Table

Display multiple data with similar format. You can sort, filter, compare your data in a table.

Basic table

Basic table is just for data display.

:::demo After setting attribute `data` of `el-table` with an object array, you can use `prop` (corresponding to a key of the object in `data` array) in `el-table-column` to insert data to table columns, and set the attribute `label` to define the column name. You can also use the attribute `width` to define the width of columns.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column
      prop="date"
      label="Date"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }]
      }
    }
  }
</script>
```

:::

Striped Table

Striped table makes it easier to distinguish different rows.

:::demo Attribute `stripe` accepts a `Boolean`. If `true`, table will be striped.

```
<template>
  <el-table
    :data="tableData"
    stripe
    style="width: 100%">
    <el-table-column
      prop="date"
      label="Date"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }]
      }
    }
  }
</script>
```

:::

Table with border

:::demo By default, Table has no vertical border. If you need it, you can set attribute `border` to `true`.

```
<template>
  <el-table
    :data="tableData"
    border
    style="width: 100%">
    <el-table-column
      prop="date"
      label="Date"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }]
      }
    }
  }
</script>
```

:::

Table with status

You can highlight your table content to distinguish between "success, information, warning, danger" and other states.

:::demo Use `row-class-name` in `el-table` to add custom classes to a certain row. Then you can style it with custom classes.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%"
    :row-class-name="tableRowClassName">
    <el-table-column
      prop="date"
      label="Date"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address">
    </el-table-column>
  </el-table>
</template>

<style>
  .el-table .warning-row {
    background: oldlace;
  }

  .el-table .success-row {
    background: #f0f9eb;
  }
```

```
</style>

<script>
  export default {
    methods: {
      tableRowClassName({row, rowIndex}) {
        if (rowIndex === 1) {
          return 'warning-row';
        } else if (rowIndex === 3) {
          return 'success-row';
        }
        return '';
      }
    },
    data() {
      return {
        tableData:  [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }]
      }
    }
  }
</script>
```

:::

Table with fixed header

When there are too many rows, you can use a fixed header.

:::demo By setting the attribute `height` of `el-table`, you can fix the table header without any other codes.

```
<template>
  <el-table
    :data="tableData"
    height="250"
    style="width: 100%">
    <el-table-column
```

```
      prop="date"
      label="Date"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-08',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-06',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-07',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }],
      }
    }
  }
</script>
```

:::

Table with fixed column

When there are too many columns, you can fix some of them.

:::demo Attribute `fixed` is used in `el-table-column`, it accepts a `Boolean`. If `true`, the column will be fixed at left. It also accepts two string literals: 'left' and 'right', both indicating that the column will be fixed at corresponding direction.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column
      fixed
      prop="date"
      label="Date"
      width="150">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="120">
    </el-table-column>
    <el-table-column
      prop="state"
      label="State"
      width="120">
    </el-table-column>
    <el-table-column
      prop="city"
      label="City"
      width="120">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address"
      width="300">
    </el-table-column>
    <el-table-column
      prop="zip"
      label="Zip"
      width="120">
    </el-table-column>
    <el-table-column
      fixed="right"
      label="Operations"
      width="120">
      <template slot-scope="scope">
        <el-button @click="handleClick" type="text" size="small">Detail</el-button>
        <el-button type="text" size="small">Edit</el-button>
      </template>
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    methods: {
      handleClick() {
        console.log('click');
      }
    },
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036',
          tag: 'Home'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036',
          tag: 'Office'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036',
          tag: 'Home'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036',
          tag: 'Office'
        }]
      }
    }
  }
</script>
```

:::

Table with fixed columns and header

When you have huge chunks of data to put in a table, you can fix the header and columns at the same time.

:::demo Fix columns and header at the same time by combining the above two examples.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%"
    height="250">
    <el-table-column
      fixed
      prop="date"
      label="Date"
      width="150">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="120">
    </el-table-column>
    <el-table-column
      prop="state"
      label="State"
      width="120">
    </el-table-column>
    <el-table-column
      prop="city"
      label="City"
      width="120">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address"
      width="300">
    </el-table-column>
    <el-table-column
      prop="zip"
      label="Zip"
      width="120">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
```

```
data() {
  return {
    tableData: [{
      date: '2016-05-03',
      name: 'Tom',
      state: 'California',
      city: 'Los Angeles',
      address: 'No. 189, Grove St, Los Angeles',
      zip: 'CA 90036'
    }, {
      date: '2016-05-02',
      name: 'Tom',
      state: 'California',
      city: 'Los Angeles',
      address: 'No. 189, Grove St, Los Angeles',
      zip: 'CA 90036'
    }, {
      date: '2016-05-04',
      name: 'Tom',
      state: 'California',
      city: 'Los Angeles',
      address: 'No. 189, Grove St, Los Angeles',
      zip: 'CA 90036'
    }, {
      date: '2016-05-01',
      name: 'Tom',
      state: 'California',
      city: 'Los Angeles',
      address: 'No. 189, Grove St, Los Angeles',
      zip: 'CA 90036'
    }, {
      date: '2016-05-08',
      name: 'Tom',
      state: 'California',
      city: 'Los Angeles',
      address: 'No. 189, Grove St, Los Angeles',
      zip: 'CA 90036'
    }, {
      date: '2016-05-06',
      name: 'Tom',
      state: 'California',
      city: 'Los Angeles',
      address: 'No. 189, Grove St, Los Angeles',
      zip: 'CA 90036'
    }, {
      date: '2016-05-07',
      name: 'Tom',
      state: 'California',
      city: 'Los Angeles',
      address: 'No. 189, Grove St, Los Angeles',
      zip: 'CA 90036'
    }]
```

