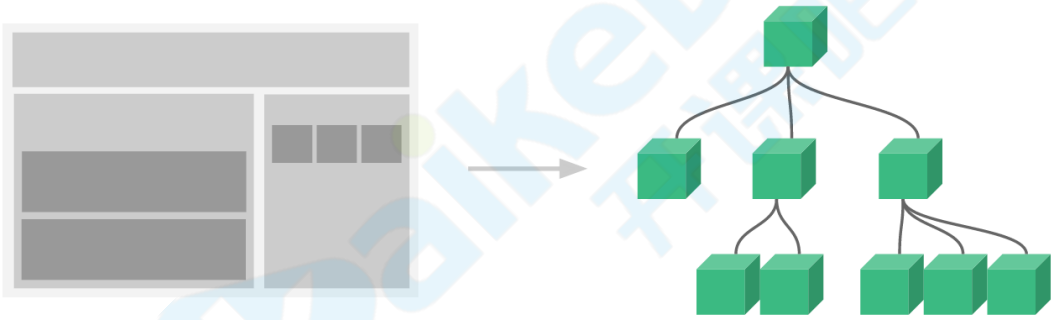
Vue预习课

核心知识06——组件化

组件基础



组件是可复用的 Vue 实例，带有一个名字，我们可以在一个通过 `new Vue` 创建的 Vue 根实例中，把这个组件作为自定义元素来使用。

组件注册、使用及数据传递

Vue.component(name, options)可用于注册组件。

范例：提取课程新增组件和课程列表组件

```
<!-- 自定义组件 -->
<course-list :courses="courses"></course-list>

<script>
    // 注册course-list组件
    Vue.component('course-list', {
        data() {
            return {
                // 改状态属于course-list内部状态，因此作为数据
                selectedCourse: '',
            }
```

```
        },
        props: {
            // 新增课程时也要访问courses，因此作为属性传递
            courses: {
                type: Array,
                default: []
            },
        },
        template: `
            <div>
                <!-- 条件渲染 -->
                <p v-if="courses.length == 0">没有任何课程信息</p>
                <!-- 列表渲染 -->
                <ul>
                    <!-- class绑定 -->
                    <li v-for="c in courses"
                        :class="{active: (selectedCourse === c)}"
                        @click="selectedCourse = c">{{c}}</li>
                    <!-- style绑定 -->
                    <!-- <li v-for="c in courses"
                    :style="{backgroundColor: (selectedCourse ===
c)?'#ddd':'transparent'}"
                    @click="selectedCourse = c">{{c}}</li> -->
                </ul>
            </div>
        `,
    })

    const app = new Vue({
        el: '#app',
        data: {
            title: '开课吧购物车'
        }
    })
</script>
```

> 组件基础

自定义事件及其监听

当子组件需要和父级组件进行通信，可以派发并监听自定义事件。

范例：新增课程组件派发新增事件

```
<!-- 监听组件事件 -->
<course-add @add-course="addCourse"></course-add>

<script>
        // 组件注册
        Vue.component('course-add', {
            data() {
                return {
                    course: '', // 将course从父组件提取到course-add维护
                }
            },
            template: `
```

```
                <div>
                    <!-- 表单输入绑定 -->
                    <input v-model="course" @keydown.enter="addCourse"/>
                    <!-- 事件处理 -->
                    <button v-on:click="addCourse">新增课程</button>
                </div>
            `,
            methods: {
                addCourse() {
                    // 发送自定义事件通知父组件
                    // 注意事件名称定义时不要有大写字母出现
                    this.$emit('add-course', this.course)
                    this.course = ''
                }
            },
        })
        const app = new Vue({
            methods: {
                // 回调函数接收事件参数
                addCourse(course) {
                    this.courses.push(course);
                }
            },
        }
</script>
```

在组件上使用v-model

[组件实现v-model]

