



JEditableTable 帮助文档

- [JEditableTable 帮助文档](#)
 - [参数配置](#)
 - [columns 参数详解](#)
 - [当 type=checkbox 时所需的参数](#)
 - [当 type=select 时所需的参数](#)
 - [options 所需参数](#)
 - [当 type=upload 时所需的参数](#)
 - [当 type=slot 时所需的参数](#)
 - [validateRules 配置规则](#)
 - [事件](#)
 - [方法](#)
 - [initialize](#)
 - [resetScrollTop](#)
 - [add](#)
 - [removeRows](#)
 - [removeSelectedRows](#)
 - [getValues](#)
 - [getValuesSync](#)
 - [getValuesPromise](#)
 - [getDeleteIds](#)
 - [getAll](#)
 - [setValues](#)
 - [clearSelection](#)
 - [内置插槽](#)
 - [\\${...} 变量使用方式](#)
 - [JEditableTableUtil 使用说明](#)
 - [export 的常量](#)
 - [FormTypes](#)
 - [VALIDATE_NO_PASSED](#)
 - [封装的方法](#)
 - [validateTables](#)
 - [拖拽排序](#)
 - [FAQ](#)
 - [方法如何调用？](#)
 - [如何获取表单的值？](#)
 - [如何进行表单验证？](#)
 - [如何添加或删除一行？](#)
 - [为什么使用了ATab组件后，切换选项卡会导致白屏或滚动条位置会归零？](#)
 - [slot\(自定义插槽\)如何使用？](#)
 - [示例一](#)
 - [示例二](#)
 - [示例三](#)
 - [示例四\(slot\)](#)
 - [示例五](#)

参数	类型	必填	说明
columns	array	✓	表格列的配置描述，具体项见下表
dataSource	array	✓	表格数据
loading	boolean		是否正在加载，加载中不会显示任何行，默认false
actionButton	boolean		是否显示操作按钮，包括"新增"、"删除"，默认false
rowNumber	boolean		是否显示行号，默认false
rowSelection	boolean		是否可选择行，默认false
dragSort	boolean		是否可拖动排序，默认false，详见 拖拽排序 章节
dragSortKey	string		拖动排序存储的Key，无需定义在columns内也能在getValues()时获取到值，默认orderNum
maxHeight	number		设定最大高度(px)，默认400
disabledRows	object		设定禁用的行，被禁用的行无法被选择和编辑，配置方法可以查看示例
disabled	boolean		是否禁用所有行，默认false

columns 参数详解

参数	类型	必填	说明
title	string	✓	表格列头显示的问题
key	string	✓	列数据在数据项中对应的 key，必须是唯一的
type	string	✓	表单的类型，可以通过 <code>JEditableTableUtil.FormTypes</code> 赋值
width	string		列的宽度，可以是百分比，也可以是 <code>px</code> 或其他单位，建议设置为百分比，且每一列的宽度加起来不应超过100%，否则可能会不能达到预期的效果。留空会自动计算百分比
placeholder	string		表单预期值的提示信息，可以使用 <code>\${...}</code> 变量替换文本（详见 <code>\${...}</code> 变量使用方式）
defaultValue	string		默认值，在新增一行时生效
validateRules	array		表单验证规则，配置方式见 validateRules 配置规则
props	object		设置添加给表单元素的自定义属性，例如： <code>props:{title: 'show title'}</code>
disabled	boolean		是否禁用当前列，默认false

当 type=checkbox 时所需的参数

参数	类型	必填	说明
----	----	----	----

☰

参数	类型	必填	说明
defaultChecked	boolean		默认值是否选中
customValue	array		自定义值，checkbox需要的是boolean值，如果数据是其他值（例如 'Y' or 'N' ）时，就会导致错误，所以提供了该属性进行转换，例： customValue: ['Y','N'] ，会将 true 转换为 'Y' ， false 转换为 'N' ，反之亦然

当 type=select 时所需的参数

参数	类型	必填	说明
options	array	✓	下拉选项列表，详见下表
allowInput	boolean		是否允许用户输入内容，并创建新的内容
allowSearch	boolean		是否允许用户搜索内容（仅搜索，不创建）
dictCode	String		数据字典Code，若options也有值，则拼接在options后面

