1.2 组件化开发

1.2.1 组件化开发基础

组件注册

- 全局注册

```
app.component('my-component-name', {
  /* ... */
})
```

- 局部注册

```
const ComponentA = {
  /* ... */
}
const app = Vue.createApp({
  components: {
    'component-a': ComponentA,
    'component-b': ComponentB
  }
})
```

数据传递

可利用属性props给组件传值。

定义属性

```
// 数组形式列出的 prop
props: ['title', 'likes', 'isPublished', 'commentIds', 'author']

// 对象形式列出 prop
props: {
  title: String,
  likes: Number,
  isPublished: Boolean,
  commentIds: Array,
  author: Object,
  callback: Function,
  contactsPromise: Promise // 或任何其他构造函数
}
```

```
// 属性校验
props: {
  propA: {
    type: String,
    required: true  // 必填的字符串
  },
  propB: {
    type: Number,
    default: 100  // 带有默认值的数字
  }
}
```

属性传值

```
<!-- 静态值 -->
<blog-post title="My journey with Vue"></blog-post>
<!-- 动态的变量值 -->
<blog-post :title="post.title"></blog-post>

<!-- 数字 -->
<blog-post :likes="42"></blog-post>
<!-- 布尔值 -->
<blog-post :is-published="false"></blog-post>
<!-- 数组 -->
<blog-post :comment-ids="[234, 266, 273]"></blog-post>

<!-- 对象 -->
<blog-post
  :author="{
    name: 'Veronica',
    company: 'Veridian Dynamics'
  }"
></blog-post>

<!--传入一个对象的所有 property-->
<blog-post v-bind="{
  id: 1,
  title: 'My Journey with Vue'
}"></blog-post>
<!--等价于-->
<blog-post v-bind:id="post.id" v-bind:title="post.title"></blog-post>
```

范例1：提取课程列表组件

```
<script type="text/x-template" id="course-list">
```

```
  <!-- 条件渲染 -->
  <p v-if="courses.length === 0">没有任何课程信息</p>
  <!-- 列表渲染 -->
  <ul>
    <li
      v-for="c in courses"
      :key="c"
      :class="{active: selectedCourse === c}"
      @click="selectedCourse = c"
    >
      {{c}}
    </li>
  </ul>
</script>
<script>
  const CourseList = {
    template: '#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,
        required: true
      },
    },
  };
</script>
```

使用

```
<course-list :courses="courses"></course-list>
```

```
Vue.createApp({
 components: {
   'course-list': CourseList
 }
})
```

1.2.2 自定义事件

当子组件需要和父级组件进行通信，可以定义自定义事件并派发事件。

自定义事件

通过 emits 选项在组件上定义已发出的事件

```
app.component('custom-comp', {
  emits: ['some-event', 'someEvent']
})
```

```
<!--监听some-event-->
<custom-comp @some-event="dosomething"></custom-comp>
<!--监听someEvent-->
<custom-comp @someevent="dosomething"></custom-comp>
```

> 注意：
>
> - 事件名不存在任何自动化的大小写转换，所以推始终使用 **kebab-case 的事件名**。
> - 建议定义所有发出的事件，以便更好地记录组件应该如何工作。

校验事件

如果使用对象语法定义发出的事件，则可以验证它。

```
app.component('custom-form', {
  emits: {
    // 没有验证
    click: null,

    // 验证submit 事件
    submit: ({ email, password }) => {
      if (email && password) {
        return true
      } else {
        console.warn('Invalid submit event payload!')
        return false
      }
    }
  },
  methods: {
    submitForm() {
      this.$emit('submit', { email, password })
    }
  }
})
```

范例：新增课程组件派发新增事件

```
<script id="course-add" type="text/x-template">
  <!-- 表单输入 -->
  <input type="text" v-model="course" @keydown.enter="addCourse">
  <!-- 事件处理 -->
  <button v-on:click="addCourse">新增课程</button>
</script>
<script>
  const CourseAdd = {
    template: "#course-add",
    data() {
      return {
        course: "",
      };
    },
    emits: ['add'],
    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", this.course);
        this.course = "";
      },
    },
  };
</script>
```

使用

```
<course-add @add="addCourse"></course-add>
```

```
Vue.createApp({
   methods: {
     addCourse(course) {
       this.courses.push(course);
     },
   },
  })
  .component('course-add', CourseAdd)
  .mount("#app");
```

1.2.3 在组件上使用v-model

自定义事件也可以用于创建支持 `v-model` 的自定义输入组件。

```
<custom-input v-model="searchText"></custom-input>

<!--等价于-->
<custom-input
  :model-value="searchText"
  @update:model-value="searchText = $event"
></custom-input>
```

costom-input组件内

```
app.component('custom-input', {
  props: ['modelValue'],
  emits: ['update:modelValue'],
  template: `
    <input
      :value="modelValue"
      @input="$emit('update:modelValue', $event.target.value)"
    >
  `
})
```

