25 | 表格：如何设计一个表格组件

2021-12-17 大圣

《玩转Vue 3全家桶》 课程介绍 >

讲述：大圣
时长 07:22 大小 6.76M

你好，我是大圣。

上一讲我们实现了树形组件，树形组件的主要难点就是对无限嵌套数据的处理。今天我们来介绍组件库中最复杂的表格组件，表格组件在项目中负责列表数据的展示，尤其是在管理系统中，比如用户信息、课程订单信息的展示，都需要使用表格组件进行渲染。

关于表单的具体交互形式和复杂程度，你可以访问ElementPlus、NaiveUi、AntDesignVue这三个主流组件库中的表格组件去体验，并且社区还提供了单独的复杂表格组件，这一讲我就给你详细说说一个复杂表格组件如何去实现。

表格组件

大部分组件库都会内置表格组件，这是总后台最常用的组件之一，用于展示大量的结构化的数据。html 也提供了内置的表格标签，由 <table> 、<thead> 、<tbody> 、<tr> 、<th> 、<td> 这些标签来组成一个最简单的表格标签。

我们先研究一下 html 的 table 标签。下面的代码中，table 标签负责表格的容器，thead 负责表头信息的容器，tbody 负责表格的主体，tr 标签负责表格的每一行，th 和 td 分别负责表头和主体的单元格。

其实标准的表格系列标签，跟 div+css 实现是有很大区别的。比如表格在做单元格合并时，要提供原生属性，这时候用 div 就很麻烦了。另外，它们的渲染原理上也有一定的区别，每一列的宽度会保持一致。

复制代码

```
<table>
  <thead>
    <tr>
      <th>课程</th>
      <th>价格</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>重学前端</td>
      <td>129</td>
    </tr>
    <tr>
      <td>玩转Vue3全家桶</td>
      <td>129</td>
    </tr>
  </tbody>
</table>
```

简单的表格数据渲染并不需要组件，我们直接使用标准的 table 系列标签就可以。但有的时候，除了呈现数据，也会带有一些额外的功能要求，比如嵌套列、性能优化等。这时候组件的好处就很明显了，它能帮我们省去这些基础的工作。

表格组件的使用风格，从设计上说也分为了两个方向。一个方向是完全由数据驱动，这里我们可以参考 Naive Ui 的使用方式，n-data-table 标签负责容器，直接通过 data 属性传递数据，通过 columns 传递表格的表头配置。

下面的代码中，我们在 colums 中去配置每行需要显示的属性，通过 render 函数可以返回定制化的结果，再使用 h 函数返回 Button，渲染出对应的按钮。

复制代码

```
<template>
  <n-data-table :columns="columns" :data="data" :pagination="pagination" />
</template>
<script>
import { h, defineComponent } from 'vue'
import { NTag, NButton, useMessage } from 'naive-ui'
const createColumns = ({ sendMail }) => {
  return [
    {
      title: 'Name',
      key: 'name',
      align: 'center'
    },
    {
      title: 'Age',
      key: 'age'
    },
    {
      title: 'Action',
      key: 'actions',
      render (row) {
        return h(
          NButton,
          {
            size: 'small',
            onClick: () => sendMail(row)
          },
          { default: () => 'Send Email' }
        )
      }
    }
  ]
}
const createData = () => [
  {
    key: 0,
    name: 'John Brown',
    age: 32,
    tags: ['nice', 'developer']
  },
  {
    key: 1,
    name: 'Jim Green',
    age: 42,
  },
  {
```

```
    key: 2,
    name: 'Joe Black',
    age: 32
  }
]
export default defineComponent({
  setup () {
    const message = useMessage()
    return {
      data: createData(),
      columns: createColumns({
        sendMail (rowData) {
          message.info('send mail to ' + rowData.name)
        }
      }),
      pagination: {
        pageSize: 10
      }
    }
  }
})
</script>

```

还有一种是 Element3 现在使用的风格，配置数据之后，具体数据的展现形式交给子元素来决定，把 columns 当成组件去使用，我们仍然通过例子来加深理解。

下面的代码中，我们配置完 data 后，使用 el-table-colum 组件去渲染组件的每一列，通过 slot 的方式去实现定制化的渲染。这两种风格各有优缺点，我们后面还会结合 Elemnt3 的源码进行讲解.

