在线 Markdown 编译器

Feedback

```
            <ul>
                <router-link v-for="(item, index) in products"
                    :to="{ path: item.path }"
                    tag="li" active-class="active"
                    :key="index"
                >
                    {{item.name}}
                </router-link>
            </ul>
        </div>

    active-class="active" 表示点击到对应的li列表时，加入active这个类
    如下面是类的样式：
    .product-board li.active,
    .product-board li:hover {
        background: #4fc08d;
        color: #fff;
    }


### 模拟从后端取数据

- json-server

        用于模拟从后端取数据，用json格式的数据进行模拟

        安装： npm install json-server --save
```

HTML

实时预览

```
<h2>安装 vue-cli</h2>

<pre><code>npm i vue-cli -g
使用：
    vue init webpack-simple my-project
        进入my-project目录
    npm install
    npm run dev

vue list 可以查看 vue-cli 提供的项目模板
</code></pre>

<h2>vue 中的命令</h2>

<pre><code>&lt;a :link="url"&gt;taobao&lt;/a&gt;
&lt;a link="url"&gt;taobao&lt;/a&gt;

上面的是由v-blind命令绑定的，下面的是原生的link,所以上面u是一个变量定义在vue data中，
下面的是字符串这个url要改成真正的url,如：http://www.taobao.com
其他的命令也一样，如：
&lt;p v-html="msg"&gt;&lt;/p&gt; 中的 msg 也是变量
</code></pre>

<h2>computed 计算属性</h2>

<pre><code>computed 里的数据和 data 里的数据是一样的，只是computed里的数据会根据内部的值而更新自己的值，如
computed:{
    computedValue () {
        return this.dataValue + 10;
    }
}
dataValue 为 data里定义的数据
computed 是 根据 dataValue的变化重新计算 computedValue的值，这个和watch 监听正好相反。
</code></pre>
```

分享到

实时预览：

安装 vue-cli

```
npm i vue-cli -g
使用：
    vue init webpack-simple my-project
        进入my-project目录
    npm install
    npm run dev

vue list 可以查看 vue-cli 提供的项目模板
```

vue 中的命令

```
<a :link="url">taobao</a>
<a link="url">taobao</a>

上面的是由v-blind命令绑定的，下面是原生的link,所以上面url是一个变量定义在vue data中，
下面的是字符串这个url要改成真正的url,如：http://www.taobao.com
其他的命令也一样，如：
<p v-html="msg"></p> 中的 msg 也是变量
```

computed 计算属性

```
computed 里的数据和 data 里的数据是一样的，只是computed里的数据会根据内部的值而更新自己的值，如
computed:{
    computedValue () {
        return this.dataValue + 10;
    }
}
dataValue 为 data里定义的数据
computed 是 根据 dataValue的变化重新计算 computedValue的值，这个和watch 监听正好相反。
```

watch 监听事件

```
    watch: {
        watchValue: function (val, oldVal) {
            console.log(val, oldVal);
        },

        example2:{

            //注意：当观察的数据为对象或数组时，curVal和oldVal是相等的，因为这两个形参指向的是同一个数据对象
            handler(curVal,oldVal){
                conosle.log(curVal,oldVal)
            },
            deep:true
        }
    }
```

这个是监听watchValue的变化，如果这个值变化了，就运行右边的函数，输出当前的值和变化之前的值，这个函数接收的两个参数是默认转递过来的。

v-if 和 v-show 区别

v-if如果条件为false时，会将当前DOM节点删除，而v-show不会删除，只会将display设置为none,让其隐藏。

```
<p v-show="isShow">  show </p>

<p v-if="isShow1"> show1 </p>
```

组件 生命周期

```
之前 1.xx:
    init
    created
    beforeCompile
    compiled
    ready       √   ->     mounted
    beforeDestroy
    destroyed

现在 2.xx:
    beforeCreate     组件实例刚刚被创建,属性都没有
    created 实例已经创建完成，属性已经绑定
    beforeMount 模板编译之前
    mounted 模板编译之后，代替之前ready  *
    beforeUpdate     组件更新之前
    updated 组件更新完毕  *
    beforeDestroy    组件销毁前
    destroyed    组件销毁后

    ================================================

    <!DOCTYPE html>
    <html lang="zh-CN">
    <head>
        <meta charset="UTF-8">
        <title>vue2.0 开始学习</title>
    </head>
    <body>
        <div id="box">
            <input type="button" value="会被更新的数据" @click="update">
            <input type="button" value="会被销毁的组件" @click="destroy">
            {{ msg }}
        </div>
    <script src="vue2.0.js"></script>
    <script>
        window.onload = function () {

            new Vue({
                el: '#box',
                data: {
                    msg: 'welcome to vue2.0'
                },
                methods: {
                    update(){
                        // 更新组件内的数据
                        this.msg = "更新页面数据"
                    },
                    destroy(){
                        // 销毁组件
                        this.$destroy();
                    }
                },

                // 组件完全显示在页面的时候，会依次触发下面4个事件
                beforeCreate(){
                    console.log('组件实例刚刚被创建');
                },
                created(){
                    console.log('实例已经创建完成');
                },
                beforeMount(){
                    console.log('模板编译之前');
                },
                mounted(){
                    console.log('模板已经编译完成')
                },
                // 更改组件的内容，会触发下面2个事件
                beforeUpdate(){
```

```
                console.log('组件更新之前');
            },
            updated(){
                console.log('组件更新完成');
            },
            // 销毁组件会触发下面2个事件
            beforeDestroy(){
                console.log('组件销毁之前')
            },
            destroyed(){
                console.log('组件销毁之后')
            }
        });
    };
</script>
</body>
</html>
```


