

组件化常用技术

组件传值、通信

父组件 => 子组件:

- 属性props

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

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

- 引用refs

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

this.$refs.hw.xx = 'xxx'
```

- 子组件children

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

子组件 => 父组件: 自定义事件

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

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

兄弟组件: 通过共同祖辈组件

通过共同的祖辈组件搭桥, \$parent或\$root。

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

祖先和后代之间

- provide/inject: 能够实现祖先给后代传值

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

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

任意两个组件之间: 事件总线 或 vuex

- 事件总线: 创建一个Bus类负责事件派发、监听和回调管理

```
// Bus: 事件派发、监听和回调管理
class Bus{
  constructor(){
    // {
    //   eventName1: [fn1, fn2],
    //   eventName2: [fn3, fn4],
    // }
    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')
```

- 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

插槽

Vue 2.6.0之后采用全新v-slot语法取代之前的slot、slot-scope

匿名插槽

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

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

具名插槽

```
// 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

```
// 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="ctx">
    来自子组件数据: {{ctx.foo}}
  </template>
</Comp3>
```

表单组件实现

- Input
 - 双向绑定: @input、:value
 - 派发校验事件

```

<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);

      this.$parent.$emit('validate');
    }
  }
};
</script>

```

- FormItem

- 给Input预留插槽 - slot
- 能够展示label和校验信息
- 能够进行校验

```

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

<script>
import Schema from 'async-validator'
export default {
  inject: ["form"],
  props: {
    label: {
      type: String,
      default: ""
    },
    prop: {
      type: String
    }
  },

```

```

data() {
  return {
    errorMessage: ""
  };
},
mounted() {
  this.$on('validate', ()=>{this.validate()})
},
methods: {
  validate() {
    // 做校验
    const value = this.form.model[this.prop]
    const rules = this.form.rules[this.prop]
    // npm i async-validator -s
    const desc = {[this.prop]: rules};
    const schema = new Schema(desc);
    // return的是校验结果的Promise
    return schema.validate([this.prop]: value}, errors => {
      if (errors) {
        this.errorMessage = errors[0].message;
      } else {
        this.errorMessage = ''
      }
    })
  }
},
};
</script>

```

- Form

- 给FormItem留插槽
- 设置数据和校验规则
- 全局校验

```

<template>
  <div>
    <slot></slot>
  </div>
</template>

<script>
export default {
  provide() {
    return {
      form: this
    };
  },
  props: {
    model: {
      type: Object,
      required: true
    }
  }
}

```

```
    },
    rules: {
      type: Object
    }
  },
  methods: {
    validate(cb) {
      const tasks = this.$children
        .filter(item => item.prop)
        .map(item => item.validate());

      // 所有任务都通过才算校验通过
      Promise.all(tasks)
        .then(() => cb(true))
        .catch(() => cb(false));
    }
  }
};
</script>
```

作业

1. 能手写Form、FormItem、Input实现
2. 尝试解决Input里面\$parent派发事件不够健壮的问题
3. 说出.sync和v-model的异同

下次课内容提示

1. 弹窗类组件设计与实现，要搞清楚vue组件实例化和挂载过程

参考资料：[渲染函数](#)、[挂载](#)

2. Tree组件实现，掌握递归组件使用

参考资料：[递归组件](#)

3. 路由实现：vue-router

参考：[vue-router官网](#)

4. vue-router实现原理

参考：[vue-router源码](#)