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

:::

Fluid-height Table with fixed header (and columns)

When the the data is dynamically changed, you might want the table to have a maximum height rather than a fixed height and to show the scroll bar if needed.

:::demo By setting the attribute `max-height` of `el-table`, you can fix the table header. The table body scrolls only if the height of the rows exceeds the max height value.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%"
    max-height="250">
    <el-table-column
      fixed
      prop="date"
      label="Date"
      width="150">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="120">
    </el-table-column>
    <el-table-column
      prop="state"
      label="State"
      width="120">
    </el-table-column>
    <el-table-column
      prop="city"
      label="City"
      width="120">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address"
      width="300">
    </el-table-column>
    <el-table-column
      prop="zip"
      label="Zip"
      width="120">
    </el-table-column>
    <el-table-column
```

```
      fixed="right"
      label="Operations"
      width="120">
      <template slot-scope="scope">
        <el-button
          @click.native.prevent="deleteRow(scope.$index, tableData)"
          type="text"
          size="small">
          Remove
        </el-button>
      </template>
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    methods: {
      deleteRow(index, rows) {
        rows.splice(index, 1);
      }
    },
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-08',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-06',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-07',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }]
      }
    }
  }
</script>
```

:::

Grouping table head

When the data structure is complex, you can use group header to show the data hierarchy.

:::demo Only need to place el-table-column inside a el-table-column, you can achieve group header.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column
      prop="date"
      label="Date"
      width="150">
    </el-table-column>
    <el-table-column label="Delivery Info">
      <el-table-column
        prop="name"
        label="Name"
        width="120">
      </el-table-column>
      <el-table-column label="Address Info">
        <el-table-column
```

```
            prop="state"
            label="State"
            width="120">
          </el-table-column>
          <el-table-column
            prop="city"
            label="City"
            width="120">
          </el-table-column>
          <el-table-column
            prop="address"
            label="Address"
            width="300">
          </el-table-column>
          <el-table-column
            prop="zip"
            label="Zip"
            width="120">
          </el-table-column>
        </el-table-column>
      </el-table-column>
    </el-table>
  </template>

  <script>
    export default {
      data() {
        return {
          tableData: [{
            date: '2016-05-03',
            name: 'Tom',
            state: 'California',
            city: 'Los Angeles',
            address: 'No. 189, Grove St, Los Angeles',
            zip: 'CA 90036'
          }, {
            date: '2016-05-02',
            name: 'Tom',
            state: 'California',
            city: 'Los Angeles',
            address: 'No. 189, Grove St, Los Angeles',
            zip: 'CA 90036'
          }, {
            date: '2016-05-04',
            name: 'Tom',
            state: 'California',
            city: 'Los Angeles',
            address: 'No. 189, Grove St, Los Angeles',
            zip: 'CA 90036'
          }, {
            date: '2016-05-01',
            name: 'Tom',
            state: 'California',
            city: 'Los Angeles',
            address: 'No. 189, Grove St, Los Angeles',
            zip: 'CA 90036'
          }, {
            date: '2016-05-08',
            name: 'Tom',
            state: 'California',
            city: 'Los Angeles',
            address: 'No. 189, Grove St, Los Angeles',
            zip: 'CA 90036'
          }, {
            date: '2016-05-06',
            name: 'Tom',
            state: 'California',
            city: 'Los Angeles',
            address: 'No. 189, Grove St, Los Angeles',
            zip: 'CA 90036'
          }, {
            date: '2016-05-07',
            name: 'Tom',
            state: 'California',
            city: 'Los Angeles',
            address: 'No. 189, Grove St, Los Angeles',
            zip: 'CA 90036'
          }]
        }
      }
    }
  </script>
```

:::

Single select

Single row selection is supported.

