实战 5：可用 Render 自定义列的表格组件——Table

表格组件 Table 是中后台产品中最常用的组件之一，用于展示大量结构化的数据。大多数组件库都提供了表格组件，比如 iView (https://www.iviewui.com/components/table)，功能也是非常强大。正规的表格，是由 <table>、<thead>、<tbody>、<tr>、<th>、<td> 这些标签组成，一般分为表头 **columns** 和数据 **data**。本小节就来开发一个最基本的表格组件 Table，它支持使用 Render 函数来自定义某一列。

分析

如果表格只是呈现数据，是比较简单的，比如下图：

Name	Age	Address
John Brown	18	New York No. 1 Lake Park
Jim Green	24	London No. 1 Lake Park
Joe Black	30	Sydney No. 1 Lake Park
Jon Snow	26	Ottawa No. 2 Lake Park

因为结构简单，我们甚至不需要组件，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，也会带有一些交互，比如有一列操作栏，可以编辑整行的数据：

姓名	年龄	出生日期	地址	操作
王小明	18	1999-2-21	北京市朝阳区芍药居	修改
张小刚	25	1992-1-23	北京市海淀区西二旗	修改
李小红	30	1987-11-10	上海市浦东新区世纪大道	修改
周小伟	26	1991-10-10	深圳市南山区深南大道	修改

写一个个的 table 系列标签是很麻烦并且重复的，而组件的好处就是省去这些基础的工作，我们直接给 Table 组件传递列的配置 **columns** 和行数据 **data**，其余的都交给 Table 组件做了。

开发 Table 组件前，有必要先了解上文说到的一系列 table 标签。一般的 table 结构是这样的：

```
<table>
  <thead>
    <tr>
      <th>姓名</th>
      <th>年龄</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>王小明</td>
      <td>18</td>
    </tr>
    <tr>
      <td>张小刚</td>
      <td>25</td>
    </tr>
  </tbody>
</table>
```

- table：定义 HTML 表格；
- thead：定义表头；

- tbody：定义表格主体；
- tr：定义表格行；
- th：定义表头单元格；
- td：定义表格单元。

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，有提供原生属性，用 div 就很麻烦了；再比如渲染原理上也有一定的区别，table 会在内容全部下载完后加载。详细的介绍可以阅读文末的扩展阅读 1。

知道了表格的结构，再来分析如何定制 API。可以看到，表格分为了两部分，表头 thead 和数据 tbody，那 props 也定义两个：

- columns：列配置，格式为数组，其中每一列 column 是一个对象，用来描述这一列的信息，它的具体说明如下：

    - title：列头显示文字；
    - key：对应列内容的字段名；
    - render：自定义渲染列，使用 Vue 的 Render 函数，不定义则直接显示为文本。

    比如：

```
[
  {
    title: '姓名',
    key: 'name'
  },
  {
    title: '年龄',
    key: 'age'
  }
]
```

- data：显示的结构化数据，格式为数组，其中每一个对象，就是一行的数据，比如：

```
[
  {
    name: '王小明',
    age: 18
  },
  {
    name: '张小刚',
    age: 25
  }
]
```

column 定义的 key 值，与 data 是一一对应的，这是一种常见的数据接口定义规则，也是 Vue.js 组件中，用数据驱动而不是 slot 驱动的经典案例。

为什么 Table 组件要用数据驱动，而不是 slot 驱动呢？slot 在很多组件中的确很好用，不过 Table 组件包含了大量的基础表格标签，如果都交给使用者由 slot 承载的话，开发成本不亚于自己实现一个 table 了，而数据驱动就简单的多，数据一般从服务端获取后就可以直接使用（或简单处理），使用者主要来定义每列的配置 **columns** 就可以了。

因为不确定使用者要对某一列做什么交互，所以不能在 Table 内来实现自定义列。使用 Render 函数可以将复杂的自定义列模板的工作交给使用者来配置，Table 内只用一个 Functional Render 做中转。

