

# Vue 核心API&&组件设计



## Vue 核心API&&组件设计

课堂目标

实现一个购物车案例

组件化

回顾

## 课堂目标

1. 实现一个购物车案例
2. 掌握vue核心API
3. 深入了解Vue的组件化机制
4. 第三方组件库element-ui应用
5. 设计并实现表单组件

## 实现一个购物车案例

- 实现课程列表

```
<template>
<div>
  <h3>添加课程</h3>
  <label for="classname">课程名称: </label>
  <input id="classname" type="text" v-model="courseInfo.name"><br/>
  <label for="classPrice">课程价格: </label>
  <input type="text" v-model.number="courseInfo.price"><br/>
  <button @click="addClassToList">添加课程到列表</button>
</div>
<hr/>
<table>
  <tr>
    <th>课程名称</th>
    <th>课程价格</th>
  </tr>
  <tr v-for="(item,index) in classList" :key="item.id">
    <td>{{item.name}}</td>
    <td>{{item.price}}</td>
    <td><button @click="addCourseToCart(index)">添加到购物车</button></td>
  </tr>
</table>
<hr/>
<cart :courseItem="courseItem" @test="removeTest(arguments[0])"></cart>
</template>
<script>
```

```

export default {
  name: '',
  data(){
    return{
      title: '开课吧-精品课程', //标题
      subTitle: false, //副标题是否显示
      courseInfo:{
        name: '',
        name: ''
      },
      courseList: [ //课程列表数据
        {
          id:0,
          name: 'web全栈开发架构师',
          price: 1000
        },
        {
          id:0,
          name: 'Python人工智能',
          price: 999
        }
      ],
      courseItem:[] //购物车数据
    }
  },
  methods:{
    removeTest(index){
      this.courseItem.splice(index,1)
    },
    addClassToList(){
      this.courseList.push(this.courseInfo)
    },
    addCourseToCart(index){
      const course = this.courseList[index];
      const isHasCourse = this.courseItem.find((item)=>item.id ==course.id)
      if(isHasClass){
        isHasClass.number += 1;
      }else{
        this.courseItem.push({
          ...course,
          number:1,
          isActive:true
        })
      }
    }
  }
}
}
</script>

```

- 购物车

```

<h2>购物车</h2>
<table>

```

```

<tr>
  <th>勾选</th>
  <th>课程名称</th>
  <th>课程价格</th>
  <th>数量</th>
  <th>价格</th>
</tr>
<tr v-for="(item,index) in courseItem" :key="item.id" :class="
{active:item.isActive}">
  <td><input type="checkbox" v-model="item.isActive"></td>
  <td>{{item.courseName}}</td>
  <td>{{item.coursePrice}}</td>
  <td>
    <button @click="minus(index)">-</button>
    {{course.number}}
    <button @click="add(index)">+</button>
  </td>
  <td>{{item.number*item.coursePrice}}</td>
</tr>
<tr>
  <td></td>
  <td colspan="2">{{activeList}}/{{allList}}</td>
  <td>
    {{activePrice}}
  </td>
</tr>
</table>

<script>
export default {
  props: ['courseItem'],
  name: "Cart",
  data(){
    return{
      title: "开课吧-购物车"
    }
  },
  methods:{
    minus(index){
      const num = this.courseItem[index].number;
      if(num > 0){
        this.courseItem[index].number -= 1;
      }else{
        // this.remove(index);
        if(window.confirm('确定要删除操作吗')){
          this.$emit('test',index)
        }
      }
    },
    add(index){
      this.courseItem[index].number += 1;
    },
    remove(index){

```

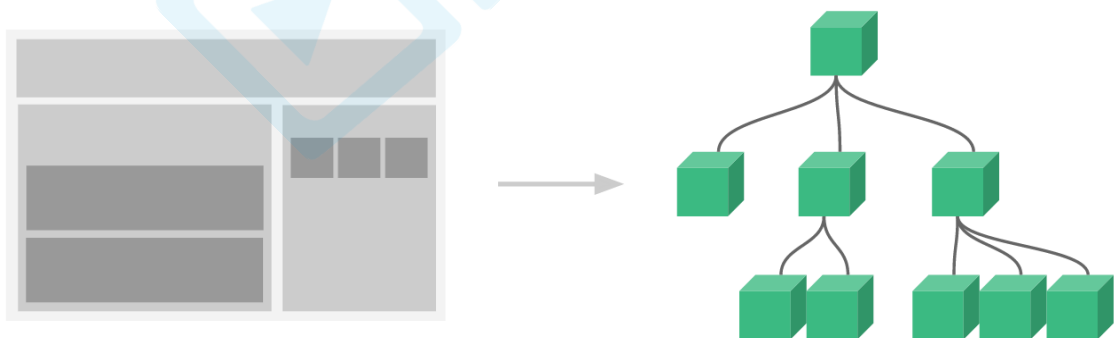
```

        if(window.confirm('确定要删除吗')){
            this.courceItem.splice(index,1)
        }
    },
    computed:{
        activeList(){
            return this.courceItem.filter(item=>item.isActive).length
        },
        allList(){
            return this.courceItem.length
        },
        activePrice(){
            let num = 0;
            this.courceItem.forEach(item=>{
                if(item.isActive){
                    num += item.courcePrice * item.number
                }
            })
            return num;
        }
    }
}
</script>

