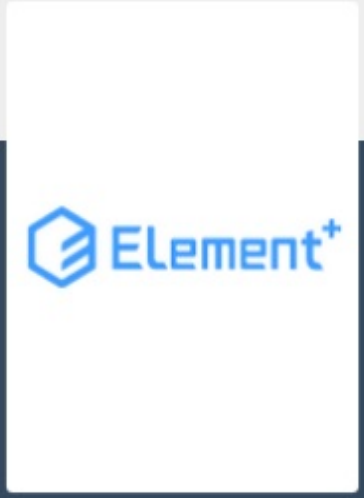
Element Plus 教程(beta版，适配 Vue 3.0 的 Element)



Element 开发团队宣布推出 Element Plus 首个 Beta 版本，官方表示 Element Plus 是 Element 对 Vue 3.0 的升级适配。Element Plus 是首个使用 TypeScript + Vue 3.0 Composition API 重构的组件库。由于 Vue 3.0 进行了大版本升…

目　录

致谢

当前文档 《Element Plus 教程(beta版，适配 Vue 3.0 的 Element)》 由 进击的皇虫 使用 书栈网(BookStack.CN) 进行构建，生成于 2020-11-30。

书栈网仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈网难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到书栈网，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到书栈网获取最新的文档，以跟上知识更新换代的步伐。

内容来源：饿了么 https://element-plus.gitee.io/#/zh-CN

文档地址：http://www.bookstack.cn/books/element-plus-beta

书栈官网：https://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

安装

npm 安装

推荐使用 npm 的方式安装，它能更好地和 webpack 打包工具配合使用。

```
npm install element-plus --save
```

CDN

目前可以通过 unpkg.com/element-plus 获取到最新版本的资源，在页面上引入 js 和 css 文件即可开始使用。

```
<!-- 引入样式 -->
<link rel="stylesheet" href="https://unpkg.com/element-plus/lib/theme-chalk/index.css">
<!-- 引入组件库 -->
<script src="https://unpkg.com/element-plus/lib/index.full.js"></script>
```

我们建议使用 CDN 引入 Element Plus 的用户在链接地址上锁定版本，以免将来 Element Plus 升级时受到非兼容性更新的影响。锁定版本的方法请查看 unpkg.com。

Hello world

通过 CDN 的方式我们可以很容易地使用 Element Plus 写出一个 Hello world 页面。在线演示

如果是通过 npm 安装，并希望配合 webpack 使用，请阅读下一节：快速上手。

快速上手

本节将介绍如何在项目中使用 Element。

使用 vue-cli@4.5

我们为新版的 vue-cli 准备了相应的 Element Plus 插件，你可以用它们快速地搭建一个基于 Element Plus 的项目。

使用 Starter Kit

我们提供了通用的项目模板，你可以直接使用，另外我们还提供了 Vite 模板。对于 Laravel 用户，我们也准备了相应的模板，同样可以直接下载使用。

如果不希望使用我们提供的模板，请继续阅读。

引入 Element Plus

你可以引入整个 Element Plus，或是根据需要仅引入部分组件。我们先介绍如何引入完整的 Element。

完整引入

在 main.js 中写入以下内容：

```
import { createApp } from 'vue'
import ElementPlus from 'element-plus';
import 'element-plus/lib/theme-chalk/index.css';
import App from './App.vue';

const app = createApp(App)
app.use(ElementPlus)
app.mount('#app')
```

以上代码便完成了 Element Plus 的引入。需要注意的是，样式文件需要单独引入。

按需引入

借助 babel-plugin-component，我们可以只引入需要的组件，以达到减小项目体积的目的。

首先，安装 babel-plugin-component：

```
npm install babel-plugin-component -D
```

然后，将 .babelrc 修改为：

```
{
  "plugins": [
    [
      "component",
      {
        "libraryName": "element-plus",
        "styleLibraryName": "theme-chalk"
      }
    ]
  ]
}
```

接下来，如果你只希望引入部分组件，比如 Button 和 Select，那么需要在 main.js 中写入以下内容：

```
import { createApp } from 'vue'
import { ElButton, ElSelect } from 'element-plus';
import App from './App.vue';

const app = createApp(App)
app.component(ElButton.name, ElButton);
app.component(ElSelect.name, ElSelect);

/* or
 * app.use(ElButton)
 * app.use(ElSelect)
 */

app.mount('#app')
```

完整组件列表和引入方式（完整组件列表以 reference 为准）

```
import { createApp } from 'vue'
import App from './App.vue';
import {
  ElAlert,
  ElAside,
```

```
    ElAutocomplete,
    ElAvatar,
    ElBacktop,
    ElBadge,
    ElBreadcrumb,
    ElBreadcrumbItem,
    ElButton,
    ElButtonGroup,
    ElCalendar,
    ElCard,
    ElCarousel,
    ElCarouselItem,
    ElCascader,
    ElCascaderPanel,
    ElCheckbox,
    ElCheckboxButton,
    ElCheckboxGroup,
    ElCol,
    ElCollapse,
    ElCollapseItem,
    ElCollapseTransition,
    ElColorPicker,
    ElContainer,
    ElDatePicker,
    ElDialog,
    ElDivider,
    ElDrawer,
    ElDropdown,
    ElDropdownItem,
    ElDropdownMenu,
    ElFooter,
    ElForm,
    ElFormItem,
    ElHeader,
    ElIcon,
    ElImage,
    ElInput,
    ElInputNumber,
    ElLink,
    ElMain,
    ElMenu,
    ElMenuItem,
```

```
48.   ElMenuItemGroup,
49.   ElOption,
50.   ElOptionGroup,
51.   ElPageHeader,
52.   ElPagination,
53.   ElPopconfirm,
54.   ElPopover,
55.   ElPopper,
56.   ElProgress,
57.   ElRadio,
58.   ElRadioButton,
59.   ElRadioGroup,
60.   ElRate,
61.   ElRow,
62.   ElScrollBar,
63.   ElSelect,
64.   ElSlider,
65.   ElStep,
66.   ElSteps,
67.   ElSubmenu,
68.   ElSwitch,
69.   ElTabPane,
70.   ElTable,
71.   ElTableColumn,
72.   ElTimeline,
73.   ElTimelineItem,
74.   ElTooltip,
75.   ElTransfer,
76.   ElTree,
77.   ElUpload,
78.   ElInfiniteScroll,
79.   ElLoading,
80.   ElMessage,
81.   ElMessageBox,
82.   ElNotification,
83. } from 'element-plus';
84.
85. const components = [
86.   ElAlert,
87.   ElAside,
88.   ElAutocomplete,
89.   ElAvatar,
```

```
    ElBacktop,
    ElBadge,
    ElBreadcrumb,
    ElBreadcrumbItem,
    ElButton,
    ElButtonGroup,
    ElCalendar,
    ElCard,
    ElCarousel,
    ElCarouselItem,
    ElCascader,
    ElCascaderPanel,
    ElCheckbox,
    ElCheckboxButton,
    ElCheckboxGroup,
    ElCol,
    ElCollapse,
    ElCollapseItem,
    ElCollapseTransition,
    ElColorPicker,
    ElContainer,
    ElDatePicker,
    ElDialog,
    ElDivider,
    ElDrawer,
    ElDropdown,
    ElDropdownItem,
    ElDropdownMenu,
    ElFooter,
    ElForm,
    ElFormItem,
    ElHeader,
    ElIcon,
    ElImage,
    ElInput,
    ElInputNumber,
    ElLink,
    ElMain,
    ElMenu,
    ElMenuItem,
    ElMenuItemGroup,
    ElOption,
```

```
    ElOptionGroup,
    ElPageHeader,
    ElPagination,
    ElPopconfirm,
    ElPopover,
    ElPopper,
    ElProgress,
    ElRadio,
    ElRadioButton,
    ElRadioGroup,
    ElRate,
    ElRow,
    ElScrollBar,
    ElSelect,
    ElSlider,
    ElStep,
    ElSteps,
    ElSubmenu,
    ElSwitch,
    ElTabPane,
    ElTable,
    ElTableColumn,
    ElTabs,
    ElTag,
    ElTimePicker,
    ElTimeSelect,
    ElTimeline,
    ElTimelineItem,
    ElTooltip,
    ElTransfer,
    ElTree,
    ElUpload,
  ]

  const plugins = [
    ElInfiniteScroll,
    ElLoading,
    ElMessage,
    ElMessageBox,
    ElNotification,
  ]

```

```
const app = createApp(App)

components.forEach(component => {
  app.component(component.name, component)
})

plugins.forEach(plugin => {
  app.use(plugin)
})
```

全局配置

在引入 Element Plus 时，可以传入一个全局配置对象。该对象目前支持 `size` 与 `zIndex` 字段。`size` 用于改变组件的默认尺寸，`zIndex` 设置弹框的初始 z-index（默认值：2000）。按照引入 Element Plus 的方式，具体操作如下：

完整引入 Element：

```
import { createApp } from 'vue'
import ElementPlus from 'element-plus';
import App from './App.vue';

const app = createApp(App)
app.use(ElementPlus, { size: 'small', zIndex: 3000 });
```

按需引入 Element：

```
import { createApp } from 'vue'
import { ElButton } from 'element-plus';
import App from './App.vue';

const app = createApp(App)
app.config.globalProperties.$ELEMENT = option
app.use(ElButton);
```

按照以上设置，项目中所有拥有 `size` 属性的组件的默认尺寸均为 ‘small’，弹框的初始 z-index 为 3000。

开始使用

至此，一个基于 Vue 和 Element Plus 的开发环境已经搭建完毕，现在就可以编写代码了。各个组件的使用方法请参阅它们各自的文档。

使用 Nuxt.js

我们还可以使用 Nuxt.js:

国际化

Element Plus 组件内部默认使用英语，若希望使用其他语言，则需要进行多语言设置。以中文为例，在 main.js 中:

```
// 完整引入 Element
import { createApp } from 'vue'
import ElementPlus from 'element-plus'
import locale from 'element-plus/lib/locale/lang/zh-cn'

createApp(App).use(ElementPlus, { locale })
```

或

```
// 按需引入 Element
import Vue from 'vue'
import { ElButton, ElSelect } from 'element-plus'
import lang from 'element-plus/lib/locale/lang/zh-cn'
import locale from 'element-plus/lib/locale'

// 设置语言
locale.use(lang)

// 引入组件
Vue.component(ElButton.name, ElButton)
Vue.component(ElSelect.name, ElSelect)
```

如果使用其它语言，默认情况下英文语言包依旧是被引入的，可以使用 webpack 的 NormalModuleReplacementPlugin 替换默认语言包。

webpack.config.js

```
{
  plugins: [
    new webpack.NormalModuleReplacementPlugin(/element-plus[\/\\]lib[\/\\]locale[\/\\]lang[\/\\]en/, 'element-plus/lib/locale/lang/zh-cn')
  ]
}
```

兼容 `vue-i18n@5.x`

Element Plus 兼容 vue-i18n@5.x，搭配使用能更方便地实现多语言切换。

```
import Vue from 'vue'
import VueI18n from 'vue-i18n'
import ElementPlus from 'element-plus'
import enLocale from 'element-plus/lib/locale/lang/en'
import zhLocale from 'element-plus/lib/locale/lang/zh-cn'
import App from './App.vue';

const app = createApp(App)
app.use(ElementPlus)
Vue.use(VueI18n)

Vue.config.lang = 'zh-cn'
Vue.locale('zh-cn', zhLocale)
Vue.locale('en', enLocale)
```

兼容其他 i18n 插件

如果不使用 `vue-i18n@5.x` ，而是用其他的 i18n 插件，Element Plus 将无法兼容，但是可以自定义 Element Plus 的 i18n 的处理方法。

```
import Vue from 'vue'
import ElementPlus from 'element-plus'
import enLocale from 'element-plus/lib/locale/lang/en'
import zhLocale from 'element-plus/lib/locale/lang/zh-cn'

Vue.use(Element, {
  i18n: function (path, options) {
    // ...
  }
})
```

兼容 `vue-i18n@6.x`

默认不支持 6.x 的 `vue-i18n` ，你需要手动处理。

```
import Vue from 'vue'
```

```
import ElementPlus from 'element-plus'
import VueI18n from 'vue-i18n'
import enLocale from 'element-plus/lib/locale/lang/en'
import zhLocale from 'element-plus/lib/locale/lang/zh-cn'

Vue.use(VueI18n)

const messages = {
  en: {
    message: 'hello',
    ...enLocale // 或者用 Object.assign({ message: 'hello' }, enLocale)
  },
  zh: {
    message: '你好',
    ...zhLocale // 或者用 Object.assign({ message: '你好' }, zhLocale)
  }
}
// Create VueI18n instance with options
const i18n = new VueI18n({
  locale: 'en', // set locale
  messages, // set locale messages
})

Vue.use(Element, {
  i18n: (key, value) => i18n.t(key, value)
})

new Vue({ i18n }).$mount('#app')
```

按需加载里定制 i18n

```
import Vue from 'vue'
import DatePicker from 'element/lib/date-picker'
import VueI18n from 'vue-i18n'

import enLocale from 'element-plus/lib/locale/lang/en'
import zhLocale from 'element-plus/lib/locale/lang/zh-cn'
import ElementLocale from 'element-plus/lib/locale'

Vue.use(VueI18n)
Vue.use(DatePicker)
```

```
11.
12. const messages = {
13.   en: {
14.     message: 'hello',
15.     ...enLocale
16.   },
17.   zh: {
18.     message: '你好',
19.     ...zhLocale
20.   }
21. }
22. // Create VueI18n instance with options
23. const i18n = new VueI18n({
24.   locale: 'en', // set locale
25.   messages, // set locale messages
26. })
27.
28. ElementLocale.i18n((key, value) => i18n.t(key, value))
```

通过 CDN 的方式加载语言文件

```
1. <script src="//unpkg.com/vue"></script>
2. <script src="//unpkg.com/element-plus"></script>
3. <script src="//unpkg.com/element-plus/lib/umd/locale/en.js"></script>
4.
5. <script>
6.   ELEMENT.locale(ELEMENT.lang.en)
7. </script>
```

搭配 `vue-i18n` 使用

```
1. <script src="//unpkg.com/vue"></script>
2. <script src="//unpkg.com/vue-i18n/dist/vue-i18n.js"></script>
3. <script src="//unpkg.com/element-plus"></script>
4. <script src="//unpkg.com/element-plus/lib/umd/locale/zh-cn.js"></script>
5. <script src="//unpkg.com/element-plus/lib/umd/locale/en.js"></script>
6.
7. <script>
8.   Vue.locale('en', ELEMENT.lang.en)
9.   Vue.locale('zh-cn', ELEMENT.lang.zhCN)
10. </script>
```

目前 Element Plus 内置了以下语言:

- 简体中文（zh-cn）
- 英语（en）
- 德语（de）
- 葡萄牙语（pt）
- 西班牙语（es）
- 丹麦语（da）
- 法语（fr）
- 挪威语（nb-no）
- 繁体中文（zh-tw）
- 意大利语（it）
- 韩语（ko）
- 日语（ja）
- 荷兰语（nl）
- 越南语（vi）
- 俄语（ru）
- 土耳其语（tr）
- 巴西葡萄牙语（pt-br）
- 波斯语（fa）
- 泰语（th）
- 印尼语（id）
- 保加利亚语（bg）
- 波兰语（pl）
- 芬兰语（fi）
- 瑞典语（sv）
- 希腊语（el）
- 斯洛伐克语（sk）
- 加泰罗尼亚语（ca）
- 捷克语（cs）
- 乌克兰语（uk）
- 土库曼语（tk）
- 泰米尔语（ta）
- 拉脱维亚语（lv）
- 南非荷兰语（af）
- 爱沙尼亚语（et）
- 斯洛文尼亚语（sl）
- 阿拉伯语（ar）
- 希伯来语（he）

- 立陶宛语（lt）
- 蒙古语（mn）
- 哈萨克斯坦语（kk）
- 匈牙利语（hu）
- 罗马尼亚语（ro）
- 库尔德语（ku）
- 维吾尔语（ug-cn）
- 高棉语（km）
- 塞尔维亚语（sr）
- 巴斯克语（eu）
- 吉尔吉斯语（ky）
- 亚美尼亚语（hy-am）
- 克罗地亚（hr）
- 世界语（eo）

如果你需要使用其他的语言，欢迎贡献 PR：只需在 这里 添加一个语言配置文件即可。

自定义主题

Element Plus 默认提供一套主题，CSS 命名采用 BEM 的风格，方便使用者覆盖样式。我们提供了四种方法，可以进行不同程度的样式自定义。

仅替换主题色

如果仅希望更换 Element Plus 的主题色，推荐使用在线主题生成工具。Element Plus 默认的主题色是鲜艳、友好的蓝色。通过替换主题色，能够让 Element Plus 的视觉更加符合具体项目的定位。

使用上述工具，可以很方便地实时预览主题色改变之后的视觉，同时它还可以基于新的主题色生成完整的样式文件包，供直接下载使用（关于如何使用下载的主题包，请参考本节「引入自定义主题」和「搭配插件按需引入组件主题」部分）。

在项目中改变 SCSS 变量

Element Plus 的 theme-chalk 使用 SCSS 编写，如果你的项目也使用了 SCSS，那么可以直接在项目中改变 Element Plus 的样式变量。新建一个样式文件，例如 `element-variables.scss` ，写入以下内容：

```
/* 改变主题色变量 */
$--color-primary: teal;

/* 改变 icon 字体路径变量，必需 */
$--font-path: '~element-plus/lib/theme-chalk/fonts';

@import "~element-plus/packages/theme-chalk/src/index";
```

之后，在项目的入口文件中，直接引入以上样式文件即可（无需引入 Element Plus 编译好的 CSS 文件）：

```
import Vue from 'vue'
import ElementPlus from 'element-plus'
import './element-variables.scss'
import App from './App.vue';

const app = createApp(App)
app.use(ElementPlus)
```

需要注意的是，覆盖字体路径变量是必需的，将其赋值为 Element Plus 中 icon 图标所在的相对路径即可。

命令行主题工具

如果你的项目没有使用 SCSS，那么可以使用命令行主题工具进行深层次的主题定制：

安装工具

首先安装「主题生成工具」，可以全局安装或者安装在当前项目下，推荐安装在项目里，方便别人 clone 项目时能直接安装依赖并启动，这里以全局安装做演示。

```
npm i element-theme -g
```

安装白垩主题，可以从 npm 安装或者从 GitHub 拉取最新代码。

```
# 从 npm
npm i element-theme-chalk -D

# 从 GitHub
npm i https://github.com/ElementUI/theme-chalk -D
```

初始化变量文件

主题生成工具安装成功后，如果全局安装可以在命令行里通过 `et` 调用工具，如果安装在当前目录下，需要通过 `node_modules/.bin/et` 访问到命令。执行 `-i` 初始化变量文件。默认输出到 `element-variables.scss` ，当然你可以传参数指定文件输出目录。

```
et -i [可以自定义变量文件]

> ✔ Generator variables file
```

如果使用默认配置，执行后当前目录会有一个 `element-variables.scss` 文件。内部包含了主题所用到的所有变量，它们使用 SCSS 的格式定义。大致结构如下：

```
$--color-primary: #409EFF !default;
$--color-primary-light-1: mix($--color-white, $--color-primary, 10%) !default;
/* 53a8ff */
$--color-primary-light-2: mix($--color-white, $--color-primary, 20%) !default;
/* 66b1ff */
$--color-primary-light-3: mix($--color-white, $--color-primary, 30%) !default;
/* 79bbff */
```

```
    $--color-primary-light-4: mix($--color-white, $--color-primary, 40%) !default;
5.  /* 8cc5ff */
    $--color-primary-light-5: mix($--color-white, $--color-primary, 50%) !default;
6.  /* a0cfff */
    $--color-primary-light-6: mix($--color-white, $--color-primary, 60%) !default;
7.  /* b3d8ff */
    $--color-primary-light-7: mix($--color-white, $--color-primary, 70%) !default;
8.  /* c6e2ff */
    $--color-primary-light-8: mix($--color-white, $--color-primary, 80%) !default;
9.  /* d9ecff */
    $--color-primary-light-9: mix($--color-white, $--color-primary, 90%) !default;
10. /* ecf5ff */
11.
12. $--color-success: #67c23a !default;
13. $--color-warning: #e6a23c !default;
14. $--color-danger: #f56c6c !default;
15. $--color-info: #909399 !default;
16.
17. ...
```

修改变量

直接编辑 `element-variables.scss` 文件，例如修改主题色为红色。

```
1.  $--color-primary: red;
```

编译主题

保存文件后，到命令行里执行 `et` 编译主题，如果你想启用 `watch` 模式，实时编译主题，增加 `-w` 参数；如果你在初始化时指定了自定义变量文件，则需要增加 `-c` 参数，并带上你的变量文件名。默认情况下编译的主题目录是放在 `./theme` 下，你可以通过 `-o` 参数指定打包目录。

```
1.  et
2.
3.  > ✔ build theme font
4.  > ✔ build element theme
```

使用自定义主题

引入自定义主题

和引入默认主题一样，在代码里直接引用「在线主题编辑器」或「命令行工具」生成的主题的 `theme/index.css` 文件即可。

```
import { createApp } from 'vue'
import '../theme/index.css'
import ElementPlus from 'element-plus'

createApp(App).use(ElementPlus)
```

搭配插件按需引入组件主题

如果是搭配 `babel-plugin-component` 一起使用，只需要修改 `.babelrc` 的配置，指定 `styleLibraryName` 路径为自定义主题相对于 `.babelrc` 的路径，注意要加 `~` 。

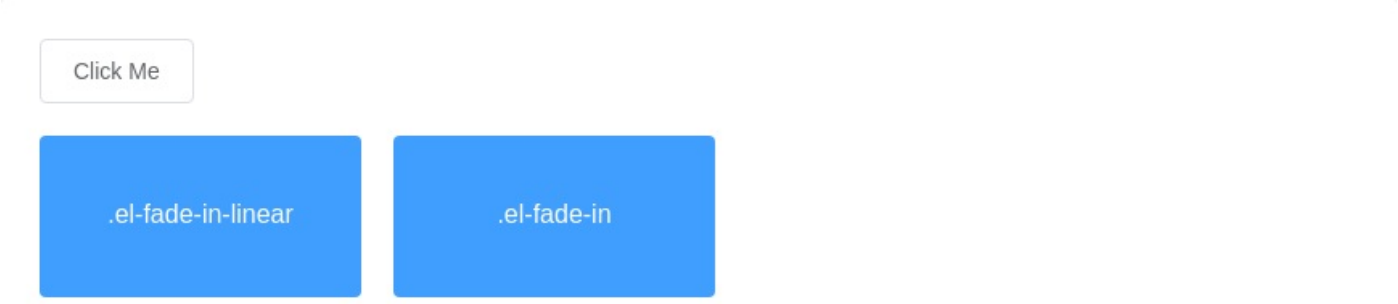
```
{
  "plugins": [
    [
      "component",
      {
        "libraryName": "element-plus",
        "styleLibraryName": "~theme"
      }
    ]
  ]
}
```

如果不清楚 `babel-plugin-component` 是什么，请阅读 快速上手 一节。更多 `element-theme` 用法请参考项目仓库。

内置过渡动画

Element Plus 内应用在部分组件的过渡动画，你也可以直接使用。在使用之前请阅读 transition 组件文档 。

fade 淡入淡出



提供 `el-fade-in-linear` 和 `el-fade-in` 两种效果。

```
<template>
  <div>
    <el-button @click="show = !show">Click Me</el-button>

    <div style="display: flex; margin-top: 20px; height: 100px;">
      <transition name="el-fade-in-linear">
        <div v-show="show" class="transition-box">.el-fade-in-linear</div>
      </transition>
      <transition name="el-fade-in">
        <div v-show="show" class="transition-box">.el-fade-in</div>
      </transition>
    </div>
  </div>
</template>

<script>
    export default {
    data: () => ({
      show: true
    })
  }
</script>

```

```
<style>
  .transition-box {
    margin-bottom: 10px;
    width: 200px;
    height: 100px;
    border-radius: 4px;
    background-color: #409EFF;
    text-align: center;
    color: #fff;
    padding: 40px 20px;
    box-sizing: border-box;
    margin-right: 20px;
  }
</style>
```

zoom 缩放

提供 `el-zoom-in-center` , `el-zoom-in-top` 和 `el-zoom-in-bottom` 三种效果。

```
<template>
  <div>
    <el-button @click="show2 = !show2">Click Me</el-button>

    <div style="display: flex; margin-top: 20px; height: 100px;">
      <transition name="el-zoom-in-center">
        <div v-show="show2" class="transition-box">.el-zoom-in-center</div>
      </transition>

      <transition name="el-zoom-in-top">
        <div v-show="show2" class="transition-box">.el-zoom-in-top</div>
      </transition>

      <transition name="el-zoom-in-bottom">
        <div v-show="show2" class="transition-box">.el-zoom-in-bottom</div>
```

```
      </transition>
    </div>
  </div>
</template>

<script>
    export default {
    data: () => ({
      show2: true
    })
  }
</script>

<style>
  .transition-box {
    margin-bottom: 10px;
    width: 200px;
    height: 100px;
    border-radius: 4px;
    background-color: #409EFF;
    text-align: center;
    color: #fff;
    padding: 40px 20px;
    box-sizing: border-box;
    margin-right: 20px;
  }
</style>
```

collapse 展开折叠

使用 `el-collapse-transition` 组件实现折叠展开效果。


```
<template>
  <div>
    <el-button @click="show3 = !show3">Click Me</el-button>

    <div style="margin-top: 20px; height: 200px;">
      <el-collapse-transition>
        <div v-show="show3">
          <div class="transition-box">el-collapse-transition</div>
          <div class="transition-box">el-collapse-transition</div>
        </div>
      </el-collapse-transition>
    </div>
  </div>
</template>

<script>
    export default {
    data: () => ({
      show3: true
    })
  }
</script>

<style>
  .transition-box {
    margin-bottom: 10px;
    width: 200px;
    height: 100px;
    border-radius: 4px;
    background-color: #409EFF;
```

```
    text-align: center;
    color: #fff;
    padding: 40px 20px;
    box-sizing: border-box;
    margin-right: 20px;
  }
</style>
```

按需引入

```
// fade/zoom 等
import 'element-plus/lib/theme-chalk/base.css';
// collapse 展开折叠
import { ElCollapseTransition } from 'element-plus';
import Vue from 'vue'

Vue.component(ElCollapseTransition.name, ElCollapseTransition)
```

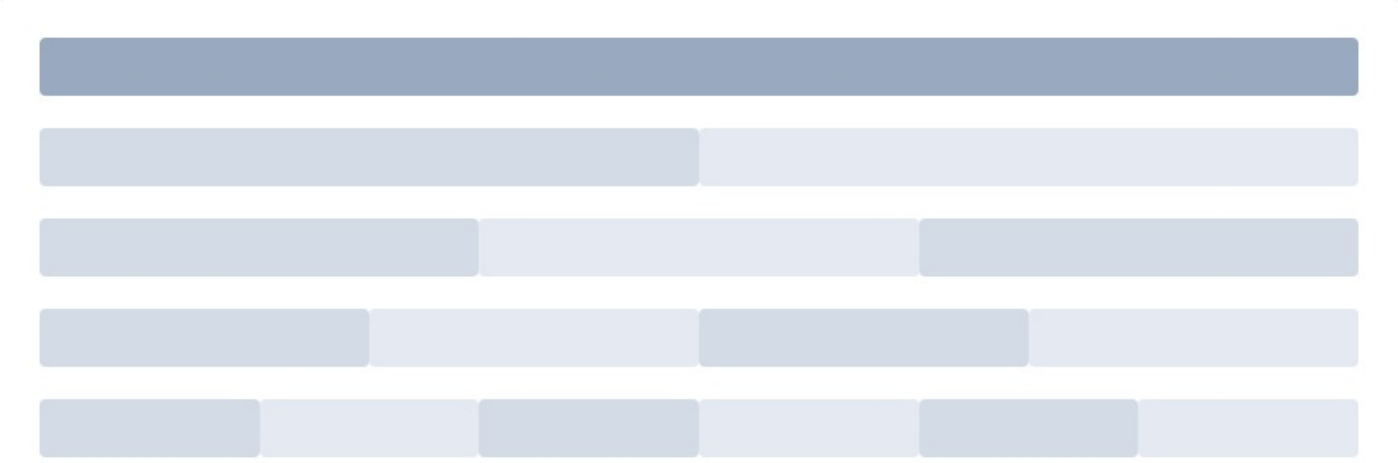
- Layout 布局
- Container 布局容器
- Color 色彩
- Typography 字体
- Border 边框
- Icon 图标
- Button 按钮
- Link 文字链接

Layout 布局

通过基础的 24 分栏，迅速简便地创建布局。

基础布局

使用单一分栏创建基础的栅格布局。



通过 row 和 col 组件，并通过 col 组件的 `span` 属性我们就可以自由地组合布局。

```
<el-row>
  <el-col :span="24"><div class="grid-content bg-purple-dark"></div></el-col>
</el-row>
<el-row>
  <el-col :span="12"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="12"><div class="grid-content bg-purple-light"></div></el-col>
</el-row>
<el-row>
  <el-col :span="8"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="8"><div class="grid-content bg-purple-light"></div></el-col>
  <el-col :span="8"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
</el-row>
<el-row>
  <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
```

```
    <el-col :span="4"><div class="grid-content bg-purple-light"></div></el-col>
    <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
    <el-col :span="4"><div class="grid-content bg-purple-light"></div></el-col>
    <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
    <el-col :span="4"><div class="grid-content bg-purple-light"></div></el-col>
  </el-row>

  <style>
    .el-row {
      margin-bottom: 20px;
      &:last-child {
        margin-bottom: 0;
      }
    }
    .el-col {
      border-radius: 4px;
    }
    .bg-purple-dark {
      background: #99a9bf;
    }
    .bg-purple {
      background: #d3dce6;
    }
    .bg-purple-light {
      background: #e5e9f2;
    }
    .grid-content {
      border-radius: 4px;
      min-height: 36px;
    }
    .row-bg {
      padding: 10px 0;
      background-color: #f9fafc;
    }
  </style>
```

分栏间隔

分栏之间存在间隔。

Row 组件 提供 `gutter` 属性来指定每一栏之间的间隔，默认间隔为 0。

```
<el-row :gutter="20">
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
</el-row>

<style>
  .el-row {
    margin-bottom: 20px;
    &:last-child {
      margin-bottom: 0;
    }
  }
  .el-col {
    border-radius: 4px;
  }
  .bg-purple-dark {
    background: #99a9bf;
  }
  .bg-purple {
    background: #d3dce6;
  }
  .bg-purple-light {
    background: #e5e9f2;
  }
  .grid-content {
    border-radius: 4px;
    min-height: 36px;
  }
  .row-bg {
    padding: 10px 0;
    background-color: #f9fafc;
  }
</style>
```

混合布局

通过基础的 1/24 分栏任意扩展组合形成较为复杂的混合布局。


```
<el-row :gutter="20">
  <el-col :span="16"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="8"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row :gutter="20">
  <el-col :span="8"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="8"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row :gutter="20">
  <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="16"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
</el-row>

<style>
  .el-row {
    margin-bottom: 20px;
    &:last-child {
      margin-bottom: 0;
    }
  }
  .el-col {
    border-radius: 4px;
  }
  .bg-purple-dark {
    background: #99a9bf;
  }
  .bg-purple {
```

```
    background: #d3dce6;
  }
  .bg-purple-light {
    background: #e5e9f2;
  }
  .grid-content {
    border-radius: 4px;
    min-height: 36px;
  }
  .row-bg {
    padding: 10px 0;
    background-color: #f9fafc;
  }
</style>
```

分栏偏移

支持偏移指定的栏数。

通过制定 col 组件的 `offset` 属性可以指定分栏偏移的栏数。

```
<el-row :gutter="20">
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6" :offset="6"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row :gutter="20">
  <el-col :span="6" :offset="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6" :offset="6"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row :gutter="20">
  <el-col :span="12" :offset="6"><div class="grid-content bg-purple"></div>
</el-col>
```

```
</el-row>

<style>
  .el-row {
    margin-bottom: 20px;
    &:last-child {
      margin-bottom: 0;
    }
  }
  .el-col {
    border-radius: 4px;
  }
  .bg-purple-dark {
    background: #99a9bf;
  }
  .bg-purple {
    background: #d3dce6;
  }
  .bg-purple-light {
    background: #e5e9f2;
  }
  .grid-content {
    border-radius: 4px;
    min-height: 36px;
  }
  .row-bg {
    padding: 10px 0;
    background-color: #f9fafc;
  }
</style>
```

对齐方式

通过 `flex` 布局来对分栏进行灵活的对齐。

将 `type` 属性赋值为 ‘flex’，可以启用 flex 布局，并可通过 `justify` 属性来指定 start, center, end, space-between, space-around 其中的值来定义子元素的排版方式。

```
<el-row type="flex" class="row-bg">
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row type="flex" class="row-bg" justify="center">
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row type="flex" class="row-bg" justify="end">
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row type="flex" class="row-bg" justify="space-between">
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
</el-row>
<el-row type="flex" class="row-bg" justify="space-around">
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
  <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
```

```
</el-row>

<style>
  .el-row {
    margin-bottom: 20px;
    &:last-child {
      margin-bottom: 0;
    }
  }
  .el-col {
    border-radius: 4px;
  }
  .bg-purple-dark {
    background: #99a9bf;
  }
  .bg-purple {
    background: #d3dce6;
  }
  .bg-purple-light {
    background: #e5e9f2;
  }
  .grid-content {
    border-radius: 4px;
    min-height: 36px;
  }
  .row-bg {
    padding: 10px 0;
    background-color: #f9fafc;
  }
</style>
```

响应式布局

参照了 Bootstrap 的 响应式设计，预设了五个响应尺寸：`xs`、`sm`、`md`、`lg` 和 `xl`。


```
<el-row :gutter="10">
```

```
    <el-col :xs="8" :sm="6" :md="4" :lg="3" :xl="1"><div class="grid-content bg-
2.  purple"></div></el-col>
    <el-col :xs="4" :sm="6" :md="8" :lg="9" :xl="11"><div class="grid-content bg-
3.  purple-light"></div></el-col>
    <el-col :xs="4" :sm="6" :md="8" :lg="9" :xl="11"><div class="grid-content bg-
4.  purple"></div></el-col>
    <el-col :xs="8" :sm="6" :md="4" :lg="3" :xl="1"><div class="grid-content bg-
5.  purple-light"></div></el-col>
6.  </el-row>
7.
8.  <style>
9.    .el-col {
10.     border-radius: 4px;
11.   }
12.   .bg-purple-dark {
13.     background: #99a9bf;
14.   }
15.   .bg-purple {
16.     background: #d3dce6;
17.   }
18.   .bg-purple-light {
19.     background: #e5e9f2;
20.   }
21.   .grid-content {
22.     border-radius: 4px;
23.     min-height: 36px;
24.   }
25. </style>
```

基于断点的隐藏类

Element Plus 额外提供了一系列类名，用于在某些条件下隐藏元素。这些类名可以添加在任何 DOM 元素或自定义组件上。如果需要，请自行引入以下文件：

```
1.  import 'element-plus/lib/theme-chalk/display.css';
```

包含的类名及其含义为：

- `hidden-xs-only` - 当视口在 `xs` 尺寸时隐藏
- `hidden-sm-only` - 当视口在 `sm` 尺寸时隐藏
- `hidden-sm-and-down` - 当视口在 `sm` 及以下尺寸时隐藏
- `hidden-sm-and-up` - 当视口在 `sm` 及以上尺寸时隐藏

- `hidden-md-only` - 当视口在 `md` 尺寸时隐藏
- `hidden-md-and-down` - 当视口在 `md` 及以下尺寸时隐藏
- `hidden-md-and-up` - 当视口在 `md` 及以上尺寸时隐藏
- `hidden-lg-only` - 当视口在 `lg` 尺寸时隐藏
- `hidden-lg-and-down` - 当视口在 `lg` 及以下尺寸时隐藏
- `hidden-lg-and-up` - 当视口在 `lg` 及以上尺寸时隐藏
- `hidden-xl-only` - 当视口在 `xl` 尺寸时隐藏

Row Attributes

参数	说明	类型	可选值	默认值
gutter	栅格间隔	number	—	0
type	布局模式，可选 flex，现代浏览器下有效	string	—	—
justify	flex 布局下的水平排列方式	string	start/end/center/space-around/space-between	start
align	flex 布局下的垂直排列方式	string	top/middle/bottom	top
tag	自定义元素标签	string	*	div

Col Attributes

参数	说明	类型	可选值	默认值
span	栅格占据的列数	number	—	24
offset	栅格左侧的间隔格数	number	—	0
push	栅格向右移动格数	number	—	0
pull	栅格向左移动格数	number	—	0
xs	`<768px` 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
sm	`≥768px` 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
md	`≥992px` 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
lg	`≥1200px` 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
xl	`≥1920px` 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
tag	自定义元素标签	string	*	div

