

单文件组件

Vue CLI3

基本配置

快速原型开发

创建项目

购物车

mock数据

数据持久化

组件深入

组件分类

使用第三方组件

组件设计

Input.vue

FormItem

Form

单文件组件

在很多Vue项目中,我们使用 `Vue.component` 来定义全局组件,紧接着用 `new Vue({ el: '#app '})` 在每个页面内指定一个容器元素。

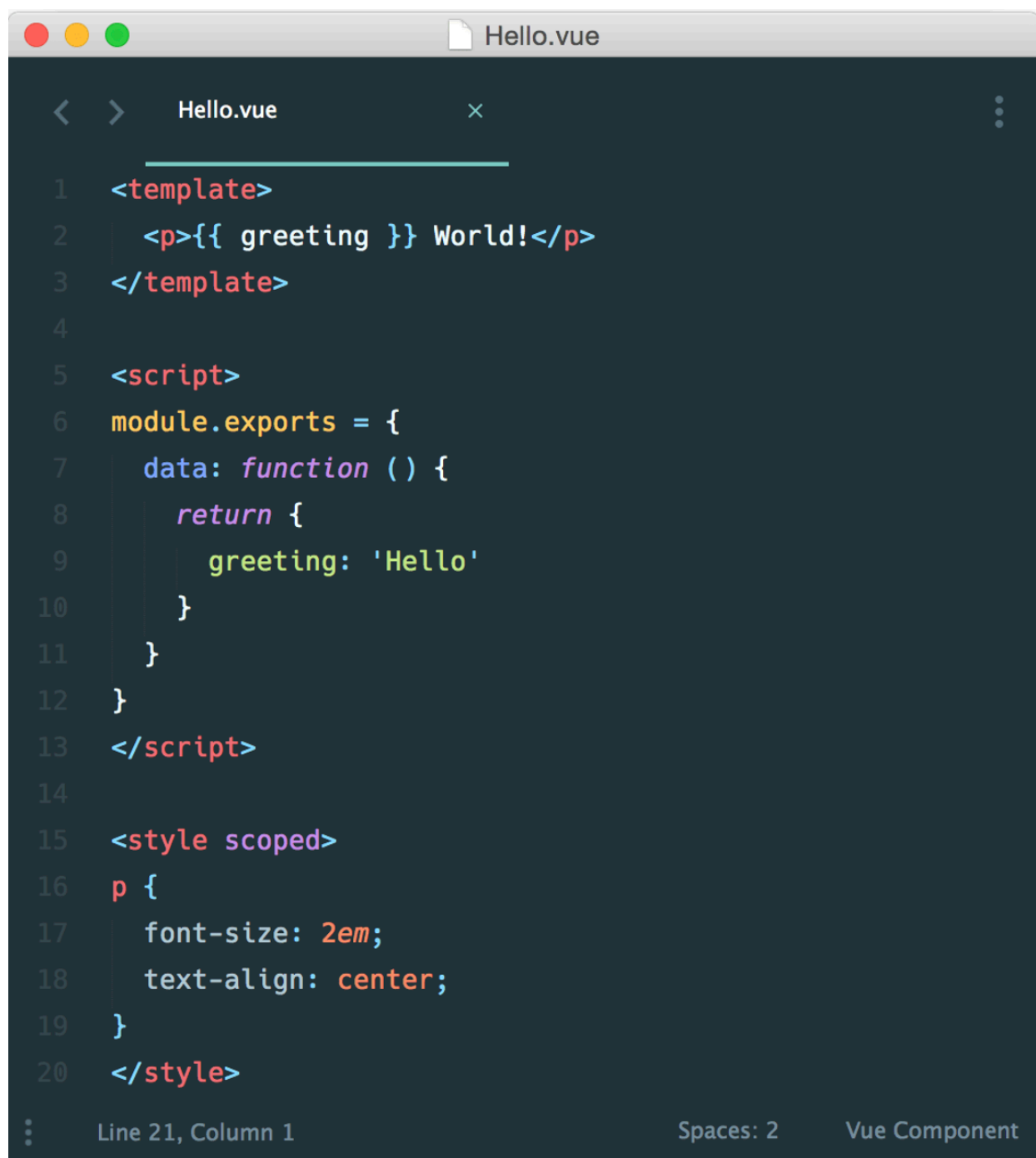
这种方式在很多中小规模的项目中运作的很好,在这些项目里 JavaScript 只被用来加强特定的视图。但当在更复杂的项目中,或者你的前端完全由 JavaScript 驱动的时候,下面这些缺点将变得非常明显:

- **全局定义** 强制要求每个 component 中的命名不得重复
- **字符串模板** 缺乏语法高亮, 在 HTML 有多行的时候, 需要用到丑陋的 `\`

- 不支持 CSS 意味着当 HTML 和 JavaScript 组件化时, CSS 明显被遗漏
- 没有构建步骤 限制只能使用 HTML 和 ES5 JavaScript, 而不能使用预处理器, 如 Pug (formerly Jade) 和 Babel

文件扩展名为 `.vue` 的 **single-file components(单文件组件)** 为以上所有问题提供了解决方法, 并且还可以使用 webpack 或 Browserify 等构建工具。

这是一个文件名为 `Hello.vue` 的简单实例:



```
1 <template>
2   <p>{{ greeting }} World!</p>
3 </template>
4
5 <script>
6   module.exports = {
7     data: function () {
8       return {
9         greeting: 'Hello'
10      }
11    }
12  }
13 </script>
14
15 <style scoped>
16   p {
17     font-size: 2em;
18     text-align: center;
19   }
20 </style>
```

Line 21, Column 1 Spaces: 2 Vue Component

现在我们获得

- [完整语法高亮](#)
- [CommonJS 模块](#)
- [组件作用域的 CSS](#)

在看完上文之后,建议使用官方提供的 [Vue CLI 3](#) 脚手架来开发工具,只要遵循提示,就能很快地运行一个带有 `.vue` 组件,ES2015,webpack和热重载的Vue项目

Vue CLI3

基本配置

- [安装Nodejs](#)
 - 保证Node.js8.9或更高版本
 - 终端中输入 `node -v`,保证已安装成功
- 安装[淘宝镜像源](#)
 - `npm install -g cnpm --registry=https://registry.npm.taobao.org`
 - 以后的npm可以用cnpm代替
- 安装Vue Cli3脚手架
 - `cnpm install -g @vue/cli`
- 检查其版本是否正确
 - `vue --version`

快速原型开发

使用 `vue serve` 和 `vue build` 命令对单个 `*.vue` 文件进行快速原型开发,不过这需要先额外安装一个全局的扩展:

```
1 | npm install -g @vue/cli-service-global
```

`vue serve` 的缺点就是它需要安装全局依赖，这使得它在不同机器上的一致性不能得到保证。因此这只适用于快速原型开发。

需要的仅仅是一个 `App.vue` 文件：

```
1 <template>
2   <div>
3     <h2>hello world 单页面组件</h2>
4   </div>
5 </template>
6 <script>
7   export default {
8
9   }
10 </script>
11 <style>
12
13 </style>
```

然后在这个 `App.vue` 文件所在的目录下运行：

```
1 vue serve
```

启动效果：

```
DONE Compiled successfully in 2790ms
```

```
App running at:
```

- Local: <http://localhost:8080/>
- Network: <http://192.168.11.7:8080/>

```
Note that the development build is not optimized.
To create a production build, run npm run build.
```



网页效果:

← → ↻ 🏠 ⓘ localhost:8080

hello world 单页面组件

但这种方式仅限于快速原型开发,终归揭底还是使用vue cli3来启动项目

创建项目

```
1 | vue create mysite
```

详细的看[官网介绍](#)

购物车

App.vue

```
1 <ul>
2   <li v-for="(item, index) in cartList" :key="index">
3     <h3>{{item.title}}</h3>
4     <p>¥{{item.price}}</p>
5     <button @click='addCart(index)'+>加购物
      车</button>
6   </li>
7 </ul>
```

```
1 cartList: [
2   {
3     id:1,
4     title:'web全栈开发',
5     price:1999
6   },
7   {
8     id: 2,
9     title: 'python全栈开发',
10    price: 2999
11  }
12 ],
```

