

Vue组件化实战



课堂目标

1. 深入理解Vue的组件化机制
2. 掌握Vue组件化常用技术
3. 能够设计并实现多种类型的组件
4. 加深对一些vue原理的理解
5. 学会看开源组件库源码

知识要点

1. 组件通信方式盘点
2. 组件复合
3. 递归组件
4. 组件构造函数和实例
5. 渲染函数
6. 组件挂载
7. ...

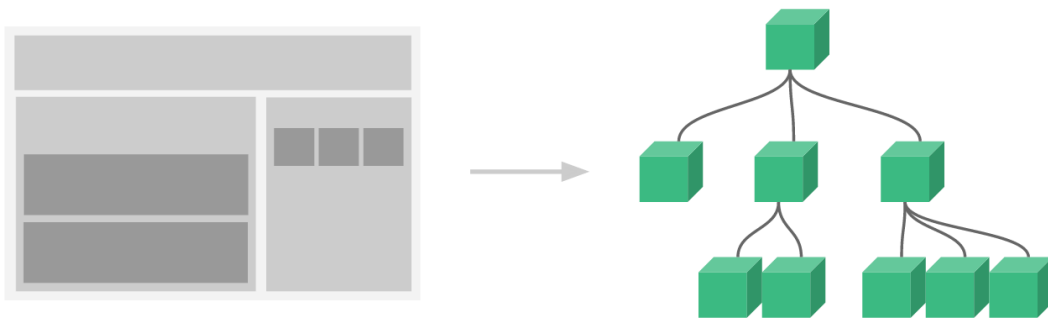
运行环境

1. [node 12.x](#)
2. [vue.js 2.6.x](#)
3. [vue-cli 4.x](#)

知识点

组件化

vue组件系统提供了一种抽象，让我们可以使用独立可复用的组件来构建大型应用，任意类型的应用界面都可以抽象为一个组件树。组件化能**提高开发效率，方便重复使用，简化调试步骤，提升项目可维护性，便于多人协同开发。**



组件通信用方式

props

eventbus

vuex

自定义事件

- 边界情况
 - \$parent
 - \$children
 - \$root
 - \$refs
 - provide/inject
- 非prop特性
 - \$attrs
 - \$listeners

组件通信

props

父给子传值

```
// child
props: { msg: String }

// parent
<HelloWorld msg="Welcome to Your Vue.js App"/>
```

自定义事件

开课吧web全栈架构师

子给父传值

```
// child
this.$emit('add', good)

// parent
<Cart @add="cartAdd($event)"></Cart>
```

事件总线

任意两个组件之间传值常用事件总线 或 vuex的方式。

```
// Bus: 事件派发、监听和回调管理
class Bus {
  constructor(){
    this.callbacks = {}
  }
  $on(name, fn){
    this.callbacks[name] = this.callbacks[name] || []
    this.callbacks[name].push(fn)
  }
  $emit(name, args){
    if(this.callbacks[name]){
      this.callbacks[name].forEach(cb => cb(args))
    }
  }
}

// main.js
Vue.prototype.$bus = new Bus()

// child1
this.$bus.$on('foo', handle)
// child2
this.$bus.$emit('foo')
```

实践中通常用Vue代替Bus，因为Vue已经实现了相应接口

vuex

创建唯一的全局数据管理者store，通过它管理数据并通知组件状态变更。

组件通信最佳实践，预习视频第12章

\$parent/\$root

兄弟组件之间通信可通过共同祖辈搭桥，\$parent或\$root。

```
// brother1
this.$parent.$on('foo', handle)
// brother2
this.$parent.$emit('foo')
```

 开课吧web全栈架构师

\$children

父组件可以通过\$children访问子组件实现父子通信。

```
// parent
this.$children[0].xx = 'xxx'
```

注意：\$children不能保证子元素顺序

\$attrs/\$listeners

包含了父作用域中不作为 prop 被识别 (且获取) 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 v-bind="\$attrs" 传入内部组件——在创建高级别的组件时非常有用。

```
// child: 并未在props中声明foo
<p>{{ $attrs.foo }}</p>

// parent
<HelloWorld foo="foo"/>
```

[文档](#)

refs

获取子节点引用

```
// parent
<HelloWorld ref="hw"/>

mounted() {
  this.$refs.hw.xx = 'xxx'
}
```

provide/inject

能够实现祖先和后代之间传值

```
// ancestor
provide() {
  return {foo: 'foo'}
}

// descendant
inject: ['foo']
```

组件通信范例代码请参考components/communication

插槽

插槽语法是Vue 实现的内容分发 API，用于复合组件开发。该技术在通用组件库开发中有大量应用。

匿名插槽

```
// comp1
<div>
  <slot></slot>
</div>

// parent
<comp>hello</comp>
```

具名插槽

将内容分发到子组件指定位置

```
// comp2
<div>
  <slot></slot>
  <slot name="content"></slot>
</div>

// parent
<Comp2>
  <!-- 默认插槽用default做参数 -->
  <template v-slot:default>具名插槽</template>
  <!-- 具名插槽用插槽名做参数 -->
  <template v-slot:content>内容...</template>
</Comp2>
```

作用域插槽

分发内容要用到子组件中的数据

```
// comp3
<div>
  <slot :foo="foo"></slot>
</div>

// parent
<Comp3>
  <!-- 把v-slot的值指定为作用域上下文对象 -->
  <template v-slot:default="slotProps">
    来自子组件数据: {{slotProps.foo}}
  </template>
</Comp3>
```