Container 布局容器

用于布局的容器组件，方便快速搭建页面的基本结构：

`<el-container>` ：外层容器。当子元素中包含 `<el-header>` 或 `<el-footer>` 时，全部子元素会垂直上下排列，否则会水平左右排列。

`<el-header>` ：顶栏容器。

`<el-aside>` ：侧边栏容器。

`<el-main>` ：主要区域容器。

`<el-footer>` ：底栏容器。

以上组件采用了 `flex` 布局，使用前请确定目标浏览器是否兼容。此外， `<el-container>` 的子元素只能是后四者，后四者的父元素也只能是 `<el-container>` 。

常见页面布局

```
Header
Main
```

```
Header
Main
Footer
```

```
Aside
Main
```

```
Header
Aside
Main
```

```
Header
Aside
Main
```

Footer

Aside

Header

Main

Aside

Header

Main

Footer

```
<el-container>
  <el-header>Header</el-header>
  <el-main>Main</el-main>
</el-container>

<el-container>
  <el-header>Header</el-header>
  <el-main>Main</el-main>
  <el-footer>Footer</el-footer>
</el-container>

<el-container>
  <el-aside width="200px">Aside</el-aside>
  <el-main>Main</el-main>
</el-container>

<el-container>
  <el-header>Header</el-header>
  <el-container>
    <el-aside width="200px">Aside</el-aside>
    <el-main>Main</el-main>
  </el-container>
</el-container>

<el-container>
  <el-header>Header</el-header>
  <el-container>
    <el-aside width="200px">Aside</el-aside>
```

```
    <el-container>
      <el-main>Main</el-main>
      <el-footer>Footer</el-footer>
    </el-container>
  </el-container>
</el-container>

<el-container>
  <el-aside width="200px">Aside</el-aside>
  <el-container>
    <el-header>Header</el-header>
    <el-main>Main</el-main>
  </el-container>
</el-container>

<el-container>
  <el-aside width="200px">Aside</el-aside>
  <el-container>
    <el-header>Header</el-header>
    <el-main>Main</el-main>
    <el-footer>Footer</el-footer>
  </el-container>
</el-container>

<style>
  .el-header, .el-footer {
    background-color: #B3C0D1;
    color: #333;
    text-align: center;
    line-height: 60px;
  }

  .el-aside {
    background-color: #D3DCE6;
    color: #333;
    text-align: center;
    line-height: 200px;
  }

  .el-main {
    background-color: #E9EEF3;
    color: #333;
```

```
    text-align: center;
    line-height: 160px;
  }

  body > .el-container {
    margin-bottom: 40px;
  }

  .el-container:nth-child(5) .el-aside,
  .el-container:nth-child(6) .el-aside {
    line-height: 260px;
  }

  .el-container:nth-child(7) .el-aside {
    line-height: 320px;
  }
</style>
```

实例

- 导航一
  - 选项1
  - 选项2
  - 选项3
  - 选项4
    - 选项4-1
- 导航二
  - 选项1
  - 选项2
  - 选项3
  - 选项4
    - 选项4-1
- 导航三

    - 选项1
    - 选项2
    - 选项3
    - 选项4
        - 选项4-1

- 查看
- 新增
- 删除

王小虎

0

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄

```
<el-container style="height: 500px; border: 1px solid #eee">
  <el-aside width="200px" style="background-color: rgb(238, 241, 246)">
    <el-menu :default-openeds="['1', '3']">
      <el-submenu index="1">
        <template #title><i class="el-icon-message"></i>导航一</template>
        <el-menu-item-group>
          <template #title>分组一</template>
          <el-menu-item index="1-1">选项1</el-menu-item>
          <el-menu-item index="1-2">选项2</el-menu-item>
        </el-menu-item-group>
        <el-menu-item-group title="分组2">
          <el-menu-item index="1-3">选项3</el-menu-item>
        </el-menu-item-group>
        <el-submenu index="1-4">
          <template #title>选项4</template>
          <el-menu-item index="1-4-1">选项4-1</el-menu-item>
        </el-submenu>
      </el-submenu>
      <el-submenu index="2">
        <template #title><i class="el-icon-menu"></i>导航二</template>
        <el-menu-item-group>
          <template #title>分组一</template>
          <el-menu-item index="2-1">选项1</el-menu-item>
          <el-menu-item index="2-2">选项2</el-menu-item>
        </el-menu-item-group>
        <el-menu-item-group title="分组2">
          <el-menu-item index="2-3">选项3</el-menu-item>
        </el-menu-item-group>
        <el-submenu index="2-4">
          <template #title>选项4</template>
          <el-menu-item index="2-4-1">选项4-1</el-menu-item>
        </el-submenu>
      </el-submenu>
      <el-submenu index="3">
        <template #title><i class="el-icon-setting"></i>导航三</template>
        <el-menu-item-group>
          <template #title>分组一</template>
          <el-menu-item index="3-1">选项1</el-menu-item>
          <el-menu-item index="3-2">选项2</el-menu-item>
        </el-menu-item-group>
        <el-menu-item-group title="分组2">
          <el-menu-item index="3-3">选项3</el-menu-item>
```

```
43.         </el-menu-item-group>
44.         <el-submenu index="3-4">
45.           <template #title>选项4</template>
46.           <el-menu-item index="3-4-1">选项4-1</el-menu-item>
47.         </el-submenu>
48.       </el-submenu>
49.     </el-menu>
50.   </el-aside>
51.
52.   <el-container>
53.     <el-header style="text-align: right; font-size: 12px">
54.       <el-dropdown>
55.         <i class="el-icon-setting" style="margin-right: 15px"></i>
56.         <template #dropdown>
57.           <el-dropdown-menu>
58.               <el-dropdown-item>查看</el-dropdown-item>
59.               <el-dropdown-item>新增</el-dropdown-item>
60.               <el-dropdown-item>删除</el-dropdown-item>
61.           </el-dropdown-menu>
62.         </template>
63.       </el-dropdown>
64.       <span>王小虎</span>
65.     </el-header>
66.
67.     <el-main>
68.       <el-table :data="tableData">
69.         <el-table-column prop="date" label="日期" width="140">
70.         </el-table-column>
71.         <el-table-column prop="name" label="姓名" width="120">
72.         </el-table-column>
73.         <el-table-column prop="address" label="地址">
74.         </el-table-column>
75.       </el-table>
76.     </el-main>
77.   </el-container>
78. </el-container>
79.
80. <style>
81.   .el-header {
82.     background-color: #B3C0D1;
83.     color: #333;
84.     line-height: 60px;
```

```
  }

  .el-aside {
    color: #333;
  }
</style>

<script>
  export default {
    data() {
      const item = {
        date: '2016-05-02',
        name: '王小虎',
        address: '上海市普陀区金沙江路 1518 弄'
      };
      return {
        tableData: Array(20).fill(item)
      }
    }
  };
</script>
```

Container Attributes

参数	说明	类型	可选值	默认值
direction	子元素的排列方向	string	horizontal / vertical	子元素中有 `el-header` 或 `el-footer` 时为 vertical，否则为 horizontal

Header Attributes

参数	说明	类型	可选值	默认值
height	顶栏高度	string	—	60px

Aside Attributes

参数	说明	类型	可选值	默认值
width	侧边栏宽度	string	—	300px

Footer Attributes

参数	说明	类型	可选值	默认值
height	底栏高度	string	—	60px

Color 色彩

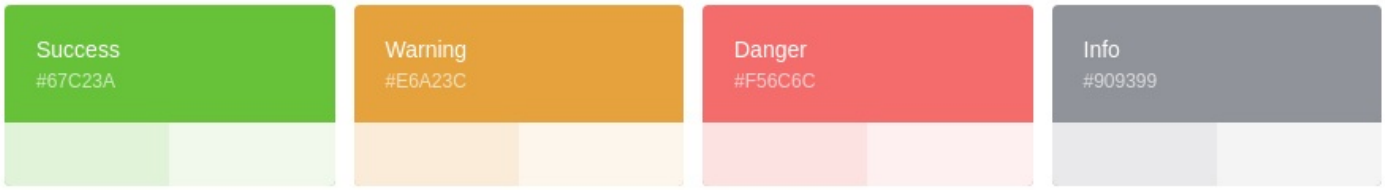
Element Plus 为了避免视觉传达差异，使用一套特定的调色板来规定颜色，为你所搭建的产品提供一致的外观视觉感受。

主色

Element Plus 主要品牌颜色是鲜艳、友好的蓝色。

辅助色

除了主色外的场景色，需要在不同的场景中使用（例如危险色表示危险的操作）。



中性色

中性色用于文本、背景和边框颜色。通过运用不同的中性色，来表现层次结构。

Typography 字体

我们对字体进行统一规范，力求在各个操作系统下都有最佳展示效果。

字体

PingFang SC

永 常规 中等 半粗体

永和九年，岁在癸丑，暮春之初，会于会稽山阴之兰亭，修禊事也。群贤毕至，少长咸集。此地有崇山峻岭，茂林修竹……

Hiragino Sans GB

永 常规 粗体

永和九年，岁在癸丑，暮春之初，会于会稽山阴之兰亭，修禊事也。群贤毕至，少长咸集。此地有崇山峻岭，茂林修竹……

Microsoft YaHei

永 细体 粗体 粗体

永和九年，岁在癸丑，暮春之初，会于会稽山阴之兰亭，修禊事也。群贤毕至，少长咸集。此地有崇山峻岭，茂林修竹……

San Francisco UI

Aa Regular Medium Blod

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890!@#$%^&*()

Helvetica Neue

Aa Regular Medium Blod

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890!@#$%^&*()

Arial

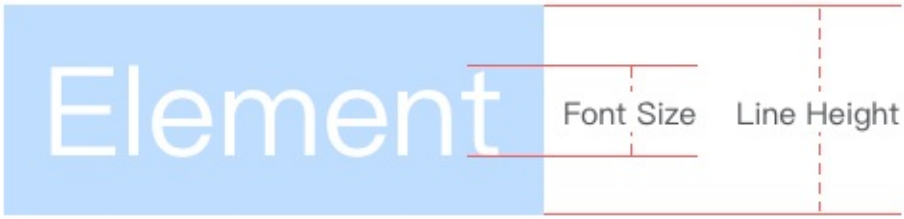
Aa Regular Bold

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890!@#$%^&*()

字号

层级	字体大小	举例
辅助文字	12px Extra Small	用 Element Plus 快速搭建页面
正文（小）	13px Small	用 Element Plus 快速搭建页面
正文	14px Base	用 Element Plus 快速搭建页面
小标题	16px Medium	用 Element Plus 快速搭建页面
标题	18px large	用 Element Plus 快速搭建页面
主标题	20px Extra large	用 Element Plus 快速搭建页面

行高



line-height:1	无行高
line-height:1.3	紧凑
line-height:1.5	常规
line-height:1.7	宽松

Font-family 代码

```
font-family: "Helvetica Neue",Helvetica,"PingFang SC","Hiragino Sans GB","Microsoft YaHei","微软雅黑",Arial,sans-serif;
```

Border 边框

我们对边框进行统一规范，可用于按钮、卡片、弹窗等组件里。

边框

我们提供了以下几种边框样式，以供选择。

名称	粗细	举例
实线	1px	
虚线	2px	

圆角

我们提供了以下几种圆角样式，以供选择。

无圆角
border-radius: 0px

小圆角
border-radius: 2px

大圆角
border-radius: 4px

圆形圆角
border-radius: 30px

投影

我们提供了以下几种投影样式，以供选择。

基础投影 `box-shadow: 0 2px 4px rgba(0, 0, 0, .12), 0 0 6px rgba(0, 0, 0, .04)`

浅色投影 `box-shadow: 0 2px 12px 0 rgba(0, 0, 0, 0.1)`

Icon 图标

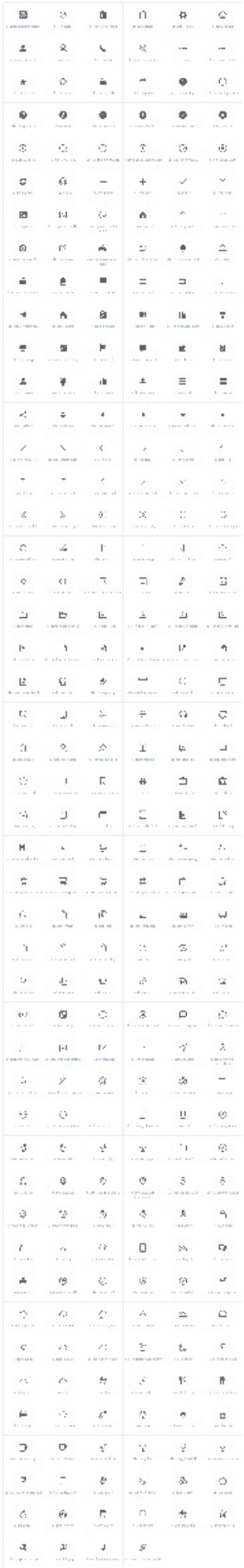
提供了一套常用的图标集合。

使用方法

直接通过设置类名为 `el-icon-iconName` 来使用即可。例如：


```
<i class="el-icon-edit"></i>
<i class="el-icon-share"></i>
<i class="el-icon-delete"></i>
<el-button type="primary" icon="el-icon-search">搜索</el-button>
```

图标集合



Button 按钮

常用的操作按钮。

基础用法

基础的按钮用法。

使用 `type` 、 `plain` 、 `round` 和 `circle` 属性来定义 Button 的样式。

```
<el-row>
  <el-button>默认按钮</el-button>
  <el-button type="primary">主要按钮</el-button>
  <el-button type="success">成功按钮</el-button>
  <el-button type="info">信息按钮</el-button>
  <el-button type="warning">警告按钮</el-button>
  <el-button type="danger">危险按钮</el-button>
</el-row>

<el-row>
  <el-button plain>朴素按钮</el-button>
  <el-button type="primary" plain>主要按钮</el-button>
  <el-button type="success" plain>成功按钮</el-button>
  <el-button type="info" plain>信息按钮</el-button>
  <el-button type="warning" plain>警告按钮</el-button>
  <el-button type="danger" plain>危险按钮</el-button>
</el-row>

<el-row>
  <el-button round>圆角按钮</el-button>
  <el-button type="primary" round>主要按钮</el-button>
```

```
    <el-button type="success" round>成功按钮</el-button>
    <el-button type="info" round>信息按钮</el-button>
    <el-button type="warning" round>警告按钮</el-button>
    <el-button type="danger" round>危险按钮</el-button>
  </el-row>

  <el-row>
    <el-button icon="el-icon-search" circle></el-button>
    <el-button type="primary" icon="el-icon-edit" circle></el-button>
    <el-button type="success" icon="el-icon-check" circle></el-button>
    <el-button type="info" icon="el-icon-message" circle></el-button>
    <el-button type="warning" icon="el-icon-star-off" circle></el-button>
    <el-button type="danger" icon="el-icon-delete" circle></el-button>
  </el-row>
```

禁用状态

按钮不可用状态。

你可以使用 `disabled` 属性来定义按钮是否可用，它接受一个 `Boolean` 值。

```
  <el-row>
    <el-button disabled>默认按钮</el-button>
    <el-button type="primary" disabled>主要按钮</el-button>
    <el-button type="success" disabled>成功按钮</el-button>
    <el-button type="info" disabled>信息按钮</el-button>
    <el-button type="warning" disabled>警告按钮</el-button>
    <el-button type="danger" disabled>危险按钮</el-button>
  </el-row>

  <el-row>
    <el-button plain disabled>朴素按钮</el-button>
    <el-button type="primary" plain disabled>主要按钮</el-button>
    <el-button type="success" plain disabled>成功按钮</el-button>
    <el-button type="info" plain disabled>信息按钮</el-button>
    <el-button type="warning" plain disabled>警告按钮</el-button>
```

```
    <el-button type="danger" plain disabled>危险按钮</el-button>
</el-row>
```

文字按钮

没有边框和背景色的按钮。


```
<el-button type="text">文字按钮</el-button>
<el-button type="text" disabled>文字按钮</el-button>
```

图标按钮

带图标的按钮可增强辨识度（有文字）或节省空间（无文字）。

设置 `icon` 属性即可，icon 的列表可以参考 Element Plus 的 icon 组件，也可以设置在文字右边的 icon ，只要使用 `i` 标签即可，可以使用自定义图标。

```
<el-button type="primary" icon="el-icon-edit"></el-button>
<el-button type="primary" icon="el-icon-share"></el-button>
<el-button type="primary" icon="el-icon-delete"></el-button>
<el-button type="primary" icon="el-icon-search">搜索</el-button>
<el-button type="primary">上传<i class="el-icon-upload el-icon--right"></i>
</el-button>
```

按钮组

以按钮组的方式出现，常用于多项类似操作。

使用 `<el-button-group>` 标签来嵌套你的按钮。

```
<el-button-group>
  <el-button type="primary" icon="el-icon-arrow-left">上一页</el-button>
  <el-button type="primary">下一页<i class="el-icon-arrow-right el-icon--right">
</i></el-button>
</el-button-group>
<el-button-group>
  <el-button type="primary" icon="el-icon-edit"></el-button>
  <el-button type="primary" icon="el-icon-share"></el-button>
  <el-button type="primary" icon="el-icon-delete"></el-button>
</el-button-group>
```

加载中

点击按钮后进行数据加载操作，在按钮上显示加载状态。

要设置为 loading 状态，只要设置 `loading` 属性为 `true` 即可。

```
<el-button type="primary" :loading="true">加载中</el-button>
```

不同尺寸

Button 组件提供除了默认值以外的三种尺寸，可以在不同场景下选择合适的按钮尺寸。

额外的尺寸： `medium` 、 `small` 、 `mini` ，通过设置 `size` 属性来配置它们。

```
<el-row>
  <el-button>默认按钮</el-button>
  <el-button size="medium">中等按钮</el-button>
  <el-button size="small">小型按钮</el-button>
  <el-button size="mini">超小按钮</el-button>
</el-row>
```

```
<el-row>
  <el-button round>默认按钮</el-button>
  <el-button size="medium" round>中等按钮</el-button>
  <el-button size="small" round>小型按钮</el-button>
  <el-button size="mini" round>超小按钮</el-button>
</el-row>
```

Attributes

参数	说明	类型	可选值	默认值
size	尺寸	string	medium / small / mini	—
type	类型	string	primary / success / warning / danger / info / text	—
plain	是否朴素按钮	boolean	—	false
round	是否圆角按钮	boolean	—	false
circle	是否圆形按钮	boolean	—	false
loading	是否加载中状态	boolean	—	false
disabled	是否禁用状态	boolean	—	false
icon	图标类名	string	—	—
autofocus	是否默认聚焦	boolean	—	false
native-type	原生 type 属性	string	button / submit / reset	button

Link 文字链接

文字超链接

基础用法

基础的文字链接用法。


```
<div>
  <el-link href="https://element.eleme.io" target="_blank">默认链接</el-link>
  <el-link type="primary">主要链接</el-link>
  <el-link type="success">成功链接</el-link>
  <el-link type="warning">警告链接</el-link>
  <el-link type="danger">危险链接</el-link>
  <el-link type="info">信息链接</el-link>
</div>
```

禁用状态

文字链接不可用状态。


```
<div>
  <el-link disabled>默认链接</el-link>
  <el-link type="primary" disabled>主要链接</el-link>
  <el-link type="success" disabled>成功链接</el-link>
  <el-link type="warning" disabled>警告链接</el-link>
  <el-link type="danger" disabled>危险链接</el-link>
  <el-link type="info" disabled>信息链接</el-link>
</div>
```

下划线

文字链接下划线。

无下划线有下划线

```
<div>
  <el-link :underline="false">无下划线</el-link>
  <el-link>有下划线</el-link>
</div>
```

图标

带图标的文字链接可增强辨识度。

编辑查看

```
<div>
  <el-link icon="el-icon-edit">编辑</el-link>
  <el-link>查看<i class="el-icon-view el-icon--right"></i> </el-link>
</div>
```

Attributes

参数	说明	类型	可选值	默认值
type	类型	string	primary / success / warning / danger / info	default
underline	是否下划线	boolean	—	true
disabled	是否禁用状态	boolean	—	false
href	原生 href 属性	string	—	-
icon	图标类名	string	—	-

- Radio 单选框
- Checkbox 多选框
- Input 输入框
- InputNumber 计数器
- Select 选择器
- Cascader 级联选择器
- Switch 开关
- Slider 滑块
- TimePicker 时间选择器
- TimeSelect 时间选择器
- DatePicker 日期选择器
- DateTimePicker 日期时间选择器
- Upload 上传
- Rate 评分
- ColorPicker 颜色选择器
- Transfer 穿梭框
- Form 表单

Radio 单选框

在一组备选项中进行单选

基础用法

由于选项默认可见，不宜过多，若选项过多，建议使用 Select 选择器。

要使用 Radio 组件，只需要设置 `v-model` 绑定变量，选中意味着变量的值为相应 Radio `label` 属性的值， `label` 可以是 `String` 、 `Number` 或 `Boolean` 。

```
<template>
  <el-radio v-model="radio" label="1">备选项</el-radio>
  <el-radio v-model="radio" label="2">备选项</el-radio>
</template>

<script>
  export default {
    data () {
      return {
        radio: '1'
      };
    }
  }
</script>
```

禁用状态

单选框不可用的状态。

只要在 `el-radio` 元素中设置 `disabled` 属性即可，它接受一个 `Boolean` ， `true` 为禁用。

```
<template>
  <el-radio disabled v-model="radio" label="禁用">备选项</el-radio>
```

```
  <el-radio disabled v-model="radio" label="选中且禁用">备选项</el-radio>
</template>

<script>
  export default {
    data () {
      return {
        radio: '选中且禁用'
      };
    }
  }
</script>
```

单选框组

适用于在多个互斥的选项中选择的场景

结合 `el-radio-group` 元素和子元素 `el-radio` 可以实现单选组，在 `el-radio-group` 中绑定 `v-model` ，在 `el-radio` 中设置好 `label` 即可，无需再给每一个 `el-radio` 绑定变量，另外，还提供了 `change` 事件来响应变化，它会传入一个参数 `value` 。

```
<template>
  <el-radio-group v-model="radio">
    <el-radio :label="3">备选项</el-radio>
    <el-radio :label="6">备选项</el-radio>
    <el-radio :label="9">备选项</el-radio>
  </el-radio-group>
</template>

<script>
  export default {
    data () {
      return {
        radio: 3
      };
    }
  }
</script>
```

按钮样式

按钮样式的单选组合。

只需要把 `el-radio` 元素换成 `el-radio-button` 元素即可，此外，Element Plus 还提供了 `size` 属性。

```
<template>
  <div>
    <el-radio-group v-model="radio1">
      <el-radio-button label="上海"></el-radio-button>
      <el-radio-button label="北京"></el-radio-button>
      <el-radio-button label="广州"></el-radio-button>
      <el-radio-button label="深圳"></el-radio-button>
    </el-radio-group>
  </div>
  <div style="margin-top: 20px">
    <el-radio-group v-model="radio2" size="medium">
      <el-radio-button label="上海" ></el-radio-button>
      <el-radio-button label="北京"></el-radio-button>
      <el-radio-button label="广州"></el-radio-button>
      <el-radio-button label="深圳"></el-radio-button>
    </el-radio-group>
  </div>
  <div style="margin-top: 20px">
    <el-radio-group v-model="radio3" size="small">
      <el-radio-button label="上海"></el-radio-button>
      <el-radio-button label="北京" disabled ></el-radio-button>
      <el-radio-button label="广州"></el-radio-button>
      <el-radio-button label="深圳"></el-radio-button>
    </el-radio-group>
  </div>
  <div style="margin-top: 20px">
```

```
    <el-radio-group v-model="radio4" disabled size="mini">
      <el-radio-button label="上海"></el-radio-button>
      <el-radio-button label="北京"></el-radio-button>
      <el-radio-button label="广州"></el-radio-button>
      <el-radio-button label="深圳"></el-radio-button>
    </el-radio-group>
  </div>
</template>

<script>
  export default {
    data () {
      return {
        radio1: '上海',
        radio2: '上海',
        radio3: '上海',
        radio4: '上海'
      };
    }
  }
</script>
```

带有边框

设置 `border` 属性可以渲染为带有边框的单选框。

```
<template>
  <div>
    <el-radio v-model="radio1" label="1" border>备选项1</el-radio>
    <el-radio v-model="radio1" label="2" border>备选项2</el-radio>
  </div>
  <div style="margin-top: 20px">
```

```
      <el-radio v-model="radio2" label="1" border size="medium">备选项1</el-radio>
      <el-radio v-model="radio2" label="2" border size="medium">备选项2</el-radio>
    </div>
    <div style="margin-top: 20px">
      <el-radio-group v-model="radio3" size="small">
        <el-radio label="1" border>备选项1</el-radio>
        <el-radio label="2" border disabled>备选项2</el-radio>
      </el-radio-group>
    </div>
    <div style="margin-top: 20px">
      <el-radio-group v-model="radio4" size="mini" disabled>
        <el-radio label="1" border>备选项1</el-radio>
        <el-radio label="2" border>备选项2</el-radio>
      </el-radio-group>
    </div>
  </template>

  <script>
    export default {
      data () {
        return {
          radio1: '1',
          radio2: '1',
          radio3: '1',
          radio4: '1'
        };
      }
    }
  </script>
```

Radio Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	string / number / boolean	—	—
label	Radio 的 value	string / number / boolean	—	—
disabled	是否禁用	boolean	—	false
border	是否显示边框	boolean	—	false
size	Radio 的尺寸，仅在 border 为真时有效	string	medium / small / mini	—
name	原生 name 属性	string	—	—

Radio Events

事件名称	说明	回调参数
change	绑定值变化时触发的事件	选中的 Radio label 值

Radio-group Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	string / number / boolean	—	—
size	单选框组尺寸，仅对按钮形式的 Radio 或带有边框的 Radio 有效	string	medium / small / mini	—
disabled	是否禁用	boolean	—	false
text-color	按钮形式的 Radio 激活时的文本颜色	string	—	#ffffff
fill	按钮形式的 Radio 激活时的填充色和边框色	string	—	#409EFF

Radio-group Events

事件名称	说明	回调参数
change	绑定值变化时触发的事件	选中的 Radio label 值

Radio-button Attributes

参数	说明	类型	可选值	默认值
label	Radio 的 value	string / number	—	—
disabled	是否禁用	boolean	—	false
name	原生 name 属性	string	—	—

Checkbox 多选框

一组备选项中进行多选

基础用法

单独使用可以表示两种状态之间的切换，写在标签中的内容为 checkbox 按钮后的介绍。

在 `el-checkbox` 元素中定义 `v-model` 绑定变量，单一的 `checkbox` 中，默认绑定变量的值会是 `Boolean` ，选中为 `true` 。

```
<template>
  <!-- `checked` 为 true 或 false -->
  <el-checkbox v-model="checked">备选项</el-checkbox>
</template>
<script>
  export default {
    data() {
      return {
        checked: true
      };
    }
  };
</script>
```

禁用状态

多选框不可用状态。

设置 `disabled` 属性即可。

```
<template>
  <el-checkbox v-model="checked1" disabled>备选项1</el-checkbox>
  <el-checkbox v-model="checked2" disabled>备选项</el-checkbox>
```

```
</template>
<script>
  export default {
    data() {
      return {
        checked1: false,
        checked2: true
      };
    }
  };
</script>
```

多选框组

适用于多个勾选框绑定到同一个数组的情景，通过是否勾选来表示这一组选项中选中的项。

`checkbox-group` 元素能把多个 checkbox 管理为一组，只需要在 Group 中使用 `v-model` 绑定 `Array` 类型的变量即可。 `el-checkbox` 的 `label` 属性是该 checkbox 对应的值，若该标签中无内容，则该属性也充当 checkbox 按钮后的介绍。 `label` 与数组中的元素值相对应，如果存在指定的值则为选中状态，否则为不选中。

```
<template>
  <el-checkbox-group v-model="checkList">
    <el-checkbox label="复选框 A"></el-checkbox>
    <el-checkbox label="复选框 B"></el-checkbox>
    <el-checkbox label="复选框 C"></el-checkbox>
    <el-checkbox label="禁用" disabled></el-checkbox>
    <el-checkbox label="选中且禁用" disabled></el-checkbox>
  </el-checkbox-group>
</template>

<script>
  export default {
    data () {
      return {
        checkList: ['选中且禁用','复选框 A']
      };
    }
```

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

indeterminate 状态

`indeterminate` 属性用以表示 checkbox 的不确定状态，一般用于实现全选的效果


```
<template>
  <el-checkbox :indeterminate="isIndeterminate" v-model="checkAll"
@change="handleCheckAllChange">全选</el-checkbox>
  <div style="margin: 15px 0;"></div>
  <el-checkbox-group v-model="checkedCities"
@change="handleCheckedCitiesChange">
    <el-checkbox v-for="city in cities" :label="city" :key="city">{{city}}</el-
checkbox>
  </el-checkbox-group>
</template>
<script>
  const cityOptions = ['上海', '北京', '广州', '深圳'];
  export default {
    data() {
      return {
        checkAll: false,
        checkedCities: ['上海', '北京'],
        cities: cityOptions,
        isIndeterminate: true
      };
    },
    methods: {
      handleCheckAllChange(val) {
        this.checkedCities = val ? cityOptions : [];
        this.isIndeterminate = false;
      },
      handleCheckedCitiesChange(value) {
        let checkedCount = value.length;
        this.checkAll = checkedCount === this.cities.length;
        this.isIndeterminate = checkedCount > 0 && checkedCount <
this.cities.length;
```

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

可选项目数量的限制

使用 `min` 和 `max` 属性能够限制可以被勾选的项目的数量。


```
<template>
  <el-checkbox-group
    v-model="checkedCities"
    :min="1"
    :max="2">
    <el-checkbox v-for="city in cities" :label="city" :key="city">{{city}}</el-checkbox>
  </el-checkbox-group>
</template>
<script>
  const cityOptions = ['上海', '北京', '广州', '深圳'];
  export default {
    data() {
      return {
        checkedCities: ['上海', '北京'],
        cities: cityOptions
      };
    }
  };
</script>
```

按钮样式

按钮样式的多选组合。

只需要把 `el-checkbox` 元素替换为 `el-checkbox-button` 元素即可。此外，Element Plus 还提供了 `size` 属性。

```
<template>
  <div>
    <el-checkbox-group v-model="checkboxGroup1">
      <el-checkbox-button v-for="city in cities" :label="city" :key="city">{{city}}</el-checkbox-button>
    </el-checkbox-group>
  </div>
  <div style="margin-top: 20px">
    <el-checkbox-group v-model="checkboxGroup2" size="medium">
      <el-checkbox-button v-for="city in cities" :label="city" :key="city">{{city}}</el-checkbox-button>
    </el-checkbox-group>
  </div>
  <div style="margin-top: 20px">
    <el-checkbox-group v-model="checkboxGroup3" size="small">
      <el-checkbox-button v-for="city in cities" :label="city" :disabled="city === '北京'" :key="city">{{city}}</el-checkbox-button>
    </el-checkbox-group>
  </div>
  <div style="margin-top: 20px">
    <el-checkbox-group v-model="checkboxGroup4" size="mini" disabled>
      <el-checkbox-button v-for="city in cities" :label="city" :key="city">{{city}}</el-checkbox-button>
    </el-checkbox-group>
  </div>
</template>
<script>
  const cityOptions = ['上海', '北京', '广州', '深圳'];
  export default {
    data () {
```

```
      return {
        checkboxGroup1: ['上海'],
        checkboxGroup2: ['上海'],
        checkboxGroup3: ['上海'],
        checkboxGroup4: ['上海'],
        cities: cityOptions
      };
    }
  }
</script>
```

带有边框

设置 `border` 属性可以渲染为带有边框的多选框。

```
<template>
  <div>
    <el-checkbox v-model="checked1" label="备选项1" border></el-checkbox>
    <el-checkbox v-model="checked2" label="备选项2" border></el-checkbox>
  </div>
  <div style="margin-top: 20px">
    <el-checkbox v-model="checked3" label="备选项1" border size="medium"></el-
checkbox>
    <el-checkbox v-model="checked4" label="备选项2" border size="medium"></el-
checkbox>
  </div>
  <div style="margin-top: 20px">
    <el-checkbox-group v-model="checkboxGroup1" size="small">
      <el-checkbox label="备选项1" border></el-checkbox>
      <el-checkbox label="备选项2" border disabled></el-checkbox>
    </el-checkbox-group>
  </div>
  <div style="margin-top: 20px">
```

```
    <el-checkbox-group v-model="checkboxGroup2" size="mini" disabled>
      <el-checkbox label="备选项1" border></el-checkbox>
      <el-checkbox label="备选项2" border></el-checkbox>
    </el-checkbox-group>
  </div>
</template>

<script>
  export default {
    data () {
      return {
        checked1: true,
        checked2: false,
        checked3: false,
        checked4: true,
        checkboxGroup1: [],
        checkboxGroup2: []
      };
    }
  }
</script>
```

Checkbox Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	string / number / boolean	—	—
label	选中状态的值（只有在 checkbox-group 或者绑定对象类型为 array 时有效）	string / number / boolean	—	—
true-label	选中时的值	string / number	—	—
false-label	没有选中时的值	string / number	—	—
disabled	是否禁用	boolean	—	false
border	是否显示边框	boolean	—	false
size	Checkbox 的尺寸，仅在 border 为真时有效	string	medium / small / mini	—
name	原生 name 属性	string	—	—
checked	当前是否勾选	boolean	—	false
indeterminate	设置 indeterminate 状态，只负责样式控制	boolean	—	false

Checkbox Events

事件名称	说明	回调参数
change	当绑定值变化时触发的事件	更新后的值

Checkbox-group Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	array	—	—
size	多选框组尺寸，仅对按钮形式的 Checkbox 或带有边框的 Checkbox 有效	string	medium / small / mini	—
disabled	是否禁用	boolean	—	false
min	可被勾选的 checkbox 的最小数量	number	—	—
max	可被勾选的 checkbox 的最大数量	number	—	—
text-color	按钮形式的 Checkbox 激活时的文本颜色	string	—	#ffffff
fill	按钮形式的 Checkbox 激活时的填充色和边框色	string	—	#409EFF

Checkbox-group Events

事件名称	说明	回调参数
change	当绑定值变化时触发的事件	更新后的值

Checkbox-button Attributes

参数	说明	类型	可选值	默认值
label	选中状态的值（只有在 `checkbox-group` 或者绑定对象类型为 `array` 时有效）	string / number / boolean	—	—
true-label	选中时的值	string / number	—	—
false-label	没有选中时的值	string / number	—	—
disabled	是否禁用	boolean	—	false
name	原生 name 属性	string	—	—
checked	当前是否勾选	boolean	—	false

Input 输入框

通过鼠标或键盘输入字符

Input 为受控组件，它总会显示 Vue 绑定值。

通常情况下，应当处理 `input` 事件，并更新组件的绑定值（或使用 `v-model`）。否则，输入框内显示的值将不会改变。

不支持 `v-model` 修饰符。

基础用法


```
<el-input v-model="input" placeholder="请输入内容"></el-input>

<script>
import { defineComponent, ref } from 'vue'

export default defineComponent ({
  setup() {
    return {
      input: ref('')
    }
  }
})
</script>
```

禁用状态

通过 `disabled` 属性指定是否禁用 input 组件

```
<el-input
```

```
  placeholder="请输入内容"
  v-model="input"
  :disabled="true">
</el-input>

<script>
import { defineComponent, ref } from 'vue'

export default defineComponent ({
  setup() {
    return {
      input: ref('')
    }
  }
})
</script>
```

可清空

使用 `clearable` 属性即可得到一个可清空的输入框

```
<el-input
  placeholder="请输入内容"
  v-model="input"
  clearable>
</el-input>

<script>
import { defineComponent, ref } from 'vue'

export default defineComponent ({
  setup() {
    return {
      input: ref('')
    }
  }
})
</script>
```

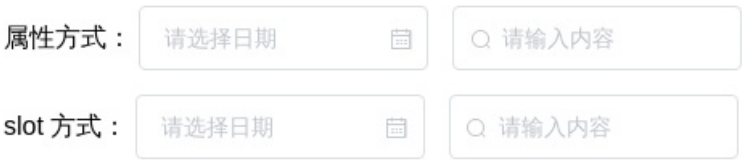
密码框

使用 `show-password` 属性即可得到一个可切换显示隐藏的密码框

```
<el-input placeholder="请输入密码" v-model="input" show-password></el-input>

<script>
import { defineComponent, ref } from 'vue'

export default defineComponent ({
  setup() {
    return {
      input: ref('')
    }
  }
})
</script>
```

