实战 6：可用 slot-scope 自定义列的表格组件——Table

上一节，我们基于 Render 函数实现了在表格中自定义列模板的组件 Table，虽说 Render 函数能够完全发挥 JavaScript 的编程能力，实现几乎所有的自定义工作，但本质上，使用者写的是一个庞大的 JS 对象，它不具备 DOM 结构，可读性和可维护性都比较差。对于大部分写 Vue.js 的开发者来说，更倾向于使用 template 的语法，毕竟它是 Vue.js 独有的特性。本小节则在上一节的 Table 组件基础上修改，实现一种达到同样渲染效果，但对使用者更友好的 slot-scope 写法。

什么是 slot-scope

slot（插槽）我们都很熟悉，它是 Vue.js 组件的 3 个 API 之一，用于分发内容。那 slot-scope 是什么呢？先来看一个场景，比如某组件拥有下面的模板：

```
<ul>
  <li v-for="book in books" :key="book.id">
    {{ book.name }}
  </li>
</ul>
```

使用者传递一个数组 books，由组件内的 v-for 循环显示，这里的 {{ book.name }} 是纯文本输出，如果想自定义它的模板（即内容分发），就要用到 slot，但 slot 只能是固定的模板，没法自定义循环体中的一个具体的项，事实上这跟上一节的 Table 场景是类似的。

常规的 slot 无法实现对组件循环体的每一项进行不同的内容分发，这就要用到 slot-scope，它本质上跟 slot 一样，只不过可以传递参数。比如上面的示例，使用 slot-scope 封装：

```
<ul>
  <li v-for="book in books" :key="book.id">
    <slot :book="book">
      <!-- 默认内容 -->
      {{ book.name }}
    </slot>
  </li>
</ul>
```

在 slot 上，传递了一个自定义的参数 book，它的值绑定的是当前循环项的数据 book，这样在父级使用时，就可以在 slot 中访问它了：

```
<book-list :books="books">
  <template slot-scope="slotProps">
    <span v-if="slotProps.book.sale">限时优惠
</span>
    {{ slotProps.book.name }}
  </template>
</book-list>
```

使用 slot-scope 指定的参数 slotProps 就是这个 slot 的全部参数，它是一个对象，在 slot-scope 中是可以传递多个参数的，上例我们只写了一个参数 book，所以访问它就是 slotProps.book。这里推荐使用 ES6 的解构，能让参数使用起来更方便：

```
<book-list :books="books">
  <template slot-scope="{ book }">
    <span v-if="book.sale">限时优惠</span>
    {{ book.name }}
  </template>
</book-list>
```

除了可以传递参数，其它用法跟 slot 是一样的，比如也可以“具名”：

```
<slot :book="book" name="book">
  {{ book.name }}
</slot>
```

```
<template slot-scope="{ book }" slot="book">
  <span v-if="book.sale">限时优惠</span>
  {{ book.name }}
</template>
```

这就是作用域 slot（slot-scope），能够在组件的循环体中做内容分发，有了它，Table 组件的自定义列模板就不用写一长串的 Render 函数了。

为了把 Render 函数和 slot-scope 理解透彻，下面我们用 3 种方法来改写 Table，实现 slot-scope 自定义列模板。

方案一

第一种方案，用最简单的 slot-scope 实现，同时也兼容 Render 函数的旧用法。拷贝上一节的 Table 组件目录，更名为 table-slot，同时也拷贝路由，更名为 table-slot.vue。为了兼容旧的 Render 函数用法，在 columns 的列配置 column 中，新增一个字段 slot 来指定 slot-scope 的名称：

```
<!-- src/components/table-slot/table.vue -->
<template>
  <table>
    <thead>
      <tr>
        <th v-for="col in columns">{{ col.title
}}</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="(row, rowIndex) in data">
        <td v-for="col in columns">
          <template v-if="'render' in col">
            <Render :row="row" :column="col"
:index="rowIndex" :render="col.render"></Render>
          </template>
          <template v-else-if="'slot' in col">
            <slot :row="row" :column="col"
:index="rowIndex" :name="col.slot"></slot>
          </template>
          <template v-else>{{ row[col.key] }}
</template>
        </td>
      </tr>
    </tbody>
  </table>
</template>
```

相比原先的文件，只在 'render' in col 的条件下新加了一个 template 的标签，如果使用者的 column 配置了 render 字段，就优先以 Render 函数渲染，然后再判断是否用 slot-scope 渲染。在定义的作用域 slot 中，将行数据 row、列数据 column 和第几行 index 作为 slot 的参数，并根据 column 中指定的 slot 字段值，