完成基础表格

我们先来完成一个基础的表格组件，之后再接入 Render 来配置自定义列。

在 src/components 目录下新建 table-render 目录，并创建 table.vue 文件：

```
<!-- src/components/table-render/table.vue -->
<template>
  <table>
    <thead>
      <tr>
        <th v-for="col in columns">{{ col.title }}</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="row in data">
        <td v-for="col in columns">{{ row[col.key] }}</td>
      </tr>
    </tbody>
  </table>
</template>
<script>
  export default {
    props: {
      columns: {
        type: Array,
        default () {
          return [];
        }
      },
      data: {
```

```
        type: Array,
        default () {
          return [];
        }
      }
    }
  }
</script>
<style>
  table{
    width: 100%;
    border-collapse: collapse;
    border-spacing: 0;
    empty-cells: show;
    border: 1px solid #e9e9e9;
  }
  table th{
    background: #f7f7f7;
    color: #5c6b77;
    font-weight: 600;
    white-space: nowrap;
  }
  table td, table th{
    padding: 8px 16px;
    border: 1px solid #e9e9e9;
    text-align: left;
  }
</style>
```

> props 中的 columns 和 data 的格式都是数组，这里要注意的是，如果 props 的类型是**对象**或**数组**，它的默认值必须从一个工厂函数获取。

tbody 内嵌套使用了两次 v-for，外层循环数据 data，内层循环列 columns，这样就填充了每个单元格。

新建路由 table-render，并在 src/views/ 目录下新建页面 table-render.vue：

```
<!-- src/views/table-render.vue -->
<template>
  <div>
    <table-render :columns="columns"
:data="data"></table-render>
  </div>
</template>
<script>
  import TableRender from '../components/table-
render/table.vue';

  export default {
    components: { TableRender },
    data () {
      return {
        columns: [
          {
            title: '姓名',
            key: 'name'
          },
          {
            title: '年龄',
```

```
      key: 'age'
    },
    {
      title: '出生日期',
      key: 'birthday'
    },
    {
      title: '地址',
      key: 'address'
    },
    {
      title: '操作'
    }
  ],
  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>
```

运行后的效果如下图：

姓名	年龄	出生日期	地址	操作
王小明	18	919526400000	北京市朝阳区芍药居	
张小刚	25	696096000000	北京市海淀区西二旗	
李小红	30	563472000000	上海市浦东新区世纪大道	
周小伟	26	687024000000	深圳市南山区深南大道	

表格已经能渲染出来了，但现在的单元格只是将 data 当作纯文本来显示，所以出生日期列显示为时间戳，因为服务端对日期有时会保存为时间戳格式。如果要显示正常的日期（如1991-5-14），目前可以另写一个计算属性（computed），手动将时间戳换算为标准日期格式后，来动态修改 data 里的 birthday 字段。这样做对于出生日期这样的数据还好，但对于**操作**这一列就不可取了，因为它带有业务逻辑，点击**编辑**按钮，是可以对当前行数据进行修改的。这时就要用到 Render 函数。

使用 Render 自定义列模板

上一节我们已经介绍过函数式组件 Functional Render 的用法，它没有状态和上下文，主要用于中转一个组件，用在本节的 Table 组件非常合适。

先在 src/components/table-render 目录下新建 render.js 文件：

```
// src/components/table-render/render.js
export default {
  functional: true,
  props: {
    row: Object,
    column: Object,
    index: Number,
    render: Function
  },
  render: (h, ctx) => {
    const params = {
      row: ctx.props.row,
      column: ctx.props.column,
      index: ctx.props.index
    };

    return ctx.props.render(h, params);
  }
};
```

render.js 定义了 4 个 props：

- **row**：当前行的数据；
- **column**：当前列的数据；
- **index**：当前是第几行；
- **render**：具体的 render 函数内容。

这里的 render 选项并没有渲染任何节点，而是直接返回 props 中定义的 render，并将 h 和当前的行、列、序号作为参数传递出去。然后在 table.vue 里就可以使用 render.js 组件：

```
<!-- table.vue，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>{{ row[col.key] }}
</template>
        </td>
      </tr>
    </tbody>
  </table>
</template>
<script>
  import Render from './render.js';

  export default {
    components: { Render },
    props: {
```

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

如果 columns 中的某一列配置了 render 字段，那就通过 render.js 完成自定义模板，否则以字符串形式渲染。比如对出生日期这列显示为标准的日期格式，可以这样定义 column：

```
// src/views/table-render.vie, 部分代码省略
export default {
  data () {
    return {
      columns: [
        // ...
        {
          title: '出生日期',
          render: (h, { row, column, index }) =>
{
            const date = new
Date(parseInt(row.birthday));
            const year = date.getFullYear();
            const month = date.getMonth() + 1;
            const day = date.getDate();

            const birthday = `${year}-${month}-
${day}`;

            return h('span', birthday);
          }
        }
      ]
    }
  }
}
```