带 icon 的输入框

带有图标标记输入类型



可以通过 `prefix-icon` 和 `suffix-icon` 属性在 input 组件首部和尾部增加显示图标，也可以通过 slot 来放置图标。

```
<div class="demo-input-suffix">
  属性方式：
  <el-input
    placeholder="请选择日期"
    suffix-icon="el-icon-date"
```

```
    v-model="input1">
  </el-input>
  <el-input
    placeholder="请输入内容"
    prefix-icon="el-icon-search"
    v-model="input2">
  </el-input>
</div>
<div class="demo-input-suffix">
  slot 方式：
  <el-input
    placeholder="请选择日期"
    v-model="input3">
    <template #suffix>
      <i class="el-input__icon el-icon-date"></i>
    </template>
  </el-input>
  <el-input
    placeholder="请输入内容"
    v-model="input4">
    <template #prefix>
      <i class="el-input__icon el-icon-search"></i>
    </template>
  </el-input>
</div>

<style>
  .demo-input-label {
    display: inline-block;
    width: 130px;
  }
</style>

<script>
import { defineComponent, ref } from 'vue'

export default defineComponent ({
  setup() {
    return {
      input1: ref(''),
      input2: ref(''),
      input3: ref(''),
```

```
      input4: ref('')
    }
  }
})
</script>
```

文本域

用于输入多行文本信息，通过将 `type` 属性的值指定为 `textarea`。

文本域高度可通过 `rows` 属性控制

```
<el-input
  type="textarea"
  :rows="2"
  placeholder="请输入内容"
  v-model="textarea">
</el-input>

<script>
import { defineComponent, ref } from 'vue'

export default defineComponent ({
  setup() {
    return {
      textarea: ref('')
    }
  }
})
</script>
```

可自适应文本高度的文本域

通过设置 `autosize` 属性可以使得文本域的高度能够根据文本内容自动进行调整，并且 `autosize` 还可以设定为一个对象，指定最小行数和最大行数。


```
<el-input
  type="textarea"
  autosize
  placeholder="请输入内容"
  v-model="textarea1">
</el-input>
<div style="margin: 20px 0;"></div>
<el-input
  type="textarea"
  :autosize="{ minRows: 2, maxRows: 4}"
  placeholder="请输入内容"
  v-model="textarea2">
</el-input>

<script>
import { defineComponent, ref } from 'vue'

export default defineComponent ({
  setup() {
    return {
      textarea1: ref(''),
      textarea2: ref('')
    }
  }
})
</script>
```

复合型输入框

可前置或后置元素，一般为标签或按钮

可通过 slot 来指定在 input 中前置或者后置内容。

```
<div>
  <el-input placeholder="请输入内容" v-model="input1">
    <template #prepend>Http://</template>
  </el-input>
</div>
<div style="margin-top: 15px">
  <el-input placeholder="请输入内容" v-model="input2">
    <template #append>.com</template>
  </el-input>
</div>
<div style="margin-top: 15px">
  <el-input
    placeholder="请输入内容"
    v-model="input3"
    class="input-with-select"
  >
    <template #prepend>
      <el-select v-model="select" placeholder="请选择">
        <el-option label="餐厅名" value="1"></el-option>
        <el-option label="订单号" value="2"></el-option>
        <el-option label="用户电话" value="3"></el-option>
      </el-select>
    </template>
    <template #append>
      <el-button icon="el-icon-search"></el-button>
    </template>
  </el-input>
</div>

<style>
  .el-select .el-input {
    width: 130px;
```

```
  }
  .input-with-select .el-input-group__prepend {
    background-color: #fff;
  }
</style>

<script>
import { defineComponent, ref } from 'vue'

export default defineComponent ({
  setup() {
    return {
      input1: ref(''),
      input2: ref(''),
      input3: ref(''),
      select: ref('')
    }
  }
})
</script>
```

尺寸

可通过 `size` 属性指定输入框的尺寸，除了默认的大小外，还提供了 large、small 和 mini 三种尺寸。

```
<div class="demo-input-size">
  <el-input
    placeholder="请输入内容"
    suffix-icon="el-icon-date"
    v-model="input1">
  </el-input>
  <el-input
    size="medium"
    placeholder="请输入内容"
    suffix-icon="el-icon-date"
    v-model="input2">
```

```
    </el-input>
    <el-input
      size="small"
      placeholder="请输入内容"
      suffix-icon="el-icon-date"
      v-model="input3">
    </el-input>
    <el-input
      size="mini"
      placeholder="请输入内容"
      suffix-icon="el-icon-date"
      v-model="input4">
    </el-input>
  </div>

  <script>
  import { defineComponent, ref } from 'vue'

  export default defineComponent ({
    setup() {
      return {
        input1: ref(''),
        input2: ref(''),
        input3: ref(''),
        input4: ref('')
      }
    }
  })
  </script>
```

带输入建议

根据输入内容提供对应的输入建议

autocomplete 是一个可带输入建议的输入框组件，`fetch-suggestions` 是一个返回输入建议的方法属性，如 querySearch(queryString, cb)，在该方法中你可以在你的输入建议数据准备好时通过 cb(data) 返回到 autocomplete 组件中。

```
<el-row class="demo-autocomplete">
  <el-col :span="12">
    <div class="sub-title">激活即列出输入建议</div>
    <el-autocomplete
      class="inline-input"
      v-model="state1"
      :fetch-suggestions="querySearch"
      placeholder="请输入内容"
      @select="handleSelect"
    ></el-autocomplete>
  </el-col>
  <el-col :span="12">
    <div class="sub-title">输入后匹配输入建议</div>
    <el-autocomplete
      class="inline-input"
      v-model="state2"
      :fetch-suggestions="querySearch"
      placeholder="请输入内容"
      :trigger-on-focus="false"
      @select="handleSelect"
    ></el-autocomplete>
  </el-col>
</el-row>
<script>
import { defineComponent, ref, onMounted } from 'vue'

export default defineComponent({
  setup() {
    const restaurants = ref([]);
    const querySearch = (queryString, cb) => {
      var results = queryString
        ? restaurants.value.filter(createFilter(queryString))
        : restaurants.value;
      // 调用 callback 返回建议列表的数据
      cb(results);
    };
    const createFilter = (queryString) => {
      return (restaurant) => {
        return (
          restaurant.value.toLowerCase().indexOf(queryString.toLowerCase()) ===
          0
        );
```

```
      };
    };
    const loadAll = () => {
      return [
        { value: "三全鲜食（北新泾店）", address: "长宁区新渔路144号" },
        {
          value: "Hot honey 首尔炸鸡（仙霞路）",
          address: "上海市长宁区淞虹路661号",
        },
        {
          value: "新旺角茶餐厅",
          address: "上海市普陀区真北路988号创邑金沙谷6号楼113",
        },
        { value: "泷千家(天山西路店)", address: "天山西路438号" },
        {
          value: "胖仙女纸杯蛋糕（上海凌空店）",
          address: "上海市长宁区金钟路968号1幢18号楼一层商铺18-101",
        },
        { value: "贡茶", address: "上海市长宁区金钟路633号" },
        {
          value: "豪大大香鸡排超级奶爸",
          address: "上海市嘉定区曹安公路曹安路1685号",
        },
        {
          value: "茶芝兰（奶茶，手抓饼）",
          address: "上海市普陀区同普路1435号",
        },
        { value: "十二泷町", address: "上海市北翟路1444弄81号B幢-107" },
        { value: "星移浓缩咖啡", address: "上海市嘉定区新郁路817号" },
        { value: "阿姨奶茶/豪大大", address: "嘉定区曹安路1611号" },
        { value: "新麦甜四季甜品炸鸡", address: "嘉定区曹安公路2383弄55号" },
        {
          value: "Monica摩托主题咖啡店",
          address: "嘉定区江桥镇曹安公路2409号1F，2383弄62号1F",
        },
        {
          value: "浮生若茶（凌空soho店）",
          address: "上海长宁区金钟路968号9号楼地下一层",
        },
        { value: "NONO JUICE  鲜榨果汁", address: "上海市长宁区天山西路119号" },
        { value: "CoCo都可(北新泾店）", address: "上海市长宁区仙霞西路" },
        {
```

```
          value: "快乐柠檬（神州智慧店）",
          address: "上海市长宁区天山西路567号1层R117号店铺",
        },
        {
          value: "Merci Paul cafe",
          address: "上海市普陀区光复西路丹巴路28弄6号楼819",
        },
        {
          value: "猫山王（西郊百联店）",
          address: "上海市长宁区仙霞西路88号第一层G05-F01-1-306",
        },
        { value: "枪会山", address: "上海市普陀区棕榈路" },
        { value: "纵食", address: "元丰天山花园(东门) 双流路267号" },
        { value: "钱记", address: "上海市长宁区天山西路" },
        { value: "壹杯加", address: "上海市长宁区通协路" },
        {
          value: "吵哇嘀咖",
          address: "上海市长宁区新泾镇金钟路999号2幢（B幢）第01层第1-02A单元",
        },
        { value: "爱茜茜里(西郊百联)", address: "长宁区仙霞西路88号1305室" },
        {
          value: "爱茜茜里(近铁广场)",
          address:
            "上海市普陀区真北路818号近铁城市广场北区地下二楼N-B2-O2-C商铺",
        },
        {
          value: "鲜果榨汁（金沙江路和美广店）",
          address: "普陀区金沙江路2239号金沙和美广场B1-10-6",
        },
        {
          value: "开心丽果（缤谷店）",
          address: "上海市长宁区威宁路天山路341号",
        },
        { value: "超级鸡车（丰庄路店）", address: "上海市嘉定区丰庄路240号" },
        { value: "妙生活果园（北新泾店）", address: "长宁区新渔路144号" },
        { value: "香宜度麻辣香锅", address: "长宁区淞虹路148号" },
        {
          value: "凡仔汉堡（老真北路店）",
          address: "上海市普陀区老真北路160号",
        },
        { value: "港式小铺", address: "上海市长宁区金钟路968号15楼15-105室" },
        { value: "蜀香源麻辣香锅（剑河路店）", address: "剑河路443-1" },
```

```
        { value: "北京饺子馆", address: "长宁区北新泾街道天山西路490-1号" },
        {
          value: "饭典*新简餐（凌空SOHO店）",
          address: "上海市长宁区金钟路968号9号楼地下一层9-83室",
        },
        {
          value: "焦耳·川式快餐（金钟路店）",
          address: "上海市金钟路633号地下一层甲部",
        },
        { value: "动力鸡车", address: "长宁区仙霞西路299弄3号101B" },
        { value: "浏阳蒸菜", address: "天山西路430号" },
        { value: "四海游龙（天山西路店）", address: "上海市长宁区天山西路" },
        {
          value: "樱花食堂（凌空店）",
          address: "上海市长宁区金钟路968号15楼15-105室",
        },
        { value: "壹分米客家传统调制米粉(天山店)", address: "天山西路428号" },
        {
          value: "福荣祥烧腊（平溪路店）",
          address: "上海市长宁区协和路福泉路255弄57-73号",
        },
        {
          value: "速记黄焖鸡米饭",
          address: "上海市长宁区北新泾街道金钟路180号1层01号摊位",
        },
        { value: "红辣椒麻辣烫", address: "上海市长宁区天山西路492号" },
        {
          value: "(小杨生煎)西郊百联餐厅",
          address: "长宁区仙霞西路88号百联2楼",
        },
        { value: "阳阳麻辣烫", address: "天山西路389号" },
        {
          value: "南拳妈妈龙虾盖浇饭",
          address: "普陀区金沙江路1699号鑫乐惠美食广场A13",
        },
      ];
    };
    const handleSelect = (item) => {
      console.log(item);
    };
    onMounted(() => {
      restaurants.value = loadAll();
```

```
    });
    return {
      restaurants,
      state1: ref(''),
      state2: ref(''),
      querySearch,
      createFilter,
      loadAll,
      handleSelect,
    };
  },
});
</script>
```

自定义模板

可自定义输入建议的显示

使用 `#default` 自定义输入建议的模板。该 scope 的参数为 `item` ，表示当前输入建议对象。

```
<el-autocomplete
  popper-class="my-autocomplete"
  v-model="state"
  :fetch-suggestions="querySearch"
  placeholder="请输入内容"
  @select="handleSelect"
>
  <template #suffix>
    <i class="el-icon-edit el-input__icon" @click="handleIconClick"> </i>
  </template>
  <template #default="{ item }">
    <div class="name">{{ item.value }}</div>
    <span class="addr">{{ item.address }}</span>
  </template>
</el-autocomplete>

<style lang="scss">
.my-autocomplete {
```

```
  li {
    line-height: normal;
    padding: 7px;

    .name {
      text-overflow: ellipsis;
      overflow: hidden;
    }
    .addr {
      font-size: 12px;
      color: #b4b4b4;
    }

    .highlighted .addr {
      color: #ddd;
    }
  }
}
</style>

<script>
import { defineComponent, ref, onMounted } from 'vue'

export default defineComponent({
  setup() {
    const restaurants = ref([]);

    const querySearch = (queryString, cb) => {
      var results = queryString
        ? restaurants.value.filter(createFilter(queryString))
        : restaurants.value;
      // 调用 callback 返回建议列表的数据
      cb(results);
    };
    const createFilter = (queryString) => {
      return (restaurant) => {
        return (
          restaurant.value.toLowerCase().indexOf(queryString.toLowerCase()) ===
          0
        );
      };
    };
```

```
    const loadAll = () => {
      return [
        { value: "三全鲜食（北新泾店）", address: "长宁区新渔路144号" },
        {
          value: "Hot honey 首尔炸鸡（仙霞路）",
          address: "上海市长宁区淞虹路661号",
        },
        {
          value: "新旺角茶餐厅",
          address: "上海市普陀区真北路988号创邑金沙谷6号楼113",
        },
        { value: "泷千家(天山西路店)", address: "天山西路438号" },
        {
          value: "胖仙女纸杯蛋糕（上海凌空店）",
          address: "上海市长宁区金钟路968号1幢18号楼一层商铺18-101",
        },
        { value: "贡茶", address: "上海市长宁区金钟路633号" },
        {
          value: "豪大大香鸡排超级奶爸",
          address: "上海市嘉定区曹安公路曹安路1685号",
        },
        {
          value: "茶芝兰（奶茶，手抓饼）",
          address: "上海市普陀区同普路1435号",
        },
        { value: "十二泷町", address: "上海市北翟路1444弄81号B幢-107" },
        { value: "星移浓缩咖啡", address: "上海市嘉定区新郁路817号" },
        { value: "阿姨奶茶/豪大大", address: "嘉定区曹安路1611号" },
        { value: "新麦甜四季甜品炸鸡", address: "嘉定区曹安公路2383弄55号" },
        {
          value: "Monica摩托主题咖啡店",
          address: "嘉定区江桥镇曹安公路2409号1F，2383弄62号1F",
        },
        {
          value: "浮生若茶（凌空soho店）",
          address: "上海长宁区金钟路968号9号楼地下一层",
        },
        { value: "NONO JUICE  鲜榨果汁", address: "上海市长宁区天山西路119号" },
        { value: "CoCo都可(北新泾店）", address: "上海市长宁区仙霞西路" },
        {
          value: "快乐柠檬（神州智慧店）",
          address: "上海市长宁区天山西路567号1层R117号店铺",
```

```
        },
        {
          value: "Merci Paul cafe",
          address: "上海市普陀区光复西路丹巴路28弄6号楼819",
        },
        {
          value: "猫山王（西郊百联店）",
          address: "上海市长宁区仙霞西路88号第一层G05-F01-1-306",
        },
        { value: "枪会山", address: "上海市普陀区棕榈路" },
        { value: "纵食", address: "元丰天山花园(东门) 双流路267号" },
        { value: "钱记", address: "上海市长宁区天山西路" },
        { value: "壹杯加", address: "上海市长宁区通协路" },
        {
          value: "吵哇嘀咖",
          address: "上海市长宁区新泾镇金钟路999号2幢（B幢）第01层第1-02A单元",
        },
        { value: "爱茜茜里(西郊百联)", address: "长宁区仙霞西路88号1305室" },
        {
          value: "爱茜茜里(近铁广场)",
          address:
            "上海市普陀区真北路818号近铁城市广场北区地下二楼N-B2-O2-C商铺",
        },
        {
          value: "鲜果榨汁（金沙江路和美广店）",
          address: "普陀区金沙江路2239号金沙和美广场B1-10-6",
        },
        {
          value: "开心丽果（缤谷店）",
          address: "上海市长宁区威宁路天山路341号",
        },
        { value: "超级鸡车（丰庄路店）", address: "上海市嘉定区丰庄路240号" },
        { value: "妙生活果园（北新泾店）", address: "长宁区新渔路144号" },
        { value: "香宜度麻辣香锅", address: "长宁区淞虹路148号" },
        {
          value: "凡仔汉堡（老真北路店）",
          address: "上海市普陀区老真北路160号",
        },
        { value: "港式小铺", address: "上海市长宁区金钟路968号15楼15-105室" },
        { value: "蜀香源麻辣香锅（剑河路店）", address: "剑河路443-1" },
        { value: "北京饺子馆", address: "长宁区北新泾街道天山西路490-1号" },
        {
```

```
            value: "饭典*新简餐（凌空SOHO店）",
            address: "上海市长宁区金钟路968号9号楼地下一层9-83室",
          },
          {
            value: "焦耳·川式快餐（金钟路店）",
            address: "上海市金钟路633号地下一层甲部",
          },
          { value: "动力鸡车", address: "长宁区仙霞西路299弄3号101B" },
          { value: "浏阳蒸菜", address: "天山西路430号" },
          { value: "四海游龙（天山西路店）", address: "上海市长宁区天山西路" },
          {
            value: "樱花食堂（凌空店）",
            address: "上海市长宁区金钟路968号15楼15-105室",
          },
          { value: "壹分米客家传统调制米粉(天山店)", address: "天山西路428号" },
          {
            value: "福荣祥烧腊（平溪路店）",
            address: "上海市长宁区协和路福泉路255弄57-73号",
          },
          {
            value: "速记黄焖鸡米饭",
            address: "上海市长宁区北新泾街道金钟路180号1层01号摊位",
          },
          { value: "红辣椒麻辣烫", address: "上海市长宁区天山西路492号" },
          {
            value: "(小杨生煎)西郊百联餐厅",
            address: "长宁区仙霞西路88号百联2楼",
          },
          { value: "阳阳麻辣烫", address: "天山西路389号" },
          {
            value: "南拳妈妈龙虾盖浇饭",
            address: "普陀区金沙江路1699号鑫乐惠美食广场A13",
          },
        ];
      };
      const handleSelect = (item) => {
        console.log(item);
      };

      const handleIconClick = (ev) => {
        console.log(ev);
      };
```

```
187.
188.     onMounted(() => {
189.       restaurants.value = loadAll();
190.     });
191.
192.     return {
193.       restaurants,
194.       state: ref(''),
195.       querySearch,
196.       createFilter,
197.       loadAll,
198.       handleSelect,
199.       handleIconClick,
200.     };
201.   },
202. });
203. </script>
```

远程搜索

从服务端搜索数据


```
1.  <el-autocomplete
2.    v-model="state"
3.    :fetch-suggestions="querySearchAsync"
4.    placeholder="请输入内容"
5.    @select="handleSelect"
6.  ></el-autocomplete>
7.  <script>
8.  import { defineComponent, ref, onMounted } from 'vue'
9.
10. export default defineComponent({
11.   setup() {
12.     const restaurants = ref([]);
13.     const loadAll = () => {
14.       return [
15.         { value: "三全鲜食（北新泾店）", address: "长宁区新渔路144号" },
16.         {
```

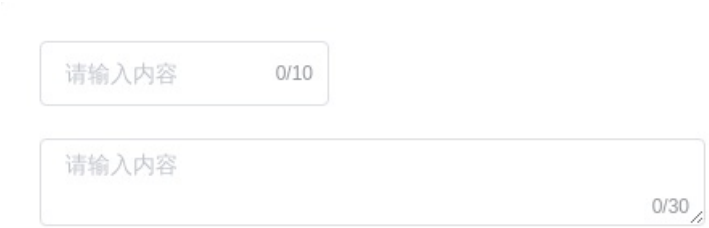
```
          value: "Hot honey 首尔炸鸡（仙霞路）",
          address: "上海市长宁区淞虹路661号",
        },
        {
          value: "新旺角茶餐厅",
          address: "上海市普陀区真北路988号创邑金沙谷6号楼113",
        },
        { value: "泷千家(天山西路店)", address: "天山西路438号" },
        {
          value: "胖仙女纸杯蛋糕（上海凌空店）",
          address: "上海市长宁区金钟路968号1幢18号楼一层商铺18-101",
        },
        { value: "贡茶", address: "上海市长宁区金钟路633号" },
        {
          value: "豪大大香鸡排超级奶爸",
          address: "上海市嘉定区曹安公路曹安路1685号",
        },
        {
          value: "茶芝兰（奶茶，手抓饼）",
          address: "上海市普陀区同普路1435号",
        },
        { value: "十二泷町", address: "上海市北翟路1444弄81号B幢-107" },
        { value: "星移浓缩咖啡", address: "上海市嘉定区新郁路817号" },
        { value: "阿姨奶茶/豪大大", address: "嘉定区曹安路1611号" },
        { value: "新麦甜四季甜品炸鸡", address: "嘉定区曹安公路2383弄55号" },
        {
          value: "Monica摩托主题咖啡店",
          address: "嘉定区江桥镇曹安公路2409号1F，2383弄62号1F",
        },
        {
          value: "浮生若茶（凌空soho店）",
          address: "上海长宁区金钟路968号9号楼地下一层",
        },
        { value: "NONO JUICE  鲜榨果汁", address: "上海市长宁区天山西路119号" },
        { value: "CoCo都可(北新泾店）", address: "上海市长宁区仙霞西路" },
        {
          value: "快乐柠檬（神州智慧店）",
          address: "上海市长宁区天山西路567号1层R117号店铺",
        },
        {
          value: "Merci Paul cafe",
          address: "上海市普陀区光复西路丹巴路28弄6号楼819",
```

```
        },
        {
          value: "猫山王（西郊百联店）",
          address: "上海市长宁区仙霞西路88号第一层G05-F01-1-306",
        },
        { value: "枪会山", address: "上海市普陀区棕榈路" },
        { value: "纵食", address: "元丰天山花园(东门) 双流路267号" },
        { value: "钱记", address: "上海市长宁区天山西路" },
        { value: "壹杯加", address: "上海市长宁区通协路" },
        {
          value: "吵哇嘀咖",
          address: "上海市长宁区新泾镇金钟路999号2幢（B幢）第01层第1-02A单元",
        },
        { value: "爱茜茜里(西郊百联)", address: "长宁区仙霞西路88号1305室" },
        {
          value: "爱茜茜里(近铁广场)",
          address:
            "上海市普陀区真北路818号近铁城市广场北区地下二楼N-B2-O2-C商铺",
        },
        {
          value: "鲜果榨汁（金沙江路和美广店）",
          address: "普陀区金沙江路2239号金沙和美广场B1-10-6",
        },
        {
          value: "开心丽果（缤谷店）",
          address: "上海市长宁区威宁路天山路341号",
        },
        { value: "超级鸡车（丰庄路店）", address: "上海市嘉定区丰庄路240号" },
        { value: "妙生活果园（北新泾店）", address: "长宁区新渔路144号" },
        { value: "香宜度麻辣香锅", address: "长宁区淞虹路148号" },
        {
          value: "凡仔汉堡（老真北路店）",
          address: "上海市普陀区老真北路160号",
        },
        { value: "港式小铺", address: "上海市长宁区金钟路968号15楼15-105室" },
        { value: "蜀香源麻辣香锅（剑河路店）", address: "剑河路443-1" },
        { value: "北京饺子馆", address: "长宁区北新泾街道天山西路490-1号" },
        {
          value: "饭典*新简餐（凌空SOHO店）",
          address: "上海市长宁区金钟路968号9号楼地下一层9-83室",
        },
        {
```

```
            value: "焦耳·川式快餐（金钟路店）",
            address: "上海市金钟路633号地下一层甲部",
          },
          { value: "动力鸡车", address: "长宁区仙霞西路299弄3号101B" },
          { value: "浏阳蒸菜", address: "天山西路430号" },
          { value: "四海游龙（天山西路店）", address: "上海市长宁区天山西路" },
          {
            value: "樱花食堂（凌空店）",
            address: "上海市长宁区金钟路968号15楼15-105室",
          },
          { value: "壹分米客家传统调制米粉(天山店)", address: "天山西路428号" },
          {
            value: "福荣祥烧腊（平溪路店）",
            address: "上海市长宁区协和路福泉路255弄57-73号",
          },
          {
            value: "速记黄焖鸡米饭",
            address: "上海市长宁区北新泾街道金钟路180号1层01号摊位",
          },
          { value: "红辣椒麻辣烫", address: "上海市长宁区天山西路492号" },
          {
            value: "(小杨生煎)西郊百联餐厅",
            address: "长宁区仙霞西路88号百联2楼",
          },
          { value: "阳阳麻辣烫", address: "天山西路389号" },
          {
            value: "南拳妈妈龙虾盖浇饭",
            address: "普陀区金沙江路1699号鑫乐惠美食广场A13",
          },
        ];
      };

      let timeout;
      const querySearchAsync = (queryString, cb) => {
        var results = queryString
          ? restaurants.value.filter(createFilter(queryString))
          : restaurants.value;

        clearTimeout(timeout);
        timeout = setTimeout(() => {
          cb(results);
        }, 3000 * Math.random());
```

```
    };
    const createFilter = (queryString) => {
      return (restaurant) => {
        return (
          restaurant.value.toLowerCase().indexOf(queryString.toLowerCase()) ===
          0
        );
      };
    };
    const handleSelect = (item) => {
      console.log(item);
    };
    onMounted(() => {
      restaurants.value = loadAll();
    });
    return {
      restaurants,
      state: ref(''),
      querySearchAsync,
      createFilter,
      loadAll,
      handleSelect,
    };
  },
});
</script>
```

输入长度限制



`maxlength` 和 `minlength` 是原生属性，用来限制输入框的字符长度，其中字符长度是用 Javascript 的字符串长度统计的。对于类型为 `text` 或 `textarea` 的输入框，在使用 `maxlength` 属性限制最大输入长度的同时，可通过设置 `show-word-limit` 属性来展示字数统计。

```
<el-input
```

```
    type="text"
    placeholder="请输入内容"
    v-model="text"
    maxlength="10"
    show-word-limit
  >
  </el-input>
  <div style="margin: 20px 0;"></div>
  <el-input
    type="textarea"
    placeholder="请输入内容"
    v-model="textarea"
    maxlength="30"
    show-word-limit
  >
  </el-input>

  <script>
  import { defineComponent, ref } from 'vue'

  export default defineComponent ({
    setup() {
      return {
        text: ref(''),
        textarea: ref('')
      }
    }
  })
  </script>
```

Input Attributes

参数	说明	类型	可选值	默认值
type	类型	string	text, textarea 和其他 原生 input 的 type 值	text
value / v-model	绑定值	string / number	—	—
maxlength	原生属性，最大输入长度	number	—	—
minlength	原生属性，最小输入长度	number	—	—
show-word-	是否显示输入字数统计，只在 type =			

limit	“text” 或 type = “textarea” 时有效	boolean	—	false
placeholder	输入框占位文本	string	—	—
clearable	是否可清空	boolean	—	false
show-password	是否显示切换密码图标	boolean	—	false
disabled	禁用	boolean	—	false
size	输入框尺寸，只在 type!=”textarea” 时有效	string	medium / small / mini	—
prefix-icon	输入框头部图标	string	—	—
suffix-icon	输入框尾部图标	string	—	—
rows	输入框行数，只对 type=”textarea” 有效	number	—	2
autosize	自适应内容高度，只对 type=”textarea” 有效，可传入对象，如，{ minRows: 2, maxRows: 6 }	boolean / object	—	false
autocomplete	原生属性，自动补全	string	on, off	off
auto-complete	下个主版本弃用	string	on, off	off
name	原生属性	string	—	—
readonly	原生属性，是否只读	boolean	—	false
max	原生属性，设置最大值	—	—	—
min	原生属性，设置最小值	—	—	—
step	原生属性，设置输入字段的合法数字间隔	—	—	—
resize	控制是否能被用户缩放	string	none, both, horizontal, vertical	—
autofocus	原生属性，自动获取焦点	boolean	true, false	false
form	原生属性	string	—	—
label	输入框关联的label文字	string	—	—
tabindex	输入框的tabindex	string	-	-
validate-event	输入时是否触发表单的校验	boolean	-	true

Input Slots

name	说明
prefix	输入框头部内容，只对 type=”text” 有效
suffix	输入框尾部内容，只对 type=”text” 有效
prepend	输入框前置内容，只对 type=”text” 有效
append	输入框后置内容，只对 type=”text” 有效

Input Events

事件名称	说明	回调参数
blur	在 Input 失去焦点时触发	(event: Event)
focus	在 Input 获得焦点时触发	(event: Event)
change	仅在输入框失去焦点或用户按下回车时触发	(value: string | number)
input	在 Input 值改变时触发	(value: string | number)
clear	在点击由 `clearable` 属性生成的清空按钮时触发	—

Input Methods

方法名	说明	参数
focus	使 input 获取焦点	—
blur	使 input 失去焦点	—
select	选中 input 中的文字	—

Autocomplete Attributes

参数	说明	类型	可选值	默认值
placeholder	输入框占位文本	string	—	—
disabled	禁用	boolean	—	false
value-key	输入建议对象中用于显示的键名	string	—	value
value	必填值，输入绑定值	string	—	—
debounce	获取输入建议的去抖延时	number	—	300
placement	菜单弹出位置	string	top / top-start / top-end / bottom / bottom-start / bottom-end	bottom-start
fetch-suggestions	返回输入建议的方法，仅当你的输入建议数据 resolve 时，通过调用 callback(data:[]) 来返回它	Function(queryString, callback)	—	—
popper-class	Autocomplete 下拉列表的类名	string	—	—
trigger-on-focus	是否在输入框 focus 时显示建议列表	boolean	—	true
name	原生属性	string	—	—
select-when-	在输入没有任何匹配建议的情况下，按下回车是否触发	boolean	—	false

unmatched	`select` 事件			
label	输入框关联的label文字	string	—	—
prefix-icon	输入框头部图标	string	—	—
suffix-icon	输入框尾部图标	string	—	—
hide-loading	是否隐藏远程加载时的加载图标	boolean	—	false
popper-append-to-body	是否将下拉列表插入至 body 元素。在下拉列表的定位出现问题时，可将该属性设置为 false	boolean	-	true
highlight-first-item	是否默认突出显示远程搜索建议中的第一项	boolean	—	false

Autocomplete Slots

name	说明
prefix	输入框头部内容
suffix	输入框尾部内容
prepend	输入框前置内容
append	输入框后置内容

Autocomplete Scoped Slot

name	说明
—	自定义输入建议，参数为 { item }

Autocomplete Events

事件名称	说明	回调参数
select	点击选中建议项时触发	选中建议项
change	在 Input 值改变时触发	(value: string | number)

Autocomplete Methods

方法名	说明	参数
focus	使 input 获取焦点	-

InputNumber 计数器

仅允许输入标准的数字值，可定义范围

基础用法

要使用它，只需要在 `el-input-number` 元素中使用 `v-model` 绑定变量即可，变量的初始值即为默认值。

```
<template>
  <el-input-number v-model="num" @change="handleChange" :min="1" :max="10"
label="描述文字"></el-input-number>
</template>
<script>
  export default {
    data() {
      return {
        num: 1
      };
    },
    methods: {
      handleChange(value) {
        console.log(value);
      }
    }
  };
</script>
```

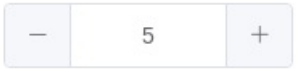
禁用状态

`disabled` 属性接受一个 `Boolean` ，设置为 `true` 即可禁用整个组件，如果你只需要控制数值在某一范围内，可以设置 `min` 属性和 `max` 属性，不设置 `min` 和 `max` 时，最小值为 `0`。

```
<template>
  <el-input-number v-model="num" :disabled="true"></el-input-number>
</template>
<script>
  export default {
    data() {
      return {
        num: 1
      }
    }
  };
</script>
```

步数

允许定义递增递减的步数控制



设置 `step` 属性可以控制步长，接受一个 `Number` 。

```
<template>
  <el-input-number v-model="num" :step="2"></el-input-number>
</template>
<script>
  export default {
    data() {
      return {
        num: 5
      }
    }
  };
</script>
```

严格步数

`step-strictly` 属性接受一个 `Boolean` 。如果这个属性被设置为 `true` ，则只能输入步数的倍数。

```
<template>
  <el-input-number v-model="num" :step="2" step-strictly></el-input-number>
</template>
<script>
  export default {
    data() {
      return {
        num: 2
      }
    }
  };
</script>
```

精度

设置 `precision` 属性可以控制数值精度，接收一个 `Number` 。

```
<template>
  <el-input-number v-model="num" :precision="2" :step="0.1" :max="10"></el-
input-number>
</template>
<script>
  export default {
    data() {
      return {
        num: 1
      }
    }
  };
</script>
```

`precision` 的值必须是一个非负整数，并且不能小于 `step` 的小数位数。

尺寸

额外提供了 `medium` 、 `small` 、 `mini` 三种尺寸的数字输入框

```
<template>
  <el-input-number v-model="num1"></el-input-number>
  <el-input-number size="medium" v-model="num2"></el-input-number>
  <el-input-number size="small" v-model="num3"></el-input-number>
  <el-input-number size="mini" v-model="num4"></el-input-number>
</template>
<script>
  export default {
    data() {
      return {
        num1: 1,
        num2: 1,
        num3: 1,
        num4: 1
      }
    }
  };
</script>
```

按钮位置

设置 `controls-position` 属性可以控制按钮位置。

```
<template>
  <el-input-number v-model="num" controls-position="right"
@change="handleChange" :min="1" :max="10"></el-input-number>
</template>
<script>
  export default {
    data() {
      return {
```

```
        num: 1
      };
    },
    methods: {
      handleChange(value) {
        console.log(value);
      }
    }
  };
</script>
```

Attributes

参数	说明	类型	可选值	默认值
modelValue / v-model	绑定值	number	—	0
min	设置计数器允许的最小值	number	—	-Infinity
max	设置计数器允许的最大值	number	—	Infinity
step	计数器步长	number	—	1
step-strictly	是否只能输入 step 的倍数	boolean	—	false
precision	数值精度	number	—	—
size	计数器尺寸	string	large/medium/small/mini	large
disabled	是否禁用计数器	boolean	—	false
controls	是否使用控制按钮	boolean	—	true
controls-position	控制按钮位置	string	right	-
name	原生属性	string	—	—
label	输入框关联的label文字	string	—	—
placeholder	输入框默认 placeholder	string	-	-

Events

事件名称	说明	回调参数
change	绑定值被改变时触发	currentValue, oldValue
blur	在组件 Input 失去焦点时触发	(event: Event)
focus	在组件 Input 获得焦点时触发	(event: Event)

Methods

方法名	说明	参数
focus	使 input 获取焦点	-
select	选中 input 中的文字	—

Select 选择器

当选项过多时，使用下拉菜单展示并选择内容。

基础用法

适用广泛的基础单选

`v-model` 的值为当前被选中的 `el-option` 的 value 属性值

```
<template>
  <el-select v-model="value" placeholder="请选择">
    <el-option
      v-for="item in options"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
</template>

<script>
  export default {
    data() {
      return {
        options: [{
          value: '选项1',
          label: '黄金糕'
        }, {
          value: '选项2',
          label: '双皮奶'
        }, {
          value: '选项3',
          label: '蚵仔煎'
        }, {
          value: '选项4',
          label: '龙须面'
```

```
        }, {
          value: '选项5',
          label: '北京烤鸭'
        }],
        value: ''
      }
    }
  }
</script>
```

有禁用选项

在 `el-option` 中，设定 `disabled` 值为 `true`，即可禁用该选项

```
<template>
  <el-select v-model="value" placeholder="请选择">
    <el-option
      v-for="item in options"
      :key="item.value"
      :label="item.label"
      :value="item.value"
      :disabled="item.disabled">
    </el-option>
  </el-select>
</template>

<script>
  export default {
    data() {
      return {
        options: [{
          value: '选项1',
          label: '黄金糕'
        }, {
          value: '选项2',
          label: '双皮奶',
          disabled: true
        }, {
```

```
          value: '选项3',
          label: '蚵仔煎'
        }, {
          value: '选项4',
          label: '龙须面'
        }, {
          value: '选项5',
          label: '北京烤鸭'
        }],
        value: ''
      }
    }
  }
</script>
```

禁用状态

选择器不可用状态

为 `el-select` 设置 `disabled` 属性，则整个选择器不可用

```
<template>
  <el-select v-model="value" disabled placeholder="请选择">
    <el-option
      v-for="item in options"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
</template>

<script>
  export default {
    data() {
      return {
        options: [{
          value: '选项1',
```

```
          label: '黄金糕'
        }, {
          value: '选项2',
          label: '双皮奶'
        }, {
          value: '选项3',
          label: '蚵仔煎'
        }, {
          value: '选项4',
          label: '龙须面'
        }, {
          value: '选项5',
          label: '北京烤鸭'
        }],
        value: ''
      }
    }
  }
</script>
```