新建Cart.vue购物车组件

```
1 <template>
2   <div>
3     <table border='1'>
4       <tr>
5         <th>#</th>
6         <th>课程</th>
7         <th>单价</th>
8         <th>数量</th>
9         <th>价格</th>
```

```
10         </tr>
11         <tr v-for="(c, index) in cart" :key="c.id"
12         :class='{active:c.active}'>
13             <td>
14                 <input type="checkbox" v-
15 model='c.active'>
16             </td>
17             <td>{{c.title}}</td>
18             <td>{{c.price}}</td>
19             <td>
20                 <button @click='subtract(index)'>-
21 </button>
22                 {{c.count}}
23                 <button @click='add(index)'>+
24 </button>
25             </td>
26             <td>¥{{c.price*c.count}}</td>
27         </tr>
28     </tr>
29 </table>
30 </div>
31 </template>
32 <script>
33     export default {
34         name: "Cart",
35         props: ['name', 'cart'],
36         methods: {
37             subtract(i) {
38                 let count = this.cart[i].count;
39                 // if(count > 1){
```

```
40         //      this.cart[i].count-=1
41         // }else{
42         //      this.remove(i)
43         // }
44         count > 1 ? this.cart[i].count -= 1 :
this.remove(i);
45     },
46     add(i) {
47         this.cart[i].count++;
48     },
49     remove(i) {
50         if (window.confirm('确定是否要删除')) {
51             this.cart.splice(i, 1);
52         }
53     }
54 },
55 data() {
56     return {}
57 },
58 created() {},
59 computed: {
60     activeCount() {
61         return this.cart.filter(v =>
v.active).length;
62     },
63     count() {
64         return this.cart.length;
65     },
66     total() {
67         // let num = 0;
68         // this.cart.forEach(c => {
69         //     if (c.active) {
70         //         num += c.price * c.count
71         //     }
72         // });
```



```

73         // return num;
74         return this.cart.reduce((sum, c) => {
75             if (c.active) {
76                 sum += c.price * c.count
77             }
78             return sum;
79         }, 0)
80     }
81 },
82
83 }
84 </script>
85 <style scoped>
86     .active {
87         color: red;
88     }
89 </style>

```

mock数据

简单的mock,使用自带的webpack-dev-server即可,新建vue.config.js
扩展webpack设置

[webpack官网介绍](#)

```

1  module.exports = {
2      configureWebpack:{
3          devServer:{
4              // mock数据模拟
5              before(app, server){
6                  app.get('/api/cartList',(req,res)=>{
7                      res.json([
8                          {
9                              id:1,

```

```

10         title: 'web全栈开发',
11         price: 1999
12     },
13     {
14         id: 2,
15         title: 'web全栈开发',
16         price: 2999
17     }
18 ]
19 })
20 }
21 }
22 }
23 }

```

访问<http://localhost:8080/api/cartList> 查看mock数据

使用axios获取接口数据 `npm install axios -S`

```

1  created() {
2      axios.get('/api/cartList').then(res=>{
3          this.cartList = res.data
4      })
5  }

```

使用ES7的async+await语法

```

1  async created() {
2      // try-catch解决async-awiat错误处理
3      try {
4          const { data } = await axios.get('/cartList')
5          this.cartList = data;
6
7      } catch (error) {
8          console.log(error);
9      }
10 },

```

数据持久化

localStorage+vue监听器

如果组件没有明显的父子关系,使用中央事件总线进行传递

Vue每个实例都有订阅/发布模式的额实现,使用\$on和\$emit

main.js

```

1  Vue.prototype.$bus = new Vue();

```

App.vue

```

1  methods: {
2      addCart(index) {
3          const good = this.cartList[index];
4          this.$bus.$emit('addGood',good);
5      }
6
7  }

```

Cart.vue

```
1 data() {
2     return {
3         cart: JSON.parse(localStorage.getItem('cart'))
4         || []
5     },
6     //数组和对象要深度监听
7     watch: {
8         cart: {
9             handler(n, o) {
10                 const total = n.reduce((total, c) => {
11                     total += c.count
12                     return total;
13                 }, 0)
14                 localStorage.setItem('total', total);
15                 localStorage.setItem('cart',
16                 JSON.stringify(n));
17                 this.$bus.$emit('add', total);
18             },
19             deep: true
20         },
21         created() {
22             this.$bus.$on('addGood', good => {
23                 const ret = this.cart.find(v => v.id ===
24                 good.id);
25                 if (ret) { //购物车已有数据
26                     ret.count += 1;
27                 } else {
28                     //购物车无数据
29                     this.cart.push({
30                         ...good,
31                         count: 1,
32                         active: true
```

```
32 |         })
33 |     }
34 | })
35 | },
```

更复杂的数据传递,可以使用vuex,后面课程会详细介绍

组件深入

组件分类

- 通用组件
 - 基础组件,大部分UI都是这种组件,比如表单 布局 弹窗等
- 业务组件
 - 与需求挂钩,会被复用,比如抽奖,摇一摇等
- 页面组件
 - 每个页面都是一个组件v