动态设置了具名 name。使用者在配置 columns 时，只要指定了某一列的 slot，那就可以在 Table 组件中使用 slot-scope。我们以上一节的可编辑整行数据为例，用 slot-scope 的写法实现完全一样的效果：

```
<!-- src/views/table-slot.vue -->
<template>
  <div>
    <table-slot :columns="columns" :data="data">
      <template slot-scope="{ row, index }"
slot="name">
        <input type="text" v-model="editName" v-
if="editIndex === index" />
        <span v-else>{{ row.name }}</span>
      </template>

      <template slot-scope="{ row, index }"
slot="age">
        <input type="text" v-model="editAge" v-
if="editIndex === index" />
        <span v-else>{{ row.age }}</span>
      </template>

      <template slot-scope="{ row, index }"
slot="birthday">
        <input type="text" v-model="editBirthday"
v-if="editIndex === index" />
        <span v-else>{{ getBirthday(row.birthday)
}}</span>
      </template>

      <template slot-scope="{ row, index }"
slot="address">
```

```
        <input type="text" v-model="editAddress"
v-if="editIndex === index" />
        <span v-else>{{ row.address }}</span>
      </template>

      <template slot-scope="{ row, index }"
slot="action">
        <div v-if="editIndex === index">
          <button @click="handleSave(index)">保
存</button>
          <button @click="editIndex = -1">取消
</button>
        </div>
        <div v-else>
          <button @click="handleEdit(row,
index)">操作</button>
        </div>
      </template>
    </table-slot>
  </div>
</template>
<script>
  import TableSlot from '../components/table-
slot/table.vue';

  export default {
    components: { TableSlot },
    data () {
      return {
        columns: [
          {
            title: '姓名',
            slot: 'name'
```

```
    },
    {
      title: '年龄',
      slot: 'age'
    },
    {
      title: '出生日期',
      slot: 'birthday'
    },
    {
      title: '地址',
      slot: 'address'
    },
    {
      title: '操作',
      slot: 'action'
    }
  ],
  data: [
    {
      name: '王小明',
      age: 18,
      birthday: '919526400000',
      address: '北京市朝阳区芍药居'
    },
    {
      name: '张小刚',
      age: 25,
      birthday: '696096000000',
      address: '北京市海淀区西二旗'
    },
    {
      name: '李小红',
```

```
            age: 30,
            birthday: '563472000000',
            address: '上海市浦东新区世纪大道'
          },
          {
            name: '周小伟',
            age: 26,
            birthday: '687024000000',
            address: '深圳市南山区深南大道'
          }
        ],
        editIndex: -1,  // 当前聚焦的输入框的行数
        editName: '',  // 第一列输入框，当然聚焦的输入
框的输入内容，与 data 分离避免重构的闪烁
        editAge: '',  // 第二列输入框
        editBirthday: '',  // 第三列输入框
        editAddress: '',  // 第四列输入框
      }
    },
    methods: {
      handleEdit (row, index) {
        this.editName = row.name;
        this.editAge = row.age;
        this.editAddress = row.address;
        this.editBirthday = row.birthday;
        this.editIndex = index;
      },
      handleSave (index) {
        this.data[index].name = this.editName;
        this.data[index].age = this.editAge;
        this.data[index].birthday =
this.editBirthday;
        this.data[index].address =
```

```
this.editAddress;
        this.editIndex = -1;
      },
      getBirthday (birthday) {
        const date = new
Date(parseInt(birthday));
        const year = date.getFullYear();
        const month = date.getMonth() + 1;
        const day = date.getDate();

        return `${year}-${month}-${day}`;
      }
    }
  }
</script>
```

示例中在 <table-slot> 内的每一个 <template> 就对应某一列的 slot-scope 模板，通过配置的 slot 字段，指定具名的 slot-scope。可以看到，基本是把 Render 函数还原成了 html 的写法，这样看起来直接多了，渲染效果是完全一样的。在 slot-scope 中，平时怎么写组件，这里就怎么写，Vue.js 所有的 API 都是可以直接使用的。