:::demo Table supports single row selection. You can activate it by adding the `highlight-current-row` attribute. An event called `current-change` will be triggered when row selection changes, and its parameters are the rows after and before this change: `currentRow` and `oldCurrentRow`. If you need to display row index, you can add a new `el-table-column` with its `type` attribute assigned to `index`, and you will see the index starting from 1.

```
<template>
  <el-table
    ref="singleTable"
    :data="tableData"
    highlight-current-row
    @current-change="handleCurrentChange"
    style="width: 100%">
    <el-table-column
```

```
      type="index"
      width="50">
    </el-table-column>
    <el-table-column
      property="date"
      label="Date"
      width="120">
    </el-table-column>
    <el-table-column
      property="name"
      label="Name"
      width="120">
    </el-table-column>
    <el-table-column
      property="address"
      label="Address">
    </el-table-column>
  </el-table>
  <div style="margin-top: 20px">
    <el-button @click="setCurrent(tableData[1])">Select second row</el-button>
    <el-button @click="setCurrent()">Clear selection</el-button>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }],
        currentRow: null
      }
    },

    methods: {
      setCurrent(row) {
        this.$refs.singleTable.setCurrentRow(row);
      },
      handleCurrentChange(val) {
        this.currentRow = val;
      }
    }
  }
</script>
```

:::

Multiple select

You can also select multiple rows.

:::demo Activating multiple selection is easy: simply add an `el-table-column` with its `type` set to `selection`. Apart from multiple selection, this example also uses `show-overflow-tooltip`: by default, if the content is too long, it will break into multiple lines. If you want to keep it in one line, use attribute `show-overflow-tooltip`, which accepts a `Boolean` value. When set `true`, the extra content will show in tooltip when hover on the cell.

```
<template>
  <el-table
    ref="multipleTable"
    :data="tableData"
    style="width: 100%"
    @selection-change="handleSelectionChange">
    <el-table-column
      type="selection"
      width="55">
    </el-table-column>
    <el-table-column
      label="Date"
      width="120">
      <template slot-scope="scope">{{ scope.row.date }}</template>
    </el-table-column>
    <el-table-column
      property="name"
      label="Name"
      width="120">
    </el-table-column>
    <el-table-column
      property="address"
      label="Address"
      show-overflow-tooltip>
    </el-table-column>
  </el-table>
  <div style="margin-top: 20px">
    <el-button @click="toggleSelection([tableData[1], tableData[2]])">Toggle
selection status of second and third rows</el-button>
    <el-button @click="toggleSelection()">Clear selection</el-button>
  </div>
```

```
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-08',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-06',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-07',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }],
        multipleSelection: []
      }
    },

    methods: {
      toggleSelection(rows) {
        if (rows) {
          rows.forEach(row => {
            this.$refs.multipleTable.toggleRowSelection(row);
          });
        } else {
          this.$refs.multipleTable.clearSelection();
        }
      },
      handleSelectionChange(val) {
        this.multipleSelection = val;
      }
    }
  }
</script>
```

:::

Sorting

Sort the data to find or compare data quickly.

:::demo Set attribute `sortable` in a certain column to sort the data based on this column. It accepts `Boolean` with a default value `false`. Set table attribute `default-sort` to determine default sort column and order. To apply your own sorting rules, use `sort-method` or `sort-by`. If you need remote sorting from backend, set `sortable` to `custom`, and listen to the `sort-change` event on Table. In the event handler, you have access to the sorting column and sorting order so that you can fetch sorted table data from API. In this example we use another attribute named `formatter` to format the value of certain columns. It accepts a function which has two parameters: `row` and `column`. You can handle it according to your own needs.

```
<template>
  <el-table
    :data="tableData"
    :default-sort = "{prop: 'date', order: 'descending'}"
    style="width: 100%">
    <el-table-column
      prop="date"
      label="Date"
      sortable
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address"
      :formatter="formatter">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }]
      }
    },
    methods: {
      formatter(row, column) {
        return row.address;
      }
    }
  }
</script>
```

:::

Filter

Filter the table to find desired data.