复制代码

```
<el-table :data="tableData" border style="width: 100%">
  <el-table-column fixed prop="date" label="日期" width="150">
  </el-table-column>
  <el-table-column prop="name" label="姓名" width="120"> </el-table-column>
  <el-table-column prop="province" label="省份" width="120"> </el-table-column>
  <el-table-column prop="city" label="市区" width="120"> </el-table-column>
  <el-table-column prop="address" label="地址" width="300"> </el-table-column>
  <el-table-column prop="zip" label="邮编" width="120"> </el-table-column>
  <el-table-column fixed="right" label="操作" width="100">
    <template v-slot="scope">
      <el-button @click="handleClick(scope.row)" type="text" size="small"
        >查看</el-button
      >
      <el-button type="text" size="small">编辑</el-button>
```

```
    </template>
  </el-table-column>
</el-table>
<script>
  export default {
    methods: {
      handleClick(row) {
        console.log(row)
      }
    },
    data() {
      return {
        tableData: [
          {
            date: '2016-05-02',
            name: '王小虎',
            province: '上海',
            city: '普陀区',
            address: '上海市普陀区金沙江路 1518 弄',
            zip: 200333
          },
          {
            date: '2016-05-04',
            name: '王小虎',
            province: '上海',
            city: '普陀区',
            address: '上海市普陀区金沙江路 1517 弄',
            zip: 200333
          },
          {
            date: '2016-05-01',
            name: '王小虎',
            province: '上海',
            city: '普陀区',
            address: '上海市普陀区金沙江路 1519 弄',
            zip: 200333
          },
          {
            date: '2016-05-03',
            name: '王小虎',
            province: '上海',
            city: '普陀区',
            address: '上海市普陀区金沙江路 1516 弄',
            zip: 200333
          }
        ]
      }
    }
  }
</script>
```

表格组件的扩展

复杂的表格组件需要对表格的显示和操作进行扩展。

首先是从表格的显示上扩展，我们可以支持表头或者某一列的锁定，在滚动的时候锁定列不受影响。一个 table 标签很难实现这个效果，这时候我们就需要分为 table-head 和 table-body 两个组件进行维护，通过 colgroup 组件限制每一列的宽度实现表格的效果，而且表头还需要支持表头嵌套。

下面的示意图中，表头就是被分组显示的。

多级表头

数据结构比较复杂的时候，可使用多级表头来展现数据的层次关系。

日期	配送信息			
	姓名	地址		
		省份	市区	地址
2016-05-03	王小虎	上海	普陀区	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海	普陀区	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海	普陀区	上海市普陀区金沙江路 1518 弄
2016-05-01	王小虎	上海	普陀区	上海市普陀区金沙江路 1518 弄
2016-05-08	王小虎	上海	普陀区	上海市普陀区金沙江路 1518 弄
2016-05-06	王小虎	上海	普陀区	上海市普陀区金沙江路 1518 弄
2016-05-07	王小虎	上海	普陀区	上海市普陀区金沙江路 1518 弄

我们还是先分析一下需求。对于表格的操作来说，首先要和树组件一样，每一样支持复选框进行选中，方便进行批量的操作。另外，表头还需要支持点击事件，点击后对当前这一列实现排序的效果，同时每一列还可能会有详情数据的展开，甚至表格内部还会有树形组件的嵌套、底部的数据显示等等。

把这些需求组合在一起，表格就成了组件库中最复杂的组件。我们需要先分解需求，把组件内部拆分成 table、table-column、table-body、table-header 组件，我们挨个来看一下。