可清空单选

包含清空按钮，可将选择器清空为初始状态

为 `el-select` 设置 `clearable` 属性，则可将选择器清空。需要注意的是， `clearable` 属性仅适用于单选。

```
<template>
  <el-select v-model="value" clearable placeholder="请选择">
    <el-option
      v-for="item in options"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
</template>

```

```
<script>
  export default {
    data() {
      return {
        options: [{
          value: '选项1',
          label: '黄金糕'
        }, {
          value: '选项2',
          label: '双皮奶'
        }, {
          value: '选项3',
          label: '蚵仔煎'
        }, {
          value: '选项4',
          label: '龙须面'
        }, {
          value: '选项5',
          label: '北京烤鸭'
        }],
        value: ''
      }
    }
  }
</script>
```

基础多选

适用性较广的基础多选，用 Tag 展示已选项

为 `el-select` 设置 `multiple` 属性即可启用多选，此时 `v-model` 的值为当前选中值所组成的数组。默认情况下选中值会以 Tag 的形式展现，你也可以设置 `collapse-tags` 属性将它们合并为一段文字。

```
<template>
  <el-select v-model="value1" multiple placeholder="请选择">
    <el-option
      v-for="item in options"
```

```
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>

  <el-select
    v-model="value2"
    multiple
    collapse-tags
    style="margin-left: 20px;"
    placeholder="请选择">
    <el-option
      v-for="item in options"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
</template>

<script>
  export default {
    data() {
      return {
        options: [{
          value: '选项1',
          label: '黄金糕'
        }, {
          value: '选项2',
          label: '双皮奶'
        }, {
          value: '选项3',
          label: '蚵仔煎'
        }, {
          value: '选项4',
          label: '龙须面'
        }, {
          value: '选项5',
          label: '北京烤鸭'
        }],
        value1: [],
```

```
        value2: []
      }
    }
  }
</script>
```

自定义模板

可以自定义备选项

将自定义的 HTML 模板插入 `el-option` 的 slot 中即可。

```
<template>
  <el-select v-model="value" placeholder="请选择">
    <el-option
      v-for="item in cities"
      :key="item.value"
      :label="item.label"
      :value="item.value">
      <span style="float: left">{{ item.label }}</span>
      <span style="float: right; color: #8492a6; font-size: 13px">{{ item.value
}}</span>
    </el-option>
  </el-select>
</template>

<script>
  export default {
    data() {
      return {
        cities: [{
          value: 'Beijing',
          label: '北京'
        }, {
          value: 'Shanghai',
          label: '上海'
        }, {
          value: 'Nanjing',
```

```
          label: '南京'
        }, {
          value: 'Chengdu',
          label: '成都'
        }, {
          value: 'Shenzhen',
          label: '深圳'
        }, {
          value: 'Guangzhou',
          label: '广州'
        }],
        value: ''
      }
    }
  }
</script>
```

分组

备选项进行分组展示

使用 `el-option-group` 对备选项进行分组，它的 `label` 属性为分组名

```
<template>
  <el-select v-model="value" placeholder="请选择">
    <el-option-group
      v-for="group in options"
      :key="group.label"
      :label="group.label">
      <el-option
        v-for="item in group.options"
        :key="item.value"
        :label="item.label"
        :value="item.value">
      </el-option>
    </el-option-group>
  </el-select>
</template>
```

```
16.
17.  <script>
18.    export default {
19.      data() {
20.        return {
21.          options: [{
22.            label: '热门城市',
23.            options: [{
24.              value: 'Shanghai',
25.              label: '上海'
26.            }, {
27.              value: 'Beijing',
28.              label: '北京'
29.            }]
30.          }, {
31.            label: '城市名',
32.            options: [{
33.              value: 'Chengdu',
34.              label: '成都'
35.            }, {
36.              value: 'Shenzhen',
37.              label: '深圳'
38.            }, {
39.              value: 'Guangzhou',
40.              label: '广州'
41.            }, {
42.              value: 'Dalian',
43.              label: '大连'
44.            }]
45.          }],
46.          value: ''
47.        }
48.      }
49.    }
50.  </script>
```

可搜索

可以利用搜索功能快速查找选项

为 `el-select` 添加 `filterable` 属性即可启用搜索功能。默认情况下，Select 会找出所有 `label` 属性包含输入值的选项。如果希望使用其他的搜索逻辑，可以通过传入一个 `filter-method` 来实现。 `filter-method` 为一个 `Function` ，它会在输入值发生变化时调用，参数为当前输入值。

```
<template>
  <el-select v-model="value" filterable placeholder="请选择">
    <el-option
      v-for="item in options"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
</template>

<script>
  export default {
    data() {
      return {
        options: [{
          value: '选项1',
          label: '黄金糕'
        }, {
          value: '选项2',
          label: '双皮奶'
        }, {
          value: '选项3',
          label: '蚵仔煎'
        }, {
          value: '选项4',
          label: '龙须面'
        }, {
          value: '选项5',
          label: '北京烤鸭'
        }],
        value: ''
      }
```

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

远程搜索

从服务器搜索数据，输入关键字进行查找

为了启用远程搜索，需要将 `filterable` 和 `remote` 设置为 `true` ，同时传入一个 `remote-method` 。 `remote-method` 为一个 `Function` ，它会在输入值发生变化时调用，参数为当前输入值。需要注意的是，如果 `el-option` 是通过 `v-for` 指令渲染出来的，此时需要为 `el-option` 添加 `key` 属性，且其值需具有唯一性，比如此例中的 `item.value` 。

```
<template>
  <el-select
    v-model="value"
    multiple
    filterable
    remote
    reserve-keyword
    placeholder="请输入关键词"
    :remote-method="remoteMethod"
    :loading="loading">
    <el-option
      v-for="item in options"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
</template>

<script>
  export default {
    data() {
      return {
        options: [],
        value: [],
```

```
        list: [],
        loading: false,
        states: ["Alabama", "Alaska", "Arizona",
        "Arkansas", "California", "Colorado",
        "Connecticut", "Delaware", "Florida",
        "Georgia", "Hawaii", "Idaho", "Illinois",
        "Indiana", "Iowa", "Kansas", "Kentucky",
        "Louisiana", "Maine", "Maryland",
        "Massachusetts", "Michigan", "Minnesota",
        "Mississippi", "Missouri", "Montana",
        "Nebraska", "Nevada", "New Hampshire",
        "New Jersey", "New Mexico", "New York",
        "North Carolina", "North Dakota", "Ohio",
        "Oklahoma", "Oregon", "Pennsylvania",
        "Rhode Island", "South Carolina",
        "South Dakota", "Tennessee", "Texas",
        "Utah", "Vermont", "Virginia",
        "Washington", "West Virginia", "Wisconsin",
        "Wyoming"]
      }
    },
    mounted() {
      this.list = this.states.map(item => {
        return { value: `value:${item}`, label: `label:${item}` };
      });
    },
    methods: {
      remoteMethod(query) {
        if (query !== '') {
          this.loading = true;
          setTimeout(() => {
            this.loading = false;
            this.options = this.list.filter(item => {
              return item.label.toLowerCase()
                .indexOf(query.toLowerCase()) > -1;
            });
          }, 200);
        } else {
          this.options = [];
        }
      }
    }
```

```
  }
</script>
```

创建条目

可以创建并选中选项中不存在的条目

使用 `allow-create` 属性即可通过在输入框中输入文字来创建新的条目。注意此时 `filterable` 必须为真。本例还使用了 `default-first-option` 属性，在该属性打开的情况下，按下回车就可以选中当前选项列表中的第一个选项，无需使用鼠标或键盘方向键进行定位。

```
<template>
  <el-select
    v-model="value"
    multiple
    filterable
    allow-create
    default-first-option
    placeholder="请选择文章标签">
    <el-option
      v-for="item in options"
      :key="item.value"
      :label="item.label"
      :value="item.value">
    </el-option>
  </el-select>
</template>

<script>
  export default {
    data() {
      return {
        options: [{
          value: 'HTML',
          label: 'HTML'
        }, {
          value: 'CSS',
          label: 'CSS'
```

```
28.         }, {
29.           value: 'JavaScript',
30.           label: 'JavaScript'
31.         }],
32.         value: []
33.       }
34.     }
35.   }
36. </script>
```

如果 Select 的绑定值为对象类型，请务必指定 `value-key` 作为它的唯一性标识。

Select Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	boolean / string / number	—	—
multiple	是否多选	boolean	—	false
disabled	是否禁用	boolean	—	false
value-key	作为 value 唯一标识的键名，绑定值为对象类型时必填	string	—	value
size	输入框尺寸	string	medium/small/mini	—
clearable	是否可以清空选项	boolean	—	false
collapse-tags	多选时是否将选中值按文字的形式展示	boolean	—	false
multiple-limit	多选时用户最多可以选择的项目数，为 0 则不限制	number	—	0
name	select input 的 name 属性	string	—	—
autocomplete	select input 的 autocomplete 属性	string	—	off
auto-complete	下个主版本弃用	string	—	off
placeholder	占位符	string	—	请选择
filterable	是否可搜索	boolean	—	false
allow-create	是否允许用户创建新条目，需配合 `filterable` 使用	boolean	—	false
filter-method	自定义搜索方法	function	—	—
remote	是否为远程搜索	boolean	—	false
remote-method	远程搜索方法	function	—	—
loading	是否正在从远程获取数据	boolean	—	false

loading-text	远程加载时显示的文字	string	—	加载中
no-match-text	搜索条件无匹配时显示的文字，也可以使用 #empty 设置	string	—	无匹配数据
no-data-text	选项为空时显示的文字，也可以使用 #empty 设置	string	—	无数据
popper-class	Select 下拉框的类名	string	—	—
reserve-keyword	多选且可搜索时，是否在选中一个选项后保留当前的搜索关键词	boolean	—	false
default-first-option	在输入框按下回车，选择第一个匹配项。需配合 filterable 或 remote 使用	boolean	-	false
popper-append-to-body	是否将弹出框插入至 body 元素。在弹出框的定位出现问题时，可将该属性设置为 false	boolean	-	true
automatic-dropdown	对于不可搜索的 Select，是否在输入框获得焦点后自动弹出选项菜单	boolean	-	false
clear-icon	自定义清空图标的类名	string	—	el-icon-circle-close

Select Events

事件名称	说明	回调参数
change	选中值发生变化时触发	目前的选中值
visible-change	下拉框出现/隐藏时触发	出现则为 true，隐藏则为 false
remove-tag	多选模式下移除tag时触发	移除的tag值
clear	可清空的单选模式下用户点击清空按钮时触发	—
blur	当 input 失去焦点时触发	(event: Event)
focus	当 input 获得焦点时触发	(event: Event)

Select Slots

name	说明
—	Option 组件列表
prefix	Select 组件头部内容
empty	无选项时的列表

Option Group Attributes

参数	说明	类型	可选值	默认值
label	分组的组名	string	—	—
disabled	是否将该分组下所有选项置为禁用	boolean	—	false

Option Attributes

参数	说明	类型	可选值	默认值
value	选项的值	string/number/object	—	—
label	选项的标签，若不设置则默认与 `value` 相同	string/number	—	—
disabled	是否禁用该选项	boolean	—	false

Methods

方法名	说明	参数
focus	使 input 获取焦点	-
blur	使 input 失去焦点，并隐藏下拉框	-

Cascader 级联选择器

当一个数据集合有清晰的层级结构时，可通过级联选择器逐级查看并选择。

基础用法

有两种触发子菜单的方式

只需为 Cascader 的 `options` 属性指定选项数组即可渲染出一个级联选择器。通过 `props.expandTrigger` 可以定义展开子级菜单的触发方式。

```
<div class="block">
  <span class="demonstration">默认 click 触发子菜单</span>
  <el-cascader
    v-model="value"
    :options="options"
    @change="handleChange"></el-cascader>
</div>
<div class="block">
  <span class="demonstration">hover 触发子菜单</span>
  <el-cascader
    v-model="value"
    :options="options"
    :props="props"
    @change="handleChange"></el-cascader>
</div>

<script>
  export default {
    data() {
      return {
        value: [],
        props: { expandTrigger: 'hover' },
        options: [{
```

```
          value: 'zhinan',
          label: '指南',
          children: [{
            value: 'shejiyuanze',
            label: '设计原则',
            children: [{
              value: 'yizhi',
              label: '一致',
            }, {
              value: 'fankui',
              label: '反馈'
            }, {
              value: 'xiaolv',
              label: '效率'
            }, {
              value: 'kekong',
              label: '可控'
            }]
          }, {
            value: 'daohang',
            label: '导航',
            children: [{
              value: 'cexiangdaohang',
              label: '侧向导航'
            }, {
              value: 'dingbudaohang',
              label: '顶部导航'
            }]
          }]
        }, {
          value: 'zujian',
          label: '组件',
          children: [{
            value: 'basic',
            label: 'Basic',
            children: [{
              value: 'layout',
              label: 'Layout 布局'
            }, {
              value: 'color',
              label: 'Color 色彩'
            }, {
```

```
              value: 'typography',
              label: 'Typography 字体'
            }, {
              value: 'icon',
              label: 'Icon 图标'
            }, {
              value: 'button',
              label: 'Button 按钮'
            }]
          }, {
            value: 'form',
            label: 'Form',
            children: [{
              value: 'radio',
              label: 'Radio 单选框'
            }, {
              value: 'checkbox',
              label: 'Checkbox 多选框'
            }, {
              value: 'input',
              label: 'Input 输入框'
            }, {
              value: 'input-number',
              label: 'InputNumber 计数器'
            }, {
              value: 'select',
              label: 'Select 选择器'
            }, {
              value: 'cascader',
              label: 'Cascader 级联选择器'
            }, {
              value: 'switch',
              label: 'Switch 开关'
            }, {
              value: 'slider',
              label: 'Slider 滑块'
            }, {
              value: 'time-picker',
              label: 'TimePicker 时间选择器'
            }, {
              value: 'date-picker',
              label: 'DatePicker 日期选择器'
```

```
            }, {
              value: 'datetime-picker',
              label: 'DateTimePicker 日期时间选择器'
            }, {
              value: 'upload',
              label: 'Upload 上传'
            }, {
              value: 'rate',
              label: 'Rate 评分'
            }, {
              value: 'form',
              label: 'Form 表单'
            }]
          }, {
            value: 'data',
            label: 'Data',
            children: [{
              value: 'table',
              label: 'Table 表格'
            }, {
              value: 'tag',
              label: 'Tag 标签'
            }, {
              value: 'progress',
              label: 'Progress 进度条'
            }, {
              value: 'tree',
              label: 'Tree 树形控件'
            }, {
              value: 'pagination',
              label: 'Pagination 分页'
            }, {
              value: 'badge',
              label: 'Badge 标记'
            }]
          }, {
            value: 'notice',
            label: 'Notice',
            children: [{
              value: 'alert',
              label: 'Alert 警告'
            }, {
```

```
              value: 'loading',
              label: 'Loading 加载'
            }, {
              value: 'message',
              label: 'Message 消息提示'
            }, {
              value: 'message-box',
              label: 'MessageBox 弹框'
            }, {
              value: 'notification',
              label: 'Notification 通知'
            }]
          }, {
            value: 'navigation',
            label: 'Navigation',
            children: [{
              value: 'menu',
              label: 'NavMenu 导航菜单'
            }, {
              value: 'tabs',
              label: 'Tabs 标签页'
            }, {
              value: 'breadcrumb',
              label: 'Breadcrumb 面包屑'
            }, {
              value: 'dropdown',
              label: 'Dropdown 下拉菜单'
            }, {
              value: 'steps',
              label: 'Steps 步骤条'
            }]
          }, {
            value: 'others',
            label: 'Others',
            children: [{
              value: 'dialog',
              label: 'Dialog 对话框'
            }, {
              value: 'tooltip',
              label: 'Tooltip 文字提示'
            }, {
              value: 'popover',
```

```
              label: 'Popover 弹出框'
            }, {
              value: 'card',
              label: 'Card 卡片'
            }, {
              value: 'carousel',
              label: 'Carousel 走马灯'
            }, {
              value: 'collapse',
              label: 'Collapse 折叠面板'
            }]
          }]
        }, {
          value: 'ziyuan',
          label: '资源',
          children: [{
            value: 'axure',
            label: 'Axure Components'
          }, {
            value: 'sketch',
            label: 'Sketch Templates'
          }, {
            value: 'jiaohu',
            label: '组件交互文档'
          }]
        }];
      },
      methods: {
        handleChange(value) {
          console.log(value);
        }
      }
    };
  </script>
```

禁用选项

通过在数据源中设置 `disabled` 字段来声明该选项是禁用的

本例中， `options` 指定的数组中的第一个元素含有 `disabled: true` 键值对，因此是禁用的。在默认情况下，Cascader 会检查数据中每一项的 `disabled` 字段是否为 `true` ，如果你的数据中表示禁用含义的字段名不为 `disabled` ，可以通过 `props.disabled` 属性来指定（详见下方 API 表格）。当然， `value` 、 `label` 和 `children` 这三个字段名也可以通过同样的方式指定。

```
<el-cascader :options="options"></el-cascader>

<script>
  export default {
    data() {
      return {
        options: [{
          value: 'zhinan',
          label: '指南',
          disabled: true,
          children: [{
            value: 'shejiyuanze',
            label: '设计原则',
            children: [{
              value: 'yizhi',
              label: '一致'
            }, {
              value: 'fankui',
              label: '反馈'
            }, {
              value: 'xiaolv',
              label: '效率'
            }, {
              value: 'kekong',
              label: '可控'
            }]
          }, {
            value: 'daohang',
            label: '导航',
            children: [{
              value: 'cexiangdaohang',
              label: '侧向导航'
            }, {
```

```
              value: 'dingbudaohang',
              label: '顶部导航'
            }]
          }]
        }, {
          value: 'zujian',
          label: '组件',
          children: [{
            value: 'basic',
            label: 'Basic',
            children: [{
              value: 'layout',
              label: 'Layout 布局'
            }, {
              value: 'color',
              label: 'Color 色彩'
            }, {
              value: 'typography',
              label: 'Typography 字体'
            }, {
              value: 'icon',
              label: 'Icon 图标'
            }, {
              value: 'button',
              label: 'Button 按钮'
            }]
          }, {
            value: 'form',
            label: 'Form',
            children: [{
              value: 'radio',
              label: 'Radio 单选框'
            }, {
              value: 'checkbox',
              label: 'Checkbox 多选框'
            }, {
              value: 'input',
              label: 'Input 输入框'
            }, {
              value: 'input-number',
              label: 'InputNumber 计数器'
            }, {
```

```
              value: 'select',
              label: 'Select 选择器'
            }, {
              value: 'cascader',
              label: 'Cascader 级联选择器'
            }, {
              value: 'switch',
              label: 'Switch 开关'
            }, {
              value: 'slider',
              label: 'Slider 滑块'
            }, {
              value: 'time-picker',
              label: 'TimePicker 时间选择器'
            }, {
              value: 'date-picker',
              label: 'DatePicker 日期选择器'
            }, {
              value: 'datetime-picker',
              label: 'DateTimePicker 日期时间选择器'
            }, {
              value: 'upload',
              label: 'Upload 上传'
            }, {
              value: 'rate',
              label: 'Rate 评分'
            }, {
              value: 'form',
              label: 'Form 表单'
            }]
          }, {
            value: 'data',
            label: 'Data',
            children: [{
              value: 'table',
              label: 'Table 表格'
            }, {
              value: 'tag',
              label: 'Tag 标签'
            }, {
              value: 'progress',
              label: 'Progress 进度条'
```

```
            }, {
              value: 'tree',
              label: 'Tree 树形控件'
            }, {
              value: 'pagination',
              label: 'Pagination 分页'
            }, {
              value: 'badge',
              label: 'Badge 标记'
            }]
          }, {
            value: 'notice',
            label: 'Notice',
            children: [{
              value: 'alert',
              label: 'Alert 警告'
            }, {
              value: 'loading',
              label: 'Loading 加载'
            }, {
              value: 'message',
              label: 'Message 消息提示'
            }, {
              value: 'message-box',
              label: 'MessageBox 弹框'
            }, {
              value: 'notification',
              label: 'Notification 通知'
            }]
          }, {
            value: 'navigation',
            label: 'Navigation',
            children: [{
              value: 'menu',
              label: 'NavMenu 导航菜单'
            }, {
              value: 'tabs',
              label: 'Tabs 标签页'
            }, {
              value: 'breadcrumb',
              label: 'Breadcrumb 面包屑'
            }, {
```

```
              value: 'dropdown',
              label: 'Dropdown 下拉菜单'
            }, {
              value: 'steps',
              label: 'Steps 步骤条'
            }]
          }, {
            value: 'others',
            label: 'Others',
            children: [{
              value: 'dialog',
              label: 'Dialog 对话框'
            }, {
              value: 'tooltip',
              label: 'Tooltip 文字提示'
            }, {
              value: 'popover',
              label: 'Popover 弹出框'
            }, {
              value: 'card',
              label: 'Card 卡片'
            }, {
              value: 'carousel',
              label: 'Carousel 走马灯'
            }, {
              value: 'collapse',
              label: 'Collapse 折叠面板'
            }]
          }]
        }, {
          value: 'ziyuan',
          label: '资源',
          children: [{
            value: 'axure',
            label: 'Axure Components'
          }, {
            value: 'sketch',
            label: 'Sketch Templates'
          }, {
            value: 'jiaohu',
            label: '组件交互文档'
          }]
```

```
        }]
      };
    }
  };
</script>
```

可清空

通过 `clearable` 设置输入框可清空


```
<el-cascader :options="options" clearable></el-cascader>

<script>
  export default {
    data() {
      return {
        options: [{
          value: 'zhinan',
          label: '指南',
          children: [{
            value: 'shejiyuanze',
            label: '设计原则',
            children: [{
              value: 'yizhi',
              label: '一致'
            }, {
              value: 'fankui',
              label: '反馈'
            }, {
              value: 'xiaolv',
              label: '效率'
            }, {
              value: 'kekong',
              label: '可控'
            }]
          }, {
            value: 'daohang',
            label: '导航',
```

```
            children: [{
              value: 'cexiangdaohang',
              label: '侧向导航'
            }, {
              value: 'dingbudaohang',
              label: '顶部导航'
            }]
          }]
        }, {
          value: 'zujian',
          label: '组件',
          children: [{
            value: 'basic',
            label: 'Basic',
            children: [{
              value: 'layout',
              label: 'Layout 布局'
            }, {
              value: 'color',
              label: 'Color 色彩'
            }, {
              value: 'typography',
              label: 'Typography 字体'
            }, {
              value: 'icon',
              label: 'Icon 图标'
            }, {
              value: 'button',
              label: 'Button 按钮'
            }]
          }, {
            value: 'form',
            label: 'Form',
            children: [{
              value: 'radio',
              label: 'Radio 单选框'
            }, {
              value: 'checkbox',
              label: 'Checkbox 多选框'
            }, {
              value: 'input',
              label: 'Input 输入框'
```

```
            }, {
              value: 'input-number',
              label: 'InputNumber 计数器'
            }, {
              value: 'select',
              label: 'Select 选择器'
            }, {
              value: 'cascader',
              label: 'Cascader 级联选择器'
            }, {
              value: 'switch',
              label: 'Switch 开关'
            }, {
              value: 'slider',
              label: 'Slider 滑块'
            }, {
              value: 'time-picker',
              label: 'TimePicker 时间选择器'
            }, {
              value: 'date-picker',
              label: 'DatePicker 日期选择器'
            }, {
              value: 'datetime-picker',
              label: 'DateTimePicker 日期时间选择器'
            }, {
              value: 'upload',
              label: 'Upload 上传'
            }, {
              value: 'rate',
              label: 'Rate 评分'
            }, {
              value: 'form',
              label: 'Form 表单'
            }]
          }, {
            value: 'data',
            label: 'Data',
            children: [{
              value: 'table',
              label: 'Table 表格'
            }, {
              value: 'tag',
```

```
              label: 'Tag 标签'
            }, {
              value: 'progress',
              label: 'Progress 进度条'
            }, {
              value: 'tree',
              label: 'Tree 树形控件'
            }, {
              value: 'pagination',
              label: 'Pagination 分页'
            }, {
              value: 'badge',
              label: 'Badge 标记'
            }]
          }, {
            value: 'notice',
            label: 'Notice',
            children: [{
              value: 'alert',
              label: 'Alert 警告'
            }, {
              value: 'loading',
              label: 'Loading 加载'
            }, {
              value: 'message',
              label: 'Message 消息提示'
            }, {
              value: 'message-box',
              label: 'MessageBox 弹框'
            }, {
              value: 'notification',
              label: 'Notification 通知'
            }]
          }, {
            value: 'navigation',
            label: 'Navigation',
            children: [{
              value: 'menu',
              label: 'NavMenu 导航菜单'
            }, {
              value: 'tabs',
              label: 'Tabs 标签页'
```

```
155.             }, {
156.               value: 'breadcrumb',
157.               label: 'Breadcrumb 面包屑'
158.             }, {
159.               value: 'dropdown',
160.               label: 'Dropdown 下拉菜单'
161.             }, {
162.               value: 'steps',
163.               label: 'Steps 步骤条'
164.             }]
165.           }, {
166.             value: 'others',
167.             label: 'Others',
168.             children: [{
169.               value: 'dialog',
170.               label: 'Dialog 对话框'
171.             }, {
172.               value: 'tooltip',
173.               label: 'Tooltip 文字提示'
174.             }, {
175.               value: 'popover',
176.               label: 'Popover 弹出框'
177.             }, {
178.               value: 'card',
179.               label: 'Card 卡片'
180.             }, {
181.               value: 'carousel',
182.               label: 'Carousel 走马灯'
183.             }, {
184.               value: 'collapse',
185.               label: 'Collapse 折叠面板'
186.             }]
187.           }]
188.         }, {
189.           value: 'ziyuan',
190.           label: '资源',
191.           children: [{
192.             value: 'axure',
193.             label: 'Axure Components'
194.           }, {
195.             value: 'sketch',
196.             label: 'Sketch Templates'
```

```
          }, {
            value: 'jiaohu',
            label: '组件交互文档'
          }]
        }]
      }
    }
  }
</script>
```

仅显示最后一级

可以仅在输入框中显示选中项最后一级的标签，而不是选中项所在的完整路径。

属性 `show-all-levels` 定义了是否显示完整的路径，将其赋值为 `false` 则仅显示最后一级

```
<el-cascader :options="options" :show-all-levels="false"></el-cascader>

<script>
  export default {
    data() {
      return {
        options: [{
          value: 'zhinan',
          label: '指南',
          children: [{
            value: 'shejiyuanze',
            label: '设计原则',
            children: [{
              value: 'yizhi',
              label: '一致'
            }, {
              value: 'fankui',
              label: '反馈'
            }, {
              value: 'xiaolv',
              label: '效率'
            }, {
```

```
            value: 'kekong',
            label: '可控'
          }]
        }, {
          value: 'daohang',
          label: '导航',
          children: [{
            value: 'cexiangdaohang',
            label: '侧向导航'
          }, {
            value: 'dingbudaohang',
            label: '顶部导航'
          }]
        }]
      }, {
        value: 'zujian',
        label: '组件',
        children: [{
          value: 'basic',
          label: 'Basic',
          children: [{
            value: 'layout',
            label: 'Layout 布局'
          }, {
            value: 'color',
            label: 'Color 色彩'
          }, {
            value: 'typography',
            label: 'Typography 字体'
          }, {
            value: 'icon',
            label: 'Icon 图标'
          }, {
            value: 'button',
            label: 'Button 按钮'
          }]
        }, {
          value: 'form',
          label: 'Form',
          children: [{
            value: 'radio',
            label: 'Radio 单选框'
```

```
            }, {
              value: 'checkbox',
              label: 'Checkbox 多选框'
            }, {
              value: 'input',
              label: 'Input 输入框'
            }, {
              value: 'input-number',
              label: 'InputNumber 计数器'
            }, {
              value: 'select',
              label: 'Select 选择器'
            }, {
              value: 'cascader',
              label: 'Cascader 级联选择器'
            }, {
              value: 'switch',
              label: 'Switch 开关'
            }, {
              value: 'slider',
              label: 'Slider 滑块'
            }, {
              value: 'time-picker',
              label: 'TimePicker 时间选择器'
            }, {
              value: 'date-picker',
              label: 'DatePicker 日期选择器'
            }, {
              value: 'datetime-picker',
              label: 'DateTimePicker 日期时间选择器'
            }, {
              value: 'upload',
              label: 'Upload 上传'
            }, {
              value: 'rate',
              label: 'Rate 评分'
            }, {
              value: 'form',
              label: 'Form 表单'
            }]
          }, {
            value: 'data',
```

```
            label: 'Data',
            children: [{
              value: 'table',
              label: 'Table 表格'
            }, {
              value: 'tag',
              label: 'Tag 标签'
            }, {
              value: 'progress',
              label: 'Progress 进度条'
            }, {
              value: 'tree',
              label: 'Tree 树形控件'
            }, {
              value: 'pagination',
              label: 'Pagination 分页'
            }, {
              value: 'badge',
              label: 'Badge 标记'
            }]
          }, {
            value: 'notice',
            label: 'Notice',
            children: [{
              value: 'alert',
              label: 'Alert 警告'
            }, {
              value: 'loading',
              label: 'Loading 加载'
            }, {
              value: 'message',
              label: 'Message 消息提示'
            }, {
              value: 'message-box',
              label: 'MessageBox 弹框'
            }, {
              value: 'notification',
              label: 'Notification 通知'
            }]
          }, {
            value: 'navigation',
            label: 'Navigation',
```

```
            children: [{
              value: 'menu',
              label: 'NavMenu 导航菜单'
            }, {
              value: 'tabs',
              label: 'Tabs 标签页'
            }, {
              value: 'breadcrumb',
              label: 'Breadcrumb 面包屑'
            }, {
              value: 'dropdown',
              label: 'Dropdown 下拉菜单'
            }, {
              value: 'steps',
              label: 'Steps 步骤条'
            }]
          }, {
            value: 'others',
            label: 'Others',
            children: [{
              value: 'dialog',
              label: 'Dialog 对话框'
            }, {
              value: 'tooltip',
              label: 'Tooltip 文字提示'
            }, {
              value: 'popover',
              label: 'Popover 弹出框'
            }, {
              value: 'card',
              label: 'Card 卡片'
            }, {
              value: 'carousel',
              label: 'Carousel 走马灯'
            }, {
              value: 'collapse',
              label: 'Collapse 折叠面板'
            }]
          }]
        }, {
          value: 'ziyuan',
          label: '资源',
```

```
          children: [{
            value: 'axure',
            label: 'Axure Components'
          }, {
            value: 'sketch',
            label: 'Sketch Templates'
          }, {
            value: 'jiaohu',
            label: '组件交互文档'
          }]
        }]
      };
    }
  };
</script>
```

多选

可通过 `props.multiple = true` 来开启多选模式

在开启多选模式后，默认情况下会展示所有已选中的选项的Tag，你可以使用 `collapse-tags` 来折叠Tag

```
<div class="block">
  <span class="demonstration">默认显示所有Tag</span>
  <el-cascader
    :options="options"
    :props="props"
    clearable></el-cascader>
</div>
<div class="block">
  <span class="demonstration">折叠展示Tag</span>
  <el-cascader
    :options="options"
    :props="props"
    collapse-tags
    clearable></el-cascader>
</div>

<script>
  export default {
    data() {
      return {
        props: { multiple: true },
        options: [{
          value: 1,
          label: '东南',
          children: [{
            value: 2,
            label: '上海',
```

```
            children: [
              { value: 3, label: '普陀' },
              { value: 4, label: '黄埔' },
              { value: 5, label: '徐汇' }
            ]
          }, {
            value: 7,
            label: '江苏',
            children: [
              { value: 8, label: '南京' },
              { value: 9, label: '苏州' },
              { value: 10, label: '无锡' }
            ]
          }, {
            value: 12,
            label: '浙江',
            children: [
              { value: 13, label: '杭州' },
              { value: 14, label: '宁波' },
              { value: 15, label: '嘉兴' }
            ]
          }]
        }, {
          value: 17,
          label: '西北',
          children: [{
            value: 18,
            label: '陕西',
            children: [
              { value: 19, label: '西安' },
              { value: 20, label: '延安' }
            ]
          }, {
            value: 21,
            label: '新疆维吾尔族自治区',
            children: [
              { value: 22, label: '乌鲁木齐' },
              { value: 23, label: '克拉玛依' }
            ]
          }]
        }];
```

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

选择任意一级选项

在单选模式下，你只能选择叶子节点；而在多选模式下，勾选父节点真正选中的都是叶子节点。启用该功能后，可让父子节点取消关联，选择任意一级选项。

可通过 `props.checkStrictly = true` 来设置父子节点取消选中关联，从而达到选择任意一级选项的目的。

```
<div class="block">
  <span class="demonstration">单选选择任意一级选项</span>
  <el-cascader
    :options="options"
    :props="{ checkStrictly: true }"
    clearable></el-cascader>
</div>
<div class="block">
  <span class="demonstration">多选选择任意一级选项</span>
  <el-cascader
    :options="options"
    :props="{ multiple: true, checkStrictly: true }"
    clearable></el-cascader>
</div>

<script>
  export default {
    data() {
      return {
        options: [{
          value: 'zhinan',
          label: '指南',
          children: [{
```

```
            value: 'shejiyuanze',
            label: '设计原则',
            children: [{
              value: 'yizhi',
              label: '一致'
            }, {
              value: 'fankui',
              label: '反馈'
            }, {
              value: 'xiaolv',
              label: '效率'
            }, {
              value: 'kekong',
              label: '可控'
            }]
          }, {
            value: 'daohang',
            label: '导航',
            children: [{
              value: 'cexiangdaohang',
              label: '侧向导航'
            }, {
              value: 'dingbudaohang',
              label: '顶部导航'
            }]
          }]
        }, {
          value: 'zujian',
          label: '组件',
          children: [{
            value: 'basic',
            label: 'Basic',
            children: [{
              value: 'layout',
              label: 'Layout 布局'
            }, {
              value: 'color',
              label: 'Color 色彩'
            }, {
              value: 'typography',
              label: 'Typography 字体'
            }, {
```

```
              value: 'icon',
              label: 'Icon 图标'
            }, {
              value: 'button',
              label: 'Button 按钮'
            }]
          }, {
            value: 'form',
            label: 'Form',
            children: [{
              value: 'radio',
              label: 'Radio 单选框'
            }, {
              value: 'checkbox',
              label: 'Checkbox 多选框'
            }, {
              value: 'input',
              label: 'Input 输入框'
            }, {
              value: 'input-number',
              label: 'InputNumber 计数器'
            }, {
              value: 'select',
              label: 'Select 选择器'
            }, {
              value: 'cascader',
              label: 'Cascader 级联选择器'
            }, {
              value: 'switch',
              label: 'Switch 开关'
            }, {
              value: 'slider',
              label: 'Slider 滑块'
            }, {
              value: 'time-picker',
              label: 'TimePicker 时间选择器'
            }, {
              value: 'date-picker',
              label: 'DatePicker 日期选择器'
            }, {
              value: 'datetime-picker',
              label: 'DateTimePicker 日期时间选择器'
```

```
108.             }, {
109.               value: 'upload',
110.               label: 'Upload 上传'
111.             }, {
112.               value: 'rate',
113.               label: 'Rate 评分'
114.             }, {
115.               value: 'form',
116.               label: 'Form 表单'
117.             }]
118.           }, {
119.             value: 'data',
120.             label: 'Data',
121.             children: [{
122.               value: 'table',
123.               label: 'Table 表格'
124.             }, {
125.               value: 'tag',
126.               label: 'Tag 标签'
127.             }, {
128.               value: 'progress',
129.               label: 'Progress 进度条'
130.             }, {
131.               value: 'tree',
132.               label: 'Tree 树形控件'
133.             }, {
134.               value: 'pagination',
135.               label: 'Pagination 分页'
136.             }, {
137.               value: 'badge',
138.               label: 'Badge 标记'
139.             }]
140.           }, {
141.             value: 'notice',
142.             label: 'Notice',
143.             children: [{
144.               value: 'alert',
145.               label: 'Alert 警告'
146.             }, {
147.               value: 'loading',
148.               label: 'Loading 加载'
149.             }, {
```

```
              value: 'message',
              label: 'Message 消息提示'
            }, {
              value: 'message-box',
              label: 'MessageBox 弹框'
            }, {
              value: 'notification',
              label: 'Notification 通知'
            }]
          }, {
            value: 'navigation',
            label: 'Navigation',
            children: [{
              value: 'menu',
              label: 'NavMenu 导航菜单'
            }, {
              value: 'tabs',
              label: 'Tabs 标签页'
            }, {
              value: 'breadcrumb',
              label: 'Breadcrumb 面包屑'
            }, {
              value: 'dropdown',
              label: 'Dropdown 下拉菜单'
            }, {
              value: 'steps',
              label: 'Steps 步骤条'
            }]
          }, {
            value: 'others',
            label: 'Others',
            children: [{
              value: 'dialog',
              label: 'Dialog 对话框'
            }, {
              value: 'tooltip',
              label: 'Tooltip 文字提示'
            }, {
              value: 'popover',
              label: 'Popover 弹出框'
            }, {
              value: 'card',
```

```
              label: 'Card 卡片'
            }, {
              value: 'carousel',
              label: 'Carousel 走马灯'
            }, {
              value: 'collapse',
              label: 'Collapse 折叠面板'
            }]
          }]
        }, {
          value: 'ziyuan',
          label: '资源',
          children: [{
            value: 'axure',
            label: 'Axure Components'
          }, {
            value: 'sketch',
            label: 'Sketch Templates'
          }, {
            value: 'jiaohu',
            label: '组件交互文档'
          }]
        }];
      }
    };
  </script>
```

