：
    1. 安装pubsub：`npm i pubsub-js`
    2. 引入：`import pubsub from 'pubsub-js'`
    3. 接收数据：A组件想接收数据，则在A组件中订阅消息，订阅的回调留在A组件自身。

```
methods(){
  demo(data){......}
}
......
mounted() {
  this.pid = pubsub.subscribe('xxx',this.demo) //订阅消息
}
```

    4. 提供数据：`pubsub.publish('xxx',数据)`
    5. 最好在beforeDestroy钩子中，用`PubSub.unsubscribe(pid)`去取消订阅。

nextTick

1. 语法：`this.$nextTick(回调函数)`
2. 作用：在下一次 DOM 更新结束后执行其指定的回调。
3. 什么时候用：当改变数据后，要基于更新后的新DOM进行某些操作时，要在nextTick所指定的回调函数中执行。

Vue封装的过度与动画

1. 作用：在插入、更新或移除 DOM元素时，在合适的时候给元素添加样式类名。
2. 图示：

3. 写法：
    1. 准备好样式：

- 元素进入的样式：
    1. v-enter：进入的起点
    2. v-enter-active：进入过程中
    3. v-enter-to：进入的终点
- 元素离开的样式：
    1. v-leave：离开的起点
    2. v-leave-active：离开过程中
    3. v-leave-to：离开的终点

2. 使用`<transition>`包裹要过度的元素，并配置name属性：

```
<transition name="hello">
    <h1 v-show="isShow">你好啊！</h1>
</transition>
```

3. 备注：若有多个元素需要过度，则需要使用：`<transition-group>`，且每个元素都要指定`key`值。

vue脚手架配置代理

方法一

在vue.config.js中添加如下配置：

```
devServer:{
  proxy:"http://localhost:5000"
}
```

说明：

1. 优点：配置简单，请求资源时直接发给前端（8080）即可。
2. 缺点：不能配置多个代理，不能灵活的控制请求是否走代理。
3. 工作方式：若按照上述配置代理，当请求了前端不存在的资源时，那么该请求会转发给服务器 （优先匹配前端资源）

方法二

编写vue.config.js配置具体代理规则：

```
module.exports = {
    devServer: {
      proxy: {
      '/api1': {// 匹配所有以 '/api1'开头的请求路径
        target: 'http://localhost:5000',// 代理目标的基础路径
        changeOrigin: true,
        pathRewrite: {'^/api1': ''}
      },
      '/api2': {// 匹配所有以 '/api2'开头的请求路径
        target: 'http://localhost:5001',// 代理目标的基础路径
        changeOrigin: true,
```

```
        pathRewrite: {'^/api2': ''}
      }
    }
  }
}
/*
   changeOrigin设置为true时，服务器收到的请求头中的host为：localhost:5000
   changeOrigin设置为false时，服务器收到的请求头中的host为：localhost:8080
   changeOrigin默认值为true
*/
```

说明：

1. 优点：可以配置多个代理，且可以灵活的控制请求是否走代理。
2. 缺点：配置略微繁琐，请求资源时必须加前缀。

插槽

1. 作用：让父组件可以向子组件指定位置插入html结构，也是一种组件间通信的方式，适用于 **父组件 ===> 子组件** 。
2. 分类：默认插槽、具名插槽、作用域插槽
3. 使用方式：
    1. 默认插槽：

```
父组件中：
        <Category>
           <div>html结构1</div>
        </Category>
子组件中：
        <template>
            <div>
               <!-- 定义插槽 -->
               <slot>插槽默认内容...</slot>
            </div>
        </template>
```

    2. 具名插槽：

```
父组件中：
        <Category>
            <template slot="center">
              <div>html结构1</div>
            </template>

            <template v-slot:footer>
               <div>html结构2</div>
            </template>
        </Category>

子组件中：
```

```
        <template>
            <div>
               <!-- 定义插槽 -->
               <slot name="center">插槽默认内容...</slot>
               <slot name="footer">插槽默认内容...</slot>
            </div>
        </template>
```

3. 作用域插槽：

    1. 理解：数据在组件的自身，但根据数据生成的结构需要组件的使用者来决定。（games数据在Category组件中，但使用数据所遍历出来的结构由App组件决定）

    2. 具体编码：

```
父组件中：
        <Category>
            <template scope="scopeData">
                <!-- 生成的是ul列表 -->
                <ul>
                    <li v-for="g in scopeData.games" :key="g">{{g}}</li>
                </ul>
            </template>
        </Category>

        <Category>
            <template slot-scope="scopeData">
                <!-- 生成的是h4标题 -->
                <h4 v-for="g in scopeData.games" :key="g">{{g}}</h4>
            </template>
        </Category>
子组件中：
        <template>
            <div>
                <slot :games="games"></slot>
            </div>
        </template>

        <script>
            export default {
                name:'Category',
                props:['title'],
                //数据在子组件自身
                data() {
                    return {
                        games:['红色警戒','穿越火线','劲舞团','超级玛丽']
                    }
                },
            }
        </script>
```

Vuex

1.概念

在Vue中实现集中式状态（数据）管理的一个Vue插件，对vue应用中多个组件的共享状态进行集中式的管理（读/写），也是一种组件间通信的方式，且适用于任意组件间通信。

2.何时使用?