方案一是最优解，一般情况下，使用这种方案就可以了，其余两种方案是基于 Render 的。

方案二

第二种方案，不需要修改原先的 Table 组件代码，只是在使用层面修改即可。先来看具体的使用代码，然后再做分析。注意，这里使用的 Table 组件，仍然是上一节 src/components/table-render 的组件，它只有 Render 函数，没有定义 slot-scope：

```
<!-- src/views/table-render.vue 的改写 -->
<template>
  <div>
    <table-render ref="table" :columns="columns"
:data="data">
      <template slot-scope="{ row, index }"
slot="name">
        <input type="text" v-model="editName" v-
if="editIndex === index" />
        <span v-else>{{ row.name }}</span>
      </template>

      <template slot-scope="{ row, index }"
slot="age">
        <input type="text" v-model="editAge" v-
if="editIndex === index" />
        <span v-else>{{ row.age }}</span>
      </template>

      <template slot-scope="{ row, index }"
slot="birthday">
        <input type="text" v-model="editBirthday"
v-if="editIndex === index" />
        <span v-else>{{ getBirthday(row.birthday)
}}</span>
      </template>

      <template slot-scope="{ row, index }"
slot="address">
        <input type="text" v-model="editAddress"
v-if="editIndex === index" />
        <span v-else>{{ row.address }}</span>
      </template>
```

```
      <template slot-scope="{ row, index }"
slot="action">
        <div v-if="editIndex === index">
          <button @click="handleSave(index)">保
存</button>
          <button @click="editIndex = -1">取消
</button>
        </div>
        <div v-else>
          <button @click="handleEdit(row,
index)">操作</button>
        </div>
      </template>
    </table-render>
  </div>
</template>
<script>
  import TableRender from '../components/table-
render/table.vue';

  export default {
    components: { TableRender },
    data () {
      return {
        columns: [
          {
            title: '姓名',
            render: (h, { row, column, index })
=> {
              return h(
                'div',
```

```
this.$refs.table.$scopedSlots.name({
                    row: row,
                    column: column,
                    index: index
                  })
                )
              }
            },
            {
              title: '年龄',
              render: (h, { row, column, index })
=> {
                return h(
                  'div',

this.$refs.table.$scopedSlots.age({
                    row: row,
                    column: column,
                    index: index
                  })
                )
              }
            },
            {
              title: '出生日期',
              render: (h, { row, column, index })
=> {
                return h(
                  'div',

this.$refs.table.$scopedSlots.birthday({
                    row: row,
                    column: column,
```

```
                  index: index
                })
              )
            }
          },
          {
            title: '地址',
            render: (h, { row, column, index })
=> {
              return h(
                'div',

this.$refs.table.$scopedSlots.address({
                  row: row,
                  column: column,
                  index: index
                })
              )
            }
          },
          {
            title: '操作',
            render: (h, { row, column, index })
=> {
              return h(
                'div',

this.$refs.table.$scopedSlots.action({
                  row: row,
                  column: column,
                  index: index
                })
              )
```

```
            }
          }
        ],
        data: [],
        editIndex: -1,  // 当前聚焦的输入框的行数
        editName: '',  // 第一列输入框，当然聚焦的输入
框的输入内容，与 data 分离避免重构的闪烁
        editAge: '',  // 第二列输入框
        editBirthday: '',  // 第三列输入框
        editAddress: '',  // 第四列输入框
      }
    },
    methods: {
      handleEdit (row, index) {
        this.editName = row.name;
        this.editAge = row.age;
        this.editAddress = row.address;
        this.editBirthday = row.birthday;
        this.editIndex = index;
      },
      handleSave (index) {
        this.data[index].name = this.editName;
        this.data[index].age = this.editAge;
        this.data[index].birthday =
this.editBirthday;
        this.data[index].address =
this.editAddress;
        this.editIndex = -1;
      },
      getBirthday (birthday) {
        const date = new
Date(parseInt(birthday));
        const year = date.getFullYear();
```

```
      const month = date.getMonth() + 1;
      const day = date.getDate();

      return `${year}-${month}-${day}`;
    }
  },
  mounted () {
    this.data = [
      {
        name: '王小明',
        age: 18,
        birthday: '919526400000',
        address: '北京市朝阳区芍药居'
      },
      {
        name: '张小刚',
        age: 25,
        birthday: '696096000000',
        address: '北京市海淀区西二旗'
      },
      {
        name: '李小红',
        age: 30,
        birthday: '563472000000',
        address: '上海市浦东新区世纪大道'
      },
      {
        name: '周小伟',
        age: 26,
        birthday: '687024000000',
        address: '深圳市南山区深南大道'
      }
    ];
```

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

在 slot-scope 的使用上（即 template 的内容），与方案一是完全一致的，可以看到，在 column 的定义上，仍然使用了 render 字段，只不过每个 render 都渲染了一个 div 节点，而这个 div 的内容，是指定来在 <table-render> 中定义的 slot-scope：

```
render: (h, { row, column, index }) => {
  return h(
    'div',
    this.$refs.table.$scopedSlots.name({
      row: row,
      column: column,
      index: index
    })
  )
}
```

这正是 Render 函数灵活的一个体现，使用 $scopedSlots 可以访问 slot-scope，所以上面这段代码的意思是，name 这一列仍然是使用 Functional Render，只不过 Render 的是一个预先定义好的 slot-scope 模板。