使用第三方组件

比如vue最流行的element,就是典型的通用组件,执行 `npm install element-ui` 安装

```

1 import Vue from 'vue';
2 import ElementUI from 'element-ui';
3 import 'element-ui/lib/theme-chalk/index.css';
4 import App from './App.vue';
5
6 Vue.use(ElementUI);
7
8 new Vue({
9   el: '#app',
10  render: h => h(App)
11 });

```

在vue-cli中可以使用vue add element 安装

安装之前注意提前提交当前工作内容,脚手架会覆盖若干文件

```

✓ All packages installed (5 packages installed from npm registry, used 833ms, speed 1
MacBookPro ~/Desktop/备课/Vue_study/single_comp/mysite $ vue add element

📦 Installing vue-cli-plugin-element...

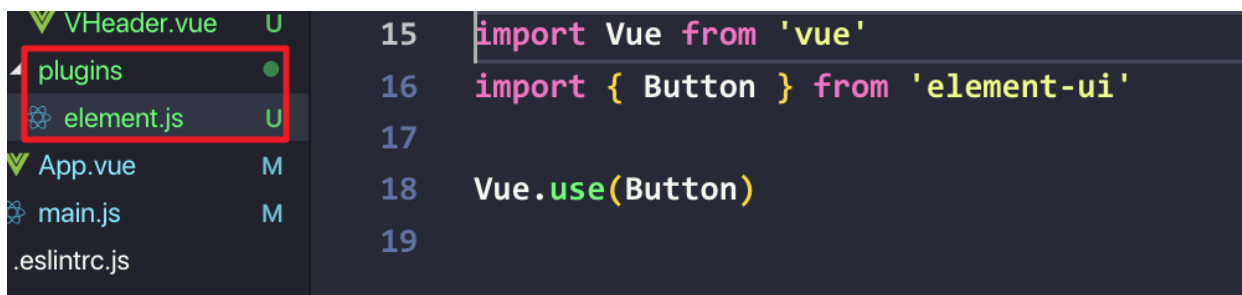
+ vue-cli-plugin-element@1.0.1
added 1 package from 1 contributor and removed 14 packages in 11.887s

✓ Successfully installed plugin: vue-cli-plugin-element

? How do you want to import Element?
  Fully import 完整导入
> Import on demand 部分导入

```

发现项目发生了变化,打开App.vue, `ctrl+z` 撤回



```

15 import Vue from 'vue'
16 import { Button } from 'element-ui'
17
18 Vue.use(Button)
19

```

此时可以在任意组件中使用 `<el-button>`

官网element-ui的通用组件,基本上都是复制粘贴使用,在这里就不一一赘述,后面项目中用到该库,咱们再一一去使用

关于组件设计,最重要的还是自己去设计组件,现在我们模仿element-ui提供的表单组件,手写实现表单组件 `m-form`

先看一下element-ui的表单

新建FormElement.vue

```
1 <template>
2   <div>
3     <h3>element表单</h3>
4     <el-form
5       :model="ruleForm"
6       status-icon
7       :rules="rules"
8       ref="ruleForm"
9       label-width="100px"
10      class="demo-ruleForm"
11    >
12      <el-form-item label="用户名" prop="name">
13        <el-input type="text" v-model="ruleForm.name"
14        autocomplete="off"></el-input>
15      </el-form-item>
16      <el-form-item label="确认密码" prop="pwd">
17        <el-input type="password" v-
18        model="ruleForm.pwd" autocomplete="off"></el-input>
19      </el-form-item>
20      <el-form-item>
21        <el-button type="primary"
22        @click="submitForm('ruleForm')">提交</el-button>
23      </el-form-item>
24    </div>
25  </template>
```

```
21     </el-form>
22   </div>
23 </template>
24
25 <script>
26 export default {
27   name: "FormElement",
28   data() {
29     return {
30       ruleForm: {
31         name: '',
32         pwd: ''
33       },
34       rules: {
35         name: [
36           {required: true, message: '请输入名称'},
37           {min: 6, max: 10, message: '请输入6~10位用
用户名'}
38         ],
39         pwd: [{require: true, message: '请输入密
码'}]],
40       }
41     }
42   },
43   methods: {
44     submitForm(name) {
45       this.$refs[name].validate(valid=>{
46         console.log(valid);
47         if(valid){
48           alert('验证成功,可以提交')
49         }else{
50           alert('error 提交');
51           return false;
52         }
53     }
```



```
54         })
55
56     }
57 },
58 };
59 </script>
```