效果如下图：

姓名	年龄	出生日期	地址	操作
王小明	18	1999-2-21	北京市朝阳区芍药居	
张小刚	25	1992-1-23	北京市海淀区西二旗	
李小红	30	1987-11-10	上海市浦东新区世纪大道	
周小伟	26	1991-10-10	深圳市南山区深南大道	

需要注意的是，columns 里定义的 render，是有两个参数的，第一个是 createElement（即 h），第二个是从 render.js 传过来的对象，它包含了当前行数据（row）、当前列配置（column）、当前是第几行（index），使用者可以基于这 3 个参数得到任意想要的结果。由于是自定义列了，显示什么都是使用者决定的，因此在使用了 render 的 column 里可以不用写字段 key 。

如果你真正理解了，应该知道 columns 里定义的 render 字段，它仅仅是名字叫 render 的一个普通函数，并非 Vue.js 实例的 *render* 选项，只是我们恰巧把它叫做 render 而已，如果愿意，也可以改为其它名字，比如 renderRow。真正的 Render 函数只有一个地方，那就是 render.js 中的 render 选项，只是它代理了 column 中的 render。这里有点绕，理清这个关系，就对 Functional Render 彻底理解了。

修改当前行

有了 render，Table 组件就已经完成了，剩余工作都是使用者来配置 columns 完成各种复杂的业务逻辑。本例来介绍最常见的表格中对整行数据编辑的功能。

操作这一列，默认是一个**修改**按钮，点击后，变为**保存**和**取消**两个按钮，同时本行其它各列都变为了输入框，并且初始值就是刚才单元格的数据。变为输入框后，可以任意修改单元格数据，点击保存按钮保存整行数据，点击取消按钮，还原至修改前的数据。

当进入编辑状态时，每一列的输入框都要有一个临时的数据使用 v-model 双向绑定来响应修改，所以在 data 里再声明四个数据：

```
// table-render.vue，部分代码省略
{
  data () {
    return {
      // ...
      editName: '',  // 第一列输入框
      editAge: '',  // 第二列输入框
      editBirthday: '',  // 第三列输入框
      editAddress: '',  // 第四列输入框
    }
  }
}
```

同时还要知道是在修改第几行的数据，所以再加一个数据标识当前正在修改的行序号（从 0 开始）：

```
// table-render.vue，部分代码省略
{
  data () {
    return {
      // ...
      editIndex: -1,  // 当前聚焦的输入框的行数
    }
  }
}
```

editIndex 默认给了 -1，也就是一个不存在的行号，当点击修改按钮时，再将它置为正确的行号。我们先定义操作列的 render：

```
// table-render.vue，部分代码省略
{
```

```
  data () {
    columns: [
      // ...
      {
        title: '操作',
        render: (h, { row, index }) => {
          // 如果当前行是编辑状态，则渲染两个按钮
          if (this.editIndex === index) {
            return [
              h('button', {
                on: {
                  click: () => {
                    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;
                  }
                }
              }, '保存'),
              h('button', {
                style: {
                  marginLeft: '6px'
                },
                on: {
                  click: () => {
                    this.editIndex = -1;
                  }
                }
```

```
              }, '取消')
            ];
          } else {  // 当前行是默认状态，渲染为一个按钮
            return h('button', {
              on: {
                click: () => {
                  this.editName = row.name;
                  this.editAge = row.age;
                  this.editAddress = row.address;
                  this.editBirthday =
row.birthday;
                  this.editIndex = index;
                }
              }
            }, '修改');
          }
        }
      }
    ]
  }
}
```

render 里的 if / else 可以先看 else，因为默认是非编辑状态，也就是说 editIndex 还是 -1。当点击**修改**按钮时，把 render 中第二个参数 { row } 中的各列数据赋值给了之前在 data 中声明的 4 个数据，这样做是因为之后点击**取消**按钮时，editName 等值已经修改了，还没有还原，所以在开启编辑状态的同时，初始化各输入框的值（当然也可以在取消时重置）。最后再把 editIndex 置为了对应的行序号 { index }，此时 render 的 if 条件 this.editIndex === index 为真，编辑列变成了两个按钮：保存和取消。点击保存，直接修改表格源数据 data 中对应的各字段值，并将 editIndex 置为 -1，退出编辑状态；点击取消，不保存源数据，直接退出编辑状态。