:::demo Set attribute `filters` and `filter-method` in `el-table-column` makes this column filterable. `filters` is an array, and `filter-method` is a function deciding which rows are displayed. It has three parameters: `value` , `row` and `column` .

```
<template>
  <el-button @click="resetDateFilter">reset date filter</el-button>
  <el-button @click="clearFilter">reset all filters</el-button>
  <el-table
    ref="filterTable"
    :data="tableData"
    style="width: 100%">
    <el-table-column
      prop="date"
      label="Date"
      sortable
      width="180"
      column-key="date"
      :filters="[{text: '2016-05-01', value: '2016-05-01'}, {text: '2016-05-02',
value: '2016-05-02'}, {text: '2016-05-03', value: '2016-05-03'}, {text: '2016-05-
04', value: '2016-05-04'}]"
      :filter-method="filterHandler"
    >
    </el-table-column>
    <el-table-column
```

```
      prop="name"
      label="Name"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address"
      :formatter="formatter">
    </el-table-column>
    <el-table-column
      prop="tag"
      label="Tag"
      width="100"
      :filters="[{ text: 'Home', value: 'Home' }, { text: 'Office', value: 'Office'
}]"
      :filter-method="filterTag"
      filter-placement="bottom-end">
      <template slot-scope="scope">
        <el-tag
          :type="scope.row.tag === 'Home' ? 'primary' : 'success'"
          disable-transitions>{{scope.row.tag}}</el-tag>
      </template>
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles',
          tag: 'Home'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles',
          tag: 'Office'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles',
          tag: 'Home'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles',
          tag: 'Office'
        }]
      }
    },
    methods: {
      resetDateFilter() {
        this.$refs.filterTable.clearFilter('date');
      },
      clearFilter() {
        this.$refs.filterTable.clearFilter();
      },
      formatter(row, column) {
        return row.address;
      },
      filterTag(value, row) {
        return row.tag === value;
      },
      filterHandler(value, row, column) {
        const property = column['property'];
        return row[property] === value;
      }
    }
  }
</script>
```

:::

Custom column template

Customize table column so it can be integrated with other components. :::demo You have access to the following data: row, column, $index and store (state management of Table) by Scoped slot.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column
      label="Date"
      width="180">
      <template slot-scope="scope">
        <i class="el-icon-time"></i>
        <span style="margin-left: 10px">{{ scope.row.date }}</span>
      </template>
    </el-table-column>
    <el-table-column
      label="Name"
      width="180">
      <template slot-scope="scope">
        <el-popover trigger="hover" placement="top">
          <p>Name: {{ scope.row.name }}</p>
          <p>Addr: {{ scope.row.address }}</p>
          <div slot="reference" class="name-wrapper">
            <el-tag size="medium">{{ scope.row.name }}</el-tag>
          </div>
        </el-popover>
      </template>
    </el-table-column>
    <el-table-column
      label="Operations">
      <template slot-scope="scope">
        <el-button
          size="mini"
          @click="handleEdit(scope.$index, scope.row)">Edit</el-button>
        <el-button
          size="mini"
          type="danger"
          @click="handleDelete(scope.$index, scope.row)">Delete</el-button>
      </template>
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }]
      }
    },
    methods: {
      handleEdit(index, row) {
        console.log(index, row);
      },
      handleDelete(index, row) {
        console.log(index, row);
      }
    }
  }
</script>
```

:::

Table with custom header

Customize table header so it can be even more customized. :::demo You can customize how the header looks by header scoped slots.

```
<template>
  <el-table
    :data="tableData.filter(data => !search ||
data.name.toLowerCase().includes(search.toLowerCase()))"
    style="width: 100%">
    <el-table-column
      label="Date"
      prop="date">
    </el-table-column>
    <el-table-column
      label="Name"
      prop="name">
    </el-table-column>
    <el-table-column
      align="right">
      <template slot="header" slot-scope="scope">
        <el-input
          v-model="search"
          size="mini"
          placeholder="Type to search"/>
      </template>
      <template slot-scope="scope">
        <el-button
          size="mini"
          @click="handleEdit(scope.$index, scope.row)">Edit</el-button>
        <el-button
          size="mini"
          type="danger"
          @click="handleDelete(scope.$index, scope.row)">Delete</el-button>
      </template>
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-02',
          name: 'John',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-04',
          name: 'Morgan',
          address: 'No. 189, Grove St, Los Angeles'
        }, {
          date: '2016-05-01',
          name: 'Jessy',
          address: 'No. 189, Grove St, Los Angeles'
        }],
        search: '',
      }
    },
    methods: {
      handleEdit(index, row) {
        console.log(index, row);
      },
      handleDelete(index, row) {
        console.log(index, row);
      }
    },
  }
</script>
```

:::

Expandable row

When the row content is too long and you do not want to display the horizontal scroll bar, you can use the expandable row feature. :::demo Activate expandable row by adding type="expand" and scoped slot. The template for el-table-column will be rendered as the contents of the expanded row, and you can access the same attributes as when you are using `Scoped slot` in custom column templates.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column type="expand">
      <template slot-scope="props">
        <p>State: {{ props.row.state }}</p>
        <p>City: {{ props.row.city }}</p>
        <p>Address: {{ props.row.address }}</p>
        <p>Zip: {{ props.row.zip }}</p>
      </template>
    </el-table-column>
    <el-table-column
      label="Date"
      prop="date">
    </el-table-column>
    <el-table-column
      label="Name"
      prop="name">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-08',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-06',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }, {
          date: '2016-05-07',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036'
        }]
      }
    }
  }
</script>
```

