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
```

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']
```

- dispatch：后代给祖先传值

```
// 定义一个dispatch方法，指定要派发事件名称和数据
function dispatch(eventName, data) {
    let parent = this.$parent
    // 只要还存在父元素就继续往上查找
    while (parent) {
        // 父元素用$emit触发
        parent.$emit(eventName,data)
        // 递归查找父元素
        parent = parent.$parent
    }
}

// 使用，HelloWorld.vue
<h1 @click="dispatch('hello', 'hello,world')">{{ msg }}</h1>

// App.vue
this.$on('hello', this.sayHello)
```

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>
```

Notice组件实现

- 组件实例创建函数：create函数

```
import Vue from 'vue';

export default function create(Component, props) {
    // 先创建实例
```

```
    const vm = new Vue({
        render(h) {
            // h就是createElement，它返回VNode
            return h(Component, {props})
        }
    }).$mount();

    // 手动挂载
    document.body.appendChild(vm.$el);

    // 销毁方法
    const comp = vm.$children[0];
    comp.remove = function() {
        document.body.removeChild(vm.$el);
        vm.$destroy();
    }
    return comp;
}
```

- Notice组件
  - 插槽预留
  - 标题、内容等属性
  - 自动关闭

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

<script>
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>
```

- 使用

```
<script>
import Notice from "@/components/notice/KNotice";

export default {
  methods: {
    submitForm(form) {
      this.$refs[form].validate(valid => {
        const notice = this.$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>
```

Tree组件实现

- 递归组件Item创建

```
<template>
  <li>
    <div @click="toggle">
      {{model.title}}
      <span v-if="isFolder">[{{open ? '-' : '+'}}]</span>
    </div>
    <ul v-show="open" v-if="isFolder">
      <item class="item" v-for="model in model.children" :model="model"
:key="model.title"></item>
    </ul>
```

```
  </li>
</template>

<script>
export default {
  name: "Item",
  props: {
    model: {
      type: Object,
      required: true
    }
  },
  data() {
    return {
      open: false
    };
  },
  computed: {
    isFolder() {
      return this.model.children && this.model.children.length;
    }
  },
  methods: {
    toggle() {
      if (this.isFolder) {
        this.open = !this.open;
      }
    }
  }
};
</script>
```

- 数据和使用

```
<template>
  <div>
    <ul>
      <item class="item" :model="treeData"></item>
    </ul>
  </div>
</template>

<script>
import Item from "./Item";
export default {
  name: "app",
  data() {
    return {
      treeData: {
        title: "Web全栈架构师",
        children: [
```

```
          {
            title: "Java架构师"
          },
          {
            title: "JS高级",
            children: [
              {
                title: "ES6"
              },
              {
                title: "动效"
              }
            ]
          },
          {
            title: "Web全栈",
            children: [
              {
                title: "Vue训练营",
                expand: true,
                children: [
                  {
                    title: "组件化"
                  },
                  {
                    title: "源码"
                  },
                  {
                    title: "docker部署"
                  }
                ]
              },
              {
                title: "React",
                children: [
                  {
                    title: "JSX"
                  },
                  {
                    title: "虚拟DOM"
                  }
                ]
              },
              {
                title: "Node"
              }
            ]
          }
        ]
      }
    };
  },
  components: { Item }
```

```
};
</script>
```