首先，在 table 组件的内部，我们使用 table-body 和 table-header 构成组件。table 提供了整个表格的标签容器；hidden-columns 负责隐藏列的显示，并且通过 table-store 进行表格内部的状态管理。每当 table 中的 table-store 被修改后，table-header、table-body 都需要重新渲染。

复制代码

```
<template>
  <div class="el-table">
    <div class="hidden-columns" ref="hiddenColumns">
      <slot></slot>
    </div>
    <div class="el-table__header-wrapper"
         ref="headerWrapper">
      <table-header ref="tableHeader"
                    :store="store">
      </table-header>
    </div>
    <div class="el-table__body-wrapper"
         ref="bodyWrapper">
      <table-body :context="context"
                  :store="store">
      </table-body>
    </div>
  </div>
</template>
```

然后在 table 组件的初始化过程中，我们首先使用 createStore 创建表格的 store 数据管理，并且通过 TableLayout 创建表格的布局，然后把 store 通过属性的方式传递给 table-header 和 table-body。

复制代码

```
let table = getCurrentInstance()
    const store = createStore(table, {
      rowKey: props.rowKey,
      defaultExpandAll: props.defaultExpandAll,
      selectOnIndeterminate: props.selectOnIndeterminate,
      // TreeTable 的相关配置
      indent: props.indent,
```

```
      lazy: props.lazy,
      lazyColumnIdentifier: props.treeProps.hasChildren || 'hasChildren',
      childrenColumnName: props.treeProps.children || 'children',
      data: props.data
    })
    table.store = store
    const layout = new TableLayout({
      store: table.store,
      table,
      fit: props.fit,
      showHeader: props.showHeader
    })
    table.layout = layout

```

再接着，table-header 组件内部会接收传递的 store，并且提供监听的事件，包括 click，mousedown 等鼠标操作后，计算出当前表头的宽高等数据进行显示。

复制代码

```
const instance = getCurrentInstance()
    const parent = instance.parent
    const storeData = parent.store.states
    const filterPanels = ref({})
    const {
      tableLayout,
      onColumnsChange,
      onScrollableChange
    } = useLayoutObserver(parent)
    const hasGutter = computed(() => {
      return !props.fixed && tableLayout.gutterWidth
    })
    onMounted(() => {
      nextTick(() => {
        const { prop, order } = props.defaultSort
        const init = true
        parent.store.commit('sort', { prop, order, init })
      })
    })
    const {
      handleHeaderClick,
      handleHeaderContextMenu,
      handleMouseDown,
      handleMouseMove,
      handleMouseOut,
      handleSortClick,
      handleFilterClick
    } = useEvent(props, emit)
    const {
      getHeaderRowStyle,
```

```
31        getHeaderRowClass,
32        getHeaderCellStyle,
33        getHeaderCellClass
34      } = useStyle(props)
35      const { isGroup, toggleAllSelection, columnRows } = useUtils(props)
36
37      instance.state = {
38        onColumnsChange,
39        onScrollableChange
40      }
41      // eslint-disable-next-line
42      instance.filterPanels = filterPanels
43
```

在 table-body 中，也是类似的实现方式和效果。不过 table-body 和 table-header 中的定制需求较多，我们需要用 render 函数来实现定制化的需求。

下面的代码中，我们利用 h 函数返回 el-table__body 的渲染，通过 state 中读取的 columns 数据依次进行数据的显示。

复制代码

```
1  render() {
2
3      return h(
4        'table',
5        {
6          class: 'el-table__body',
7          cellspacing: '0',
8          cellpadding: '0',
9          border: '0'
10       },
11       [
12         hColgroup(this.store.states.columns.value),
13         h('tbody', {}, [
14           data.reduce((acc, row) => {
15             return acc.concat(this.wrappedRowRender(row, acc.length))
16           }, []),
17           h(
18             ElTooltip,
19             {
20               modelValue: this.tooltipVisible,
21               content: this.tooltipContent,
22               manual: true,
23               effect: this.$parent.tooltipEffect,
24               placement: 'top'
25             },
26             {
```

```
              default: () => this.tooltipTrigger
            }
          )
        ])
      ]
    )
  }