:::

Tree data and lazy mode

:::demo You can display tree structure data. When row contains the `children` field, it is treated as nested data. For rendering nested data, the prop `row-key` is required。Also, child row data can be loaded asynchronously. Set `lazy` property of Table to true and the function `load` . Specify `hasChildren` attribute in row to determine which row contains children. Both `children` and `hasChildren` can be configured via `tree-props` .

```
<template>
<div>
  <el-table
    :data="tableData"
    style="width: 100%;margin-bottom: 20px;"
    row-key="id"
    border
    default-expand-all>
    <el-table-column
      prop="date"
      label="date"
      sortable
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      sortable
      width="180">
    </el-table-column>
  </el-table>

  <el-table
    :data="tableData1"
    style="width: 100%"
    row-key="id"
    border
    lazy
    :load="load"
    :tree-props="{children: 'children', hasChildren: 'hasChildren'}">
    <el-table-column
```

```
        prop="date"
        label="Date"
        width="180">
      </el-table-column>
      <el-table-column
        prop="name"
        label="Name"
        width="180">
      </el-table-column>
    </el-table>
  </div>
</template>
<script>
  export default {
    data() {
      return {
        tableData: [{
          id: 1,
          date: '2016-05-02',
          name: 'wangxiaohu'
        }, {
          id: 2,
          date: '2016-05-04',
          name: 'wangxiaohu'
        }, {
          id: 3,
          date: '2016-05-01',
          name: 'wangxiaohu',
          children: [{
              id: 31,
              date: '2016-05-01',
              name: 'wangxiaohu'
            }, {
              id: 32,
              date: '2016-05-01',
              name: 'wangxiaohu'
          }]
        }, {
          id: 4,
          date: '2016-05-03',
          name: 'wangxiaohu'
        }],
        tableData1: [{
          id: 1,
          date: '2016-05-02',
          name: 'wangxiaohu'
        }, {
          id: 2,
          date: '2016-05-04',
          name: 'wangxiaohu'
        }, {
          id: 3,
            date: '2016-05-01',
            name: 'wangxiaohu',
            hasChildren: true
          }, {
            id: 4,
            date: '2016-05-03',
            name: 'wangxiaohu'
          }]
        }
      },
      methods: {
        load(tree, treeNode, resolve) {
          setTimeout(() => {
            resolve([
              {
                id: 31,
                date: '2016-05-01',
                name: 'wangxiaohu'
              }, {
                id: 32,
                date: '2016-05-01',
                name: 'wangxiaohu'
              }
            ])
          }, 1000)
        }
      },
    }
</script>
```

:::

Summary row

For table of numbers, you can add an extra row at the table footer displaying each column's sum. :::demo You can add the summary row by setting `show-summary` to `true`. By default, for the summary row, the first column does not sum anything up but always displays 'Sum' (you can configure the displayed text using `sum-text`), while other columns sum every number in that column up and display them. You can of course define your own sum behaviour. To do so, pass a method to `summary-method`, which returns an array, and each element of the returned array will be displayed in the columns of the summary row. The second table of this example is a detailed demo.

```
<template>
  <el-table
    :data="tableData"
    border
    show-summary
    style="width: 100%">
    <el-table-column
      prop="id"
      label="ID"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name">
    </el-table-column>
    <el-table-column
      prop="amount1"
      sortable
      label="Amount 1">
    </el-table-column>
    <el-table-column
      prop="amount2"
      sortable
      label="Amount 2">
    </el-table-column>
    <el-table-column
      prop="amount3"
      sortable
      label="Amount 3">
    </el-table-column>
  </el-table>

  <el-table
    :data="tableData"
    border
    height="200"
    :summary-method="getSummaries"
    show-summary
    style="width: 100%; margin-top: 20px">
    <el-table-column
      prop="id"
      label="ID"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name">
    </el-table-column>
    <el-table-column
      prop="amount1"
      label="Cost 1 ($)">
    </el-table-column>
    <el-table-column
      prop="amount2"
      label="Cost 2 ($)">
    </el-table-column>
    <el-table-column
      prop="amount3"
      label="Cost 3 ($)">
    </el-table-column>
  </el-table>
</template>
```

```
<script>
  export default {
    data() {
      return {
        tableData: [{
          id: '12987122',
          name: 'Tom',
          amount1: '234',
          amount2: '3.2',
          amount3: 10
        }, {
          id: '12987123',
          name: 'Tom',
          amount1: '165',
          amount2: '4.43',
          amount3: 12
        }, {
          id: '12987124',
          name: 'Tom',
          amount1: '324',
          amount2: '1.9',
          amount3: 9
        }, {
          id: '12987125',
          name: 'Tom',
          amount1: '621',
          amount2: '2.2',
          amount3: 17
        }, {
          id: '12987126',
          name: 'Tom',
          amount1: '539',
          amount2: '4.1',
          amount3: 15
        }]
      };
    },
    methods: {
      getSummaries(param) {
        const { columns, data } = param;
        const sums = [];
        columns.forEach((column, index) => {
          if (index === 0) {
            sums[index] = 'Total Cost';
            return;
          }
          const values = data.map(item => Number(item[column.property]));
          if (!values.every(value => isNaN(value))) {
            sums[index] = '$ ' + values.reduce((prev, curr) => {
              const value = Number(curr);
              if (!isNaN(value)) {
                return prev + curr;
              } else {
                return prev;
              }
            }, 0);
          } else {
            sums[index] = 'N/A';
          }
        });

        return sums;
      }
    }
  };
</script>
```