组件间通信

A组件发出信号：

```
import { eventBus } from '../../eventBus'

    // 点击最外部组件 处理程序
    resetSelect() {
        // 发送全局事件，监听有这个事件的组件会作出相应的反应
        eventBus.$emit('reset-component');
    }
```

B组件监听：

```
import { eventBus } from '../../eventBus'

mounted() {
    eventBus.$on('reset-component', () => {
        this.isDrop = false;
    })
},
```

监听到A发出事件时，就作出对应处理，其中eventBus.js数据就是一个全局的vue：

```
import Vue from 'vue'
// 用作--发送--公共事件
const eventBus = new Vue();

export {eventBus }
```

父子组件通信

子组件接收数用props

```
    props: {
        isShow: {
            type: Boolean,
            default: false
        }
    },
```

父组件传递方法： 是用属性邦定：

```
    <my-dialog :is-show="isShowLogDialog" @on-close="closeDialog('isShowLogDialog')">
        <log-form @has-log="onSuccessLog"></log-form>
    </my-dialog>
```

同时父组件监听子组件的事件 on-close 当子组件运行下面函数：

```
       closeMyself () {
            this.$emit('on-close', "传递给父组件的数据也可以没有")
        }
```

发送事件时，父组件就会接收到，并且作出对应的处理，同时子组件也可以传递数据。

slot分发内容

父组件中的内容

```
 // game 为子组件
<game>
   <h1 slot="k1">王者荣耀</h1>
   <h2 slot="k2">王者荣耀</h2>
   <p>小学生要背锅</p>
</game>
```

子组件中的slot有name属性，与父组件的slot的值相对应，那么久会匹配到。没有匹配到的先会显示在默认的slot中。

```
// 子组件内容
```

```
<template>
   <div>
      <h4>游戏</h4>
      <slot name="k1"></slot>
      <slot name="k2"></slot>
      <slot></slot>
   </div>
</template>

结果：
    游戏
    王者荣耀
    王者荣耀
    小学生要背锅
```

$refs vue操作DOM方法

```
<div ref="wrapper">
    <slot></slot>
 </div>

获取上面的DIV节点： this.$refs.wrapper
```

路由和路由跳转

安装

```
npm install vue-router --save
```

路由形式

```
<router-link tag="div" class="tab-item" to="/recommend">
    <span class="tab-link">推荐</span>
</router-link>

路由默认是 a 标签，tag是重新设置路由的标签
路由展示： <router-view></router-view>
```

软件自动跳转

```
toOrderList () {
    this.$router.push({path: '/orderList'});
    或者将当前路由替换成其他的路由，如：
    this.$router.replace({path: '/otherList'});
}
```

点击跳转

```
<ul>
    <router-link v-for="(item, index) in products"
        :to="{ path: item.path }"
        tag="li" active-class="active"
        :key="index"
    >
        {{ item.name }}
    </router-link>
</ul>

active-class="active" 表示点击到对应的li列表时，加入active这个类似
```

访问路由配置中的参数

```
路由配置如下：
{
        path: '/user',
        component: User,
        children: [
            {path: ':username/age/:age', component: UserDetail}
        ],
}
要访问username,age两个参数，可以通过： this.$route.params.xxxx来访问，如：

        getMessage () {
            this.username = this.$route.params.username

            this.age =  this.$route.params.age

        }
```