- 注意事项
 - allowInput 和 allowSearch 不能同时使用，若同时使用，则 allowSearch 的优先级更高
 - 若 options 的某一项设置了 disabled:true ，则搜索不到该项

options 所需参数

参数	类型	必填	说明
text	string	✓	显示标题
value	string	✓	真实值
disabled	boolean	✓	是否禁用当前选项
title	string		显示标题（已废弃，若同时填写子title和text那么优先使用text）

当 type=upload 时所需的参数

参数	类型	必填	说明
action	string	✓	上传文件路径
token	boolean		上传的时候是否传递token
responseName	string	✓	若要从上传成功后从response中取出返回的文件名，那么这里填后台返回的包含文件名的字段名

当 type=slot 时所需的参数

参数	类型	必填	说明
slotName	string	✓	slot的名称

validateRules 配置规则

validateRules 需要的是一个数组，数组里每项都是一个规则，规则是object类型，规则的各个参数如下

-
- `required` 是否必填，可选值为 `true` or `false`
 - `unique` 唯一校验，不可重复，可选值为 `true` or `false`
 - `pattern` 正则表达式验证，只有成功匹配该正则的值才能成功通过验证
 - `handler` 自定义函数校验，使用方法请见[示例五](#)，注：设置了handler将不会走validateRules里handler以下的校验了，所以建议将handler放到validateRules的最下面
 - `message` 当验证未通过时显示的提示文本，可以使用 `${...}` 变量替换文本（详见 `${...}` 变量使用方式 ）
 - 配置示例请看[示例二](#)

事件

事件名	触发时机	参数
added	当添加行操作完成后触发	
deleted	当删除行操作完成后触发（批量删除操作只会触发一次）	<code>deleteIds</code> 被逻辑删除的id
selectRowChange	当行被选中或取消选中时触发	<code>selectedRowIds</code> 被选中行的id
valueChange	当数据发生改变的时候触发的事件	<code>{ type, row, column, value, target }</code> Event对象

方法

关于方法的如何调用的问题，请在[FAQ](#)中查看[方法如何调用](#)

initialize

用于初始化表格（清空表格）

- `参数：` 无
- `返回值：` 无

resetScrollTop

重置滚动条Top位置

- `参数：`

参数名	类型	必填	说明
top	number		新top位置，留空则滚动到上次记录的位置，用于解决切换tab选项卡时导致白屏以及自动将滚动条滚动到顶部的问题

- `返回值：` 无

add

主动添加行，默认情况下，当用户的滚动条已经在底部的时候，会将滚动条固定在底部，即添加后无需用户手动滚动，而会自动滚动到底部

- `参数：`

参数名	类型	必填	说明
num	number		添加几行，默认为1
forceScrollToBottom	boolean		是否在添加后无论用户的滚动条在什么位置都强制滚动到底部，默认为false

- `返回值：` 无

removeRows

- 参数：

参数名	类型	必填	说明
id	string 或 array	✓	被删除行的id。如果要删除一个，可以直接传id，如果要删除多个，需要将多个id封装成一个数组传入

- 返回值： 无

removeSelectedRows

主动删除被选中的行

- 参数： 无
- 返回值： 无

getValues

用于获取表格里所有表单的值，可进行表单验证

- 参数：

参数名	类型	必填	说明
callback	function	✓	获取值的回调方法，会传入 <code>error</code> 和 <code>values</code> 两个参数。 <code>error</code> ：未通过验证的数量，当等于 <code>0</code> 时代表验证通过； <code>values</code> ：获取的值（即使未通过验证该字段也有数据）
validate	boolean		是否进行表单验证，默认为 <code>true</code> ，设为 <code>false</code> 则代表忽略表单验证
rowIds	array		默认返回所有行的数据，如果传入了 <code>rowIds</code> ，那么就会只返回与该 <code>rowIds</code> 相匹配的数据，如果没有匹配的数据，就会返回空数组

- 返回值： 无

getValuesSync

`getValues` 的同步版，会直接将获取到的数据返回

- 参数：

参数名	类型	必填	说明
options	object		选项，详见下方所需参数

- `options` 所需参数

参数名	类型	必填	说明
validate	boolean		是否进行表单验证，默认为 <code>true</code> ，设为 <code>false</code> 则代表忽略表单验证

参数名	类型	必填	说明
rowIds	array		默认返回所有行的数据，如果传入了 rowIds ，那么就会只返回与该 rowIds 相匹配的数据，如果没有匹配的数据，就会返回空数组