```

## 组件化

- 组件化思想



- 第三方组件应用
  - Element-UI: <http://element-cn.eleme.io/>
  - element集成: vue add element
  - 组件使用: 创建一个登陆表单并可以校验用户输入

```

<template>
  <div>

```

```

<h3>Element表单</h3>
<hr>
<el-form :model="model" :rules="rules" ref="loginForm">
  <el-form-item label="用户名" prop="username">
    <el-input v-model="model.username" autocomplete="off"></el-input>
  </el-form-item>
  <el-form-item label="确认密码" prop="password">
    <el-input type="password" v-model="model.password" autocomplete="off">
  </el-input>
  </el-form-item>
  <el-form-item>
    <el-button type="primary" @click="submitForm('loginForm')">提交</el-
button>
  </el-form-item>
</el-form>
</div>
</template>

<script>
export default {
  data() {
    return {
      model: { username: "tom", password: "" },
      rules: {
        username: [{ required: true, message: "请输入用户名" }],
        password: [{ required: true, message: "请输入密码" }],
      }
    };
  },
  methods: {
    submitForm(form) {
      this.$refs[form].validate(valid=>{
        if (valid) {
          alert('请求登录!')
        } else {
          alert('校验失败! ')
        }
      })
    }
  },
};
</script>

```

需要在element.js导入Form、FormItem和Input

```
import { Button,Form,FormItem,Input } from 'element-ui'
```

```
Vue.use(Form)
```

```
Vue.use(FormItem)
```


```
Vue.use(Input)
```

- 组件设计：实现Form、FormItem、Input

需要思考的几个问题？

1. Input是自定义组件，它是怎么实现双向绑定的？
2. FormItem是怎么知道执行校验的，它是怎么知道Input状态的？它是怎么获得对应数据模型的？
3. Form是怎么进行全局校验的？它用什么办法把数据模型和校验规则传递给内部组件？

设计思想

 formsheji

#### ◦ 实现Input

- 任务1：实现自定义组件双向绑定功能

v-model是语法糖，实现自定义组件双绑需要指定:value和@input即可

- 任务2：值发生变化能够通知FormItem组件

```
<template>
  <div>
    <input :type="type" :value="value" @input="onInput">
  </div>
</template>

<script>
  export default {
    props: {
      value: {
        type: String,
        default: ''
      },
      type: {
        type: String,
        default: 'text'
      }
    },
    methods: { // input事件触发设置模型的值并通知父组件
      onInput(e) {
        let inputValue = e.target.value;
        this.$emit('input', inputValue);
      }
    },
  }
</script>
```

#### ◦ 实现FormItem

- 任务1：给Input预留插槽 - slot
  - 匿名插槽

```
// 定义parent中插槽
<div><slot></slot></div>
// 使用parent并指定插槽内容
<parent>content</parent>
```

- 具名插槽

```
// 定义parent中插槽
<div><slot name="top"></slot><slot></slot></div>
// 使用parent并指定插槽内容
<parent><template slot="top">top content</template>content</parent>
```

- 任务2: 能够展示label和校验信息
- 任务3: 能够进行校验

```
<template>
  <div>
    <label v-if="label">{{label}}</label>
    <slot></slot>
    <p v-if="error">{{error}}</p>
  </div>
</template>

<script>
export default {
  props: {
    label: { // 输入项标签
      type: String,
      default: ''
    },
    prop: { // 字段名
      type: String,
      default: ''
    },
  },
  data() {
    return {
      error: '' // 校验错误
    }
  },
};
</script>
```

- 实现Form:

- 给form-item预留槽位
- 将数据传递给后代便于它们访问数据模型和校验规则
  - provide && inject

```

<template>
  <form>
    <slot></slot>
  </form>
</template>

<script>
export default {
  provide() {
    return {
      form: this // 将组件实例作为提供者，子代组件可方便获取
    };
  },
  props: {
    model: { type: Object, required: true },
    rules: { type: Object }
  }
};
</script>

```

#### ○ 数据校验

- 思路：校验发生在FormItem，它需要知道何时校验（让Input通知它），还需要知道怎么校验（注入校验规则）
- 任务1：Input通知校验

```

onInput(e) {
  // ...
  // $parent指FormItem
  this.$parent.$emit('validate');
}

```

- 任务2：FormItem监听校验通知，获取规则并执行校验

```

inject: ['form'], // 注入
mounted() { // 监听校验事件
  this.$on('validate', this.validate)
},
methods: {
  validate() {
    // 获取对应FormItem校验规则
    console.log(this.form.rules[this.prop]);
    // 获取校验值
    console.log(this.form.model[this.prop]);
  }
},

```

- 安装async-validator: npm i async-validator -S



```
import schema from "async-validator";

validate() {
  // 获取对应FormItem校验规则
  const rules = this.form.rules[this.prop];
  // 获取校验值
  const value = this.form.model[this.prop];
  // 校验描述对象
  const descriptor = { [this.prop]: rules };
  // 创建校验器
  const schema = new Schema(descriptor);
  schema.validate({ [this.prop]: value }, errors => {
    if (errors) {
      // 将错误信息显示
      this.error = errors[0].message;
    } else {
      // 校验通过
      this.error = "";
    }
  });
}
```

## 回顾

### Vue 核心API&&组件设计

课堂目标

实现一个购物车案例

组件化

回顾