动态加载

当选中某一级时，动态加载该级下的选项。

通过 `lazy` 开启动态加载，并通过 `lazyload` 来设置加载数据源的方法。 `lazyload` 方法有两个参数，第一个参数 `node` 为当前点击的节点，第二个 `resolve` 为数据加载完成的回调(必须调用)。为了更准确的显示节点的状态，还可以对节点数据添加是否为叶子节点的标志位 (默认字段为 `leaf` ，可通过 `props.leaf` 修改)，否则会简单的以有无子节点来判断是否为叶子节点。

```
<el-cascader :props="props"></el-cascader>
```

```
2.
3.  <script>
4.    let id = 0;
5.
6.    export default {
7.      data() {
8.        return {
9.          props: {
10.           lazy: true,
11.           lazyLoad (node, resolve) {
12.             const { level } = node;
13.             setTimeout(() => {
14.               const nodes = Array.from({ length: level + 1 })
15.                 .map(item => ({
16.                   value: ++id,
17.                   label: `选项${id}`,
18.                   leaf: level >= 2
19.                 }));
20.               // 通过调用resolve将子节点数据返回，通知组件数据加载完成
21.               resolve(nodes);
22.             }, 1000);
23.           }
24.         }
25.       };
26.     }
27.   };
28. </script>
```

可搜索

可以快捷地搜索选项并选择。

将 `filterable` 赋值为 `true` 即可打开搜索功能，默认会匹配节点的 `label` 或所有父节点的 `label` (由 `show-all-levels` 决定)中包含输入值的选项。你也可以用 `filter-method` 自定义搜索逻辑，接受一个函数，第一个参数是节点 `node` ，第二个参数是搜索关键词 `keyword` ，通过

返回布尔值表示是否命中。

```
<div class="block">
  <span class="demonstration">单选可搜索</span>
  <el-cascader
    placeholder="试试搜索：指南"
    :options="options"
    filterable></el-cascader>
</div>
<div class="block">
  <span class="demonstration">多选可搜索</span>
  <el-cascader
    placeholder="试试搜索：指南"
    :options="options"
    :props="{ multiple: true }"
    filterable></el-cascader>
</div>

<script>
  export default {
    data() {
      return {
        options: [{
          value: 'zhinan',
          label: '指南',
          children: [{
            value: 'shejiyuanze',
            label: '设计原则',
            children: [{
              value: 'yizhi',
              label: '一致'
            }, {
              value: 'fankui',
              label: '反馈'
            }, {
              value: 'xiaolv',
              label: '效率'
            }, {
              value: 'kekong',
              label: '可控'
            }]
          }, {
```

```
            value: 'daohang',
            label: '导航',
            children: [{
              value: 'cexiangdaohang',
              label: '侧向导航'
            }, {
              value: 'dingbudaohang',
              label: '顶部导航'
            }]
          }]
        }, {
          value: 'zujian',
          label: '组件',
          children: [{
            value: 'basic',
            label: 'Basic',
            children: [{
              value: 'layout',
              label: 'Layout 布局'
            }, {
              value: 'color',
              label: 'Color 色彩'
            }, {
              value: 'typography',
              label: 'Typography 字体'
            }, {
              value: 'icon',
              label: 'Icon 图标'
            }, {
              value: 'button',
              label: 'Button 按钮'
            }]
          }, {
            value: 'form',
            label: 'Form',
            children: [{
              value: 'radio',
              label: 'Radio 单选框'
            }, {
              value: 'checkbox',
              label: 'Checkbox 多选框'
            }, {
```

```
              value: 'input',
              label: 'Input 输入框'
            }, {
              value: 'input-number',
              label: 'InputNumber 计数器'
            }, {
              value: 'select',
              label: 'Select 选择器'
            }, {
              value: 'cascader',
              label: 'Cascader 级联选择器'
            }, {
              value: 'switch',
              label: 'Switch 开关'
            }, {
              value: 'slider',
              label: 'Slider 滑块'
            }, {
              value: 'time-picker',
              label: 'TimePicker 时间选择器'
            }, {
              value: 'date-picker',
              label: 'DatePicker 日期选择器'
            }, {
              value: 'datetime-picker',
              label: 'DateTimePicker 日期时间选择器'
            }, {
              value: 'upload',
              label: 'Upload 上传'
            }, {
              value: 'rate',
              label: 'Rate 评分'
            }, {
              value: 'form',
              label: 'Form 表单'
            }]
          }, {
            value: 'data',
            label: 'Data',
            children: [{
              value: 'table',
              label: 'Table 表格'
```

```
125.             }, {
126.               value: 'tag',
127.               label: 'Tag 标签'
128.             }, {
129.               value: 'progress',
130.               label: 'Progress 进度条'
131.             }, {
132.               value: 'tree',
133.               label: 'Tree 树形控件'
134.             }, {
135.               value: 'pagination',
136.               label: 'Pagination 分页'
137.             }, {
138.               value: 'badge',
139.               label: 'Badge 标记'
140.             }]
141.           }, {
142.             value: 'notice',
143.             label: 'Notice',
144.             children: [{
145.               value: 'alert',
146.               label: 'Alert 警告'
147.             }, {
148.               value: 'loading',
149.               label: 'Loading 加载'
150.             }, {
151.               value: 'message',
152.               label: 'Message 消息提示'
153.             }, {
154.               value: 'message-box',
155.               label: 'MessageBox 弹框'
156.             }, {
157.               value: 'notification',
158.               label: 'Notification 通知'
159.             }]
160.           }, {
161.             value: 'navigation',
162.             label: 'Navigation',
163.             children: [{
164.               value: 'menu',
165.               label: 'NavMenu 导航菜单'
166.             }, {
```

```
              value: 'tabs',
              label: 'Tabs 标签页'
            }, {
              value: 'breadcrumb',
              label: 'Breadcrumb 面包屑'
            }, {
              value: 'dropdown',
              label: 'Dropdown 下拉菜单'
            }, {
              value: 'steps',
              label: 'Steps 步骤条'
            }]
          }, {
            value: 'others',
            label: 'Others',
            children: [{
              value: 'dialog',
              label: 'Dialog 对话框'
            }, {
              value: 'tooltip',
              label: 'Tooltip 文字提示'
            }, {
              value: 'popover',
              label: 'Popover 弹出框'
            }, {
              value: 'card',
              label: 'Card 卡片'
            }, {
              value: 'carousel',
              label: 'Carousel 走马灯'
            }, {
              value: 'collapse',
              label: 'Collapse 折叠面板'
            }]
          }]
        }, {
          value: 'ziyuan',
          label: '资源',
          children: [{
            value: 'axure',
            label: 'Axure Components'
          }, {
```

```
            value: 'sketch',
            label: 'Sketch Templates'
          }, {
            value: 'jiaohu',
            label: '组件交互文档'
          }]
        }]
      };
    }
  };
</script>
```

自定义节点内容

可以自定义备选项的节点内容

可以通过 `scoped slot` 对级联选择器的备选项的节点内容进行自定义，scoped slot会传入两个字段 `node` 和 `data`，分别表示当前节点的 Node 对象和数据。

```
<el-cascader :options="options">
  <template #default="{ node, data }">
    <span>{{ data.label }}</span>
    <span v-if="!node.isLeaf"> ({{ data.children.length }}) </span>
  </template>
</el-cascader>

<script>
  export default {
    data() {
      return {
        options: [{
          value: 'zhinan',
          label: '指南',
          children: [{
            value: 'shejiyuanze',
            label: '设计原则',
            children: [{
              value: 'yizhi',
```

```
              label: '一致'
            }, {
              value: 'fankui',
              label: '反馈'
            }, {
              value: 'xiaolv',
              label: '效率'
            }, {
              value: 'kekong',
              label: '可控'
            }]
          }, {
            value: 'daohang',
            label: '导航',
            children: [{
              value: 'cexiangdaohang',
              label: '侧向导航'
            }, {
              value: 'dingbudaohang',
              label: '顶部导航'
            }]
          }]
        }, {
          value: 'zujian',
          label: '组件',
          children: [{
            value: 'basic',
            label: 'Basic',
            children: [{
              value: 'layout',
              label: 'Layout 布局'
            }, {
              value: 'color',
              label: 'Color 色彩'
            }, {
              value: 'typography',
              label: 'Typography 字体'
            }, {
              value: 'icon',
              label: 'Icon 图标'
            }, {
              value: 'button',
```

```
              label: 'Button 按钮'
            }]
          }, {
            value: 'form',
            label: 'Form',
            children: [{
              value: 'radio',
              label: 'Radio 单选框'
            }, {
              value: 'checkbox',
              label: 'Checkbox 多选框'
            }, {
              value: 'input',
              label: 'Input 输入框'
            }, {
              value: 'input-number',
              label: 'InputNumber 计数器'
            }, {
              value: 'select',
              label: 'Select 选择器'
            }, {
              value: 'cascader',
              label: 'Cascader 级联选择器'
            }, {
              value: 'switch',
              label: 'Switch 开关'
            }, {
              value: 'slider',
              label: 'Slider 滑块'
            }, {
              value: 'time-picker',
              label: 'TimePicker 时间选择器'
            }, {
              value: 'date-picker',
              label: 'DatePicker 日期选择器'
            }, {
              value: 'datetime-picker',
              label: 'DateTimePicker 日期时间选择器'
            }, {
              value: 'upload',
              label: 'Upload 上传'
            }, {
```

```
              value: 'rate',
              label: 'Rate 评分'
            }, {
              value: 'form',
              label: 'Form 表单'
            }]
          }, {
            value: 'data',
            label: 'Data',
            children: [{
              value: 'table',
              label: 'Table 表格'
            }, {
              value: 'tag',
              label: 'Tag 标签'
            }, {
              value: 'progress',
              label: 'Progress 进度条'
            }, {
              value: 'tree',
              label: 'Tree 树形控件'
            }, {
              value: 'pagination',
              label: 'Pagination 分页'
            }, {
              value: 'badge',
              label: 'Badge 标记'
            }]
          }, {
            value: 'notice',
            label: 'Notice',
            children: [{
              value: 'alert',
              label: 'Alert 警告'
            }, {
              value: 'loading',
              label: 'Loading 加载'
            }, {
              value: 'message',
              label: 'Message 消息提示'
            }, {
              value: 'message-box',
```

```
              label: 'MessageBox 弹框'
            }, {
              value: 'notification',
              label: 'Notification 通知'
            }]
          }, {
            value: 'navigation',
            label: 'Navigation',
            children: [{
              value: 'menu',
              label: 'NavMenu 导航菜单'
            }, {
              value: 'tabs',
              label: 'Tabs 标签页'
            }, {
              value: 'breadcrumb',
              label: 'Breadcrumb 面包屑'
            }, {
              value: 'dropdown',
              label: 'Dropdown 下拉菜单'
            }, {
              value: 'steps',
              label: 'Steps 步骤条'
            }]
          }, {
            value: 'others',
            label: 'Others',
            children: [{
              value: 'dialog',
              label: 'Dialog 对话框'
            }, {
              value: 'tooltip',
              label: 'Tooltip 文字提示'
            }, {
              value: 'popover',
              label: 'Popover 弹出框'
            }, {
              value: 'card',
              label: 'Card 卡片'
            }, {
              value: 'carousel',
              label: 'Carousel 走马灯'
```

```
            }, {
              value: 'collapse',
              label: 'Collapse 折叠面板'
            }]
          }]
        }, {
          value: 'ziyuan',
          label: '资源',
          children: [{
            value: 'axure',
            label: 'Axure Components'
          }, {
            value: 'sketch',
            label: 'Sketch Templates'
          }, {
            value: 'jiaohu',
            label: '组件交互文档'
          }]
        }]
      }
    }
  }
</script>
```

级联面板

级联面板是级联选择器的核心组件，与级联选择器一样，有单选、多选、动态加载等多种功能。

和级联选择器一样，通过 `options` 来指定选项，也可通过 `props` 来设置多选、动态加载等功能，具体详情见下方API表格。

```
<el-cascader-panel :options="options"></el-cascader-panel>

<script>
  export default {
    data() {
      return {
        options: [{
          value: 'zhinan',
          label: '指南',
          children: [{
            value: 'shejiyuanze',
            label: '设计原则',
            children: [{
              value: 'yizhi',
              label: '一致'
            }, {
              value: 'fankui',
              label: '反馈'
            }, {
              value: 'xiaolv',
              label: '效率'
            }, {
              value: 'kekong',
              label: '可控'
            }]
```

```
        }, {
          value: 'daohang',
          label: '导航',
          children: [{
            value: 'cexiangdaohang',
            label: '侧向导航'
          }, {
            value: 'dingbudaohang',
            label: '顶部导航'
          }]
        }]
      }, {
        value: 'zujian',
        label: '组件',
        children: [{
          value: 'basic',
          label: 'Basic',
          children: [{
            value: 'layout',
            label: 'Layout 布局'
          }, {
            value: 'color',
            label: 'Color 色彩'
          }, {
            value: 'typography',
            label: 'Typography 字体'
          }, {
            value: 'icon',
            label: 'Icon 图标'
          }, {
            value: 'button',
            label: 'Button 按钮'
          }]
        }, {
          value: 'form',
          label: 'Form',
          children: [{
            value: 'radio',
            label: 'Radio 单选框'
          }, {
            value: 'checkbox',
            label: 'Checkbox 多选框'
```

```
68.             }, {
69.               value: 'input',
70.               label: 'Input 输入框'
71.             }, {
72.               value: 'input-number',
73.               label: 'InputNumber 计数器'
74.             }, {
75.               value: 'select',
76.               label: 'Select 选择器'
77.             }, {
78.               value: 'cascader',
79.               label: 'Cascader 级联选择器'
80.             }, {
81.               value: 'switch',
82.               label: 'Switch 开关'
83.             }, {
84.               value: 'slider',
85.               label: 'Slider 滑块'
86.             }, {
87.               value: 'time-picker',
88.               label: 'TimePicker 时间选择器'
89.             }, {
90.               value: 'date-picker',
91.               label: 'DatePicker 日期选择器'
92.             }, {
93.               value: 'datetime-picker',
94.               label: 'DateTimePicker 日期时间选择器'
95.             }, {
96.               value: 'upload',
97.               label: 'Upload 上传'
98.             }, {
99.               value: 'rate',
100.              label: 'Rate 评分'
101.            }, {
102.              value: 'form',
103.              label: 'Form 表单'
104.            }]
105.          }, {
106.            value: 'data',
107.            label: 'Data',
108.            children: [{
109.              value: 'table',
```

```
              label: 'Table 表格'
            }, {
              value: 'tag',
              label: 'Tag 标签'
            }, {
              value: 'progress',
              label: 'Progress 进度条'
            }, {
              value: 'tree',
              label: 'Tree 树形控件'
            }, {
              value: 'pagination',
              label: 'Pagination 分页'
            }, {
              value: 'badge',
              label: 'Badge 标记'
            }]
          }, {
            value: 'notice',
            label: 'Notice',
            children: [{
              value: 'alert',
              label: 'Alert 警告'
            }, {
              value: 'loading',
              label: 'Loading 加载'
            }, {
              value: 'message',
              label: 'Message 消息提示'
            }, {
              value: 'message-box',
              label: 'MessageBox 弹框'
            }, {
              value: 'notification',
              label: 'Notification 通知'
            }]
          }, {
            value: 'navigation',
            label: 'Navigation',
            children: [{
              value: 'menu',
              label: 'NavMenu 导航菜单'
```

```
            }, {
              value: 'tabs',
              label: 'Tabs 标签页'
            }, {
              value: 'breadcrumb',
              label: 'Breadcrumb 面包屑'
            }, {
              value: 'dropdown',
              label: 'Dropdown 下拉菜单'
            }, {
              value: 'steps',
              label: 'Steps 步骤条'
            }]
          }, {
            value: 'others',
            label: 'Others',
            children: [{
              value: 'dialog',
              label: 'Dialog 对话框'
            }, {
              value: 'tooltip',
              label: 'Tooltip 文字提示'
            }, {
              value: 'popover',
              label: 'Popover 弹出框'
            }, {
              value: 'card',
              label: 'Card 卡片'
            }, {
              value: 'carousel',
              label: 'Carousel 走马灯'
            }, {
              value: 'collapse',
              label: 'Collapse 折叠面板'
            }]
          }]
        }, {
          value: 'ziyuan',
          label: '资源',
          children: [{
            value: 'axure',
            label: 'Axure Components'
```

```
          }, {
            value: 'sketch',
            label: 'Sketch Templates'
          }, {
            value: 'jiaohu',
            label: '组件交互文档'
          }]
        }]
      };
    }
  };
</script>
```

Cascader Attributes

参数	说明	类型	可选值	默认值
value / v-model	选中项绑定值	-	—	—
options	可选项数据源，键名可通过 `Props` 属性配置	array	—	—
props	配置选项，具体见下表	object	—	—
size	尺寸	string	medium / small / mini	—
placeholder	输入框占位文本	string	—	请选择
disabled	是否禁用	boolean	—	false
clearable	是否支持清空选项	boolean	—	false
show-all-levels	输入框中是否显示选中值的完整路径	boolean	—	true
collapse-tags	多选模式下是否折叠Tag	boolean	-	false
separator	选项分隔符	string	—	斜杠'/'
filterable	是否可搜索选项	boolean	—	—
filter-method	自定义搜索逻辑，第一个参数是节点 `node` ，第二个参数是搜索关键词 `keyword` ，通过返回布尔值表示是否命中	function(node, keyword)	-	-
debounce	搜索关键词输入的去抖延迟，毫秒	number	—	300
before-filter	筛选之前的钩子，参数为输入的值，若返回 false 或者返回 Promise 且被 reject，则停止筛选	function(value)	—	—
popper-class	自定义浮层类名	string	—	—

Cascader Events

事件名称	说明	回调参数
change	当选中节点变化时触发	选中节点的值
expand-change	当展开节点发生变化时触发	各父级选项值组成的数组
blur	当失去焦点时触发	(event: Event)
focus	当获得焦点时触发	(event: Event)
visible-change	下拉框出现/隐藏时触发	出现则为 true，隐藏则为 false
remove-tag	在多选模式下，移除Tag时触发	移除的Tag对应的节点的值

Cascader Methods

方法名	说明	参数
getCheckedNodes	获取选中的节点	(leafOnly) 是否只是叶子节点，默认值为 `false`

Cascader Slots

名称	说明
-	自定义备选项的节点内容，参数为 { node, data }，分别为当前节点的 Node 对象和数据
empty	无匹配选项时的内容

CascaderPanel Attributes

参数	说明	类型	可选值	默认值
value / v-model	选中项绑定值	-	—	—
options	可选项数据源，键名可通过 `Props` 属性配置	array	—	—
props	配置选项，具体见下表	object	—	—

CascaderPanel Events

事件名称	说明	回调参数
change	当选中节点变化时触发	选中节点的值
expand-change	当展开节点发生变化时触发	各父级选项值组成的数组

CascaderPanel Methods

方法名	说明	参数
getCheckedNodes	获取选中的节点数组	(leafOnly) 是否只是叶子节点，默认值为 `false`
clearCheckedNodes	清空选中的节点	-

CascaderPanel Slots

名称	说明
-	自定义备选项的节点内容，参数为 { node, data }，分别为当前节点的 Node 对象和数据

Props

参数	说明	类型	可选值	默认值
expandTrigger	次级菜单的展开方式	string	click / hover	‘click’
multiple	是否多选	boolean	-	false
checkStrictly	是否严格的遵守父子节点不互相关联	boolean	-	false
emitPath	在选中节点改变时，是否返回由该节点所在的各级菜单的值所组成的数组，若设置 false，则只返回该节点的值	boolean	-	true
lazy	是否动态加载子节点，需与 lazyLoad 方法结合使用	boolean	-	false
lazyLoad	加载动态数据的方法，仅在 lazy 为 true 时有效	function(node, resolve)， node 为当前点击的节点， resolve 为数据加载完成的回调(必须调用)	-	-
value	指定选项的值为选项对象的某个属性值	string	—	‘value’
label	指定选项标签为选项对象的某个属性值	string	—	‘label’
children	指定选项的子选项为选项对象的某个属性值	string	—	‘children’
disabled	指定选项的禁用为选项对象的某个属性值	string	—	‘disabled’
leaf	指定选项的叶子节点的标志位为选项对象的某个属性值	string	—	‘leaf’

Switch 开关

表示两种相互对立的状态间的切换，多用于触发「开/关」。

基本用法

绑定 `v-model` 到一个 `Boolean` 类型的变量。可以使用 `active-color` 属性与 `inactive-color` 属性来设置开关的背景色。

```
<el-switch
  v-model="value"
  active-color="#13ce66"
  inactive-color="#ff4949">
</el-switch>

<script>
  export default {
    data() {
      return {
        value: true
      }
    }
  };
</script>
```

文字描述

使用 `active-text` 属性与 `inactive-text` 属性来设置开关的文字描述。

```
<el-switch
  v-model="value1"
  active-text="按月付费"
  inactive-text="按年付费">
</el-switch>
<el-switch
  style="display: block"
  v-model="value2"
  active-color="#13ce66"
  inactive-color="#ff4949"
  active-text="按月付费"
  inactive-text="按年付费">
</el-switch>

<script>
  export default {
    data() {
      return {
        value1: true,
        value2: true
      }
    }
  };
</script>
```

扩展的 value 类型

设置 `active-value` 和 `inactive-value` 属性，接受 `Boolean` ， `String` 或 `Number` 类型的值。

```
<el-tooltip :content="'Switch value: ' + value" placement="top">
  <el-switch
    v-model="value"
    active-color="#13ce66"
    inactive-color="#ff4949"
    active-value="100"
    inactive-value="0">
```

```
  </el-switch>
</el-tooltip>

<script>
  export default {
    data() {
      return {
        value: '100'
      }
    }
  };
</script>
```

禁用状态

设置 `disabled` 属性，接受一个 `Boolean` ，设置 `true` 即可禁用。

```
<el-switch
  v-model="value1"
  disabled>
</el-switch>
<el-switch
  v-model="value2"
  disabled>
</el-switch>
<script>
  export default {
    data() {
      return {
        value1: true,
        value2: false
      }
    }
  };
</script>
```

加载中

设置 `loading` 属性，接受一个 `Boolean` ，设置 `true` 即加载中状态。

```
<el-switch
  v-model="value1"
  loading>
</el-switch>
<el-switch
  v-model="value2"
  loading>
</el-switch>
<script>
  export default {
    data() {
      return {
        value1: true,
        value2: false
      }
    }
  };
</script>
```

Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	boolean / string / number	—	—
disabled	是否禁用	boolean	—	false
loading	是否显示加载中	boolean	—	false
width	switch 的宽度（像素）	number	—	40
active-icon-class	switch 打开时所显示图标的类名，设置此项会忽略 `active-text`	string	—	—
inactive-icon-class	switch 关闭时所显示图标的类名，设置此项会忽略 `inactive-text`	string	—	—
active-text	switch 打开时的文字描述	string	—	—

inactive-text	switch 关闭时的文字描述	string	—	—
active-value	switch 打开时的值	boolean / string / number	—	true
inactive-value	switch 关闭时的值	boolean / string / number	—	false
active-color	switch 打开时的背景色	string	—	#409EFF
inactive-color	switch 关闭时的背景色	string	—	#C0CCDA
name	switch 对应的 name 属性	string	—	—
validate-event	改变 switch 状态时是否触发表单的校验	boolean	-	true

Events

事件名称	说明	回调参数
change	switch 状态发生变化时的回调函数	新状态的值

Methods

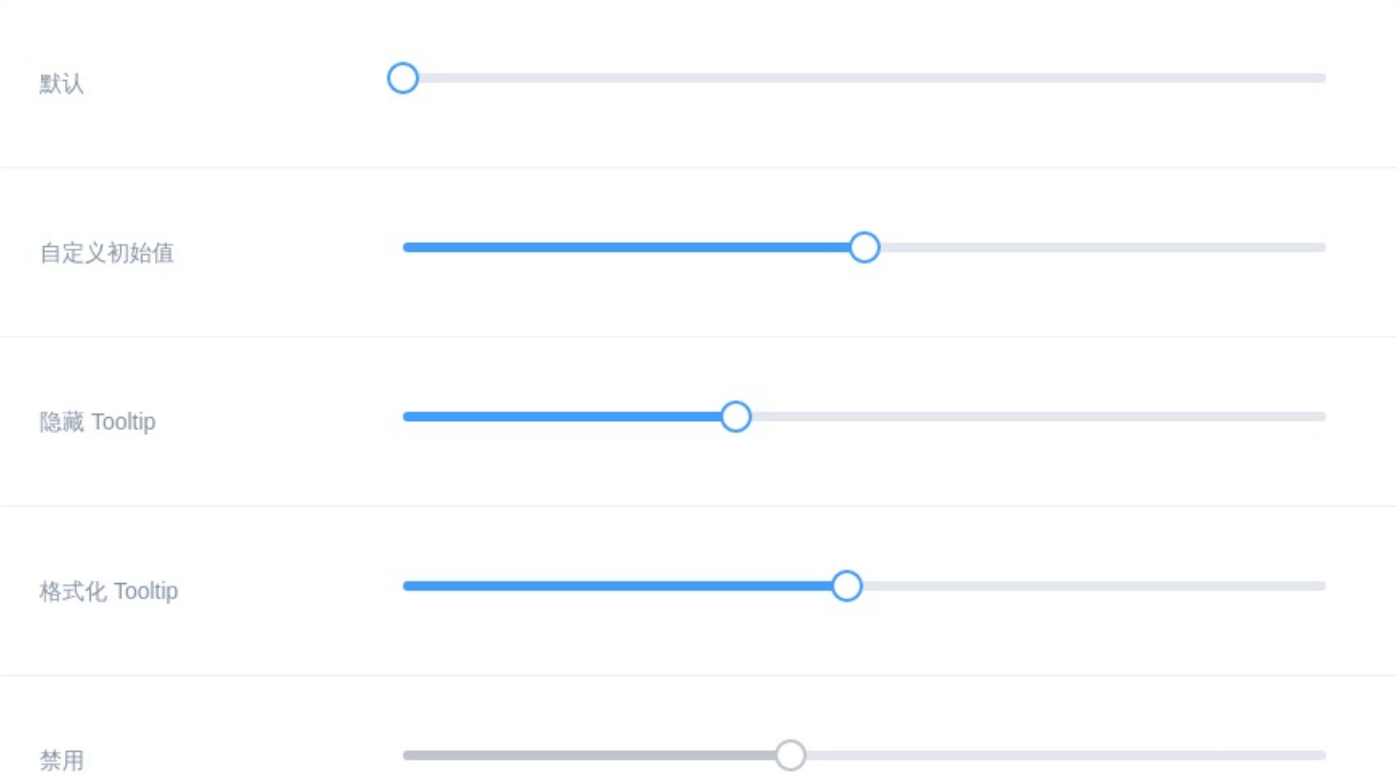
方法名	说明	参数
focus	使 Switch 获取焦点	-

Slider 滑块

通过拖动滑块在一个固定区间内进行选择

基础用法

在拖动滑块时，显示当前值



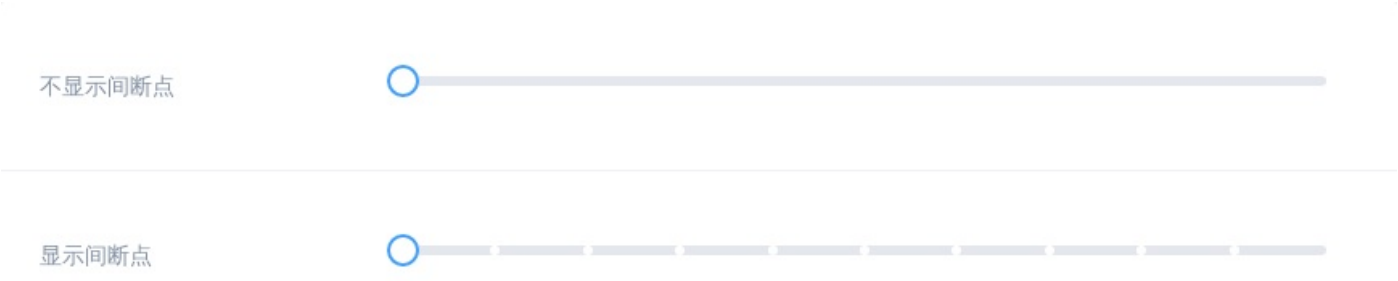
通过设置绑定值自定义滑块的初始值

```
<template>
  <div class="block">
    <span class="demonstration">默认</span>
    <el-slider v-model="value1"></el-slider>
  </div>
  <div class="block">
    <span class="demonstration">自定义初始值</span>
    <el-slider v-model="value2"></el-slider>
  </div>
  <div class="block">
    <span class="demonstration">隐藏 Tooltip</span>
    <el-slider v-model="value3" :show-tooltip="false"></el-slider>
```

```
    </div>
    <div class="block">
      <span class="demonstration">格式化 Tooltip</span>
      <el-slider v-model="value4" :format-tooltip="formatTooltip"></el-slider>
    </div>
    <div class="block">
      <span class="demonstration">禁用</span>
      <el-slider v-model="value5" disabled></el-slider>
    </div>
  </template>

  <script>
    export default {
      data() {
        return {
          value1: 0,
          value2: 50,
          value3: 36,
          value4: 48,
          value5: 42
        }
      },
      methods: {
        formatTooltip(val) {
          return val / 100;
        }
      }
    }
  </script>
```

离散值

选项可以是离散的



改变 `step` 的值可以改变步长，通过设置 `show-stops` 属性可以显示间断点

```
<template>
  <div class="block">
    <span class="demonstration">不显示间断点</span>
    <el-slider
      v-model="value1"
      :step="10">
    </el-slider>
  </div>
  <div class="block">
    <span class="demonstration">显示间断点</span>
    <el-slider
      v-model="value2"
      :step="10"
      show-stops>
    </el-slider>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        value1: 0,
        value2: 0
      }
    }
  }
</script>
```

带有输入框

通过输入框设置精确数值

设置 `show-input` 属性会在右侧显示一个输入框

```
<template>
  <div class="block">
    <el-slider
      v-model="value"
      show-input>
    </el-slider>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        value: 0
      }
    }
  }
</script>
```

范围选择

支持选择某一数值范围

设置 `range` 即可开启范围选择，此时绑定值是一个数组，其元素分别为最小边界值和最大边界值

```
<template>
  <div class="block">
    <el-slider
      v-model="value"
      range
      show-stops
      :max="10">
    </el-slider>
  </div>
</template>

<script>
```

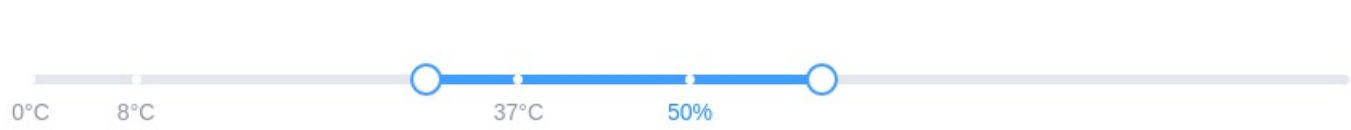
```
  export default {
    data() {
      return {
        value: [4, 8]
      }
    }
  }
</script>
```

竖向模式

设置 `vertical` 可使 Slider 变成竖向模式，此时必须设置高度 `height` 属性

```
<template>
  <div class="block">
    <el-slider
      v-model="value"
      vertical
      height="200px">
    </el-slider>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        value: 0
      }
    }
  }
</script>
```

展示标记



设置 `marks` 属性可以展示标记

```
<template>
  <div class="block">
    <el-slider
      v-model="value"
      range
      :marks="marks">
    </el-slider>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        value: [30, 60],
        marks: {
          0: '0°C',
          8: '8°C',
          37: '37°C',
          50: {
            style: {
              color: '#1989FA'
            },
            label: '50%'
          }
        }
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	number	—	0
min	最小值	number	—	0
max	最大值	number	—	100
disabled	是否禁用	boolean	—	false
step	步长	number	—	1
show-input	是否显示输入框，仅在非范围选择时有效	boolean	—	false
show-input-controls	在显示输入框的情况下，是否显示输入框的控制按钮	boolean	—	true
input-size	输入框的尺寸	string	large / medium / small / mini	small
show-stops	是否显示间断点	boolean	—	false
show-tooltip	是否显示 tooltip	boolean	—	true
format-tooltip	格式化 tooltip message	function(value)	—	—
range	是否为范围选择	boolean	—	false
vertical	是否竖向模式	boolean	—	false
height	Slider 高度，竖向模式时必填	string	—	—
label	屏幕阅读器标签	string	—	—
debounce	输入时的去抖延迟，毫秒，仅在 `show-input` 等于true时有效	number	—	300
tooltip-class	tooltip 的自定义类名	string	—	—
marks	标记， key 的类型必须为 number 且取值在闭区间 `[min, max]` 内，每个标记可以单独设置样式	object	—	—

Events

事件名称	说明	回调参数
change	值改变时触发（使用鼠标拖曳时，只在松开鼠标后触发）	改变后的值
input	数据改变时触发（使用鼠标拖曳时，活动过程实时触发）	改变后的值

TimePicker 时间选择器

用于选择或输入日期

任意时间点

可以选择任意时间

使用 el-time-picker 标签，通过 `selectableRange` 限制可选时间范围。提供了两种交互方式：默认情况下通过鼠标滚轮进行选择，打开 `arrow-control` 属性则通过界面上的箭头进行选择。

```
<template>
  <el-time-picker
    v-model="value1"
    :picker-options="{
      selectableRange: '18:30:00 - 20:30:00'
    }"
    placeholder="任意时间点">
  </el-time-picker>
  <el-time-picker
    arrow-control
    v-model="value2"
    :picker-options="{
      selectableRange: '18:30:00 - 20:30:00'
    }"
    placeholder="任意时间点">
  </el-time-picker>
</template>

<script>
  export default {
    data() {
      return {
        value1: new Date(2016, 9, 10, 18, 40),
        value2: new Date(2016, 9, 10, 18, 40)
      };
    }
```

```
  }
</script>
```

任意时间范围

可选择任意的时间范围

添加 `is-range` 属性即可选择时间范围，同样支持 `arrow-control` 属性。

```
<template>
  <el-time-picker
    is-range
    v-model="value1"
    range-separator="至"
    start-placeholder="开始时间"
    end-placeholder="结束时间"
    placeholder="选择时间范围">
  </el-time-picker>
  <el-time-picker
    is-range
    arrow-control
    v-model="value2"
    range-separator="至"
    start-placeholder="开始时间"
    end-placeholder="结束时间"
    placeholder="选择时间范围">
  </el-time-picker>
</template>

<script>
  export default {
    data() {
      return {
        value1: [new Date(2016, 9, 10, 8, 40), new Date(2016, 9, 10, 9, 40)],
        value2: [new Date(2016, 9, 10, 8, 40), new Date(2016, 9, 10, 9, 40)]
      };
    }
  }
```

```
30.  </script>
```

Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	date(TimePicker) / string(TimeSelect)	—	—
readonly	完全只读	boolean	—	false
disabled	禁用	boolean	—	false
editable	文本框可输入	boolean	—	true
clearable	是否显示清除按钮	boolean	—	true
size	输入框尺寸	string	medium / small / mini	—
placeholder	非范围选择时的占位内容	string	—	—
start-placeholder	范围选择时开始日期的占位内容	string	—	—
end-placeholder	范围选择时开始日期的占位内容	string	—	—
is-range	是否为时间范围选择，仅对 `<el-time-picker>` 有效	boolean	—	false
arrow-control	是否使用箭头进行时间选择，仅对 `<el-time-picker>` 有效	boolean	—	false
align	对齐方式	string	left / center / right	left
popper-class	TimePicker 下拉框的类名	string	—	—
range-separator	选择范围时的分隔符	string	-	‘-‘
string	见日期格式	—		
default-value	可选，选择器打开时默认显示的时间	Date(TimePicker) / string(TimeSelect)	可被 `new Date()` 解析(TimePicker) / 可选值(TimeSelect)	—
name	原生属性	string	—	—
prefix-icon	自定义头部图标的类名	string	—	el-icon-time
clear-icon	自定义清空图标的类名	string	—	el-icon-circle-close

Time Picker Options

参数	说明	类型	可选值	默认值
selectableRange	可选时间段，例如 ‘18:30:00 - 20:30:00’ 或者传入数组 [‘09:30:00 - 12:00:00’, ‘14:30:00 - 18:30:00’]	string / array	—	—
format	时间格式化(TimePicker)	string	小时： HH ，分： mm ，秒： ss ，AM/PM A	‘HH:mm:ss’

Events

事件名	说明	参数
change	用户确认选定的值时触发	组件绑定值
blur	当 input 失去焦点时触发	组件实例
focus	当 input 获得焦点时触发	组件实例

Methods

方法名	说明	参数
focus	使 input 获取焦点	-

TimeSelect 时间选择

用于选择或输入日期

固定时间点

提供几个固定的时间点供用户选择

使用 el-time-select 标签，分别通过 `start` 、 `end` 和 `step` 指定可选的起始时间、结束时间和步长