> vue3中没有sync修饰符，v-model转换结果和它行为相同

范例：改造course-add为支持v-model的版本

```
<script id="course-add" type="text/x-template">
  <input
    type="text"
    <!-- 绑定value和input -->
    :value="modelValue"
    @input="$emit('update:model-value', $event.target.value)"
    @keydown.enter="addCourse">
  <button v-on:click="addCourse">新增课程</button>
</script>
<script>
  const CourseAdd = {
    template: "#course-add",
    // 定义modelValue属性
    props: {
      modelValue: {
        type: String,
        required: true,
      },
```

```
    },
    // 定义update：model-value事件
    emits: ["add", 'update:model-value'],
    methods: {
      addCourse() {
        // 只需要发送add事件，不需要参数
        this.$emit("add");
      },
    },
  };
</script>
```

使用时

```
<course-add v-model="course" @add="addCourse"></course-add>
```

```
Vue.createApp({
  data() {
    return {
      // course定义在父组件中
      course: ''
    };
  },
  methods: {
    // 不需要参数了
    addCourse() {
      this.courses.push(this.course);
      this.course = ''
    },
  },
})
```

`v-model` 参数

默认情况下，组件上的 `v-model` 使用 `modelValue` 作为 prop 和 `update:modelValue` 作为事件。我们可以通过向 `v-model` 传递参数来修改这些名称：

```
<my-component v-model:foo="bar"></my-component>
```

组件中就需要一个foo属性并发出update:foo事件

```
app.component('my-component', {
  props: ['foo'],
  template: `
    <input
      type="text"
      :value="foo"
      @input="$emit('update:foo', $event.target.value)">
  `
})
```

> 这和vue2中sync修饰符行为相同，并完全替代了sync

多个 `v-model` 绑定

现在可以在单个组件实例上创建多个 v-model 绑定。每个 v-model 将同步到不同的 prop，而不需要在组件中添加额外的选项：

```
<user-name
  v-model:first-name="firstName"
  v-model:last-name="lastName"
></user-name>
```

1.2.4 通过插槽分发内容

Vue 实现了一套内容分发的 API，称为插槽，将 `<slot>` 元素作为承载分发内容的出口。

```
<!-- 使用 todo-button 组件时传递内容 -->
<todo-button>
  button label
</todo-button>

<!-- todo-button 组件模板中使用slot作为内容出口 -->
<button class="btn-primary">
  <slot></slot>
</button>

<!-- 未来渲染的 HTML -->
<button class="btn-primary">
  button label
</button>
```

具名插槽

有时我们需要多个插槽，这样可以将指定内容放到指定位置。

```
<!--layout组件-->
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>

<!--向具名插槽提供内容的时候可在一个 <template> 元素上使用 v-slot 指令，并以参数形式提供其名称-->
<layout>
  <template v-slot:header>
    <h1>Here might be a page title</h1>
  </template>
  <template v-slot:default>
    <p>A paragraph for the main content.</p>
    <p>And another one.</p>
  </template>
  <template v-slot:footer>
    <p>Here's some contact info</p>
  </template>
</layout>
```

作用域插槽

有时插槽内容需要访问子组件中才有的数据，可使用作用域插槽实现。

```
const Child = {
  template: '<div><slot :foo="foo"></slot></div>',
  data() {
    return {
      foo: 'bar'
    }
  }
}
const Parent = {
  template: `
    <Child>
```

```
      <template v-slot:default="slotProps">
        abc
        {{slotProps.foo}}
        efg
      </template>
    </Child>
  `,
  components: { Child }
}
```

> 工作中常用场景：
>
> - 希望能够自定义每个项目的渲染方式，但是数据是在列表组件内定义的
> - 希望能够自定义table某个列的渲染方式，但是数据是在table中定义的

范例：添加课程成功之后，显示消息组件

```
<style>
  .message-box {
    padding: 10px 20px;
    background: #4fc08d;
    border: 1px solid #42b983;
    color: #fff;
  }
  .message-box-close {
    float: right;
    cursor: pointer;
  }
</style>
<script type="text/x-template" id="message">
  <div class="message-box">
    <h3>
      <slot name="title"></slot>
      <span class="message-box-close" @click="$emit('close')">X</span>
    </h3>
    <slot></slot>
  </div>
</script>
<script>
  const Message = {
    template: '#message',
    emits: ['close']
  }
</script>
```

使用

```
<message v-if="showMsg" @close="showMsg = false">
  <template v-slot:title>恭喜</template>
  <template v-slot:default>新增课程成功！</template>
</message>
```

```
Vue.createApp({
  data() {
    return {
      showMsg: false
    };
  },
  methods: {
    addCourse() {
      this.showMsg = true
    },
  },
}).component("message", Message)
```