多个组件需要共享数据时

3.搭建vuex环境

1. 创建文件：`src/store/index.js`

```
//引入Vue核心库
import Vue from 'vue'
//引入Vuex
import Vuex from 'vuex'
//应用Vuex插件
Vue.use(Vuex)

//准备actions对象—响应组件中用户的动作
const actions = {}
//准备mutations对象—修改state中的数据
const mutations = {}
//准备state对象—保存具体的数据
const state = {}

//创建并暴露store
export default new Vuex.Store({
    actions,
    mutations,
    state
})
```

2. 在`main.js`中创建vm时传入`store`配置项

```
......
//引入store
import store from './store'
......

//创建vm
new Vue({
    el:'#app',
    render: h => h(App),
    store
})
```

4.基本使用

1. 初始化数据、配置`actions`、配置`mutations`，操作文件`store.js`

```
//引入Vue核心库
import Vue from 'vue'
//引入Vuex
import Vuex from 'vuex'
//引用Vuex
Vue.use(Vuex)

const actions = {
    //响应组件中加的动作
    jia(context,value){
        // console.log('actions中的jia被调用了',miniStore,value)
        context.commit('JIA',value)
    },
}

const mutations = {
    //执行加
    JIA(state,value){
        // console.log('mutations中的JIA被调用了',state,value)
        state.sum += value
    }
}

//初始化数据
const state = {
   sum:0
}

//创建并暴露store
export default new Vuex.Store({
    actions,
    mutations,
    state,
})
```

2. 组件中读取vuex中的数据：`$store.state.sum`
3. 组件中修改vuex中的数据：`$store.dispatch('action中的方法名',数据)` 或 `$store.commit('mutations中的方法名',数据)`

> 备注：若没有网络请求或其他业务逻辑，组件中也可以越过actions，即不写`dispatch`，直接编写`commit`

5.getters的使用

1. 概念：当state中的数据需要经过加工后再使用时，可以使用getters加工。
2. 在`store.js`中追加`getters`配置

```
......

const getters = {
    bigSum(state){
        return state.sum * 10
    }
}

//创建并暴露store
export default new Vuex.Store({
    ......
    getters
})
```

3. 组件中读取数据：`$store.getters.bigSum`

6.四个map方法的使用

1. **mapState方法：**用于帮助我们映射`state`中的数据为计算属性

```
computed: {
    //借助mapState生成计算属性：sum、school、subject（对象写法）
     ...mapState({sum:'sum',school:'school',subject:'subject'}),

    //借助mapState生成计算属性：sum、school、subject（数组写法）
    ...mapState(['sum','school','subject']),
},
```

2. **mapGetters方法：**用于帮助我们映射`getters`中的数据为计算属性

```
computed: {
    //借助mapGetters生成计算属性：bigSum（对象写法）
    ...mapGetters({bigSum:'bigSum'}),

    //借助mapGetters生成计算属性：bigSum（数组写法）
    ...mapGetters(['bigSum'])
},
```

3. **mapActions方法：**用于帮助我们生成与`actions`对话的方法，即：包含`$store.dispatch(xxx)`的函数

```
methods:{
    //靠mapActions生成：incrementOdd、incrementWait（对象形式）
    ...mapActions({incrementOdd:'jiaOdd',incrementWait:'jiaWait'})

    //靠mapActions生成：incrementOdd、incrementWait（数组形式）
    ...mapActions(['jiaOdd','jiaWait'])
}
```

4. **mapMutations方法：**用于帮助我们生成与`mutations`对话的方法，即：包含`$store.commit(xxx)`的函数

```
methods:{
    //靠mapActions生成：increment、decrement（对象形式）
    ...mapMutations({increment:'JIA',decrement:'JIAN'}),

    //靠mapMutations生成：JIA、JIAN（对象形式）
    ...mapMutations(['JIA','JIAN']),
}
```