:::

Rowspan and colspan

Configuring rowspan and colspan allows you to merge cells :::demo Use the `span-method` attribute to configure rowspan and colspan. It accepts a method, and passes an object to that method including current row `row`, current column `column`, current row index `rowIndex` and current column index `columnIndex`. The method should return an array of two numbers, the first number being `rowspan` and second `colspan`. It can also return an object with `rowspan` and `colspan` props.

```
<template>
  <div>
    <el-table
      :data="tableData"
      :span-method="arraySpanMethod"
      border
      style="width: 100%">
      <el-table-column
        prop="id"
        label="ID"
        width="180">
      </el-table-column>
      <el-table-column
        prop="name"
        label="Name">
      </el-table-column>
      <el-table-column
        prop="amount1"
        sortable
        label="Amount 1">
      </el-table-column>
      <el-table-column
        prop="amount2"
        sortable
        label="Amount 2">
      </el-table-column>
      <el-table-column
        prop="amount3"
        sortable
        label="Amount 3">
      </el-table-column>
    </el-table>

    <el-table
      :data="tableData"
      :span-method="objectSpanMethod"
      border
      style="width: 100%; margin-top: 20px">
      <el-table-column
        prop="id"
        label="ID"
        width="180">
      </el-table-column>
      <el-table-column
        prop="name"
        label="Name">
      </el-table-column>
      <el-table-column
        prop="amount1"
        label="Amount 1">
      </el-table-column>
      <el-table-column
        prop="amount2"
        label="Amount 2">
      </el-table-column>
      <el-table-column
        prop="amount3"
        label="Amount 3">
      </el-table-column>
    </el-table>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          id: '12987122',
          name: 'Tom',
          amount1: '234',
          amount2: '3.2',
          amount3: 10
        }, {
          id: '12987123',
          name: 'Tom',
          amount1: '165',
          amount2: '4.43',
          amount3: 12
        }, {
          id: '12987124',
          name: 'Tom',
          amount1: '324',
          amount2: '1.9',
          amount3: 9
        }, {
          id: '12987125',
          name: 'Tom',
          amount1: '621',
          amount2: '2.2',
          amount3: 17
        }, {
          id: '12987126',
          name: 'Tom',
          amount1: '539',
          amount2: '4.1',
          amount3: 15
        }]
      };
    },
    methods: {
      arraySpanMethod({ row, column, rowIndex, columnIndex }) {
        if (rowIndex % 2 === 0) {
          if (columnIndex === 0) {
            return [1, 2];
          } else if (columnIndex === 1) {
            return [0, 0];
          }
        }
      },

      objectSpanMethod({ row, column, rowIndex, columnIndex }) {
        if (columnIndex === 0) {
          if (rowIndex % 2 === 0) {
            return {
              rowspan: 2,
              colspan: 1
            };
          } else {
            return {
              rowspan: 0,
              colspan: 0
            };
          }
        }
      }
    }
  };
</script>
```

:::

Custom index

You can customize row index in `type=index` columns. :::demo To customize row indices, use `index` attribute on `el-table-column` with `type=index`. If it is assigned to a number, all indices will have an offset of that number. It also accepts a method with each index (starting from `0`) as parameter, and the returned value will be displayed as index.