```

整体表格组件的渲染逻辑和过程比较复杂。为了帮你抽丝剥茧，这节课我重点给你说说 Element3 中 table 标签的渲染过程，至于具体的表格实现代码，你可以课后参考 Element3 的源码。

表格组件除了显示的效果非常复杂、交互非常复杂之外，还有一个非常棘手的性能问题。由于表格是二维渲染，而且表格组件如果想支持表头或者某一列锁定的定制效果，内部需要渲染不止一个 table 标签。一旦数据量庞大之后，表格就成了最容易导致性能瓶颈的组件，那这种场景如何去做优化呢？

这里我们要快速回顾一下性能优化那一讲的思路：性能优化主要的思路就是如何能够减少计算量。比如我们的表格如果有 1000 行要显示，但是我们浏览器最多只能显示 100 条，其他的需要通过滚动条的方式进行滚动显示，屏幕之外，成千上万个 dom 元素就成了性能消耗的主要原因。

针对这种情况，我们可以考虑类似图片懒加载的方案，对屏幕之外的 dom 元素做懒渲染，也就是非常常见的**虚拟列表解决方案**。

在虚拟列表解决方案中，我们首先要获取窗口的高度、元素的高度以及当前滚动的距离，通过这些数据计算出当前屏幕显示出来的数据。然后创建这些元素标签，设置元素的 transform 属性模拟滚动效果。这样表面看是 1000 条数据在表格里显示，实际只渲染了屏幕中间的这 100 行数据，当我们滚动鼠标的同时，去维护这 100 个数据列表，这样就完成了标签过多的性能问题。

如果表格内部每一行的高度不同的话，我们就需要对每一个元素的高度进行估计。具体操作时，先进行渲染，然后等待渲染完毕之后获取高度并且缓存下来，即可实现虚拟列表元素高度的自适应。

总结

今天要我们学习了表格组件如何实现，我给你做个总结吧。

表格组件是组件库中最复杂的组件，核心的难点除了**数据的嵌套渲染**和**复杂的交互**之外，**复杂的 dom 节点**也是表格的特点之一。我们通过对 table-header、table-body 和 table-footer 的组件分析，掌握了表格组件设计思路的实现细节。

除此之外，表格也是最容易导致页面卡顿的组件，所以我们除了数据驱动渲染之外，还需要考虑通过虚拟滚动的方式进行渲染的优化，这也是列表数据常见的优化策略，属于懒渲染的解决方案。

无论数据有多少行，我们只渲染用户可视窗口之内的，控制 top 的属性来模拟滚动效果，通过 computed 计算出需要渲染的数据。最后，我还想提醒你注意，虚拟滚动也是面试的热门解决方案，你一定要手敲一遍才能加深理解。

思考题

最后留个思考题吧，你现在基础的复杂项目或者组件库中，有哪些组件适合用虚拟滚动做性能优化呢？欢迎你在评论区分享你的答案，也欢迎你把这一讲分享给你的同事和朋友们，我们下一讲再见

上一篇 24 | 树：如何设计一个树形组件？

精选留言 (3)

写留言

海阔天空
2021-12-17

表格算是elment中比较难的组件了。感觉 Naive Ui 的使用方式配置得比较多，element 让人更好理解一点

展开

1

银太
2021-12-17

请教下，我是想对table封装成xtable，那么在page中使用的时候原来的table的slot就会写到xtable的slot，那table怎么渲染？
x-table：

<a-table>...

展开

Geek_a964f4
2021-12-17

这里有代码有点看不懂了

展开