在App.vue组件中导入该组件,挂载,使用

组件设计

表单组件,组件分层

1. Form负责定义校验规则
2. FormItem负责显示错误信息
3. Input负责数据双向绑定
4. 使用provide和inject内部共享数据

Form 绑定数据模型 添加校验规则

FormItem 绑定label prop 校验和显示错误

Input 实现双向数据绑定,
通知FormItem做校验

表单控件实现双向的数据绑定

Input.vue

```
1 <template>
2   <div>
3     <input :type="type" @input="handleInput"
4       :value="inputVal">
5   </div>
6 </template>
7 <script>
8 export default {
```

```
9     props: {
10       value: {
11         type: String,
12         default: ""
13       },
14       type: {
15         type: String,
16         default: "text"
17       }
18     },
19     data() {
20       return {
21         //单向数据流的原则:组件内不能修改props
22         inputVal: this.value
23       };
24     },
25     methods: {
26       handleInput(e) {
27         this.inputVal = e.target.value;
28         // 通知父组件值的更新
29         this.$emit("input", this.inputVal);
30       }
31     }
32   };
33 </script>
34
35 <style scoped>
36 </style>
```

FormElement.vue

```
1 如果不传type表示默认值,在Input.vue的props中有说明
2 <m-input v-model="ruleForm.name"></m-input>
3 <m-input v-model="ruleForm.name" type='password'></m-
  input>
```

```
1 //数据
2 data() {
3   return {
4     ruleForm: {
5       name: "",
6       pwd: ""
7     },
8     rules: {
9       name: [
10        { required: true, message: "请输入名称" },
11        { min: 6, max: 10, message: "请输入6~10位用户
    名" }
12      ],
13       pwd: [{ require: true, message: "请输入密码" }]
14     }
15   };
16 },
```

FormItem

1. 获取当前输入框的规则
2. 如果输入框和rule不匹配 显示错误信息
3. Input组件中用户输入内容时,通知FormItem做校验
4. 使用[async-validator](#)做出校验

FormItem.vue

```
1
```

```
2 <template>
3   <div>
4     <label v-if="label">{{label}}</label>
5     <slot></slot>
6     <!-- 校验的错误信息 -->
7     <p v-if="validateStatus=='error'" class="error">
      {{errorMessage}}</p>
8   </div>
9 </template>
10
11 <script>
12 import schema from "async-validator";
13 export default {
14   name: "FormItem",
15   data() {
16     return {
17       validateStatus: "",
18       errorMessage: ""
19     };
20   },
21   props: {
22     label: {
23       type: String,
24       default: ""
25     },
26     prop: {
27       type: String
28     }
29   }
30 };
31 </script>
32
33 <style scoped>
34 .error {
35   color: red;
```

```
36 | }  
37 </style>
```

FormElement.vue

```
1 <m-form-item label="用户名" prop="name">  
2   <m-input v-model="ruleForm.name"></m-input>  
3 </m-form-item>  
4 <m-form-item label="密码" prop="pwd">  
5   <m-input v-model="ruleForm.pwd" type="password"></m-  
   input>  
6 </m-form-item>
```

此时网页正常显示,但没有校验规则,添加校验规则

思路:比如对用户名进行校验,用户输入的用户名必须是6~10位

```
1 npm i async-validator -S
```

Input.vue

```
1 methods: {  
2   handleInput(e) {  
3     this.inputVal = e.target.value;  
4     //....  
5     //通知父组件校验,将输入框的值实时传进去  
6     this.$parent.$emit("validate", this.inputVal);  
7   }  
8 }
```

FormItem.vue

```
1 import schema from "async-validator";  
2 export default {
```

```
3   name: "FormItem",
4   data() {
5     return {
6       validateStatus: "",
7       errorMessage: ""
8     };
9   },
10  methods: {
11    validate(value) {//value为当前输入框的值
12      // 校验当前项:依赖async-validate
13      let descriptor = {};
14      descriptor[this.prop] =
15      this.form.rules[this.prop];
16      // const descriptor = { [this.prop]:
17      this.form.rules[this.prop] };
18      const validator = new schema(descriptor);
19
20      let obj = {};
21      obj[this.prop] = value;
22      // let obj =
23      {[this.prop]:this.form.model[this.prop]};
24      validator.validate(obj, errors => {
25        if (errors) {
26          this.validateStatus = "error";
27          this.errorMessage = errors[0].message;
28        } else {
29          this.validateStatus = "";
30          this.errorMessage = "";
31        }
32      });
33    },
34    created() {
35      //监听子组件Input的派发的validate事件
```

```

35     this.$on("validate", this.validate);
36   },
37   //注入名字 获取父组件Form 此时Form我们还没创建
38   inject: ["form"],
39   props: {
40     label: {
41       type: String,
42       default: ""
43     },
44     prop: {
45       type: String
46     }
47   }
48 };