范例

插槽相关范例请参考components/slots中代码

组件化实战

通用表单组件

收集数据、校验数据并提交。

需求分析

- 实现KForm
 - 指定数据、校验规则
- KFormItem
 - 执行校验
 - 显示错误信息
- KInput
 - 维护数据

最终效果: [Element表单](#)

范例代码查看components/form/ElementTest.vue

KInput

创建components/form/**KInput.vue**

```
<template>
  <div>
    <input :value="value" @input="onInput" v-bind="$attrs">
  </div>
</template>

<script>
  export default {
    inheritAttrs: false,
    props: {
      value: {
        type: String,
        default: ''
      },
    },
    methods: {
      onInput(e) {
        this.$emit('input', e.target.value)
      }
    },
  }
</script>
```

使用KInput

创建components/form/index.vue，添加如下代码：

```
<template>
  <div>
    <h3>KForm表单</h3>
    <hr>
    <k-input v-model="model.username"></k-input>
    <k-input type="password" v-model="model.password"></k-input>
  </div>
</template>

<script>
import KInput from "../KInput";

export default {
  components: {
    KInput
  },
  data() {
    return {
      model: { username: "tom", password: "" },
    };
  }
};
</script>
```

实现KFormItem

创建components/form/KFormItem.vue

```
<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>

```

使用KFormItem

components/form/index.vue, 添加基础代码:

```

<template>
  <div>
    <h3>KForm表单</h3>
    <hr>
    <k-form-item label="用户名" prop="username">
      <k-input v-model="model.username"></k-input>
    </k-form-item>
    <k-form-item label="确认密码" prop="password">
      <k-input type="password" v-model="model.password"></k-input>
    </k-form-item>
  </div>
</template>

```

实现KForm

```

<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>

```

使用KForm

components/form/index.vue, 添加基础代码:


```

<template>
  <div>
    <h3>KForm表单</h3>
    <hr>
    <k-form :model="model" :rules="rules" ref="loginForm">
      ...
    </k-form>
  </div>
</template>

<script>
import KForm from './KForm';

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

```

数据校验

Input通知校验

```

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

```

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);
  // 返回Promise, 没有触发catch就说明验证通过
  return 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 = "";
    }
  });
}

```

表单全局验证, 为Form提供validate方法

```

validate(cb) {
  // 调用所有含有prop属性的子组件的validate方法并得到Promise数组
  const tasks = this.$children
    .filter(item => item.prop)
    .map(item => item.validate());
  // 所有任务必须全部成功才算校验通过, 任一失败则校验失败
  Promise.all(tasks)
    .then(() => cb(true))
    .catch(() => cb(false))
}

```

实现弹窗组件

弹窗这类组件的特点是它们**在当前vue实例之外独立存在**，通常挂载于body；它们是通过JS动态创建的，不需要在任何组件中声明。常见使用姿势：

```
this.$create(Notice, {
  title: '社会你杨哥喊你来搬砖',
  message: '提示信息',
  duration: 1000
}).show();
```

create函数

```
import Vue from "vue";

// 创建函数接收要创建组件定义
function create(Component, props) {
  // 创建一个Vue新实例
  const vm = new Vue({
    render(h) {
      // render函数将传入组件配置对象转换为虚拟dom
      console.log(h(Component, { props }));
      return h(Component, { props });
    }
  }).$mount(); // 执行挂载函数，但未指定挂载目标，表示只执行初始化工作

  // 将生成dom元素追加至body
  document.body.appendChild(vm.$el);

  // 给组件实例添加销毁方法
  const comp = vm.$children[0];
  comp.remove = () => {
    document.body.removeChild(vm.$el);
    vm.$destroy();
  };
  return comp;
}

// 暴露调用接口
export default create;
```

另一种创建组件实例的方式：`vue.extend(Component)`

通知组件

建通知组件，Notice.vue

```
<template>
  <div class="box" v-if="isShow">
    <h3>{{title}}</h3>
    <p class="box-content">{{message}}</p>
  </div>
</template>

<script>
```

开课吧web全栈架构师

```

export default {
  props: {
    title: {
      type: String,
      default: ""
    },
    message: {
      type: String,
      default: ""
    },
    duration: {
      type: Number,
      default: 1000
    }
  },
  data() {
    return {
      isShow: false
    };
  },
  methods: {
    show() {
      this.isShow = true;
      setTimeout(this.hide, this.duration);
    },
    hide() {
      this.isShow = false;
      this.remove();
    }
  }
};
</script>

```

```

<style>
.box {
  position: fixed;
  width: 100%;
  top: 16px;
  left: 0;
  text-align: center;
  pointer-events: none;
  background-color: #fff;
  border: grey 3px solid;
  box-sizing: border-box;
}
.box-content {
  width: 200px;
  margin: 10px auto;
  font-size: 14px;
  padding: 8px 16px;
  background: #fff;
  border-radius: 3px;
  margin-bottom: 8px;
}
</style>

```

测试, components/form/index.vue

```
<script>
import create from "@/utils/create";
import Notice from "@/components/Notice";

export default {
  methods: {
    submitForm(form) {
      this.$refs[form].validate(valid => {
        const notice = create(Notice, {
          title: "社会你杨哥喊你来搬砖",
          message: valid ? "请求登录!" : "校验失败!",
          duration: 1000
        });
        notice.show();
      });
    }
  }
};
</script>
```

作业

1. 修正input中\$parent写法的问题
2. 使用Vue.extend方式实现create方法
3. 学习element源码

今年就到这里了。各位亲，明年见！ mua~~