- 返回值： object
 - error 未通过验证的数量，当等于 0 时代表验证通过
 - values 获取的值（即使未通过验证该字段也有数据）
- 使用示例

```
let { error, values } = this.$refs.editableTable.getValuesSync({ validate: true, rowIds: ['rowId1', 'rowId2'] });
if (error === 0) {
  console.log('表单验证通过，数据：', values);
} else {
  console.log('未通过表单验证，数据：', values);
}
```

getValuesPromise

getValues 的promise版，会在 resolve 中传入获取到的值，会在 reject 中传入失败原因，例如 VALIDATE_NO_PASSED

- 参数：

参数名	类型	必填	说明
validate	boolean		同 getValues 的 validate 参数
rowIds	array		默认返回所有行的数据，如果传入了 rowIds ，那么就会只返回与该 rowIds 相匹配的数据，如果没有匹配的数据，就会返回空数组

- 返回值： Promise

getDeletelds

用于获取被逻辑删除的行的id，返回一个数组，用户可将该数组传入后台，并进行批量删除

- 参数： 无
- 返回值： array

getAll

获取所有的数据，包括values、deletelds

会在 resolve 中传入获取到的值： {values, deleteIds}

会在 reject 中传入失败原因，例如 VALIDATE_NO_PASSED

- 参数：

参数名	类型	必填	说明
validate	boolean		同 getValues 的 validate 参数

- 返回值： Promise

setValues

- 参数：

参数名	类型	必填	说明
values	array		传入一个数组，数组中的每项都是一行的新值，具体见下面的示例

- 返回值： 无
- 示例：

```
setValues([
  {
    rowKey: id1, // 行的id
    values: { // 在这里 values 中的 name 是你 columns 中配置的 key
      'name': 'zhangsan',
      'age': '20'
    }
  },
  {
    rowKey: id2,
    values: {
      'name': 'lisi',
      'age': '23'
    }
  }
])
```

clearSelection

主动清空选择的行

- 参数： 无
- 返回值： 无

内置插槽

插槽名	说明
buttonBefore	在操作按钮的 前面 插入插槽，不受 <code>actionButton</code> 属性的影响
buttonAfter	在操作按钮的 后面 插入插槽，不受 <code>actionButton</code> 属性的影响

\${...} 变量使用方式

在 `placeholder` 和 `message` 这两个属性中可以使用 `${...}` 变量来替换文本
在[示例二](#)中，配置了 `title` 为 `名称` 的一列，而 `placeholder` 配置成了 `请输入${title}`，那么最终显示效果为 `请输入名称`
这就是 `${...}` 变量的使用方式，在 `${}` 中可以使用的变量有 `title`、`key`、`defaultValue` 这三个属性的值

JEditableTableUtil 使用说明

在之前配置 `columns` 时提到过 `JEditableTableUtil` 这个工具类，那么如果要知道详细的使用说明就请看[这里](#)

export 的常量

FormTypes

这是配置 `columns.type` 时用到的常量值，其中包括

- `normal` 默认，直接显示值，不渲染表单
- `input` 显示输入框
- `inputNumber` 显示数字输入框
- `checkbox` 显示多选框

-
- `select` 显示选择器（下拉框）
 - `date` 日期选择器
 - `datetime` 日期时间选择器
 - `upload` 上传组件（文件域）
 - `slot` 自定义插槽
 - `hidden` 隐藏列，被隐藏的列可以正常获取赋予的原始值

VALIDATE_NO_PASSED

在判断表单验证是否通过时使用，如果 `reject` 的值 === `VALIDATE_NO_PASSED` 则代表表单验证未通过，你可以做相应的其他处理，反之则可能是发生了报错，可以使用 `console.error` 输出

封装的方法

validateTables

当你的页面中存在多个 `JEDataTable`实例的时候，如果要获取每个实例的值、判断表单验证是否通过，就会让代码变得极其冗余、繁琐，于是我们就将该操作封装成了一个函数供你调用，它可以同时获取并验证多个 `JEDataTable`实例的值，只有当所有实例的表单验证都通过后会返回 `值`，否则将会告诉你具体哪个实例没有通过验证。具体使用方法请看下面的示例