除编辑列，其它各数据列都有两种状态：

1. 当 editIndex 等于当前行号 index 时，呈现输入框状态；
2. 当 editIndex 不等于当前行号 index 时，呈现默认数据。

以姓名为例：

```
// table-render.vue，部分代码省略
{
  data () {
    columns: [
      // ...
      {
        title: '姓名',
        key: 'name',
        render: (h, { row, index }) => {
          let edit;

          // 当前行为聚焦行时
          if (this.editIndex === index) {
            edit = [h('input', {
              domProps: {
                value: row.name
              },
              on: {
                input: (event) => {
                  this.editName =
event.target.value;
                }
              }
            })];
          } else {
            edit = row.name;
```

```
          }

          return h('div', [
            edit
          ]);
        }
      }
    ]
  }
}
```

变量 edit 根据 editIndex 呈现不同的节点，还是先看 else，直接显示了对应字段的数据。在聚焦时（this.editIndex === index），渲染一个 input 输入框，初始值 value 通过 render 的 domProps 绑定了 row.name（这里也可绑定 editName），并监听了 input 事件，将输入的内容，实时缓存在数据 editName 中，供保存时使用。事实上，这里绑定的 value 和事件 input 就是语法糖 v-model 在 Render 函数中的写法，在 template 中，经常写作 <input v-model="editName">。

其它列与姓名类似，只是对于的字段不同：

```
// table-render.vue, 部分代码省略
{
  data () {
    return {
      columns: [
        // ...
        {
          title: '年龄',
          key: 'age',
          render: (h, { row, index }) => {
            let edit;
```

```
            // 当前行为聚焦行时
            if (this.editIndex === index) {
              edit = [h('input', {
                domProps: {
                  value: row.age
                },
                on: {
                  input: (event) => {
                    this.editAge =
event.target.value;
                  }
                }
              })];
            } else {
              edit = row.age;
            }

            return h('div', [
              edit
            ]);
          }
        },
        {
          title: '出生日期',
          render: (h, { row, index }) => {
            let edit;

            // 当前行为聚焦行时
            if (this.editIndex === index) {
              edit = [h('input', {
                domProps: {
                  value: row.birthday
```

```
                },
                on: {
                  input: (event) => {
                    this.editBirthday =
event.target.value;
                  }
                }
              })];
            } else {
              const date = new
Date(parseInt(row.birthday));
              const year = date.getFullYear();
              const month = date.getMonth() + 1;
              const day = date.getDate();

              edit = `${year}-${month}-${day}`;
            }

            return h('div', [
              edit
            ]);
          }
        },
        {
          title: '地址',
          key: 'address',
          render: (h, { row, index }) => {
            let edit;

            // 当前行为聚焦行时
            if (this.editIndex === index) {
              edit = [h('input', {
                domProps: {
```

```
                  value: row.address
                },
                on: {
                  input: (event) => {
                    this.editAddress =
event.target.value;
                  }
                }
              })];
            } else {
              edit = row.address;
            }

            return h('div', [
              edit
            ]);
          }
        },
      ]
    }
  }
}
```

> 完整的代码见：https://github.com/icarusion/vue-component-book/blob/master/src/views/table-render.vue (https://github.com/icarusion/vue-component-book/blob/master/src/views/table-render.vue)

这样，可编辑行的表格示例就完成了：

结语

本示例的 Table 组件，只展现了表格最核心的功能——自定义列模板，一个完整的 Table 组件功能要复杂的多，比如排序、筛选、列固定、表头固定、表头嵌套等。万事开头难，打好了 Table 的地基，后面的功能可以持续开发。

事实上，很多 Vue.js 的开发难题，都可以用 Render 函数来解决，它比 template 模板更灵活，可以完全发挥 JavaScript 的编程能力，因此很多 JS 的开发思想都可以借鉴。如果你习惯 JSX，那完全可以抛弃传统的 template 写法。

Render 函数虽好，但也是有弊端的，通过上面的示例可以发现，写出来的 VNode 对象是很难读的，维护性也比 template 差。下一节，我们将改写 Table 组件，用另一种思想来实现同样的功能。

扩展阅读

- [Div 和 Table 的区别 (https://www.cnblogs.com/lovebear/archive/2012/04/1