## 1.4 可复用性

### 1.4.1 混入

混入 (mixin) 提供了一种非常灵活的方式，来分发 Vue 组件中的可复用功能。一个混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被“混合”进入该组件本身的选项。

```
// 定义一个混入对象
const myMixin = {
  created() {
    console.log('hello from mixin!')
  },
}
const app = Vue.createApp({
  // 使用混入
  mixins: [myMixin],
  // 混入对象的选项会和组件本身的选项合并
  created() {
    console.log('hello from app!')
  }
})

// 全局混入
app.mixin({
  beforeCreate() {

  }
})
app.mount('#demo') // 'hello from app!' 'hello from mixin!'
```

> mixin是vue2就有的功能，存在**来源不明**、**命名冲突**等问题
>
> vue3中使用composition-api复用逻辑是更好的解决方案

### 1.4.2 自定义指令

需要对普通 DOM 元素进行底层操作，会用到自定义指令。

```
const app = Vue.createApp({})
// 全局注册指令 `v-focus`
app.directive('focus', {
  mounted(el) {
    el.focus()
  }
})
```

```
// 局部注册指令 `v-focus`
directives: {
  focus: {
    mounted(el) {
      el.focus()
    }
  }
}
```

指令钩子函数：钩子函数和vue2相较有一些变化，现在和组件钩子一致：

- `beforeMount`：当指令第一次绑定到元素并且在挂载父组件之前调用，这里可以做一次性初始化设置。
- `mounted`：在挂载绑定元素到父组件时调用。
- `beforeUpdate`：在更新包含组件的 VNode 之前调用。
- `updated`：在包含组件的 VNode **及其子组件的 VNode** 更新后调用。
- `beforeUnmount`：在卸载绑定元素的父组件之前调用
- `unmounted`：当指令与元素解除绑定且父组件已卸载时，只调用一次。

### 1.4.3 Teleport 传送

有时组件模板的一部分在逻辑上属于该组件，而从技术角度来看，最好将模板的这一部分移动到 DOM 中 Vue app 之外的其他位置，比如一个弹窗内容、消息通知等。

```
app.component('modal-button', {
  template: `
        <button @click="modalOpen = true">
            打开弹窗
        </button>

        <teleport to="body">
          <div v-if="modalOpen" class="modal">
            <div>
              <slot></slot>
              <button @click="modalOpen = false">关闭</button>
            </div>
          </div>
        </teleport>
  `,
  data() {
    return {
      modalOpen: false
    }
  }
```

```
})
```

```
<modal-button>
  <template v-slot>
    弹窗内容。。。
  </template>
</model-button>

<style>
  .modal {
    position: absolute;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
    background-color: rgba(0, 0, 0, .5);
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
  }

  .modal div {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    background-color: white;
    width: 300px;
    height: 300px;
    padding: 5px;
  }
</style>
```

### 1.4.4 渲染函数

**渲染函数**给我们提供完全JS编程能力，可以解决更复杂的模板需求。

```
const app = Vue.createApp({})

app.component('x-heading', {
  render() {
    return Vue.h(
      'h' + this.level, // tag name
      {}, // props/attributes
      this.$slots.default() // array of children
```

```
    )
  },
  props: {
    level: {
      type: Number,
      required: true
    }
  }
})
```

渲染函数中用 `if`/`else` 和 `map()` 来替代 `v-if` 和 `v-for`

```
props: ['items'],
render() {
  if (this.items.length) {
    return Vue.h('ul', this.items.map((item) => {
      return Vue.h('li', item.name)
    }))
  } else {
    return Vue.h('p', 'No items found.')
  }
}
```

`v-model` 指令展开为 `modelValue` 和 `onUpdate:modelValue`，要实现同等功能必须提供这些 prop：

```
props: ['modelValue'],
render() {
  return Vue.h(SomeComponent, {
    modelValue: this.modelValue,
    'onUpdate:modelValue': value => this.$emit('update:modelValue', value)
  })
}
```

事件处理需要提供一个正确的prop名称，例如，要处理 `click` 事件，prop名称应该是 `onClick`。

```
render() {
  return Vue.h('div', {
    onClick: $event => console.log('clicked', $event.target)
  })
}
```

对于 `.passive` 、 `.capture` 和 `.once` 事件修饰符，Vue提供了专属的对象语法：

```
render() {
  return Vue.h('input', {
    onClick: {
      handler: this.doThisOnceInCapturingMode,
      once: true,
      capture: true
    },
  })
}
```

> 对于所有其它的修饰符，需要在事件处理函数中手动使用事件方法

通过 `this.$slots` 访问静态插槽的内容，每个插槽都是一个 VNode 数组：

```
render() {
  // `<div><slot></slot></div>`
  return Vue.h('div', {}, this.$slots.default())
}
```

如果要将插槽传递给子组件：

```
render() {
  // `<child v-slot="props"><span>{{ props.text }}</span></child>`
  return Vue.h('div', [
    Vue.h('child', {}, {
      // 传递一个对象作为children
      // 形如 { name: props => VNode | Array<VNode> }
      default: (props) => Vue.h('span', props.text)
    })
  ])
}
```

### 1.4.5 插件

插件是自包含的代码，通常给 Vue 添加全局功能。插件可以是包含 `install()` 方法的 `object`，也可以是 `function`

```
export default {
  install: (app, options) => {
    // 插件接收应用实例和插件选项
  }
}
```

**插件常见任务**

添加指令/组件/过渡等全局资源

```
export default {
  install: (app, options) => {
    app.component('comp', {})
  }
}
```

全局混入一些组件选项

```
export default {
  install: (app, options) => {
    app.mixin({})
  }
}
```

添加实例方法

```
export default {
  install: (app, options) => {
    app.config.globalProperties.xx = xx
  }
}
```

**使用插件**

实例挂载之前调用use()注册插件

```
app.use(plugin)
```

范例：实现一个Message插件

```
const MessagePlugin = function (app) {
  const MyMessage = {
    props: {
      msg: {
        type: String,
        required: true
      },
      duration: {
        type: Number,
        default: 1000
      }
    },
    template: `
      <div class="message-box">
        <p>{{msg}}</p>
      </div>
    `,
    mounted() {
      setTimeout(() => {
        app.config.globalProperties.$message(null)
      }, this.duration);
    },
  }

  const container = document.createElement('div')
  document.body.appendChild(container)

  app.config.globalProperties.$message = function (props) {
    if (props) {
      render(h(MyMessage, props), container)
    } else {
      render(null, container)
    }
  }
}
```