路由重定向

```
    {
        path: '/',
        redirect: '/hello'  // 重定向到路由 hello

    },
    {
        path: '/hello',
        component: HelloWorld
    }
```

router-active-class

```
点击跳转到对应的路由时，会加载这个类
```

路由跳转回到页面顶部

```
在router/index.js 文件下修改Router内部即可：

export default new Router({
mode: 'history',
routes: [
        {
            path: '/',
            name: 'HomePage',
            component: HomePage
        },
        {
            path: '/product/:id',
            component: ProductDetail
        }
    ],
    // 让路由跳转到顶部
    scrollBehavior (to, from, savedPosition) {
        return { x: 0, y: 0 }
    }
})
```

Vuex

```
先安装： npm install vuex --save

Vuex中接收外部参数的方法
// 可以异步操作
const actions = {
    fetchProductList ({commit}) {
        commit('updateProductList', products);
    },
    // 获取对应ID的产品，ID为外部传递过来
    getProductOfId ({commit}, id) {
        commit('updateProductList', products);
        commit('updateProductOfId', id);
    }

};

调用：
    方法一：
created(){
        // vuex 中传递参数方法 调用actions中的方法
        this.$store.dispatch('getProductOfId',
            parseInt(this.$route.params.id));

        // 调用mutations中的方法
        this.$store.commit("setProductId", "12")
    },


    方法二：

    import {mapMutations, mapActions} from 'vuex'

      methods: {

          ...mapMutations({
            setSinger: 'getProductOfId'
          }),
          ...mapActions({
                "setProductId"
            })

        或者不要别名：
```

```
        ...mapMutations({
            'getProductOfId'
          }),
            或者将大括号{}改为中括号[]，上面用大括号是用于重命名
            ...mapMutations([
            'getProductOfId'
          ]),

        其他函数直接调用：
        created() {
            getProductOfId(11);
        }
      },
```

这里需要注意的是：

mutations 中的方法是不分组件的 ，假如你在 a.js 文件中的定义了 switch_dialog 方法 ，在其他文件中定义另一个 switch_dialog 方法 ，那么 $store.commit('switch_dialog') 会执行所有的 switch_dialog 方法。
mutations里的操作必须是同步的。

You may have an infinite update loop in a component render function

在vue中报这个错误可能是

- 直接在 template 里操作vue里的data 变量，可以把操作变量放到函数中
- 在v-for中对data里的变量进行++操作，可以把data里的变量放到export default外面进行声明

失去焦点事件

```
@blur="checkTitle"
```

webpack跨域设置

用webpack-simple模板的vue项目设置方法

在webpack.config.js中进行配置即可。

在module.exports 中配置

```
devServer: {
      historyApiFallback: true,
      noInfo: true,

    // 设置跨域相关
    proxy: {
      // 请求到 '/api' 下 的请求都会被代理到 target：http://localhost:8002 中
      '/api': {
          target: 'http://localhost:8002',
          secure: false, // 接受 运行在 https 上的服务
          changeOrigin: true
      }
    }
    },
```

其他的配置都不变。

用webpack模板的vue项目设置方法

这个网上有，可以参考，因为没用过webpack模板的vue进行跨域开发，所以没有配置过。

vue-lazyload vue中的图片懒加载插件

vue 中的 event的用法

函数如果前面有参数的话，event这个参数是放在最后的。

```
/* ... */
{
  methods: {
    submit: function (msg, e) {
      e.stopPropagation()
        console.log(msg+e)
    },
    show: function(event) {
        event.cancelBubble = true;   //这种就阻止了事件冒泡
         console.log(event);   //event 这个就是事件对象了
    }
  }
}
```

方法中传入$event即可，如

```
<button @click="submit('hello!', $event)">Submit</button>

div id="box">
```

```
    <input type="button" value="按钮" @click="show($event)">
</div>
```

vue 中class 相关用法

```
    <div
        class="index-board-item"
        v-for="(item, index) in boardList"
        :class="[{'line-last' : index % 2 !== 0},
                {'line-bottom' : index >= (boardList.length -2)},
                 'index-board-' + item.id]"
    </div>
```

或者router-link中

```
    <div class="product-board">
        <img :src="productIcon">
        <ul>
            <router-link v-for="(item, index) in products"
                :to="{ path: item.path }"
                tag="li" active-class="active"
                :key="index"
            >
                {{item.name}}
            </router-link>
        </ul>
    </div>
```

active-class="active" 表示点击到对应的li列表时，加入active这个类
如下面是类的样式：

```
.product-board li.active,
.product-board li:hover {
    background: #4fc08d;
    color: #fff;
}
```

模拟从后端取数据

json-server

用于模拟从后端取数据，用json格式的数据进行模拟

安装： npm install json-server --save

对于webpack框架，修改dev-server.js文件

body-parser

安装： npm install body-parser --save

对于webpack框架，修改dev-server.js文件

04：Vue2.0 实战二 电商项目（2017年3月最新） 项目中有说明