```
<el-time-select
  v-model="value"
  :picker-options="{
    start: '08:30',
    step: '00:15',
    end: '18:30'
  }"
  placeholder="选择时间">
</el-time-select>

<script>
  export default {
    data() {
      return {
        value: ''
      };
    }
  }
</script>
```

固定时间范围

若先选择开始时间，则结束时间内备选项的状态会随之改变


```
<template>
  <el-time-select
    placeholder="起始时间"
    v-model="startTime"
    :picker-options="{
      start: '08:30',
      step: '00:15',
      end: '18:30'
    }">
  </el-time-select>
  <el-time-select
    placeholder="结束时间"
    v-model="endTime"
    :picker-options="{
      start: '08:30',
      step: '00:15',
      end: '18:30',
      minTime: startTime
    }">
  </el-time-select>
</template>

<script>
  export default {
    data() {
      return {
        startTime: '',
        endTime: ''
      };
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	date(TimePicker) / string(TimeSelect)	—	—

editable	文本框可输入	boolean	—	true
clearable	是否显示清除按钮	boolean	—	true
size	输入框尺寸	string	medium / small / mini	—
placeholder	非范围选择时的占位内容	string	—	—
picker-options	当前时间日期选择器特有的选项参考下表	object	—	{}
name	原生属性	string	—	—
prefix-icon	自定义头部图标的类名	string	—	el-icon-time
clear-icon	自定义清空图标的类名	string	—	el-icon-circle-close

Time Select Options

参数	说明	类型	可选值	默认值
start	开始时间	string	—	09:00
end	结束时间	string	—	18:00
step	间隔时间	string	—	00:30
minTime	最小时间，小于该时间的时间段将被禁用	string	—	00:00
maxTime	最大时间，大于该时间的时间段将被禁用	string	—	—

Events

事件名	说明	参数
change	用户确认选定的值时触发	组件绑定值
blur	当 input 失去焦点时触发	组件实例
focus	当 input 获得焦点时触发	组件实例

Methods

方法名	说明	参数
focus	使 input 获取焦点	-
blur	去掉 input 获取焦点	-

DatePicker 日期选择器

用于选择或输入日期

选择日

以「日」为基本单位，基础的日期选择控件

基本单位由 `type` 属性指定。快捷选项需配置 `picker-options` 对象中的 `shortcuts` ，禁用日期通过 `disabledDate` 设置，传入函数

```
<template>
  <div class="block">
    <span class="demonstration">默认</span>
    <el-date-picker
      v-model="value1"
      type="date"
      placeholder="选择日期">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">带快捷选项</span>
    <el-date-picker
      v-model="value2"
      align="right"
      type="date"
      placeholder="选择日期"
      :disabled-date="disabledDate"
      :shortcuts="shortcuts"
    >
    </el-date-picker>
  </div>
</template>

```

```
<script>
  export default {
    data() {
      return {
        disabledDate(time) {
          return time.getTime() > Date.now()
        },
        shortcuts: [{
          text: 'Today',
          value: new Date(),
        }, {
          text: 'Yesterday',
          value: (() => {
            const date = new Date()
            date.setTime(date.getTime() - 3600 * 1000 * 24)
            return date
          })(),
        }, {
          text: 'A week ago',
          value: (() => {
            const date = new Date()
            date.setTime(date.getTime() - 3600 * 1000 * 24 * 7)
            return date
          })(),
        }],
        value1: '',
        value2: '',
      };
    }
  };
</script>
```

其他日期单位

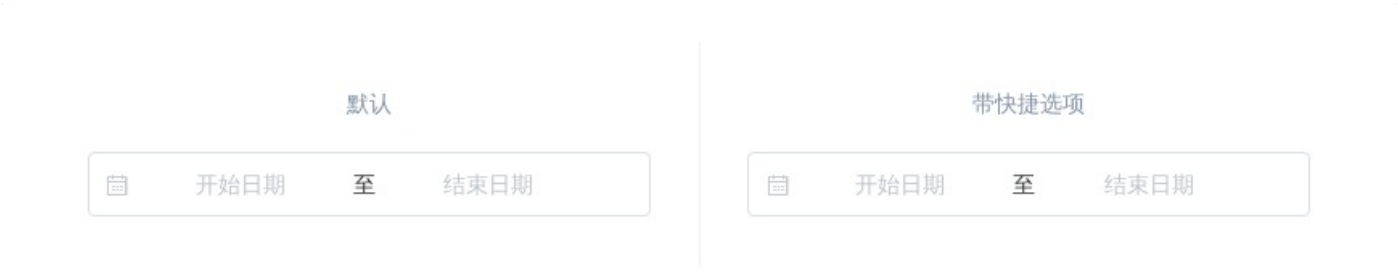
通过扩展基础的日期选择，可以选择周、月、年或多个日期


```
<div class="container">
  <div class="block">
    <span class="demonstration">周</span>
    <el-date-picker
      v-model="value1"
      type="week"
      format="gggg 第 ww 周"
      placeholder="选择周">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">月</span>
    <el-date-picker
      v-model="value2"
      type="month"
      placeholder="选择月">
    </el-date-picker>
  </div>
</div>
<div class="container">
  <div class="block">
    <span class="demonstration">年</span>
    <el-date-picker
      v-model="value3"
      type="year"
      placeholder="选择年">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">多个日期</span>
```

```
    <el-date-picker
      type="dates"
      v-model="value4"
      placeholder="选择一个或多个日期">
    </el-date-picker>
  </div>
</div>

<script>
  export default {
    data() {
      return {
        value1: '',
        value2: '',
        value3: '',
        value4: ''
      };
    }
  };
</script>
```

选择日期范围

可在一个选择器中便捷地选择一个时间范围



在选择日期范围时，默认情况下左右面板会联动。如果希望两个面板各自独立切换当前月份，可以使用 `unlink-panels` 属性解除联动。

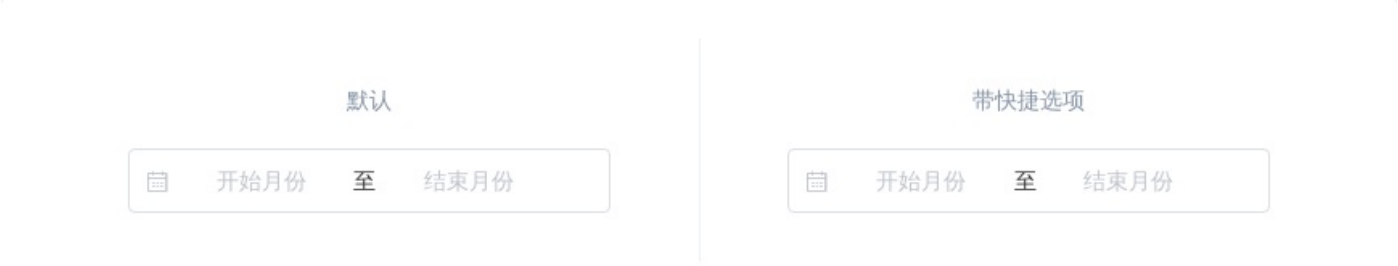
```
<template>
  <div class="block">
    <span class="demonstration">默认</span>
    <el-date-picker
      v-model="value1"
      type="daterange"
      range-separator="至"
```

```
      start-placeholder="开始日期"
      end-placeholder="结束日期">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">带快捷选项</span>
    <el-date-picker
      v-model="value2"
      type="daterange"
      align="right"
      unlink-panels
      range-separator="至"
      start-placeholder="开始日期"
      end-placeholder="结束日期"
      :shortcuts="shortcuts"
    >
    </el-date-picker>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        shortcuts: [{
          text: '最近一周',
          value: (() => {
            const end = new Date()
            const start = new Date()
            start.setTime(start.getTime() - 3600 * 1000 * 24 * 7)
            return [start, end]
          })(),
        }, {
          text: '最近一个月',
          value: (() => {
            const end = new Date()
            const start = new Date()
            start.setTime(start.getTime() - 3600 * 1000 * 24 * 30)
            return [start, end]
          })(),
        }, {
          text: '最近三个月',
```

```
          value: (() => {
            const end = new Date()
            const start = new Date()
            start.setTime(start.getTime() - 3600 * 1000 * 24 * 90)
            return [start, end]
          })(),
        }],
        value1: '',
        value2: ''
      };
    }
  };
</script>
```

选择月份范围

可在一个选择器中便捷地选择一个月份范围



在选择月份范围时，默认情况下左右面板会联动。如果希望两个面板各自独立切换当前年份，可以使用 `unlink-panels` 属性解除联动。

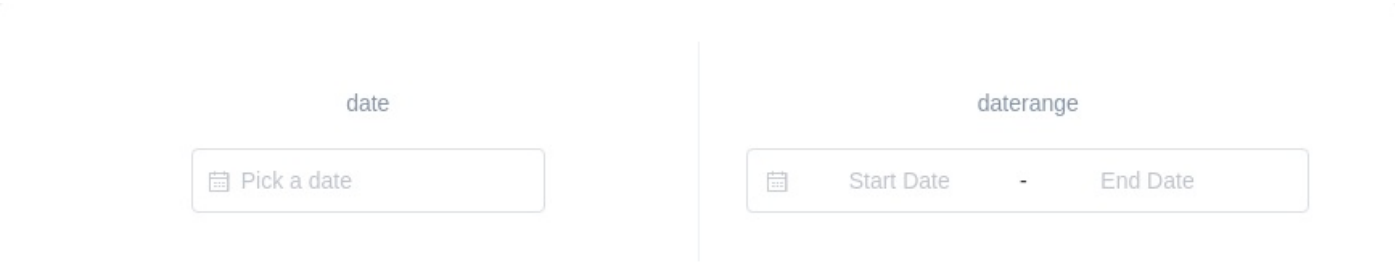
```
<template>
  <div class="block">
    <span class="demonstration">默认</span>
    <el-date-picker
      v-model="value1"
      type="monthrange"
      range-separator="至"
      start-placeholder="开始月份"
      end-placeholder="结束月份">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">带快捷选项</span>
    <el-date-picker
```

```
      v-model="value2"
      type="monthrange"
      align="right"
      unlink-panels
      range-separator="至"
      start-placeholder="开始月份"
      end-placeholder="结束月份"
      :shortcuts="shortcuts"
    >
    </el-date-picker>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        shortcuts: [{
          text: '本月',
          value: [new Date(), new Date()],
        }, {
          text: '今年至今',
          value: (() => {
            const end = new Date()
            const start = new Date(new Date().getFullYear(), 0)
            return [start, end]
          })(),
        }, {
          text: '最近六个月',
          value: (() => {
            const end = new Date()
            const start = new Date()
            start.setMonth(start.getMonth() - 6)
            return [start, end]
          })(),
        }],
        value1: '',
        value2: ''
      };
    }
  };
</script>
```

Default Value (需要翻译)

If user hasn’t picked a date, shows today’s calendar by default. You can use `default-value` to set another date. Its value should be parsable by `new Date()` .

If type is `daterange` , `default-value` sets the left side calendar.




```
<template>
  <div class="block">
    <span class="demonstration">date</span>
    <el-date-picker
      v-model="value1"
      type="date"
      placeholder="Pick a date"
      :default-value="new Date(2010, 9, 1)">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">daterange</span>
    <el-date-picker
      v-model="value2"
      type="daterange"
      align="right"
      start-placeholder="Start Date"
      end-placeholder="End Date"
      :default-value="[new Date(2010, 9, 1), new Date(2010, 10, 1)]">
    </el-date-picker>
  </div>
</template>

<script>
  export default {
    data() {
      return {
```

```
        value1: '',
        value2: ''
      };
    }
  };
</script>
```

日期格式

使用 `format` 指定输入框的格式。

默认情况下，组件接受并返回 `Date` 对象。

在 这里 查看 Day.js 支持的 format 参数。

请注意大小写


```
<template>
  <div class="block">
    <span class="demonstration">默认为 Date 对象</span>
    <div class="demonstration">值：{{ value1 }}</div>
    <el-date-picker
      v-model="value1"
      type="date"
      placeholder="选择日期"
      format="YYYY 年 MM 月 DD 日">
    </el-date-picker>
  </div>
</template>

<script>
  export default {
    data() {
      return {
```

```
        value1: '',
        value2: '',
        value3: ''
      };
    }
  };
</script>
```

默认显示日期

在选择日期范围时，指定起始日期和结束日期的默认时刻。

选择日期范围时，默认情况下，起始日期和结束日期的时间部分均为当天的 `0` 点 `0` 分 `0` 秒。通过 `default-time` 可以分别指定二者的具体时刻。 `default-time` 接受一个数组，其中的值为形如 `12:00:00` 的字符串，第一个值控制起始日期的时刻，第二个值控制结束日期的时刻。

```
<template>
  <div class="block">
    <p>组件值：{{ value }}</p>
    <el-date-picker
      v-model="value"
      type="daterange"
      start-placeholder="开始日期"
      end-placeholder="结束日期"
      :default-time="[new Date(2000, 1, 1, 0 , 0,0), new Date(2000, 2, 1, 23 ,
59,59)]">
    </el-date-picker>
  </div>
</template>

<script>
  export default {
    data() {
      return {
```

```
        value: ''
      };
    }
  };
</script>
```

Attributes

参数	说明	类型	可选值	
value / v-model	绑定值	date(DatePicker) / array(DateRangePicker)	—	—
readonly	完全只读	boolean	—	f
disabled	禁用	boolean	—	f
editable	文本框可输入	boolean	—	t
clearable	是否显示清除按钮	boolean	—	t
size	输入框尺寸	string	large/medium/small/mini	l
placeholder	非范围选择时的占位内容	string	—	—
start-placeholder	范围选择时开始日期的占位内容	string	—	—
end-placeholder	范围选择时结束日期的占位内容	string	—	—
type	显示类型	string	year/month/date/dates/ week/datetime/datetimerange/ daterange/monthrange	d
format	显示在输入框中的格式	string	见日期格式	y M
align	对齐方式	string	left, center, right	l
popper-class	DatePicker 下拉框的类名	string	—	—
range-separator	选择范围时的分隔符	string	—	'
default-value	可选，选择器打开时默认显示的时间	Date	可被 `new Date()` 解析	—
default-time	范围选择时选中日期所使用的当日内具体时刻	string[]	数组，长度为 2，每项值为字符串，形如 `12:00:00` ，第一项指定开始日期的时刻，第二项指定结束日期的时刻，不指定会使用时刻 `00:00:00`	—
name	原生属性	string	—	—
unlink-panels	在范围选择器里取消两个日期面板之间的	boolean	—	f

	联动			
prefix-icon	自定义头部图标的类名	string	—	e i d
clear-icon	自定义清空图标的类名	string	—	e i c c
validate-event	输入时是否触发表单的校验	boolean	-	t

Picker Options

参数	说明	类型	可选值	默认值
shortcuts	设置快捷选项，需要传入 { text, onClick } 对象用法参考 demo 或下表	Object[]	—	—
disabledDate	设置禁用状态，参数为当前日期，要求返回 Boolean	Function	—	—
cellClassName	设置日期的 className	Function(Date)	—	—
firstDayOfWeek	周起始日	Number	1 到 7	7
onPick	选中日期后会执行的回调，只有当 `daterange` 或 `datetimerange` 才生效	Function({ maxDate, minDate })	—	—

Shortcuts

参数	说明	类型	可选值	默认值
text	标题文本	string	—	—
onClick	选中后的回调函数，参数是 vm，可通过触发 ‘pick’ 事件设置选择器的值。例如 vm.$emit(‘pick’, new Date())	function	—	—

Events

事件名称	说明	回调参数
change	用户确认选定的值时触发	组件绑定值
blur	当 input 失去焦点时触发	组件实例
focus	当 input 获得焦点时触发	组件实例

Methods

方法名	说明	参数

focus	使 input 获取焦点	—

DateTimePicker 日期时间选择器

在同一个选择器里选择日期和时间

DateTimePicker 由 DatePicker 和 TimePicker 派生，`Picker Options` 或者其他选项可以参照 DatePicker 和 TimePicker。

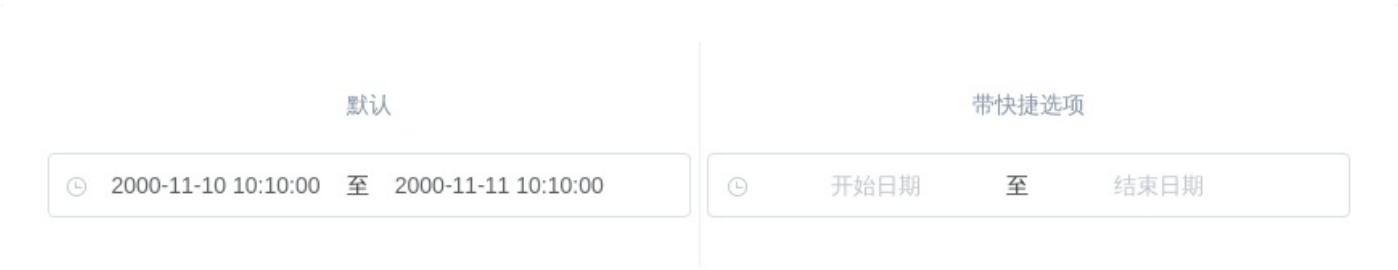
日期和时间点

通过设置 `type` 属性为 `datetime` ，即可在同一个选择器里同时进行日期和时间的选择。快捷选项的使用方法与 Date Picker 相同。

```
<template>
  <div class="block">
    <span class="demonstration">默认</span>
    <el-date-picker
      v-model="value1"
      type="datetime"
      placeholder="选择日期时间">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">带快捷选项</span>
    <el-date-picker
      v-model="value2"
      type="datetime"
      placeholder="选择日期时间"
      align="right"
      :picker-options="pickerOptions">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">设置默认时间</span>
```

```
    <el-date-picker
      v-model="value3"
      type="datetime"
      placeholder="选择日期时间"
      default-time="12:00:00">
    </el-date-picker>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        pickerOptions: {
          shortcuts: [{
            text: '今天',
            onClick(picker) {
              picker.$emit('pick', new Date());
            }
          }, {
            text: '昨天',
            onClick(picker) {
              const date = new Date();
              date.setTime(date.getTime() - 3600 * 1000 * 24);
              picker.emit('pick', date);
            }
          }, {
            text: '一周前',
            onClick(picker) {
              const date = new Date();
              date.setTime(date.getTime() - 3600 * 1000 * 24 * 7);
              picker.emit('pick', date);
            }
          }]
        },
        value1: '',
        value2: '',
        value3: ''
      };
    }
  };
</script>
```

日期和时间范围



设置 `type` 为 `datetimerange` 即可选择日期和时间范围

```
<template>
  <div class="block">
    <span class="demonstration">默认</span>
    <el-date-picker
      v-model="value1"
      type="datetimerange"
      range-separator="至"
      start-placeholder="开始日期"
      end-placeholder="结束日期">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">带快捷选项</span>
    <el-date-picker
      v-model="value2"
      type="datetimerange"
      :picker-options="pickerOptions"
      range-separator="至"
      start-placeholder="开始日期"
      end-placeholder="结束日期"
      align="right">
    </el-date-picker>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        pickerOptions: {
          shortcuts: [{
```

```
            text: '最近一周',
            onClick(picker) {
              const end = new Date();
              const start = new Date();
              start.setTime(start.getTime() - 3600 * 1000 * 24 * 7);
              picker.$emit('pick', [start, end]);
            }
          }, {
            text: '最近一个月',
            onClick(picker) {
              const end = new Date();
              const start = new Date();
              start.setTime(start.getTime() - 3600 * 1000 * 24 * 30);
              picker.$emit('pick', [start, end]);
            }
          }, {
            text: '最近三个月',
            onClick(picker) {
              const end = new Date();
              const start = new Date();
              start.setTime(start.getTime() - 3600 * 1000 * 24 * 90);
              picker.$emit('pick', [start, end]);
            }
          }]
        },
        value1: [new Date(2000, 10, 10, 10, 10), new Date(2000, 10, 11, 10,
10)],
        value2: ''
      };
    }
  };
</script>
```

默认的起始与结束时刻

使用 `datetimerange` 进行范围选择时，在日期选择面板中选定起始与结束的日期，默认会使用该日期的 `00:00:00` 作为起始与结束的时刻；通过选项 `default-time` 可以控制选中起始与结束日期时所使用的具体时刻。 `default-time` 接受一个数组，数组每项值为字符串，形如 `12:00:00` ，其中第一项控制起始日期的具体时刻，第二项控制结束日期的具体时刻。

```
<template>
  <div class="block">
    <span class="demonstration">起始日期时刻为 12:00:00</span>
    <el-date-picker
      v-model="value1"
      type="datetimerange"
      start-placeholder="开始日期"
      end-placeholder="结束日期"
      :default-time="['12:00:00']">
    </el-date-picker>
  </div>
  <div class="block">
    <span class="demonstration">起始日期时刻为 12:00:00，结束日期时刻为 08:00:00</span>
    <el-date-picker
      v-model="value2"
      type="datetimerange"
      align="right"
      start-placeholder="开始日期"
      end-placeholder="结束日期"
      :default-time="['12:00:00', '08:00:00']">
    </el-date-picker>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        value1: '',
        value2: ''
      };
    }
  };
</script>
```

Attributes

参数	说明	类型	可选值
value / v-model	绑定值	date(DateTimePicker) / array(DateTimeRangePicker)	—
readonly	完全只读	boolean	—
disabled	禁用	boolean	—
editable	文本框可输入	boolean	—
clearable	是否显示清除按钮	boolean	—
size	输入框尺寸	string	large/medium/small/mini
placeholder	非范围选择时的占位内容	string	—
start-placeholder	范围选择时开始日期的占位内容	string	—
end-placeholder	范围选择时结束日期的占位内容	string	—
time-arrow-control	是否使用箭头进行时间选择	boolean	—
type	显示类型	string	year/month/date/week/datetime/datetimerange/d
format	显示在输入框中的格式	string	见日期格式
align	对齐方式	string	left, center, right
popper-class	DateTimePicker 下拉框的类名	string	—
range-separator	选择范围时的分隔符	string	-
default-value	可选，选择器打开时默认显示的时间	Date	可被 `new Date()` 解析
default-time	选中日期后的默认具体时刻	非范围选择时：string / 范围选择时：string[]	非范围选择时：形如 `12:00:00` 串；范围选择时：数组，长度为 为字符串，形如 `12:00:00` ，第 开始日期的时刻，第二项指定结 刻。不指定会使用时刻 `00:00:`
name	原生属性	string	—
unlink-panels	在范围选择器里取消两个日期面板之间的联动	boolean	—
prefix-icon	自定义头部图标的类名	string	—
clear-icon	自定义清空图标的类名	string	—

Picker Options

参数	说明	类型	可选值	默认值
shortcuts	设置快捷选项，需要传入 { text, onClick } 对象用法参考 demo 或下表	Object[]	—	—
disabledDate	设置禁用状态，参数为当前日期，要求返回 Boolean	Function	—	—
cellClassName	设置日期的 className	Function(Date)	—	—
firstDayOfWeek	周起始日	Number	1 到 7	7

Shortcuts

参数	说明	类型	可选值	默认值
text	标题文本	string	—	—
onClick	选中后的回调函数，参数是 vm，可通过触发 ‘pick’ 事件设置选择器的值。例如 vm.$emit(‘pick’, new Date())	function	—	—

Events

Event Name	Description	Parameters
change	用户确认选定的值时触发	组件绑定值
blur	当 input 失去焦点时触发	组件实例
focus	当 input 获得焦点时触发	组件实例

Methods

方法名	说明	参数
focus	使 input 获取焦点	—

Slots

Name	说明
range-separator	自定义分隔符

Upload 上传

通过点击或者拖拽上传文件

点击上传

通过 slot 你可以传入自定义的上传按钮类型和文字提示。可通过设置 `limit` 和 `on-exceed` 来限制上传文件的个数和定义超出限制时的行为。可通过设置 `before-remove` 来阻止文件移除操作。

```
<el-upload
  class="upload-demo"
  action="https://jsonplaceholder.typicode.com/posts/"
  :on-preview="handlePreview"
  :on-remove="handleRemove"
  :before-remove="beforeRemove"
  multiple
  :limit="3"
  :on-exceed="handleExceed"
  :file-list="fileList"
>
  <el-button size="small" type="primary">点击上传</el-button>
  <template #tip>
    <div class="el-upload__tip">只能上传 jpg/png 文件，且不超过 500kb</div>
  </template>
</el-upload>
<script>
  export default {
    data() {
      return {
        fileList: [{name: 'food.jpeg', url:
'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg?
imageMogr2/thumbnail/360x360/format/webp/quality/100'}, {name: 'food2.jpeg',
url: 'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg?
imageMogr2/thumbnail/360x360/format/webp/quality/100'}]
```

```
      };
    },
    methods: {
      handleRemove(file, fileList) {
        console.log(file, fileList);
      },
      handlePreview(file) {
        console.log(file);
      },
      handleExceed(files, fileList) {
        this.$message.warning(`当前限制选择 3 个文件，本次选择了 ${files.length} 个文件，共选择了 ${files.length + fileList.length} 个文件`);
      },
      beforeRemove(file, fileList) {
        return this.$confirm(`确定移除 ${ file.name }？`);
      }
    }
  }
</script>
```

用户头像上传

使用 `before-upload` 限制用户上传的图片格式和大小。

```
<el-upload
  class="avatar-uploader"
  action="https://jsonplaceholder.typicode.com/posts/"
  :show-file-list="false"
  :on-success="handleAvatarSuccess"
  :before-upload="beforeAvatarUpload"
>
  <img v-if="imageUrl" :src="imageUrl" class="avatar">
  <i v-else class="el-icon-plus avatar-uploader-icon"></i>
```

```
</el-upload>

<style>
  .avatar-uploader .el-upload {
    border: 1px dashed #d9d9d9;
    border-radius: 6px;
    cursor: pointer;
    position: relative;
    overflow: hidden;
  }
  .avatar-uploader .el-upload:hover {
    border-color: #409EFF;
  }
  .avatar-uploader-icon {
    font-size: 28px;
    color: #8c939d;
    width: 178px;
    height: 178px;
    line-height: 178px;
    text-align: center;
  }
  .avatar {
    width: 178px;
    height: 178px;
    display: block;
  }
</style>

<script>
  export default {
    data() {
      return {
        imageUrl: ''
      };
    },
    methods: {
      handleAvatarSuccess(res, file) {
        this.imageUrl = URL.createObjectURL(file.raw);
      },
      beforeAvatarUpload(file) {
        const isJPG = file.type === 'image/jpeg';
        const isLt2M = file.size / 1024 / 1024 < 2;
```

```
52.
53.         if (!isJPG) {
54.           this.$message.error('上传头像图片只能是 JPG 格式!');
55.         }
56.         if (!isLt2M) {
57.           this.$message.error('上传头像图片大小不能超过 2MB!');
58.         }
59.         return isJPG && isLt2M;
60.       }
61.     }
62.   }
63. </script>
```

照片墙

使用 `list-type` 属性来设置文件列表的样式。

```
1.  <el-upload
2.    action="https://jsonplaceholder.typicode.com/posts/"
3.    list-type="picture-card"
4.    :on-preview="handlePictureCardPreview"
5.    :on-remove="handleRemove">
6.    <i class="el-icon-plus"></i>
7.  </el-upload>
8.  <el-dialog :visible.sync="dialogVisible">
9.    <img width="100%" :src="dialogImageUrl" alt="">
10. </el-dialog>
11. <script>
12.   export default {
13.     data() {
14.       return {
15.         dialogImageUrl: '',
16.         dialogVisible: false
17.       };
```

```
    },
    methods: {
      handleRemove(file, fileList) {
        console.log(file, fileList);
      },
      handlePictureCardPreview(file) {
        this.dialogImageUrl = file.url;
        this.dialogVisible = true;
      }
    }
  }
</script>
```

文件缩略图

使用 `scoped-slot` 去设置缩略图模版。

```
<el-upload
  action="#"
  list-type="picture-card"
  :auto-upload="false">
    <template #default>
      <i class="el-icon-plus"></i>
    </template>
    <template #file="{file}">
      <div>
        <img class="el-upload-list__item-thumbnail" :src="file.url" alt="" />
        <span class="el-upload-list__item-actions">
          <span
            class="el-upload-list__item-preview"
            @click="handlePictureCardPreview(file)"
          >
            <i class="el-icon-zoom-in"></i>
          </span>
```

```
          <span
            v-if="!disabled"
            class="el-upload-list__item-delete"
            @click="handleDownload(file)"
          >
            <i class="el-icon-download"></i>
          </span>
          <span
            v-if="!disabled"
            class="el-upload-list__item-delete"
            @click="handleRemove(file)"
          >
            <i class="el-icon-delete"></i>
          </span>
        </span>
      </div>
    </template>
</el-upload>
<el-dialog :visible.sync="dialogVisible">
  <img width="100%" :src="dialogImageUrl" alt="">
</el-dialog>
<script>
  export default {
    data() {
      return {
        dialogImageUrl: '',
        dialogVisible: false,
        disabled: false
      };
    },
    methods: {
      handleRemove(file) {
        console.log(file);
      },
      handlePictureCardPreview(file) {
        this.dialogImageUrl = file.url;
        this.dialogVisible = true;
      },
      handleDownload(file) {
        console.log(file);
      }
    }
```

```
  }
</script>
```

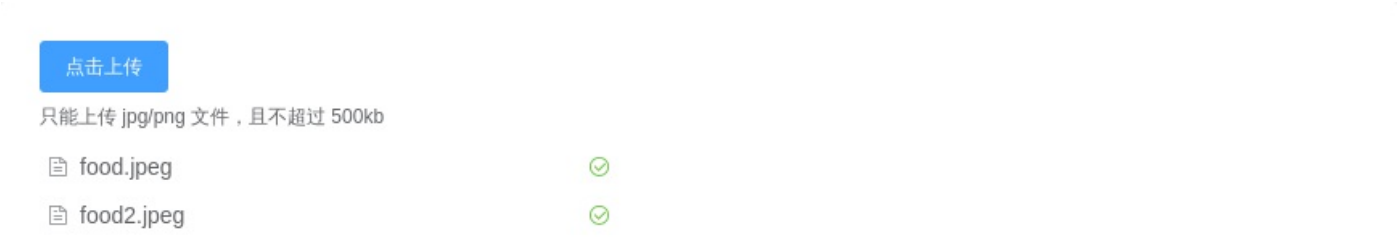
图片列表缩略图


```
<el-upload
  class="upload-demo"
  action="https://jsonplaceholder.typicode.com/posts/"
  :on-preview="handlePreview"
  :on-remove="handleRemove"
  :file-list="fileList"
  list-type="picture">
  <el-button size="small" type="primary">点击上传</el-button>
  <template #tip>
    <div class="el-upload__tip">
      只能上传 jpg/png 文件，且不超过 500kb
    </div>
  </template>
</el-upload>
<script>
  export default {
    data() {
      return {
        fileList: [{name: 'food.jpeg', url:
'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg?
imageMogr2/thumbnail/360x360/format/webp/quality/100'}, {name: 'food2.jpeg',
url: 'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg?
imageMogr2/thumbnail/360x360/format/webp/quality/100'}]
      };
    },
```

```
    methods: {
      handleRemove(file, fileList) {
        console.log(file, fileList);
      },
      handlePreview(file) {
        console.log(file);
      }
    }
  }
</script>
```

上传文件列表控制

通过 `on-change` 钩子函数来对列表进行控制




```
<el-upload
  class="upload-demo"
  action="https://jsonplaceholder.typicode.com/posts/"
  :on-change="handleChange"
  :file-list="fileList">
  <el-button size="small" type="primary">点击上传</el-button>
  <template #tip>
    <div class="el-upload__tip">
      只能上传 jpg/png 文件，且不超过 500kb
    </div>
  </template>
</el-upload>
<script>
  export default {
    data() {
      return {
        fileList: [{
          name: 'food.jpeg',
```

```
            url:
    'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg?
19. imageMogr2/thumbnail/360x360/format/webp/quality/100'
20.         }, {
21.           name: 'food2.jpeg',
            url:
    'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg?
22. imageMogr2/thumbnail/360x360/format/webp/quality/100'
23.         }]
24.       };
25.     },
26.     methods: {
27.       handleChange(file, fileList) {
28.         this.fileList = fileList.slice(-3);
29.       }
30.     }
31.   }
32. </script>
```

拖拽上传


```
1.  <el-upload
2.    class="upload-demo"
3.    drag
4.    action="https://jsonplaceholder.typicode.com/posts/"
5.    multiple>
6.    <i class="el-icon-upload"></i>
7.    <div class="el-upload__text">将文件拖到此处，或<em>点击上传</em></div>
8.    <template #tip>
9.      <div class="el-upload__tip">
10.       只能上传 jpg/png 文件，且不超过 500kb
11.     </div>
```

```
  </template>
</el-upload>
```

手动上传


```
<el-upload
  class="upload-demo"
  ref="upload"
  action="https://jsonplaceholder.typicode.com/posts/"
  :on-preview="handlePreview"
  :on-remove="handleRemove"
  :file-list="fileList"
  :auto-upload="false">
  <template #trigger>
    <el-button size="small" type="primary">选取文件</el-button>
  </template>
  <el-button style="margin-left: 10px;" size="small" type="success"
@click="submitUpload">上传到服务器</el-button>
  <template #tip>
    <div class="el-upload__tip">
      只能上传 jpg/png 文件，且不超过 500kb
    </div>
  </template>
</el-upload>
<script>
  export default {
    data() {
      return {
        fileList: [{name: 'food.jpeg', url:
'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg?
imageMogr2/thumbnail/360x360/format/webp/quality/100'}, {name: 'food2.jpeg',
url: 'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg?
imageMogr2/thumbnail/360x360/format/webp/quality/100'}]
      };
    },
```

```
    methods: {
      submitUpload() {
        this.$refs.upload.submit();
      },
      handleRemove(file, fileList) {
        console.log(file, fileList);
      },
      handlePreview(file) {
        console.log(file);
      }
    }
  }
</script>
```

Attribute

参数	说明	类型	可选值
action	必选参数，上传的地址	string	—
headers	设置上传的请求头部	object	—
multiple	是否支持多选文件	boolean	—
data	上传时附带的额外参数	object	—
name	上传的文件字段名	string	—
with-credentials	支持发送 cookie 凭证信息	boolean	—
show-file-list	是否显示已上传文件列表	boolean	—
drag	是否启用拖拽上传	boolean	—
accept	接受上传的文件类型（thumbnail-mode 模式下此参数无效）	string	—
on-preview	点击文件列表中已上传的文件时的钩子	function(file)	—
on-remove	文件列表移除文件时的钩子	function(file, fileList)	—
on-success	文件上传成功时的钩子	function(response, file, fileList)	—
on-error	文件上传失败时的钩子	function(err, file, fileList)	—
on-progress	文件上传时的钩子	function(event, file, fileList)	—
on-change	文件状态改变时的钩子，添加文件、上传成功和上传失败时都会被调用	function(file, fileList)	—
before-upload	上传文件之前的钩子，参数为上传的文件，若返回 false 或者返回 Promise 且被 reject，则停止上传。	function(file)	—

before-remove	删除文件之前的钩子，参数为上传的文件和文件列表，若返回 false 或者返回 Promise 且被 reject，则停止删除。	function(file, fileList)	—
list-type	文件列表的类型	string	text/picture/picture-card
auto-upload	是否在选取文件后立即进行上传	boolean	—
file-list	上传的文件列表，例如：[{name: 'food.jpg', url: 'https://xxx.cdn.com/xxx.jpg'}]	array	—
http-request	覆盖默认的上传行为，可以自定义上传的实现	function	—
disabled	是否禁用	boolean	—
limit	最大允许上传个数	number	—
on-exceed	文件超出个数限制时的钩子	function(files, fileList)	—

Slot

name	说明
trigger	触发文件选择框的内容
tip	提示说明文字

Methods

方法名	说明	参数
clearFiles	清空已上传的文件列表（该方法不支持在 before-upload 中调用）	—
abort	取消上传请求	（ file: fileList 中的 file 对象 ）
submit	手动上传文件列表	—