范例：改造course-add为支持v-model的版本

```
<!-- 自定义组件支持v-model需要实现内部input的:value和@input -->
<course-add v-model="course" @add-course="addCourse"></course-add>

<script>
        Vue.component('course-add', {
            // 接收父组件传递value，不需要额外维护course
            props: ['value'],
            template: `
                <div>
                    <!-- 需要实现input的:value和@input -->
                    <input :value="value" @input="onInput"
                           @keydown.enter="addCourse"/>
                    <button v-on:click="addCourse">新增课程</button>
                </div>
            `,
            methods: {
                addCourse() {
                    // 派发事件不再需要传递数据
                    this.$emit('add-course')
                    // this.course = ''
                },
                onInput(e) {
                    this.$emit('input', e.target.value)
```

```
                }
            },
        })

        const app = new Vue({
            data: {
                course: '', // 还原course
            },
            methods: {
                addCourse() {// 还原addCourse
                    this.courses.push(this.course);
                    this.course = '';
                }
            }
        })
</script>
```

> v-model默认转换是:value和@input，如果想要修改这个行为，可以通过定义model选项
>
> ```
> Vue.component('course-add', {
>     model: {
>         prop: 'value',
>         event: 'change'
>     }
> })
> ```

通过插槽分发内容

通过使用vue提供的`<slot>`元素可以给组件传递内容

范例：弹窗组件

```
<style>
    .message-box {
        padding: 10px 20px;
        background: #4fc08d;
        border: 1px solid #42b983;
    }
    .message-box-close {
        float: right;
    }
</style>

<!-- 插槽实现内容分发 -->
<message :show.sync="show">新增课程成功！</message>

<script>
    // slot作为占位符
    Vue.component('message', {
        props: ['show'],
        template: `
            <div class="message-box" v-if="show">
                <slot></slot>
                <span class="message-box-close"
                    @click="$emit('update:show', false)">X</span>
```

```
            </div>
        `
    })
    const app = new Vue({
        data: {
            show: false, // 提示框状态
        },
        methods: {
            addCourse() {
                // 提示新增信息
                this.show = true;
            }
        },
    }
</script>
```

如果存在多个独立内容要分发，可以使用具名插槽v-slot:name

范例：添加一个title部分

```
<message :show.sync="show">
    <template v-slot:title>恭喜</template>
    <template>新增课程成功！</template>
</message>
```

```
Vue.component('message', {
    props: ['show'],
    template: `
        <div class="message-box" v-if="show">
            <strong><slot name="title"></slot></strong>
            <slot></slot>
        </div>
    `
})
```

> 插槽

Vue组件化的理解

组件化是Vue的精髓，Vue应用就是由一个个组件构成的。Vue的组件化涉及到的内容非常多，当面试时被问到：谈一下你对Vue组件化的理解。这时候有可能无从下手，可以从以下几点进行阐述：

定义：组件是**可复用的 Vue 实例**，准确讲它们是VueComponent的实例，继承自Vue。

优点：从上面案例可以看出组件化可以增加代码的**复用性**、**可维护性**和**可测试性**。

使用场景：什么时候使用组件？以下分类可作为参考：

- 通用组件：实现最基本的功能，具有通用性、复用性，例如按钮组件、输入框组件、布局组件等。
- 业务组件：它们完成具体业务，具有一定的复用性，例如登录组件、轮播图组件。
- 页面组件：组织应用各部分独立内容，需要时在不同页面组件间切换，例如列表页、详情页组件

如何使用组件

- 定义：Vue.component()，components选项，sfc
- 分类：有状态组件，functional，abstract
- 通信：props，$emit()/$on()，provide/inject，$children/$parent/$root/$attrs/$listeners
- 内容分发：<slot>，<template>，v-slot
- 使用及优化：is，keep-alive，异步组件

组件的本质

vue中的组件经历如下过程

组件配置 => VueComponent实例 => render() => Virtual DOM=> DOM

所以**组件的本质是产生虚拟DOM**