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column
      type="index"
      :index="indexMethod">
    </el-table-column>
    <el-table-column
      prop="date"
      label="Date"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="Name"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="Address">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036',
          tag: 'Home'
        }, {
          date: '2016-05-02',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036',
          tag: 'Office'
        }, {
          date: '2016-05-04',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036',
          tag: 'Home'
        }, {
          date: '2016-05-01',
          name: 'Tom',
          state: 'California',
          city: 'Los Angeles',
          address: 'No. 189, Grove St, Los Angeles',
          zip: 'CA 90036',
          tag: 'Office'
        }],
      }
    },
    methods: {
      indexMethod(index) {
        return index * 2;
      }
    }
  };
</script>
```

:::

Table Attributes

Attribute	Description	Type	Accepted Values	Default
data	Table data	array	—	—
height	Table's height. By default it has an `auto` height. If its value is a number, the height is measured in pixels; if its value is a string, the value will be assigned to element's style.height, the height is affected by external styles	string/number	—	—
max-height	Table's max-height. The legal value is a number or the height in px.	string/number	—	—
stripe	whether Table is striped	boolean	—	false

border	whether Table has vertical border	boolean	—	false
size	size of Table	string	medium / small / mini	—
fit	whether width of column automatically fits its container	boolean	—	true
show-header	whether Table header is visible	boolean	—	true
highlight-current-row	whether current row is highlighted	boolean	—	false
current-row-key	key of current row, a set only prop	string,number	—	—
row-class-name	function that returns custom class names for a row, or a string assigning class names for every row	Function({row, rowIndex})/String	—	—
row-style	function that returns custom style for a row, or an object assigning custom style for every row	Function({row, rowIndex})/Object	—	—
cell-class-name	function that returns custom class names for a cell, or a string assigning class names for every cell	Function({row, column, rowIndex, columnIndex})/String	—	—
cell-style	function that returns custom style for a cell, or an object assigning custom style for every cell	Function({row, column, rowIndex, columnIndex})/Object	—	—
header-row-class-name	function that returns custom class names for a row in table header, or a string assigning class names for every row in table header	Function({row, rowIndex})/String	—	—
header-row-style	function that returns custom style for a row in table header, or an object assigning custom style for every row in table header	Function({row, rowIndex})/Object	—	—
header-cell-class-name	function that returns custom class names for a cell in table header, or a string assigning class names for every cell in table header	Function({row, column, rowIndex, columnIndex})/String	—	—

header-cell-style	function that returns custom style for a cell in table header, or an object assigning custom style for every cell in table header	Function({row, column, rowIndex, columnIndex})/Object	—	—
row-key	key of row data, used for optimizing rendering. Required if `reserve-selection` is on or display tree data. When its type is String, multi-level access is supported, e.g. `user.info.id`, but `user.info[0].id` is not supported, in which case `Function` should be used.	Function(row)/String	—	—
empty-text	Displayed text when data is empty. You can customize this area with `slot="empty"`	String	—	No Data
default-expand-all	whether expand all rows by default, works when the table has a column type="expand" or contains tree structure data	Boolean	—	false
expand-row-keys	set expanded rows by this prop, prop's value is the keys of expand rows, you should set row-key before using this prop	Array	—	
default-sort	set the default sort column and order. property `prop` is used to set default sort column, property `order` is used to set default sort order	Object	`order`: ascending, descending	if `prop` is set, and `order` is not set, then `order` is default to ascending
tooltip-effect	tooltip `effect` property	String	dark/light	
show-summary	whether to display a summary row	Boolean	—	false
sum-text	displayed text for the first column of summary row	String	—	Sum
summary-method	custom summary method	Function({ columns, data })	—	—
span-method	method that returns rowspan and colspan	Function({ row, column, rowIndex, columnIndex })	—	—
select-on-indeterminate	controls the behavior of master checkbox in multi-select tables	Boolean	—	true

	when only some rows are selected (but not all). If true, all rows will be selected, else deselected.			
indent	horizontal indentation of tree data	Number	—	16
lazy	whether to lazy loading data	Boolean	—	—
load	method for loading child row data, only works when `lazy` is true	Function(row, treeNode, resolve)	—	—
tree-props	configuration for rendering nested data	Object	—	{ hasChildren: 'hasChildren', children: 'children' }

Table Events

Event Name	Description	Parameters
select	triggers when user clicks the checkbox in a row	selection, row
select-all	triggers when user clicks the checkbox in table header	selection
selection-change	triggers when selection changes	selection
cell-mouse-enter	triggers when hovering into a cell	row, column, cell, event
cell-mouse-leave	triggers when hovering out of a cell	row, column, cell, event
cell-click	triggers when clicking a cell	row, column, cell, event
cell-dblclick	triggers when double clicking a cell	row, column, cell, event
row-click	triggers when clicking a row	row, column, event
row-contextmenu	triggers when user right clicks on a row	row, column, event
row-dblclick	triggers when double clicking a row	row, column, event