Rate 评分

评分组件

基础用法

评分默认被分为三个等级，可以利用颜色数组对分数及情感倾向进行分级（默认情况下不区分颜色）。三个等级所对应的颜色用 `colors` 属性设置，而它们对应的两个阈值则通过 `low-threshold` 和 `high-threshold` 设定。你也可以通过传入颜色对象来自定义分段，键名为分段的界限值，键值为对应的颜色。

```
<div class="block">
  <span class="demonstration">默认不区分颜色</span>
  <el-rate v-model="value1"></el-rate>
</div>
<div class="block">
  <span class="demonstration">区分颜色</span>
  <el-rate
    v-model="value2"
    :colors="colors">
  </el-rate>
</div>

<script>
  export default {
    data() {
      return {
        value1: null,
        value2: null,
        colors: ['#99A9BF', '#F7BA2A', '#FF9900']  // 等同于 { 2: '#99A9BF', 4:
{ value: '#F7BA2A', excluded: true }, 5: '#FF9900' }
      }
    }
  }
```

```
</script>
```

允许半选

属性 `allow-half` 允许出现半星

```
<div class="block">
  <el-rate  v-model="value"  allow-half />
</div>


<script>
import { defineComponent, ref } from 'vue'
  export default {
    setup() {
      return {
        value: ref(null)
      }
    }
  }
</script>
```

辅助文字

用辅助文字直接地表达对应分数

为组件设置 `show-text` 属性会在右侧显示辅助文字。通过设置 `texts` 可以为每一个分值指定对应的辅助文字。 `texts` 为一个数组，长度应等于最大值 `max` 。

```
<el-rate
  v-model="value"
  show-text>
```

```
</el-rate>

<script>
  export default {
    data() {
      return {
        value: null
      }
    }
  }
</script>
```

其它 icon

当有多层评价时，可以用不同类型的 icon 区分评分层级

设置 `icon-classes` 属性可以自定义不同分段的图标。若传入数组，共有 3 个元素，为 3 个分段所对应的类名；若传入对象，可自定义分段，键名为分段的界限值，键值为对应的类名。本例还使用 `void-icon-class` 指定了未选中时的图标类名。

```
<el-rate
  v-model="value"
  :icon-classes="iconClasses"
  void-icon-class="icon-rate-face-off"
  :colors="['#99A9BF', '#F7BA2A', '#FF9900']">
</el-rate>

<script>
  export default {
    data() {
      return {
        value: null,
        iconClasses: ['icon-rate-face-1', 'icon-rate-face-2', 'icon-rate-face-3'] // 等同于 { 2: 'icon-rate-face-1', 4: { value: 'icon-rate-face-2', excluded: true }, 5: 'icon-rate-face-3' }
      }
    }
  }
</script>
```

只读

只读的评分用来展示分数，允许出现半星

为组件设置 `disabled` 属性表示组件为只读，支持小数分值。此时若设置 `show-score` ，则会在右侧显示目前的分值。可以提供 `score-template` 作为显示模板，模板为一个包含了 `{value}` 的字符串， `{value}` 会被解析为分值。

```
<el-rate
  v-model="value"
  disabled
  show-score
  text-color="#ff9900"
  score-template="{value}">
</el-rate>

<script>
  export default {
    data() {
      return {
        value: 3.7
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	number	—	0
max	最大分值	number	—	5
disabled	是否为只读	boolean	—	false
allow-half	是否允许半选	boolean	—	false
low-	低分和中等分数的界限值，值本身被划分在低分	number	—	2

threshold	中			
high-threshold	高分和中等分数的界限值，值本身被划分在高分中	number	—	4
colors	icon 的颜色。若传入数组，共有 3 个元素，为 3 个分段所对应的颜色；若传入对象，可自定义分段，键名为分段的界限值，键值为对应的颜色	array/object	—	[‘#F7BA2A’, ‘#F7BA2A’, ‘#F7BA2A’]
void-color	未选中 icon 的颜色	string	—	#C6D1DE
disabled-void-color	只读时未选中 icon 的颜色	string	—	#EFF2F7
icon-classes	icon 的类名。若传入数组，共有 3 个元素，为 3 个分段所对应的类名；若传入对象，可自定义分段，键名为分段的界限值，键值为对应的类名	array/object	—	[‘el-icon-star-on’, ‘el-icon-star-on’,’el-icon-star-on’]
void-icon-class	未选中 icon 的类名	string	—	el-icon-star-off
disabled-void-icon-class	只读时未选中 icon 的类名	string	—	el-icon-star-on
show-text	是否显示辅助文字，若为真，则会从 texts 数组中选取当前分数对应的文字内容	boolean	—	false
show-score	是否显示当前分数，show-score 和 show-text 不能同时为真	boolean	—	false
text-color	辅助文字的颜色	string	—	#1F2D3D
texts	辅助文字数组	array	—	[‘极差’, ‘失望’, ‘一般’, ‘满意’, ‘惊喜’]
score-template	分数显示模板	string	—	{value}

Events

事件名称	说明	回调参数
change	分值改变时触发	改变后的分值

ColorPicker 颜色选择器

用于颜色选择，支持多种格式。

基础用法

使用 v-model 与 Vue 实例中的一个变量进行双向绑定，绑定的变量需要是字符串类型。

```
<div class="block">
  <span class="demonstration">有默认值</span>
  <el-color-picker v-model="color1"></el-color-picker>
</div>
<div class="block">
  <span class="demonstration">无默认值</span>
  <el-color-picker v-model="color2"></el-color-picker>
</div>

<script>
  export default {
    data() {
      return {
        color1: '#409EFF',
        color2: null
      }
    }
  };
</script>
```

选择透明度

ColorPicker 支持普通颜色，也支持带 Alpha 通道的颜色，通过 `show-alpha` 属性即可控制是否支持透明度的选择。

```
<el-color-picker v-model="color" show-alpha></el-color-picker>

<script>
  export default {
    data() {
      return {
        color: 'rgba(19, 206, 102, 0.8)'
      }
    }
  };
</script>
```

预定义颜色

ColorPicker 支持预定义颜色

```
<el-color-picker
  v-model="color"
  show-alpha
  :predefine="predefineColors">
</el-color-picker>

<script>
  export default {
    data() {
      return {
        color: 'rgba(255, 69, 0, 0.68)',
        predefineColors: [
          '#ff4500',
          '#ff8c00',
```

```
          '#ffd700',
          '#90ee90',
          '#00ced1',
          '#1e90ff',
          '#c71585',
          'rgba(255, 69, 0, 0.68)',
          'rgb(255, 120, 0)',
          'hsv(51, 100, 98)',
          'hsva(120, 40, 94, 0.5)',
          'hsl(181, 100%, 37%)',
          'hsla(209, 100%, 56%, 0.73)',
          '#c7158577'
        ]
      }
    }
  };
</script>
```

不同尺寸

```
<el-color-picker v-model="color"></el-color-picker>
<el-color-picker v-model="color" size="medium"></el-color-picker>
<el-color-picker v-model="color" size="small"></el-color-picker>
<el-color-picker v-model="color" size="mini"></el-color-picker>

<script>
  export default {
    data() {
      return {
        color: '#409EFF'
      }
    }
  };
</script>
```

Attributes

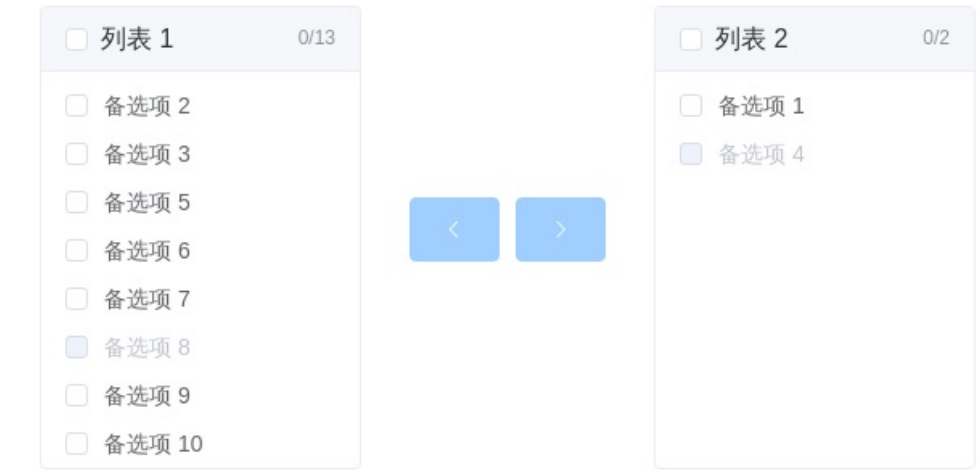
参数	说明	类型	可选值	默认值
value / v-model	绑定值	string	—	—
disabled	是否禁用	boolean	—	false
size	尺寸	string	—	medium / small / mini
show-alpha	是否支持透明度选择	boolean	—	false
color-format	写入 v-model 的颜色的格式	string	hsl / hsv / hex / rgb	hex（show-alpha 为 false）/ rgb（show-alpha 为 true）
popper-class	ColorPicker 下拉框的类名	string	—	—
predefine	预定义颜色	array	—	—

Events

事件名称	说明	回调参数
change	当绑定值变化时触发	当前值
active-change	面板中当前显示的颜色发生改变时触发	当前显示的颜色值

Transfer 穿梭框

基础用法



Transfer 的数据通过 `data` 属性传入。数据需要是一个对象数组，每个对象有以下属性：`key` 为数据的唯一性标识，`label` 为显示文本，`disabled` 表示该项数据是否禁止转移。目标列表中的数据项会同步到绑定至 `v-model` 的变量，值为数据项的 `key` 所组成的数组。当然，如果希望在初始状态时目标列表不为空，可以像本例一样为 `v-model` 绑定的变量赋予一个初始值。

```
<template>
  <el-transfer v-model="value" :data="data" />
</template>

<script>
  export default {
    data() {
      const generateData = _ => {
        const data = [];
        for (let i = 1; i <= 15; i++) {
          data.push({
            key: i,
            label: `备选项 ${ i }`,
            disabled: i % 4 === 0
          });
        }
        return data;
      };
```

```
      return {
        data: generateData(),
        value: [1, 4]
      };
    }
  };
</script>
```

可搜索

在数据很多的情况下，可以对数据进行搜索和过滤。

设置 `filterable` 为 `true` 即可开启搜索模式。默认情况下，若数据项的 `label` 属性包含搜索关键字，则会在搜索结果中显示。你也可以使用 `filter-method` 定义自己的搜索逻辑。`filter-method` 接收一个方法，当搜索关键字变化时，会将当前的关键字和每个数据项传给该方法。若方法返回 `true`，则会在搜索结果中显示对应的数据项。

```
<template>
  <el-transfer
    v-model="value"
    filterable
    :filter-method="filterMethod"
    filter-placeholder="请输入城市拼音"
    :data="data"
  />
</template>

<script>
  export default {
```

```
    data() {
      const generateData = _ => {
        const data = [];
        const cities = ['上海', '北京', '广州', '深圳', '南京', '西安', '成都'];
        const spell = ['shanghai', 'beijing', 'guangzhou', 'shenzhen',
'nanjing', 'xian', 'chengdu'];
        cities.forEach((city, index) => {
          data.push({
            label: city,
            key: index,
            spell: spell[index]
          });
        });
        return data;
      };
      return {
        data: generateData(),
        value: [],
        filterMethod(query, item) {
          return item.spell.indexOf(query) > -1;
        }
      };
    }
  };
</script>
```

可自定义

可以对列表标题文案、按钮文案、数据项的渲染函数、列表底部的勾选状态文案、列表底部的内容区等进行自定义。

可以使用 `titles`、`button-texts`、`render-content` 和 `format` 属性分别对列表标题文案、按钮文案、数据项的渲染函数和列表顶部的勾选状态文案进行自定义。数据项的渲染还可以使用 `scoped-slot` 进行自定义。对于列表底部的内容区，提供了两个具名 slot：`left-footer` 和 `right-footer`。此外，如果希望某些数据项在初始化时就被勾选，可以使用 `left-default-checked` 和 `right-default-checked` 属性。最后，本例还展示了 `change` 事件的用法。注意：由于 jsfiddle 不支持 JSX 语法，所以使用 `render-content` 自定义数据项的例子在 jsfiddle 中无法运行。但是在实际的项目中，只要正确地配置了相关依赖，就可以正常运行。

```
1.  <template>
      <p style="text-align: center; margin: 0 0 20px">使用 render-content 自定义数据
2.  项</p>
3.    <div style="text-align: center">
```

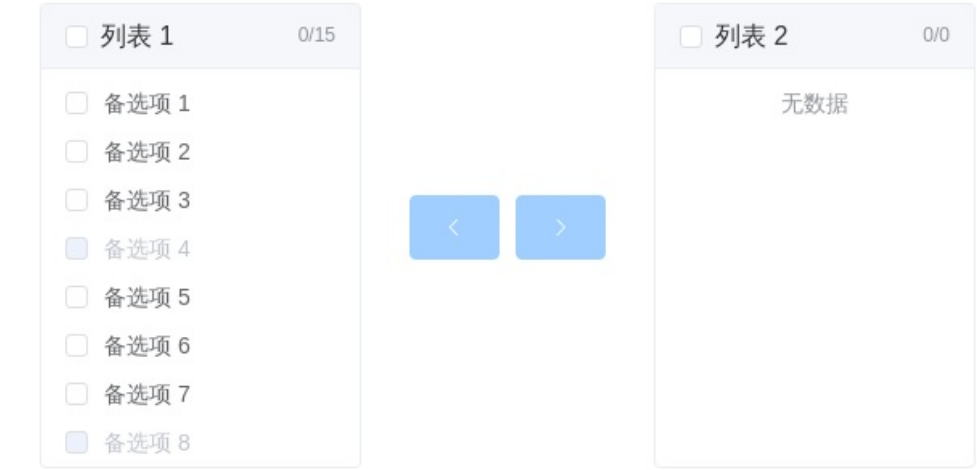
```
    <el-transfer
      v-model="leftValue"
      style="text-align: left; display: inline-block"
      filterable
      :left-default-checked="[2, 3]"
      :right-default-checked="[1]"
      :render-content="renderFunc"
      :titles="['Source', 'Target']"
      :button-texts="['到左边', '到右边']"
      :format="{
        noChecked: '${total}',
        hasChecked: '${checked}/${total}'
      }"
      :data="data"
      @change="handleChange"
    >
      <template #left-footer>
        <el-button class="transfer-footer" size="small">操作</el-button>
      </template>
      <template #right-footer>
        <el-button class="transfer-footer" size="small">操作</el-button>
      </template>
    </el-transfer>
  </div>
  <p style="text-align: center; margin: 50px 0 20px">使用 scoped-slot 自定义数据项</p>
  <div style="text-align: center">
    <el-transfer
      v-model="rightValue"
      style="text-align: left; display: inline-block"
      filterable
      :left-default-checked="[2, 3]"
      :right-default-checked="[1]"
      :titles="['Source', 'Target']"
      :button-texts="['到左边', '到右边']"
      :format="{
        noChecked: '${total}',
        hasChecked: '${checked}/${total}'
      }"
      :data="data"
      @change="handleChange"
    >
```

```
      <template #default="{option}">
        <span>{{ option.key }} - {{ option.label }}</span>
      </template>
      <template #left-footer>
        <el-button class="transfer-footer" size="small">操作</el-button>
      </template>
      <template #right-footer>
        <el-button class="transfer-footer" size="small">操作</el-button>
      </template>
    </el-transfer>
  </div>
</template>

<style>
.transfer-footer {
  margin-left: 20px;
  padding: 6px 5px;
}
</style>

<script>
  export default {
    data() {
      const generateData = _ => {
        const data = [];
        for (let i = 1; i <= 15; i++) {
          data.push({
            key: i,
            label: `备选项 ${ i }`,
            disabled: i % 4 === 0
          });
        }
        return data;
      };
      return {
        data: generateData(),
        leftValue: [1],
        rightValue: [1],
        renderFunc(h, option) {
          return h("span", null, option.key, " - ", option.label);
        }
      };
```

```
    },

    methods: {
      handleChange(value, direction, movedKeys) {
        console.log(value, direction, movedKeys);
      }
    }
  };
</script>
```

数据项属性别名

默认情况下，Transfer 仅能识别数据项中的 `key`、`label` 和 `disabled` 字段。如果你的数据的字段名不同，可以使用 `props` 属性为它们设置别名。



本例中的数据源没有 `key` 和 `label` 字段，在功能上与它们相同的字段名为 `value` 和 `desc`。因此可以使用 `props` 属性为 `key` 和 `label` 设置别名。

```
<template>
  <el-transfer
    v-model="value"
    :props="{
      key: 'value',
      label: 'desc'
    }"
    :data="data"
  />
</template>

```

```
<script>
  export default {
    data() {
      const generateData = _ => {
        const data = [];
        for (let i = 1; i <= 15; i++) {
          data.push({
            value: i,
            desc: `备选项 ${ i }`,
            disabled: i % 4 === 0
          });
        }
        return data;
      };
      return {
        data: generateData(),
        value: []
      };
    }
  };
</script>
```

Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	array	—	—
data	Transfer 的数据源	array[{ key, label, disabled }]	—	[ ]
filterable	是否可搜索	boolean	—	false
filter-placeholder	搜索框占位符	string	—	请输入搜索内容
filter-method	自定义搜索方法	function	—	—
target-order	右侧列表元素的排序策略：若为 `original`，则保持与数据源相同的顺序；若为 `push`，则新加入的元素排在最后；若为 `unshift`，则新加入的元素排在最前	string	original / push / unshift	original

titles	自定义列表标题	array	—	[‘列表 1’, ‘列表 2’]
button-texts	自定义按钮文案	array	—	[ ]
render-content	自定义数据项渲染函数	function(h, option)	—	—
format	列表顶部勾选状态文案	object{noChecked, hasChecked}	—	{ noChecked: ‘${checked}/${total}’, hasChecked: ‘${checked}/${total}’ }
props	数据源的字段别名	object{key, label, disabled}	—	—
left-default-checked	初始状态下左侧列表的已勾选项的 key 数组	array	—	[ ]
right-default-checked	初始状态下右侧列表的已勾选项的 key 数组	array	—	[ ]

Slot

name	说明
left-footer	左侧列表底部的内容
right-footer	右侧列表底部的内容

Scoped Slot

name	说明
—	自定义数据项的内容，参数为 { option }

Methods

方法名	说明	参数
clearQuery	清空某个面板的搜索关键词	‘left’ / ‘right’，指定需要清空的面板

Events

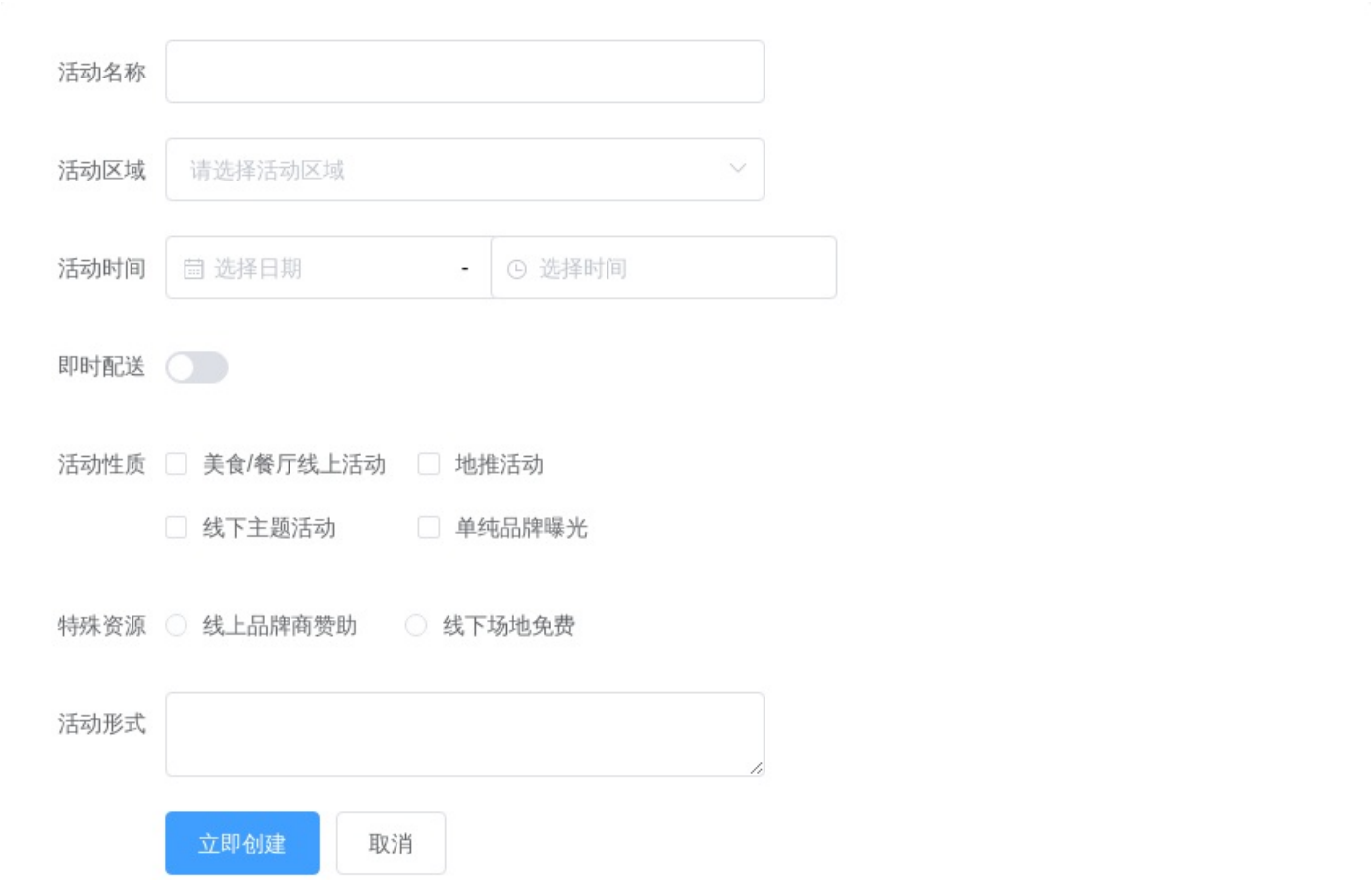
事件名称	说明	回调参数
change	右侧列表元素变化时触发	当前值、数据移动的方向（’left’ / ‘right’）、发生移动的数据 key 数组
left-check-change	左侧列表元素被用户选中 / 取消选中时触发	当前被选中的元素的 key 数组、选中状态发生变化的元素的 key 数组
right-check-change	右侧列表元素被用户选中 / 取消选中时触发	当前被选中的元素的 key 数组、选中状态发生变化的元素的 key 数组

Form 表单

由输入框、选择器、单选框、多选框等控件组成，用以收集、校验、提交数据

典型表单

包括各种表单项，比如输入框、选择器、开关、单选框、多选框等。



在 Form 组件中，每一个表单域由一个 Form-Item 组件构成，表单域中可以放置各种类型的表单控件，包括 Input、Select、Checkbox、Radio、Switch、DatePicker、TimePicker

```
<el-form ref="form" :model="form" label-width="80px">
  <el-form-item label="活动名称">
    <el-input v-model="form.name"></el-input>
  </el-form-item>
  <el-form-item label="活动区域">
    <el-select v-model="form.region" placeholder="请选择活动区域">
      <el-option label="区域一" value="shanghai"></el-option>
      <el-option label="区域二" value="beijing"></el-option>
    </el-select>
```

```
  </el-form-item>
  <el-form-item label="活动时间">
    <el-col :span="11">
      <el-date-picker type="date" placeholder="选择日期" v-model="form.date1"
style="width: 100%;"></el-date-picker>
    </el-col>
    <el-col class="line" :span="2">-</el-col>
    <el-col :span="11">
      <el-time-picker placeholder="选择时间" v-model="form.date2" style="width:
100%;"></el-time-picker>
    </el-col>
  </el-form-item>
  <el-form-item label="即时配送">
    <el-switch v-model="form.delivery"></el-switch>
  </el-form-item>
  <el-form-item label="活动性质">
    <el-checkbox-group v-model="form.type">
      <el-checkbox label="美食/餐厅线上活动" name="type"></el-checkbox>
      <el-checkbox label="地推活动" name="type"></el-checkbox>
      <el-checkbox label="线下主题活动" name="type"></el-checkbox>
      <el-checkbox label="单纯品牌曝光" name="type"></el-checkbox>
    </el-checkbox-group>
  </el-form-item>
  <el-form-item label="特殊资源">
    <el-radio-group v-model="form.resource">
      <el-radio label="线上品牌商赞助"></el-radio>
      <el-radio label="线下场地免费"></el-radio>
    </el-radio-group>
  </el-form-item>
  <el-form-item label="活动形式">
    <el-input type="textarea" v-model="form.desc"></el-input>
  </el-form-item>
  <el-form-item>
    <el-button type="primary" @click="onSubmit">立即创建</el-button>
    <el-button>取消</el-button>
  </el-form-item>
</el-form>
<script>
  export default {
    data() {
      return {
        form: {
          name: '',
```

```
          region: '',
          date1: '',
          date2: '',
          delivery: false,
          type: [],
          resource: '',
          desc: ''
        }
      }
    },
    methods: {
      onSubmit() {
        console.log('submit!');
      }
    }
  }
</script>
```

W3C 标准中有如下规定：

> *When there is only one single-line text input field in a form, the user agent should accept Enter in that field as a request to submit the form.*

即：当一个 form 元素中只有一个输入框时，在该输入框中按下回车应提交该表单。如果希望阻止这一默认行为，可以在 `<el-form>` 标签上添加 `@submit.native.prevent` 。

行内表单

当垂直方向空间受限且表单较简单时，可以在一行内放置表单。

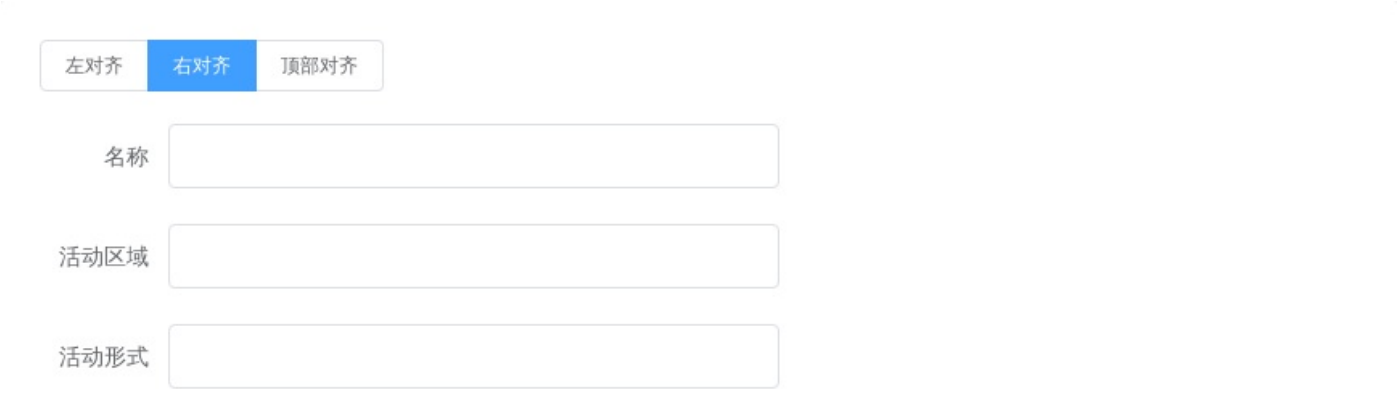
设置 `inline` 属性可以让表单域变为行内的表单域

```
<el-form :inline="true" :model="formInline" class="demo-form-inline">
  <el-form-item label="审批人">
    <el-input v-model="formInline.user" placeholder="审批人"></el-input>
  </el-form-item>
  <el-form-item label="活动区域">
    <el-select v-model="formInline.region" placeholder="活动区域">
```

```
        <el-option label="区域一" value="shanghai"></el-option>
        <el-option label="区域二" value="beijing"></el-option>
      </el-select>
    </el-form-item>
    <el-form-item>
      <el-button type="primary" @click="onSubmit">查询</el-button>
    </el-form-item>
  </el-form>
  <script>
    export default {
      data() {
        return {
          formInline: {
            user: '',
            region: ''
          }
        }
      },
      methods: {
        onSubmit() {
          console.log('submit!');
        }
      }
    }
  </script>
```

对齐方式

根据具体目标和制约因素，选择最佳的标签对齐方式。



通过设置 `label-position` 属性可以改变表单域标签的位置，可选值为 `top` 、 `left` ，当设

为 `top` 时标签会置于表单域的顶部

```
<el-radio-group v-model="labelPosition" size="small">
  <el-radio-button label="left">左对齐</el-radio-button>
  <el-radio-button label="right">右对齐</el-radio-button>
  <el-radio-button label="top">顶部对齐</el-radio-button>
</el-radio-group>
<div style="margin: 20px;"></div>
<el-form :label-position="labelPosition" label-width="80px"
:model="formLabelAlign">
  <el-form-item label="名称">
    <el-input v-model="formLabelAlign.name"></el-input>
  </el-form-item>
  <el-form-item label="活动区域">
    <el-input v-model="formLabelAlign.region"></el-input>
  </el-form-item>
  <el-form-item label="活动形式">
    <el-input v-model="formLabelAlign.type"></el-input>
  </el-form-item>
</el-form>
<script>
  export default {
    data() {
      return {
        labelPosition: 'right',
        formLabelAlign: {
          name: '',
          region: '',
          type: ''
        }
      };
    }
  }
</script>
```

表单验证

在防止用户犯错的前提下，尽可能让用户更早地发现并纠正错误。

Form 组件提供了表单验证的功能，只需要通过 `rules` 属性传入约定的验证规则，并将 Form-Item 的 `prop` 属性设置为需校验的字段名即可。校验规则参见 async-validator

```
    <el-form :model="ruleForm" :rules="rules" ref="ruleForm" label-width="100px"
1.  class="demo-ruleForm">
2.    <el-form-item label="活动名称" prop="name">
3.      <el-input v-model="ruleForm.name"></el-input>
4.    </el-form-item>
5.    <el-form-item label="活动区域" prop="region">
6.      <el-select v-model="ruleForm.region" placeholder="请选择活动区域">
7.        <el-option label="区域一" value="shanghai"></el-option>
8.        <el-option label="区域二" value="beijing"></el-option>
9.      </el-select>
10.   </el-form-item>
11.   <el-form-item label="活动时间" required>
12.     <el-col :span="11">
13.       <el-form-item prop="date1">
            <el-date-picker type="date" placeholder="选择日期" v-
14. model="ruleForm.date1" style="width: 100%;"></el-date-picker>
15.       </el-form-item>
16.     </el-col>
```

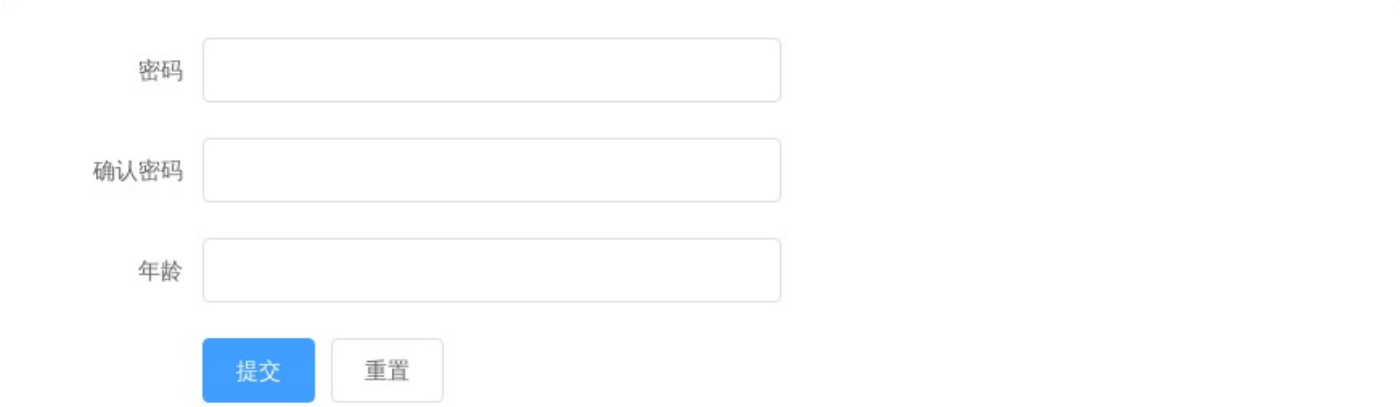
```
      <el-col class="line" :span="2">-</el-col>
      <el-col :span="11">
        <el-form-item prop="date2">
          <el-time-picker placeholder="选择时间" v-model="ruleForm.date2"
style="width: 100%;"></el-time-picker>
        </el-form-item>
      </el-col>
    </el-form-item>
    <el-form-item label="即时配送" prop="delivery">
      <el-switch v-model="ruleForm.delivery"></el-switch>
    </el-form-item>
    <el-form-item label="活动性质" prop="type">
      <el-checkbox-group v-model="ruleForm.type">
        <el-checkbox label="美食/餐厅线上活动" name="type"></el-checkbox>
        <el-checkbox label="地推活动" name="type"></el-checkbox>
        <el-checkbox label="线下主题活动" name="type"></el-checkbox>
        <el-checkbox label="单纯品牌曝光" name="type"></el-checkbox>
      </el-checkbox-group>
    </el-form-item>
    <el-form-item label="特殊资源" prop="resource">
      <el-radio-group v-model="ruleForm.resource">
        <el-radio label="线上品牌商赞助"></el-radio>
        <el-radio label="线下场地免费"></el-radio>
      </el-radio-group>
    </el-form-item>
    <el-form-item label="活动形式" prop="desc">
      <el-input type="textarea" v-model="ruleForm.desc"></el-input>
    </el-form-item>
    <el-form-item>
      <el-button type="primary" @click="submitForm('ruleForm')">立即创建</el-
button>
      <el-button @click="resetForm('ruleForm')">重置</el-button>
    </el-form-item>
</el-form>
<script>
  export default {
    data() {
      return {
        ruleForm: {
          name: '',
          region: '',
          date1: '',
          date2: '',
```

```
58.           delivery: false,
59.           type: [],
60.           resource: '',
61.           desc: ''
62.         },
63.         rules: {
64.           name: [
65.             { required: true, message: '请输入活动名称', trigger: 'blur' },
66.             { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
67.           ],
68.           region: [
69.             { required: true, message: '请选择活动区域', trigger: 'change' }
70.           ],
71.           date1: [
                { type: 'date', required: true, message: '请选择日期', trigger:
72. 'change' }
73.           ],
74.           date2: [
                { type: 'date', required: true, message: '请选择时间', trigger:
75. 'change' }
76.           ],
77.           type: [
                { type: 'array', required: true, message: '请至少选择一个活动性质',
78. trigger: 'change' }
79.           ],
80.           resource: [
81.             { required: true, message: '请选择活动资源', trigger: 'change' }
82.           ],
83.           desc: [
84.             { required: true, message: '请填写活动形式', trigger: 'blur' }
85.           ]
86.         }
87.       };
88.     },
89.     methods: {
90.       submitForm(formName) {
91.         this.$refs[formName].validate((valid) => {
92.           if (valid) {
93.             alert('submit!');
94.           } else {
95.             console.log('error submit!!');
96.             return false;
97.           }
```

```
        });
      },
      resetForm(formName) {
        this.$refs[formName].resetFields();
      }
    }
  }
</script>
```

自定义校验规则

这个例子中展示了如何使用自定义验证规则来完成密码的二次验证。



本例还使用 `status-icon` 属性为输入框添加了表示校验结果的反馈图标。

```
<el-form :model="ruleForm" status-icon :rules="rules" ref="ruleForm" label-width="100px" class="demo-ruleForm">
  <el-form-item label="密码" prop="pass">
    <el-input type="password" v-model="ruleForm.pass" autocomplete="off"></el-input>
  </el-form-item>
  <el-form-item label="确认密码" prop="checkPass">
    <el-input type="password" v-model="ruleForm.checkPass" autocomplete="off"></el-input>
  </el-form-item>
  <el-form-item label="年龄" prop="age">
    <el-input v-model.number="ruleForm.age"></el-input>
  </el-form-item>
  <el-form-item>
    <el-button type="primary" @click="submitForm('ruleForm')">提交</el-button>
    <el-button @click="resetForm('ruleForm')">重置</el-button>
```