有一点需要注意的是，示例中的 data 默认是空数组，而在 mounted 里才赋值的，是因为这样定义的 slot-scope，初始时读取 this.$refs.table.$scopedSlots 是读不到的，会报错，当没有数据时，也就不会去渲染，也就避免了报错。

这种方案虽然可行，但归根到底是一种 hack，不是非常推荐，之所以列出来，是为了对 Render 和 slot-scope 有进一步的认识。

方案三

第 3 中方案的思路和第 2 种是一样的，它介于方案 1 与方案 2 之间。这种方案要修改 Table 组件代码，但是用例与方案 1 完全一致。

在方案 2 中，我们是通过修改用例使用 slot-scope 的，也就是说 Table 组件本身没有支持 slot-scope，是我们“强加”上去的，如果把强加的部分，集成到 Table 内，那对使用者就很友好了，同时也避免了初始化报错，不得不把 data 写在 mounted 的问题。

保持方案 1 的用例不变，修改 src/components/table-render 中的代码。为了同时兼容 Render 与 slot-scope，我们在 table-render 下新建一个 slot.js 的文件：

```
// src/components/table-render/slot.js
export default {
  functional: true,
  inject: ['tableRoot'],
  props: {
    row: Object,
    column: Object,
    index: Number
  },
  render: (h, ctx) => {
    return h('div',
ctx.injections.tableRoot.$scopedSlots[ctx.props.c
olumn.slot]({
      row: ctx.props.row,
      column: ctx.props.column,
      index: ctx.props.index
    }));
  }
};
```

它仍然是一个 Functional Render，使用 inject 注入了父级组件 table.vue（下文改写） 中提供的实例 tableRoot。在 render 里，也是通过方案 2 中使用 $scopedSlots 定义的 slot，不过这是在组件级别定义，对用户来说是透明的，只要按方案 1 的用例来写就可以了。

table.vue 也要做一点修改：

```
<!-- src/components/table-slot/table.vue 的改写，部
分代码省略 -->
<template>
  <table>
    <thead>
```

```
      <tr>
        <th v-for="col in columns">{{ col.title
}}</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="(row, rowIndex) in data">
        <td v-for="col in columns">
          <template v-if="'render' in col">
            <Render :row="row" :column="col"
:index="rowIndex" :render="col.render"></Render>
          </template>
          <template v-else-if="'slot' in col">
            <slot-scope :row="row" :column="col"
:index="rowIndex"></slot-scope>
          </template>
          <template v-else>{{ row[col.key] }}
</template>
        </td>
      </tr>
    </tbody>
  </table>
</template>
<script>
  import Render from './render.js';
  import SlotScope from './slot.js';

  export default {
    components: { Render, SlotScope },
    provide () {
      return {
        tableRoot: this
      };
```

```
    },
    props: {
      columns: {
        type: Array,
        default () {
          return [];
        }
      },
      data: {
        type: Array,
        default () {
          return [];
        }
      }
    }
  }
</script>
```

因为 slot-scope 模板是写在 table.vue 中的（对使用者来说，相当于写在组件 <table-slot></table-slot> 之间），所以在 table.vue 中使用 provide 向下提供了 Table 的实例，这样在 slot.js 中就可以通过 inject 访问到它，继而通过 $scopedSlots 获取到 slot。需要注意的是，在 Functional Render 是没有 this 上下文的，都是通过 h 的第二个参数临时上下文 ctx 来访问 prop、inject 等的。

方案 3 也是推荐使用的，当 Table 的功能足够复杂，层级会嵌套的比较深，那时方案 1 的 slot 就不会定义在第一级组件中，中间可能会隔许多组件，slot 就要一层层中转，相比在任何地方都能直接使用的 Render 就要麻烦了。所以，如果你的组件层级简单，推荐用第一种方案；如果你的组件已经成型（某 API 基于 Render 函数），但一时间不方便支持 slot-scope，而使用者又想用，那就选方案 2；如果你的组件已经成型（某 API 基于 Render 函数），但组件层

级复杂，要按方案 1 那样支持 slot-scope 可能改动较大，还有可能带来新的 bug，那就用方案 3，它不会破坏原有的任何内容，但会额外支持 slot-scope 用法，关键是改动简单。

结语

理论上，绝大多数能用 Render 的地方，都可以用 slot-scope。对于极客来说，喜欢挑战各种新奇的写法，所以会在 Vue.js 中大量使用 Render 函数、JSX 甚至 TS；而对于求稳的开发者来说，常规的 template、slot、slot-scope 写法会是好的选择。如果非要选一种，那要从你团队的整体情况来定，如果团队大部分是写后端为主的，那可能更倾向于 TS；如果写过 React，或许 JSX 是不错的选择；如果实在不知道选什么，那就求稳吧！