```

Form

1. 声明props中获取数据模型(model)和检验规则(rules)
2. 当FormItem组件挂载完成时,通知Form组件开始缓存需要校验的表单项
3. 将缓存的表单项进行统一处理,如果有一个是错误,则返回false.(思路:使用 `promise.all()` 进行处理)
4. 声明校验方法,供父级组件方法调用validate()方法

Form.vue

声明props中获取数据模型(model)和检验规则(rules)

```

1  <template>
2    <div>
3      <slot></slot>
4    </div>
5  </template>
6

```



```

7 <script>
8   export default {
9     name: 'Form',
10    //依赖
11    provide(){
12      return {
13        // 将表单的实例传递给后代,在子组件中我们就
        可以获取this.form.rules和this.form.rules
14        form: this
15      }
16    },
17    props:{
18      model:{
19        type:Object,
20        required:true
21      },
22      rules:{
23        type:Object
24      }
25    },
26  }
27 }
28 </script>

```

当FormItem组件挂载完成时,通知Form组件开始缓存需要校验的表单项

FormItem.vue

```

1  mounted() {
2      //挂载到form上时,派发一个添加事件
3      //必须做判断,因为Form组件的子组件可能不是FormItem
4      if (this.prop) {
5          //通知将表单项缓存
6          this.$parent.$emit("formItemAdd", this);
7      }
8  }

```

Form.vue

```

1  created () {
2      // 缓存需要校验的表单项
3      this.fileds = []
4      this.$on('formItemAdd',(item)=>{
5          this.fileds.push(item);
6      })
7  },

```

将缓存的表单项进行统一处理,如果有一个是错误,则返回false.(思路:使用 `Promise.all()` 进行处理).

注意:因为Promise.all方法的第一个参数是数组对象,该数组对象保存多个promise对象,所以要对FormItem的validate方法进行改造

FormItem.vue

```

1  validate() {
2      // 校验当前项:依赖async-validate
3      return new Promise(resolve => {
4          const descriptor = { [this.prop]:
this.form.rules[this.prop] };
5          const validator = new schema(descriptor);

```

```

6      validator.validate({[this.prop]:this.form.model[this.p
rop]}], errors => {
7          if (errors) {
8              this.validateStatus = "error";
9              this.errorMessage = errors[0].message;
10             resolve(false);
11         } else {
12             this.validateStatus = "";
13             this.errorMessage = "";
14             resolve(true);
15         }
16     });
17 });
18
19 }

```

Form.vue

```

1  methods: {
2      validate(callback) {
3          // 获取所有的验证结果统一处理 只要有一个失败就失败,
4          // 将formItem的validate方法 验证修改为promise对
          象,并且保存验证之后的布尔值
5          // tasks保存着验证之后的多个promise对象
6          const tasks =
this.filesds.map(item=>item.validate());
7          let ret = true;
8          // 统一处理多个promise对象来验证,只要有一个错误,就
          返回false,
9          Promise.all(tasks).then(results=>{
10             results.forEach(valid=>{
11                 if(!valid){
12                     ret = false;

```

```
13         }
14     })
15     callback(ret);
16 })
17 }
18 },
```

测试:

```
1 <m-form :model="ruleForm" :rules="rules"
  ref="ruleForm2">
2   <m-form-item label="用户名" prop="name">
3     <m-input v-model="ruleForm.name"></m-input>
4   </m-form-item>
5   <m-form-item label="密码" prop="pwd">
6     <m-input v-model="ruleForm.pwd"
7     type="password"></m-input>
8   </m-form-item>
9   <m-form-item>
10     <m-button type="danger"
11     @click="submitForm2('ruleForm2')">提交</m-button>
12   </m-form-item>
13 </m-form>
```

```
1 methods:{
2     submitForm2(name) {
3         this.$refs[name].validate(valid=>{
4             console.log(valid);
5             if(valid){
6                 alert('验证成功');
7             }else{
8                 alert('验证失败')
9             }
10        });
11    }
12 }
```