> 备注：mapActions与mapMutations使用时，若需要传递参数需要：在模板中绑定事件时传递好参数，否则参数是事件对象。

7.模块化+命名空间

1. 目的：让代码更好维护，让多种数据分类更加明确。
2. 修改 `store.js`

```
const countAbout = {
  namespaced:true,//开启命名空间
  state:{x:1},
  mutations: { ... },
  actions: { ... },
  getters: {
    bigSum(state){
       return state.sum * 10
    }
  }
}

const personAbout = {
  namespaced:true,//开启命名空间
  state:{ ... },
  mutations: { ... },
  actions: { ... }
}

const store = new Vuex.Store({
  modules: {
    countAbout,
    personAbout
  }
})
```

3. 开启命名空间后，组件中读取state数据：

```
//方式一：自己直接读取
this.$store.state.personAbout.list
//方式二：借助mapState读取：
...mapState('countAbout',['sum','school','subject']),
```

4. 开启命名空间后，组件中读取getters数据：

```
//方式一：自己直接读取
this.$store.getters['personAbout/firstPersonName']
//方式二：借助mapGetters读取：
...mapGetters('countAbout',['bigSum'])
```

5. 开启命名空间后，组件中调用dispatch

```
//方式一：自己直接dispatch
this.$store.dispatch('personAbout/addPersonWang',person)
//方式二：借助mapActions：
...mapActions('countAbout',{incrementOdd:'jiaOdd',incrementWait:'jiaWait'})
```

6. 开启命名空间后，组件中调用commit

```
//方式一：自己直接commit
this.$store.commit('personAbout/ADD_PERSON',person)
//方式二：借助mapMutations：
...mapMutations('countAbout',{increment:'JIA',decrement:'JIAN'}),
```

路由

1. 理解： 一个路由（route）就是一组映射关系（key - value），多个路由需要路由器（router）进行管理。
2. 前端路由：key是路径，value是组件。

1.基本使用

1. 安装vue-router，命令：`npm i vue-router`
2. 应用插件：`Vue.use(VueRouter)`
3. 编写router配置项:

```
//引入VueRouter
import VueRouter from 'vue-router'
//引入Luyou 组件
import About from '../components/About'
import Home from '../components/Home'

//创建router实例对象，去管理一组一组的路由规则
const router = new VueRouter({
    routes:[
        {
            path:'/about',
            component:About
        },
        {
            path:'/home',
            component:Home
```

```
        }
    ]
})

//暴露router
export default router
```

4. 实现切换（active-class可配置高亮样式）

```
<router-link active-class="active" to="/about">About</router-link>
```

5. 指定展示位置

```
<router-view></router-view>
```

2.几个注意点

1. 路由组件通常存放在`pages`文件夹，一般组件通常存放在`components`文件夹。
2. 通过切换，“隐藏”了的路由组件，默认是被销毁掉的，需要的时候再去挂载。
3. 每个组件都有自己的`$route`属性，里面存储着自己的路由信息。
4. 整个应用只有一个router，可以通过组件的`$router`属性获取到。

3.多级路由（多级路由）

1. 配置路由规则，使用children配置项：

```
routes:[
    {
        path:'/about',
        component:About,
    },
    {
        path:'/home',
        component:Home,
        children:[ //通过children配置子级路由
            {
                path:'news', //此处一定不要写：/news
                component:News
            },
            {
                path:'message',//此处一定不要写：/message
                component:Message
            }
        ]
    }
]
```

2. 跳转（要写完整路径）：

```
<router-link to="/home/news">News</router-link>
```

4.路由的query参数

1. 传递参数

```
<!-- 跳转并携带query参数，to的字符串写法 -->
<router-link :to="/home/message/detail?id=666&title=你好">跳转</router-link>

<!-- 跳转并携带query参数，to的对象写法 -->
<router-link
    :to="{
        path:'/home/message/detail',
        query:{
           id:666,
            title:'你好'
        }
    }"
>跳转</router-link>
```

2. 接收参数：

```
$route.query.id
$route.query.title
```

5.命名路由

1. 作用：可以简化路由的跳转。
2. 如何使用
    1. 给路由命名：

```
{
    path:'/demo',
    component:Demo,
    children:[
        {
            path:'test',
            component:Test,
            children:[
                {
                      name:'hello' //给路由命名
                    path:'welcome',
                    component:Hello,
                }
            ]
        }
    ]
}
```

2. 简化跳转：

```
<!--简化前，需要写完整的路径 -->
<router-link to="/demo/test/welcome">跳转</router-link>

<!--简化后，直接通过名字跳转 -->
<router-link :to="{name:'hello'}">跳转</router-link>

<!--简化写法配合传递参数 -->
<router-link
    :to="{
        name:'hello',
        query:{
           id:666,
            title:'你好'
        }
    }"
>跳转</router-link>
```

6.路由的params参数

1. 配置路由，声明接收params参数

```
{
    path:'/home',
    component:Home,
    children:[
        {
            path:'news',
            component:News
        },
        {
            component:Message,
            children:[
                {
                    name:'xiangqing',
                    path:'detail/:id/:title', //使用占位符声明接收params参数
                    component:Detail
                }
            ]
        }
    ]
}
```