- 参数：

参数名	类型	必填	说明
cases	array		传入一个数组，数组中的每项都是一个 <code>JEDataTable</code> 的实例

- 返回值： `Promise`
- 示例：

```
import { validateTables, VALIDATE_NO_PASSED } from '@/utils/JEDataTableUtil'
// 封装cases
let cases = []
cases.push(this.$refs.editableTable1)
cases.push(this.$refs.editableTable2)
cases.push(this.$refs.editableTable3)
cases.push(this.$refs.editableTable4)
cases.push(this.$refs.editableTable5)
// 同时验证并获取多个实例的值
validateTables(cases).then((all) => {
  // all 是一个数组，每项都对应传入cases的下标，包含values和deleteIds
  console.log('所有实例的值：', all)
}).catch((e = {}) => {
  // 判断表单验证是否未通过
  if (e.error === VALIDATE_NO_PASSED) {
    console.log('未通过验证的实例下标:', e.index)
  } else {
    console.error('发生异常:', e)
  }
})
```

拖拽排序

若要启用拖拽排序，只需将 `dragSort` 参数置为 `true` 即可。

开启后，在列表的最左侧会出现一个方块，如果用鼠标左键按住不动拖动的话，就可以移动整行，如果单击一下小方块的话，就会出现一个菜单，如下图所示。

拖拽这里可以排序整行



单击可弹出以下菜单



若要持久化排序顺序的话，需要传递 `dragSortKey` 参数，即你数据库中记录排序顺序的字段。该参数也可不传，不传默认为 `orderNum`

即使你 `columns` 里没有定义这个字段，也可以正常获取到值，并存到数据库里。

但当给 `dataSouce` 赋值的时候，并不会自动处理顺序，因为这样系统性能开销太大了，所以建议查询的时候用SQL语句排好序再返回给组件使用。

FAQ

方法如何调用？

在[示例一](#)中，设定了一个 `ref="editableTable"` 的属性，那么在vue中就可以使用 `this.$refs.editableTable` 获取到该表格的实例，并调取其中的方法。

假如我要调取 `initialize` 方法，就可以这么写：`this.$refs.editableTable.initialize()`

如何获取表单的值？

使用 `getValue` 方法进行获取，详见[示例三](#)

如何进行表单验证？

在获取值的时候默认会进行表单验证操作，用户在输入的时候也会对正在输入的表单进行验证，只要配置好规则就可以了

如何添加或删除一行？

该功能已封装到组件中，你只需要将 `actionButton` 设置为 `true` 即可，当然你也可以在代码中主动调用新增方法或修改，具体见上方的方法介绍。

为什么使用了ATab组件后，切换选项卡会导致白屏或滚动条位置会归零？

在ATab组件中确实会导致滚动条位置归零，且不会触发 `onscroll` 方法，所以无法动态加载行，导致白屏的问题出现。

解决方法是在ATab组件的 `onChange` 事件触发时执行实例提供的 `resetScrollTop()` 方法即可，但是需要注意的是：代码主动改变ATab的 `activeKey` 不会触发 `onChange` 事件，还需要你手动调用下。

- [示例](#)

```
<template>
  <a-tabs @change="handleChangeTab">
    <a-tab-pane tab="表格1" :forceRender="true" key="1">
      <j-editable-table
        ref="editableTable1"
        :loading="tab1.loading"
        :columns="tab1.columns"
        :dataSource="tab1.dataSource"/>
    </a-tab-pane>
    <a-tab-pane tab="表格2" :forceRender="true" key="2">
      <j-editable-table
        ref="editableTable2"
        :loading="tab2.loading"
        :columns="tab2.columns"
        :dataSource="tab2.dataSource"/>
    </a-tab-pane>
  </a-tabs>
</template>
```



```
</a-tabs>
</template>
```

```
/*--- 忽略部分代码片段 ---*/
methods: {

    /** 切换tab选项卡的时候重置editableTable的滚动条状态 */
    handleChangeTab(key) {
        this.$refs[`editableTable${key}`].resetScrollTop()
    }

}
/*--- 忽略部分代码片段 ---*/
```

slot(自定义插槽)如何使用？

代码示例请看：[示例四\(slot\)](#)

示例一

```
<j-editable-table
  ref="editableTable"
  :loading="loading"
  :columns="columns"
  :dataSource="dataSource"
  :rowNumber="true"
  :rowSelection="true"
  :actionButton="true"
  style="margin-top: 8px;"
  @selectRowChange="handleSelectRowChange"/>
```