header-click	triggers when clicking a column header	column, event
header-contextmenu	triggers when user right clicks on a column header	column, event

sort-change	triggers when Table's sorting changes	{ column, prop, order }
filter-change	column's key. If you need to use the filter-change event, this attribute is mandatory to identify which column is being filtered	filters
current-change	triggers when current row changes	currentRow, oldCurrentRow
header-dragend	triggers after changing a column's width by dragging the column header's border	newWidth, oldWidth, column, event
expand-change	triggers when user expands or collapses a row (for expandable table, second param is expandedRows; for tree Table, second param is expanded)	row, (expandedRows | expanded)

Table Methods

Method	Description	Parameters
clearSelection	used in multiple selection Table, clear user selection	—
toggleRowSelection	used in multiple selection Table, toggle if a certain row is selected. With the second parameter, you can directly set if this row is selected	row, selected
toggleAllSelection	used in multiple selection Table, toggle the selected state of all rows	-
toggleRowExpansion	used in expandable Table or tree Table, toggle if a certain row is expanded. With the second parameter, you can directly set if this row is expanded or collapsed	row, expanded
setCurrentRow	used in single selection Table, set a certain row selected. If called without any parameter, it will clear selection.	row
clearSort	clear sorting, restore data to the original order	—
clearFilter	clear filters of the columns whose `columnKey` are passed in. If no params, clear all filters	columnKeys
doLayout	refresh the layout of Table. When the visibility of Table changes, you may need to call this method to get a correct layout	—
sort	sort Table manually. Property `prop` is used to set sort column, property `order` is used to set sort order	prop: string, order: string

Table Slot

Name	Description
append	Contents to be inserted after the last row. You may need this slot if you want to implement infinite scroll for the table. This slot will be displayed above the summary row if there is one.

Table-column Attributes

Attribute	Description	Type	Accepted Values	Default
type	type of the column. If set to `selection`, the column will display checkbox. If set to `index`, the column will display index of the row (staring from 1). If set to `expand`, the column will display expand icon.	string	selection/index/expand	—
index	customize indices for each row, works on columns with `type=index`	number, Function(index)	-	-
label	column label	string	—	—
column-key	column's key. If you need to use the filter-change event, you need this attribute to identify which column is being filtered	string	string	—
prop	field name. You can also use its alias: `property`	string	—	—
width	column width	string	—	—
min-width	column minimum width. Columns with `width` has a fixed width, while columns with `min-width` has a width that is distributed in proportion	string	—	—
fixed	whether column is fixed at left/right. Will be fixed at left if `true`	string/boolean	true/left/right	—
render-header	render function for table header of this column	Function(h, { column, $index })	—	—
sortable	whether column can be sorted. Remote sorting can be done by setting this attribute to 'custom' and listening to the `sort-change` event of Table	boolean, string	true, false, custom	false
sort-method	sorting method, works when `sortable` is `true`. Should return a number, just like Array.sort	Function(a, b)	—	—

sort-by	specify which property to sort by, works when `sortable` is `true` and `sort-method` is `undefined`. If set to an Array, the column will sequentially sort by the next property if the previous one is equal	Function(row, index)/String/Array	—	—
sort-orders	the order of the sorting strategies used when sorting the data, works when `sortable` is `true`. Accepts an array, as the user clicks on the header, the column is sorted in order of the elements in the array	array	the elements in the array need to be one of the following: `ascending`, `descending` and `null` (restores to the original order)	['ascending', 'descending', null]
resizable	whether column width can be resized, works when `border` of `el-table` is `true`	boolean	—	false
formatter	function that formats cell content	Function(row, column, cellValue, index)	—	—
show-overflow-tooltip	whether to hide extra content and show them in a tooltip when hovering on the cell	boolean	—	false
align	alignment	string	left/center/right	left
header-align	alignment of the table header. If omitted, the value of the above `align` attribute will be applied	String	left/center/right	—
class-name	class name of cells in the column	string	—	—
label-class-name	class name of the label of this column	string	—	—
selectable	function that determines if a certain row can be selected, works when `type` is 'selection'	Function(row, index)	—	—
reserve-selection	whether to reserve selection after data refreshing, works when	boolean	—	false

	`type` is 'selection'. Note that `row-key` is required for this to work			
filters	an array of data filtering options. For each element in this array, `text` and `value` are required	Array[{ text, value }]	—	—
filter-placement	placement for the filter dropdown	String	same as Tooltip's `placement`	—
filter-multiple	whether data filtering supports multiple options	Boolean	—	true
filter-method	data filtering method. If `filter-multiple` is on, this method will be called multiple times for each row, and a row will display if one of the calls returns `true`	Function(value, row, column)	—	—
filtered-value	filter value for selected data, might be useful when table header is rendered with `render-header`	Array	—	—

Table-column Scoped Slot

Name	Description
—	Custom content for table columns. The scope parameter is { row, column, $index }
header	Custom content for table header. The scope parameter is { column, $index }