2. 传递参数

```vue
<!-- 跳转并携带params参数，to的字符串写法 -->
<router-link :to="/home/message/detail/666/你好">跳转</router-link>

<!-- 跳转并携带params参数，to的对象写法 -->
<router-link
    :to="{
        name:'xiangqing',
        params:{
           id:666,
            title:'你好'
        }
    }"
>跳转</router-link>
```

> 特别注意：路由携带params参数时，若使用to的对象写法，则不能使用path配置项，必须使用name配置！

3. 接收参数：

```vue
$route.params.id
$route.params.title
```

7.路由的props配置

作用：让路由组件更方便的收到参数

```vue
{
    name:'xiangqing',
    path:'detail/:id',
    component:Detail,

    //第一种写法：props值为对象，该对象中所有的key-value的组合最终都会通过props传给Detail组件
    // props:{a:900}

    //第二种写法：props值为布尔值，布尔值为true，则把路由收到的所有params参数通过props传给Detail组件
    // props:true

    //第三种写法：props值为函数，该函数返回的对象中每一组key-value都会通过props传给Detail组件
    props(route){
        return {
            id:route.query.id,
            title:route.query.title
        }
    }
}
```

8. `<router-link>` 的replace属性

1. 作用：控制路由跳转时操作浏览器历史记录的模式

2. 浏览器的历史记录有两种写入方式：分别为`push`和`replace`，`push`是追加历史记录，`replace`是替换当前记录。路由跳转时候默认为`push`
3. 如何开启`replace`模式：`<router-link replace .....>News</router-link>`

9.编程式路由导航

1. 作用：不借助`<router-link>`实现路由跳转，让路由跳转更加灵活
2. 具体编码：

```
//$router的两个API
this.$router.push({
    name:'xiangqing',
        params:{
            id:xxx,
            title:xxx
        }
})

this.$router.replace({
    name:'xiangqing',
        params:{
            id:xxx,
            title:xxx
        }
})
this.$router.forward() //前进
this.$router.back() //后退
this.$router.go() //可前进也可后退
```

10.缓存路由组件

1. 作用：让不展示的路由组件保持挂载，不被销毁。
2. 具体编码：

```
<keep-alive include="News">
    <router-view></router-view>
</keep-alive>
```

11.两个新的生命周期钩子

1. 作用：路由组件所独有的两个钩子，用于捕获路由组件的激活状态。
2. 具体名字：
    1. `activated`路由组件被激活时触发。
    2. `deactivated`路由组件失活时触发。

12.路由守卫

1. 作用：对路由进行权限控制

2. 分类：全局守卫、独享守卫、组件内守卫

3. 全局守卫:

```
//全局前置守卫：初始化时执行、每次路由切换前执行
router.beforeEach((to,from,next)=>{
    console.log('beforeEach',to,from)
    if(to.meta.isAuth){ //判断当前路由是否需要进行权限控制
        if(localStorage.getItem('school') === 'atguigu'){ //权限控制的具体规则
            next() //放行
        }else{
            alert('暂无权限查看')
            // next({name:'guanyu'})
        }
    }else{
        next() //放行
    }
})

//全局后置守卫：初始化时执行、每次路由切换后执行
router.afterEach((to,from)=>{
    console.log('afterEach',to,from)
    if(to.meta.title){
        document.title = to.meta.title //修改网页的title
    }else{
        document.title = 'vue_test'
    }
})
```

4. 独享守卫:

```
beforeEnter(to,from,next){
    console.log('beforeEnter',to,from)
    if(to.meta.isAuth){ //判断当前路由是否需要进行权限控制
        if(localStorage.getItem('school') === 'atguigu'){
            next()
        }else{
            alert('暂无权限查看')
            // next({name:'guanyu'})
        }
    }else{
        next()
    }
}
```

5. 组件内守卫：

```
//进入守卫：通过路由规则，进入该组件时被调用
beforeRouteEnter (to, from, next) {
},
//离开守卫：通过路由规则，离开该组件时被调用
beforeRouteLeave (to, from, next) {
}
```

13.路由器的两种工作模式

1. 对于一个url来说，什么是hash值？—— #及其后面的内容就是hash值。
2. hash值不会包含在 HTTP 请求中，即：hash值不会带给服务器。
3. hash模式：
    1. 地址中永远带着#号，不美观 。
    2. 若以后将地址通过第三方手机app分享，若app校验严格，则地址会被标记为不合法。
    3. 兼容性较好。
4. history模式：
    1. 地址干净，美观 。
    2. 兼容性和hash模式相比略差。
    3. 应用部署上线时需要后端人员支持，解决刷新页面服务端404的问题。