示例二

```
import { FormTypes } from '@/utils/JEditableTableUtil'

/*--- 忽略部分代码片断 ---*/
columns: [
  {
    title: '名称',
    key: 'name',
    type: FormTypes.input,
    placeholder: '请输入${title}',
    defaultValue: '称名',
    // 表单验证规则
    validateRules: [
      {
        required: true, // 必填
        message: '${title}不能为空' // 提示的文本
      },
      {
        pattern: /^[a-z|A-Z][a-z|A-Z\d_-]{0,}$/ , // 正则
        message: '${title}必须以字母开头，可包含数字、下划线、横杠'
      }
    ]
  },
  {
    title: '年龄',
    key: 'age',
    type: FormTypes.inputNumber,
    placeholder: '请输入${title}',
    defaultValue: 18,
    validateRules: [{required: true, message: '${title}不能为空'}]
  }
]
/*--- 忽略部分代码片断 ---*/
```

```
// 获取被逻辑删除的字段id
let deleteIds = this.$refs.editableTable.getDeleteIds();
// 获取所有表单的值，并进行验证
this.$refs.editableTable.getValues((error, values) => {
  // 错误数 = 0 则代表验证通过
  if (error === 0) {
    this.$message.success('验证通过')
    // 将通过后的数组提交到后台或自行进行其他处理
    console.log(deleteIds, values)
  } else {
    this.$message.error('验证未通过')
  }
})
})
```

示例四(slot)

```
<template>
  <j-editable-table :columns="columns" :dataSource="dataSource">
    <!-- 定义插槽 -->
    <!-- 这种定义插槽的写法是vue推荐的新版写法 (https://cn.vuejs.org/v2/guide/components-slots.html#具名插槽) -->
    <!-- 若webstorm这样写报错，请看这篇文章：https://blog.csdn.net/lxq_9532/article/details/81870651 -->
    <template v-slot:action="props">
      <a @click="handleDelete(props)">删除</a>
    </template>
  </j-editable-table>
</template>
<script>
import { FormTypes } from '@/utils/JEditableTableUtil'
import JEditableTable from '@/components/jeecg/JEditableTable'
export default {
  components: { JEditableTable },
  data() {
    return {
      columns: [
        // ...
        {
          title: '操作',
          key: 'action',
          width: '8%',
          type: FormTypes.slot, // 定义该列为 自定义插值列
          slotName: 'action' // slot 的名称，对应 v-slot 冒号后面和等号前面的内容
        }
      ]
    }
  },
  methods: {
    /* a 标签的点击事件，删除当前选中的行 */
    handleDelete(props) {
      // 参数解释
      // props.index : 当前行的下标
      // props.text : 当前值，可能是defaultValue定义的值，也可能是从dataSource中取出的值
      // props.rowId : 当前选中行的id，如果是新增行则是临时id
      // props.column : 当前操作的列
      // props.getValue : 这是一个function，执行后可以获取当前行的所有值（禁止在template中使用）
      // 例: const value = props.getValue()
      // props.target : 触发当前事件的实例，可直接调用该实例内的方法（禁止在template中使用）
      // 例: target.add()

      // 使用实例：删除当前操作的行
      let { rowId, target } = props
      target.removeRows(rowId)
    }
  }
}
</script>
```

```
// 该示例是自定义函数校验
columns: [
  {
    title: '字段名称',
    key: 'dbFieldName',
    type: FormTypes.input,
    defaultValue: '',
    validateRules: [
      {
        // 自定义函数校验 handler
        handler(type, value, row, column, callback, target) {
          // type 触发校验的类型 (input、change、blur)
          // value 当前校验的值
          // callback(flag, message) 方法必须执行且只能执行一次
          // flag = 是否通过了校验, 不填写或者填写 null 代表不进行任何操作
          // message = 提示的消息, 默认使用配置的 message
          // target 行编辑的实例对象

          if (type === 'blur') {

            if (value === 'abc') {
              callback(false, '${title}不能是abc') // false = 未通过, 可以跟自定义提示
              return
            }

            let { values } = target.getValuesSync({ validate: false })
            let count = 0
            for (let val of values) {
              if (val['dbFieldName'] === value) {
                if (++count >= 2) {
                  callback(false, '${title}不能重复')
                  return
                }
              }
            }
            callback(true) // true = 通过验证
          } else {
            callback() // 不填写或者填写 null 代表不进行任何操作
          }
        },
        message: '${title}默认提示'
      }
    ]
  },
],
],
```