```
  </el-form-item>
</el-form>
<script>
  export default {
    data() {
      var checkAge = (rule, value, callback) => {
        if (!value) {
          return callback(new Error('年龄不能为空'));
        }
        setTimeout(() => {
          if (!Number.isInteger(value)) {
            callback(new Error('请输入数字值'));
          } else {
            if (value < 18) {
              callback(new Error('必须年满18岁'));
            } else {
              callback();
            }
          }
        }, 1000);
      };
      var validatePass = (rule, value, callback) => {
        if (value === '') {
          callback(new Error('请输入密码'));
        } else {
          if (this.ruleForm.checkPass !== '') {
            this.$refs.ruleForm.validateField('checkPass');
          }
          callback();
        }
      };
      var validatePass2 = (rule, value, callback) => {
        if (value === '') {
          callback(new Error('请再次输入密码'));
        } else if (value !== this.ruleForm.pass) {
          callback(new Error('两次输入密码不一致!'));
        } else {
          callback();
        }
      };
      return {
        ruleForm: {
```

```
          pass: '',
          checkPass: '',
          age: ''
        },
        rules: {
          pass: [
            { validator: validatePass, trigger: 'blur' }
          ],
          checkPass: [
            { validator: validatePass2, trigger: 'blur' }
          ],
          age: [
            { validator: checkAge, trigger: 'blur' }
          ]
        }
      };
    },
    methods: {
      submitForm(formName) {
        this.$refs[formName].validate((valid) => {
          if (valid) {
            alert('submit!');
          } else {
            console.log('error submit!!');
            return false;
          }
        });
      },
      resetForm(formName) {
        this.$refs[formName].resetFields();
      }
    }
  }
</script>
```

自定义校验 `callback` 必须被调用。 更多高级用法可参考 `async-validator`。

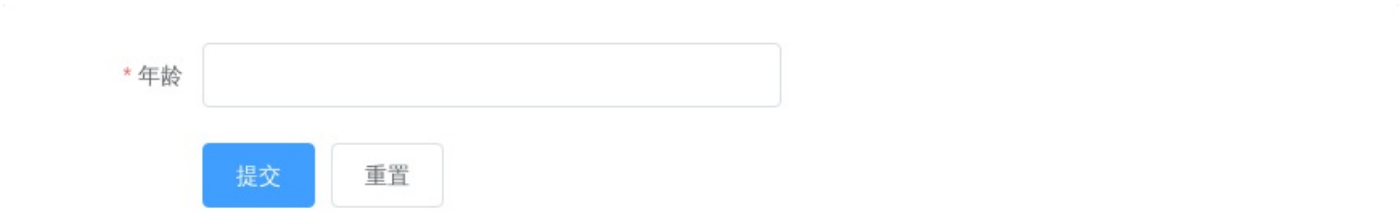
动态增减表单项

除了在 Form 组件上一次性传递所有的验证规则外还可以在单个的表单域上传递属性的验证规则

```
<el-form :model="dynamicValidateForm" ref="dynamicValidateForm" label-
width="100px" class="demo-dynamic">
  <el-form-item
    prop="email"
    label="邮箱"
    :rules="[
      { required: true, message: '请输入邮箱地址', trigger: 'blur' },
      { type: 'email', message: '请输入正确的邮箱地址', trigger: ['blur',
'change'] }
    ]"
  >
    <el-input v-model="dynamicValidateForm.email"></el-input>
  </el-form-item>
  <el-form-item
    v-for="(domain, index) in dynamicValidateForm.domains"
    :label="'域名' + index"
    :key="domain.key"
    :prop="'domains.' + index + '.value'"
    :rules="{
      required: true, message: '域名不能为空', trigger: 'blur'
    }"
  >
    <el-input v-model="domain.value"></el-input><el-button
@click.prevent="removeDomain(domain)">删除</el-button>
  </el-form-item>
  <el-form-item>
    <el-button type="primary" @click="submitForm('dynamicValidateForm')">提交
</el-button>
    <el-button @click="addDomain">新增域名</el-button>
    <el-button @click="resetForm('dynamicValidateForm')">重置</el-button>
  </el-form-item>
</el-form>
```

```
<script>
  export default {
    data() {
      return {
        dynamicValidateForm: {
          domains: [{
            value: ''
          }],
          email: ''
        }
      };
    },
    methods: {
      submitForm(formName) {
        this.$refs[formName].validate((valid) => {
          if (valid) {
            alert('submit!');
          } else {
            console.log('error submit!!');
            return false;
          }
        });
      },
      resetForm(formName) {
        this.$refs[formName].resetFields();
      },
      removeDomain(item) {
        var index = this.dynamicValidateForm.domains.indexOf(item)
        if (index !== -1) {
          this.dynamicValidateForm.domains.splice(index, 1)
        }
      },
      addDomain() {
        this.dynamicValidateForm.domains.push({
          value: '',
          key: Date.now()
        });
      }
    }
  }
</script>
```

数字类型验证



数字类型的验证需要在 `v-model` 处加上 `.number` 的修饰符，这是 `Vue` 自身提供的用于将绑定值转化为 `number` 类型的修饰符。

```
    <el-form :model="numberValidateForm" ref="numberValidateForm" label-
1.  width="100px" class="demo-ruleForm">
2.    <el-form-item
3.      label="年龄"
4.      prop="age"
5.      :rules="[
6.        { required: true, message: '年龄不能为空'},
7.        { type: 'number', message: '年龄必须为数字值'}
8.      ]"
9.    >
        <el-input type="age" v-model.number="numberValidateForm.age"
10. autocomplete="off"></el-input>
11.   </el-form-item>
12.   <el-form-item>
        <el-button type="primary" @click="submitForm('numberValidateForm')">提交
13. </el-button>
14.     <el-button @click="resetForm('numberValidateForm')">重置</el-button>
15.   </el-form-item>
16. </el-form>
17. <script>
18.   export default {
19.     data() {
20.       return {
21.         numberValidateForm: {
22.           age: ''
23.         }
24.       };
25.     },
26.     methods: {
27.       submitForm(formName) {
28.         this.$refs[formName].validate((valid) => {
```

```
29.           if (valid) {
30.             alert('submit!');
31.           } else {
32.             console.log('error submit!!');
33.             return false;
34.           }
35.         });
36.       },
37.       resetForm(formName) {
38.         this.$refs[formName].resetFields();
39.       }
40.     }
41.   }
42. </script>
```

嵌套在 `el-form-item` 中的 `el-form-item` 标签宽度默认为零，不会继承 `el-form` 的 `label-width`。如果需要可以为其单独设置 `label-width` 属性。

表单内组件尺寸控制

通过设置 Form 上的 `size` 属性可以使该表单内所有可调节大小的组件继承该尺寸。Form-Item 也具有该属性。

如果希望某个表单项或某个表单组件的尺寸不同于 Form 上的 `size` 属性，直接为这个表单项或表单组件设置自己的 `size` 即可。

```
1. <el-form ref="form" :model="sizeForm" label-width="80px" size="mini">
2.   <el-form-item label="活动名称">
3.     <el-input v-model="sizeForm.name"></el-input>
```

```
  </el-form-item>
  <el-form-item label="活动区域">
    <el-select v-model="sizeForm.region" placeholder="请选择活动区域">
      <el-option label="区域一" value="shanghai"></el-option>
      <el-option label="区域二" value="beijing"></el-option>
    </el-select>
  </el-form-item>
  <el-form-item label="活动时间">
    <el-col :span="11">
      <el-date-picker type="date" placeholder="选择日期" v-
model="sizeForm.date1" style="width: 100%;"></el-date-picker>
    </el-col>
    <el-col class="line" :span="2">-</el-col>
    <el-col :span="11">
      <el-time-picker placeholder="选择时间" v-model="sizeForm.date2"
style="width: 100%;"></el-time-picker>
    </el-col>
  </el-form-item>
  <el-form-item label="活动性质">
    <el-checkbox-group v-model="sizeForm.type">
      <el-checkbox-button label="美食/餐厅线上活动" name="type"></el-checkbox-
button>
      <el-checkbox-button label="地推活动" name="type"></el-checkbox-button>
      <el-checkbox-button label="线下主题活动" name="type"></el-checkbox-button>
    </el-checkbox-group>
  </el-form-item>
  <el-form-item label="特殊资源">
    <el-radio-group v-model="sizeForm.resource" size="medium">
      <el-radio border label="线上品牌商赞助"></el-radio>
      <el-radio border label="线下场地免费"></el-radio>
    </el-radio-group>
  </el-form-item>
  <el-form-item size="large">
    <el-button type="primary" @click="onSubmit">立即创建</el-button>
    <el-button>取消</el-button>
  </el-form-item>
</el-form>

<script>
  export default {
    data() {
      return {
        sizeForm: {
```

```
            name: '',
            region: '',
            date1: '',
            date2: '',
            delivery: false,
            type: [],
            resource: '',
            desc: ''
          }
        };
      },
      methods: {
        onSubmit() {
          console.log('submit!');
        }
      }
    };
  </script>
```

Form Attributes

参数	说明	类型	可选值	默认值
model	表单数据对象	object	—	—
rules	表单验证规则	object	—	—
inline	行内表单模式	boolean	—	false
label-position	表单域标签的位置，如果值为 left 或者 right 时，则需要设置 `label-width`	string	right/left/top	right
label-width	表单域标签的宽度，例如 ‘50px’。作为 Form 直接子元素的 form-item 会继承该值。支持 `auto` 。	string	—	—
label-suffix	表单域标签的后缀	string	—	—
hide-required-asterisk	是否显示必填字段的标签旁边的红色星号	boolean	—	false
show-message	是否显示校验错误信息	boolean	—	true
inline-message	是否以行内形式展示校验信息	boolean	—	false
status-icon	是否在输入框中显示校验结果反馈图标	boolean	—	false
validate-on-rule-change	是否在 `rules` 属性改变后立即触发一次验证	boolean	—	true

size	用于控制该表单内组件的尺寸	string	medium / small / mini	—
disabled	是否禁用该表单内的所有组件。若设置为 true，则表单内组件上的 disabled 属性不再生效	boolean	—	false

Form Methods

方法名	说明	参数
validate	对整个表单进行校验的方法，参数为一个回调函数。该回调函数会在校验结束后被调用，并传入两个参数：是否校验成功和未通过校验的字段。若不传入回调函数，则会返回一个 promise	Function(callback: Function(boolean, object))
validateField	对部分表单字段进行校验的方法	Function(props: array | string, callback: Function(errorMessage: string))
resetFields	对整个表单进行重置，将所有字段值重置为初始值并移除校验结果	—
clearValidate	移除表单项的校验结果。传入待移除的表单项的 prop 属性或者 prop 组成的数组，如不传则移除整个表单的校验结果	Function(props: array | string)

Form Events

事件名称	说明	回调参数
validate	任一表单项被校验后触发	被校验的表单项 prop 值，校验是否通过，错误消息（如果存在）

Form-Item Attributes

参数	说明	类型	可选值	默认值
prop	表单域 model 字段，在使用 validate、resetFields 方法的情况下，该属性是必填的	string	传入 Form 组件的 `model` 中的字段	—
label	标签文本	string	—	—
label-width	表单域标签的的宽度，例如 ‘50px’。支持 `auto` 。	string	—	—
required	是否必填，如不设置，则会根据校验规则自动生成	boolean	—	false
rules	表单验证规则	object	—	—
error	表单域验证错误信息，设置该值会使表单验证状态变为 `error` ，并显示该错误信息	string	—	—
show-message	是否显示校验错误信息	boolean	—	true
inline-message	以行内形式展示校验信息	boolean	—	false
size	用于控制该表单域下组件的尺寸	string	medium / small / mini	-

Form-Item Slot

name	说明
—	Form Item 的内容
label	标签文本的内容

Form-Item Scoped Slot

name	说明
error	自定义表单校验信息的显示方式，参数为 { error }

Form-Item Methods

方法名	说明	参数
resetField	对该表单项进行重置，将其值重置为初始值并移除校验结果	-
clearValidate	移除该表单项的校验结果	-

- Table 表格
- Tag 标签
- Progress 进度条
- Tree 树形控件
- Pagination 分页
- Badge 标记
- Avatar 头像

Table 表格

用于展示多条结构类似的数据，可对数据进行排序、筛选、对比或其他自定义操作。

基础表格

基础的表格展示用法。

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1517 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1516 弄

当 `el-table` 元素中注入 `data` 对象数组后，在 `el-table-column` 中用 `prop` 属性来对应对象中的键名即可填入数据，用 `label` 属性来定义表格的列名。可以使用 `width` 属性来定义列宽。

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

```
  </template>

  <script>
    export default {
      data() {
        return {
          tableData: [{
            date: '2016-05-02',
            name: '王小虎',
            address: '上海市普陀区金沙江路 1518 弄'
          }, {
            date: '2016-05-04',
            name: '王小虎',
            address: '上海市普陀区金沙江路 1517 弄'
          }, {
            date: '2016-05-01',
            name: '王小虎',
            address: '上海市普陀区金沙江路 1519 弄'
          }, {
            date: '2016-05-03',
            name: '王小虎',
            address: '上海市普陀区金沙江路 1516 弄'
          }]
        }
      }
    }
  </script>
```

带斑马纹表格

使用带斑马纹的表格，可以更容易区分出不同行的数据。

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1517 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1516 弄

`stripe` 属性可以创建带斑马纹的表格。它接受一个 `Boolean` ，默认为 `false` ，设置为 `true` 即为启用。

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

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1517 弄'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1519 弄'
        }, {
```

```
            date: '2016-05-03',
            name: '王小虎',
            address: '上海市普陀区金沙江路 1516 弄'
          }]
        }
      }
    }
</script>
```

带边框表格

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1517 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1516 弄

默认情况下，Table 组件是不具有竖直方向的边框的，如果需要，可以使用 `border` 属性，它接受一个 `Boolean` ，设置为 `true` 即可启用。

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

```
      label="地址">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1517 弄'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1519 弄'
        }, {
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1516 弄'
        }]
      }
    }
  }
</script>
```

带状态表格

可将表格内容 highlight 显示，方便区分「成功、信息、警告、危险」等内容。

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1518 弄

可以通过指定 Table 组件的 `row-class-name` 属性来为 Table 中的某一行添加 class，表明该行处于某种状态。

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%"
    :row-class-name="tableRowClassName">
    <el-table-column
      prop="date"
      label="日期"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址">
    </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: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄',
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄',
        }, {
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }]
      }
    }
  }
</script>
```

固定表头

纵向内容过多时，可选择固定表头。

日期	姓名	地址
2016-05-03	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1518 弄

只要在 `el-table` 元素中定义了 `height` 属性，即可实现固定表头的表格，而不需要额外的代码。

```
<template>
  <el-table
    :data="tableData"
    height="250"
    border
    style="width: 100%">
    <el-table-column
      prop="date"
      label="日期"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
```

```
        tableData: [{
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-08',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-06',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-07',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }]
      }
    }
  }
</script>
```

固定列

横向内容过多时，可选择固定列。

日期	姓名	省份	市区	地址	操作
2016-05-02	王小虎	上海	普陀区	上海市普陀区金沙江路 1518 弄	查看 编辑
2016-05-04	王小虎	上海	普陀区	上海市普陀区金沙江路 1517 弄	查看 编辑
2016-05-01	王小虎	上海	普陀区	上海市普陀区金沙江路 1519 弄	查看 编辑
2016-05-03	王小虎	上海	普陀区	上海市普陀区金沙江路 1516 弄	查看 编辑

固定列需要使用 `fixed` 属性，它接受 Boolean 值或者 `left` `right`，表示左边固定还是右边固定。

```
<template>
  <el-table
    :data="tableData"
    border
    style="width: 100%">
    <el-table-column
      fixed
      prop="date"
      label="日期"
      width="150">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      width="120">
    </el-table-column>
    <el-table-column
      prop="province"
      label="省份"
      width="120">
    </el-table-column>
    <el-table-column
      prop="city"
      label="市区"
      width="120">
    </el-table-column>
    <el-table-column
```

```
      prop="address"
      label="地址"
      width="300">
    </el-table-column>
    <el-table-column
      prop="zip"
      label="邮编"
      width="120">
    </el-table-column>
    <el-table-column
      fixed="right"
      label="操作"
      width="100">
      <template #default="scope">
        <el-button @click="handleClick(scope.row)" type="text" size="small">查
看</el-button>
        <el-button type="text" size="small">编辑</el-button>
      </template>
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    methods: {
      handleClick(row) {
        console.log(row);
      }
    },

    data() {
      return {
        tableData: [{
          date: '2016-05-02',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-04',
          name: '王小虎',
```

```
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1517 弄',
          zip: 200333
        }, {
          date: '2016-05-01',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1519 弄',
          zip: 200333
        }, {
          date: '2016-05-03',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1516 弄',
          zip: 200333
        }]
      }
    }
  }
</script>
```

固定列和表头

横纵内容过多时，可选择固定列和表头。

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

固定列和表头可以同时使用，只需要将上述两个属性分别设置好即可。

```
<template>
```

```
  <el-table
    :data="tableData"
    style="width: 100%"
    height="250">
    <el-table-column
      fixed
      prop="date"
      label="日期"
      width="150">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      width="120">
    </el-table-column>
    <el-table-column
      prop="province"
      label="省份"
      width="120">
    </el-table-column>
    <el-table-column
      prop="city"
      label="市区"
      width="120">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址"
      width="300">
    </el-table-column>
    <el-table-column
      prop="zip"
      label="邮编"
      width="120">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
```

```
        tableData: [{
          date: '2016-05-03',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-02',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-04',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-01',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-08',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-06',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
```

```
        }, {
          date: '2016-05-07',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }]
      }
    }
  }
</script>
```

流体高度

当数据量动态变化时，可以为 Table 设置一个最大高度。

通过设置 `max-height` 属性为 Table 指定最大高度。此时若表格所需的高度大于最大高度，则会显示一个滚动条。

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%"
    max-height="250">
    <el-table-column
      fixed
      prop="date"
      label="日期"
      width="150">
    </el-table-column>
```

```
12.     <el-table-column
13.       prop="name"
14.       label="姓名"
15.       width="120">
16.     </el-table-column>
17.     <el-table-column
18.       prop="province"
19.       label="省份"
20.       width="120">
21.     </el-table-column>
22.     <el-table-column
23.       prop="city"
24.       label="市区"
25.       width="120">
26.     </el-table-column>
27.     <el-table-column
28.       prop="address"
29.       label="地址"
30.       width="300">
31.     </el-table-column>
32.     <el-table-column
33.       prop="zip"
34.       label="邮编"
35.       width="120">
36.     </el-table-column>
37.     <el-table-column
38.       fixed="right"
39.       label="操作"
40.       width="120">
41.       <template #default="scope">
42.         <el-button
43.           @click.native.prevent="deleteRow(scope.$index, tableData)"
44.           type="text"
45.           size="small">
46.           移除
47.         </el-button>
48.       </template>
49.     </el-table-column>
50.   </el-table>
51. </template>
52.
53. <script>
```

```
54.   export default {
55.     methods: {
56.       deleteRow(index, rows) {
57.         rows.splice(index, 1);
58.       }
59.     },
60.     data() {
61.       return {
62.         tableData: [{
63.           date: '2016-05-03',
64.           name: '王小虎',
65.           province: '上海',
66.           city: '普陀区',
67.           address: '上海市普陀区金沙江路 1518 弄',
68.           zip: 200333
69.         }, {
70.           date: '2016-05-02',
71.           name: '王小虎',
72.           province: '上海',
73.           city: '普陀区',
74.           address: '上海市普陀区金沙江路 1518 弄',
75.           zip: 200333
76.         }, {
77.           date: '2016-05-04',
78.           name: '王小虎',
79.           province: '上海',
80.           city: '普陀区',
81.           address: '上海市普陀区金沙江路 1518 弄',
82.           zip: 200333
83.         }, {
84.           date: '2016-05-01',
85.           name: '王小虎',
86.           province: '上海',
87.           city: '普陀区',
88.           address: '上海市普陀区金沙江路 1518 弄',
89.           zip: 200333
90.         }, {
91.           date: '2016-05-08',
92.           name: '王小虎',
93.           province: '上海',
94.           city: '普陀区',
95.           address: '上海市普陀区金沙江路 1518 弄',
```

```
          zip: 200333
        }, {
          date: '2016-05-06',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-07',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }]
      }
    }
  }
</script>
```

多级表头

数据结构比较复杂的时候，可使用多级表头来展现数据的层次关系。

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

只需要在 el-table-column 里面嵌套 el-table-column，就可以实现多级表头。

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column
      prop="date"
      label="日期"
      width="150">
    </el-table-column>
    <el-table-column label="配送信息">
      <el-table-column
        prop="name"
        label="姓名"
        width="120">
      </el-table-column>
      <el-table-column label="地址">
        <el-table-column
          prop="province"
          label="省份"
          width="120">
        </el-table-column>
```

```
22.         <el-table-column
23.           prop="city"
24.           label="市区"
25.           width="120">
26.         </el-table-column>
27.         <el-table-column
28.           prop="address"
29.           label="地址"
30.           width="300">
31.         </el-table-column>
32.         <el-table-column
33.           prop="zip"
34.           label="邮编"
35.           width="120">
36.         </el-table-column>
37.       </el-table-column>
38.     </el-table-column>
39.   </el-table>
40. </template>
41.
42. <script>
43.   export default {
44.     data() {
45.       return {
46.         tableData: [{
47.           date: '2016-05-03',
48.           name: '王小虎',
49.           province: '上海',
50.           city: '普陀区',
51.           address: '上海市普陀区金沙江路 1518 弄',
52.           zip: 200333
53.         }, {
54.           date: '2016-05-02',
55.           name: '王小虎',
56.           province: '上海',
57.           city: '普陀区',
58.           address: '上海市普陀区金沙江路 1518 弄',
59.           zip: 200333
60.         }, {
61.           date: '2016-05-04',
62.           name: '王小虎',
63.           province: '上海',
```

```
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-01',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-08',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-06',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }, {
          date: '2016-05-07',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333
        }]
      }
    }
  }
</script>
```

单选

选择单行数据时使用色块表示。

Table 组件提供了单选的支持，只需要配置 `highlight-current-row` 属性即可实现单选。之后由 `current-change` 事件来管理选中时触发的事件，它会传入 `currentRow` ， `oldCurrentRow` 。如果需要显示索引，可以增加一列 `el-table-column` ，设置 `type` 属性为 `index` 即可显示从 1 开始的索引号。

```
<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="日期"
      width="120">
    </el-table-column>
    <el-table-column
      property="name"
      label="姓名"
      width="120">
    </el-table-column>
    <el-table-column
      property="address"
      label="地址">
```

```
25.     </el-table-column>
26.   </el-table>
27.   <div style="margin-top: 20px">
28.     <el-button @click="setCurrent(tableData[1])">选中第二行</el-button>
29.     <el-button @click="setCurrent()">取消选择</el-button>
30.   </div>
31. </template>
32.
33. <script>
34.   export default {
35.     data() {
36.       return {
37.         tableData: [{
38.           date: '2016-05-02',
39.           name: '王小虎',
40.           address: '上海市普陀区金沙江路 1518 弄'
41.         }, {
42.           date: '2016-05-04',
43.           name: '王小虎',
44.           address: '上海市普陀区金沙江路 1517 弄'
45.         }, {
46.           date: '2016-05-01',
47.           name: '王小虎',
48.           address: '上海市普陀区金沙江路 1519 弄'
49.         }, {
50.           date: '2016-05-03',
51.           name: '王小虎',
52.           address: '上海市普陀区金沙江路 1516 弄'
53.         }],
54.         currentRow: null
55.       }
56.     },
57.
58.     methods: {
59.       setCurrent(row) {
60.         this.$refs.singleTable.setCurrentRow(row);
61.       },
62.       handleCurrentChange(val) {
63.         this.currentRow = val;
64.       }
65.     }
66.   }
```

```
67.  </script>
```

多选

选择多行数据时使用 Checkbox。

实现多选非常简单：手动添加一个 `el-table-column`，设 `type` 属性为 `selection` 即可；默认情况下若内容过多会折行显示，若需要单行显示可以使用 `show-overflow-tooltip` 属性，它接受一个 `Boolean`，为 `true` 时多余的内容会在 hover 时以 tooltip 的形式显示出来。

```
1.  <template>
2.    <el-table
3.      ref="multipleTable"
4.      :data="tableData"
5.      tooltip-effect="dark"
6.      style="width: 100%"
7.      @selection-change="handleSelectionChange">
8.      <el-table-column
9.        type="selection"
10.       width="55">
11.     </el-table-column>
12.     <el-table-column
13.       label="日期"
14.       width="120">
```

```
15.       <template #default="scope">{{ scope.row.date }}</template>
16.     </el-table-column>
17.     <el-table-column
18.       prop="name"
19.       label="姓名"
20.       width="120">
21.     </el-table-column>
22.     <el-table-column
23.       prop="address"
24.       label="地址"
25.       show-overflow-tooltip>
26.     </el-table-column>
27.   </el-table>
28.   <div style="margin-top: 20px">
      <el-button @click="toggleSelection([tableData[1], tableData[2]])">切换第二、
29. 第三行的选中状态</el-button>
30.     <el-button @click="toggleSelection()">取消选择</el-button>
31.   </div>
32. </template>
33.
34. <script>
35.   export default {
36.     data() {
37.       return {
38.         tableData: [{
39.           date: '2016-05-03',
40.           name: '王小虎',
41.           address: '上海市普陀区金沙江路 1518 弄'
42.         }, {
43.           date: '2016-05-02',
44.           name: '王小虎',
45.           address: '上海市普陀区金沙江路 1518 弄'
46.         }, {
47.           date: '2016-05-04',
48.           name: '王小虎',
49.           address: '上海市普陀区金沙江路 1518 弄'
50.         }, {
51.           date: '2016-05-01',
52.           name: '王小虎',
53.           address: '上海市普陀区金沙江路 1518 弄'
54.         }, {
55.           date: '2016-05-08',
```

```
            name: '王小虎',
            address: '上海市普陀区金沙江路 1518 弄'
          }, {
            date: '2016-05-06',
            name: '王小虎',
            address: '上海市普陀区金沙江路 1518 弄'
          }, {
            date: '2016-05-07',
            name: '王小虎',
            address: '上海市普陀区金沙江路 1518 弄'
          }],
          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>
```

排序

对表格进行排序，可快速查找或对比数据。

日期	姓名	地址
2016-05-04	王小虎	上海市普陀区金沙江路 1517 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1516 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄

在列中设置 `sortable` 属性即可实现以该列为基准的排序，接受一个 `Boolean` ，默认为 `false` 。可以通过 Table 的 `default-sort` 属性设置默认的排序列和排序顺序。可以使用 `sort-method` 或者 `sort-by` 使用自定义的排序规则。如果需要后端排序，需将 `sortable` 设置为 `custom` ，同时在 Table 上监听 `sort-change` 事件，在事件回调中可以获取当前排序的字段名和排序顺序，从而向接口请求排序后的表格数据。在本例中，我们还使用了 `formatter` 属性，它用于格式化指定列的值，接受一个 `Function` ，会传入两个参数： `row` 和 `column` ，可以根据自己的需求进行处理。

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

```
    </el-table-column>
  </el-table>
</template>

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

筛选

对表格进行筛选，可快速查找到自己想看的数据。

在列中设置 `filters` `filter-method` 属性即可开启该列的筛选，filters 是一个数组， `filter-method` 是一个方法，它用于决定某些数据是否显示，会传入三个参数： `value` ， `row` 和 `column` 。

```
<template>
  <el-button @click="resetDateFilter">清除日期过滤器</el-button>
  <el-button @click="clearFilter">清除所有过滤器</el-button>
  <el-table
    row-key="date"
    ref="filterTable"
    :data="tableData"
    style="width: 100%">
    <el-table-column
      prop="date"
      label="日期"
      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="姓名"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址"
      :formatter="formatter">
    </el-table-column>
    <el-table-column
      prop="tag"
      label="标签"
      width="100"
      :filters="[{ text: '家', value: '家' }, { text: '公司', value: '公司' }]"
      :filter-method="filterTag"
      filter-placement="bottom-end">
      <template #default="scope">
        <el-tag
          :type="scope.row.tag === '家' ? 'primary' : 'success'"
          disable-transitions>{{scope.row.tag}}</el-tag>
      </template>
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄',
          tag: '家'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1517 弄',
          tag: '公司'
        }, {
          date: '2016-05-01',
```

```
          name: '王小虎',
          address: '上海市普陀区金沙江路 1519 弄',
          tag: '家'
        }, {
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1516 弄',
          tag: '公司'
        }]
      }
    },
    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>
```

自定义列模板

自定义列的显示内容，可组合其他组件使用。

通过 `Scoped slot` 可以获取到 row, column, $index 和 store（table 内部的状态管理）的数据，用法参考 demo。

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column
      label="日期"
      width="180">
      <template #default="scope">
        <i class="el-icon-time"></i>
        <span style="margin-left: 10px">{{ scope.row.date }}</span>
      </template>
    </el-table-column>
    <el-table-column
      label="姓名"
      width="180">
      <template #default="scope">
        <el-popover effect="light" trigger="hover" placement="top">
          <template #default>
            <p>姓名: {{ scope.row.name }}</p>
            <p>住址: {{ scope.row.address }}</p>
          </template>
          <template #trigger>
            <div class="name-wrapper">
              <el-tag size="medium">{{ scope.row.name }}</el-tag>
            </div>
          </template>
        </el-popover>
      </template>
```

```
    </el-table-column>
    <el-table-column label="操作">
      <template #default="scope">
        <el-button
          size="mini"
          @click="handleEdit(scope.$index, scope.row)">编辑</el-button>
        <el-button
          size="mini"
          type="danger"
          @click="handleDelete(scope.$index, scope.row)">删除</el-button>
      </template>
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1517 弄'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1519 弄'
        }, {
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1516 弄'
        }]
      }
    },
    methods: {
      handleEdit(index, row) {
        console.log(index, row);
      },
```

```
        handleDelete(index, row) {
          console.log(index, row);
        }
      }
    }
  </script>
```

展开行

当行内容过多并且不想显示横向滚动条时，可以使用 Table 展开行功能。

通过设置 type=”expand” 和 `Scoped slot` 可以开启展开行功能，`el-table-column` 的模板会被渲染成为展开行的内容，展开行可访问的属性与使用自定义列模板时的 `Scoped slot` 相同。

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%">
    <el-table-column type="expand">
      <template #default="props">
        <el-form label-position="left" inline class="demo-table-expand">
          <el-form-item label="商品名称">
            <span>{{ props.row.name }}</span>
          </el-form-item>
          <el-form-item label="所属店铺">
            <span>{{ props.row.shop }}</span>
          </el-form-item>
          <el-form-item label="商品 ID">
            <span>{{ props.row.id }}</span>
          </el-form-item>
```

```
          <el-form-item label="店铺 ID">
            <span>{{ props.row.shopId }}</span>
          </el-form-item>
          <el-form-item label="商品分类">
            <span>{{ props.row.category }}</span>
          </el-form-item>
          <el-form-item label="店铺地址">
            <span>{{ props.row.address }}</span>
          </el-form-item>
          <el-form-item label="商品描述">
            <span>{{ props.row.desc }}</span>
          </el-form-item>
        </el-form>
      </template>
    </el-table-column>
    <el-table-column
      label="商品 ID"
      prop="id">
    </el-table-column>
    <el-table-column
      label="商品名称"
      prop="name">
    </el-table-column>
    <el-table-column
      label="描述"
      prop="desc">
    </el-table-column>
  </el-table>
</template>

<style>
  .demo-table-expand {
    font-size: 0;
  }
  .demo-table-expand label {
    width: 90px;
    color: #99a9bf;
  }
  .demo-table-expand .el-form-item {
    margin-right: 0;
    margin-bottom: 0;
    width: 50%;
```

```
  }
</style>

<script>
  export default {
    data() {
      return {
        tableData: [{
          id: '12987122',
          name: '好滋好味鸡蛋仔',
          category: '江浙小吃、小吃零食',
          desc: '荷兰优质淡奶，奶香浓而不腻',
          address: '上海市普陀区真北路',
          shop: '王小虎夫妻店',
          shopId: '10333'
        }, {
          id: '12987123',
          name: '好滋好味鸡蛋仔',
          category: '江浙小吃、小吃零食',
          desc: '荷兰优质淡奶，奶香浓而不腻',
          address: '上海市普陀区真北路',
          shop: '王小虎夫妻店',
          shopId: '10333'
        }, {
          id: '12987125',
          name: '好滋好味鸡蛋仔',
          category: '江浙小吃、小吃零食',
          desc: '荷兰优质淡奶，奶香浓而不腻',
          address: '上海市普陀区真北路',
          shop: '王小虎夫妻店',
          shopId: '10333'
        }, {
          id: '12987126',
          name: '好滋好味鸡蛋仔',
          category: '江浙小吃、小吃零食',
          desc: '荷兰优质淡奶，奶香浓而不腻',
          address: '上海市普陀区真北路',
          shop: '王小虎夫妻店',
          shopId: '10333'
        }]
      }
    }
```

```
    }
</script>
```

树形数据与懒加载

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1517 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1516 弄

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1517 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1516 弄

支持树类型的数据的显示。当 row 中包含 `children` 字段时，被视为树形数据。渲染树形数据时，必须要指定 `row-key` 。支持子节点数据异步加载。设置 Table 的 `lazy` 属性为 true 与加载函数 `load` 。通过指定 row 中的 `hasChildren` 字段来指定哪些行是包含子节点。 `children` 与 `hasChildren` 都可以通过 `tree-props` 配置。

```
<template>
<div>
  <el-table
    :data="tableData"
    style="width: 100%;margin-bottom: 20px;"
    row-key="id"
    border
    default-expand-all
```

```
    :tree-props="{children: 'children', hasChildren: 'hasChildren'}">
    <el-table-column
      prop="date"
      label="日期"
      sortable
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      sortable
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址">
    </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="日期"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址">
    </el-table-column>
  </el-table>
```

```
51.  </div>
52.  </template>
53.  <script>
54.    export default {
55.      data() {
56.        return {
57.          tableData: [{
58.            id: 1,
59.            date: '2016-05-02',
60.            name: '王小虎',
61.            address: '上海市普陀区金沙江路 1518 弄'
62.          }, {
63.            id: 2,
64.            date: '2016-05-04',
65.            name: '王小虎',
66.            address: '上海市普陀区金沙江路 1517 弄'
67.          }, {
68.            id: 3,
69.            date: '2016-05-01',
70.            name: '王小虎',
71.            address: '上海市普陀区金沙江路 1519 弄',
72.            children: [{
73.                id: 31,
74.                date: '2016-05-01',
75.                name: '王小虎',
76.                address: '上海市普陀区金沙江路 1519 弄'
77.              }, {
78.                id: 32,
79.                date: '2016-05-01',
80.                name: '王小虎',
81.                address: '上海市普陀区金沙江路 1519 弄'
82.            }]
83.          }, {
84.            id: 4,
85.            date: '2016-05-03',
86.            name: '王小虎',
87.            address: '上海市普陀区金沙江路 1516 弄'
88.          }],
89.          tableData1: [{
90.            id: 1,
91.            date: '2016-05-02',
92.            name: '王小虎',
```

```
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          id: 2,
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1517 弄'
        }, {
          id: 3,
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1519 弄',
          hasChildren: true
        }, {
          id: 4,
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1516 弄'
        }]
      }
    },
    methods: {
      load(tree, treeNode, resolve) {
        setTimeout(() => {
          resolve([
            {
              id: 31,
              date: '2016-05-01',
              name: '王小虎',
              address: '上海市普陀区金沙江路 1519 弄'
            }, {
              id: 32,
              date: '2016-05-01',
              name: '王小虎',
              address: '上海市普陀区金沙江路 1519 弄'
            }
          ])
        }, 1000)
      }
    },
  }
</script>
```

自定义表头

表头支持自定义。

通过设置 `Scoped slot` 来自定义表头。

```
<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 #header #default="scope">
        <el-input
          v-model="search"
          size="mini"
          placeholder="输入关键字搜索"/>
      </template>
      <template #default="scope">
        <el-button
          size="mini"
          @click="handleEdit(scope.$index, scope.row)">Edit</el-button>
```

```
25.         <el-button
26.           size="mini"
27.           type="danger"
28.           @click="handleDelete(scope.$index, scope.row)">Delete</el-button>
29.       </template>
30.     </el-table-column>
31.   </el-table>
32. </template>
33.
34. <script>
35.   export default {
36.     data() {
37.       return {
38.         tableData: [{
39.           date: '2016-05-02',
40.           name: '王小虎',
41.           address: '上海市普陀区金沙江路 1518 弄'
42.         }, {
43.           date: '2016-05-04',
44.           name: '王小虎',
45.           address: '上海市普陀区金沙江路 1517 弄'
46.         }, {
47.           date: '2016-05-01',
48.           name: '王小虎',
49.           address: '上海市普陀区金沙江路 1519 弄'
50.         }, {
51.           date: '2016-05-03',
52.           name: '王小虎',
53.           address: '上海市普陀区金沙江路 1516 弄'
54.         }],
55.         search: ''
56.       }
57.     },
58.     methods: {
59.       handleEdit(index, row) {
60.         console.log(index, row);
61.       },
62.       handleDelete(index, row) {
63.         console.log(index, row);
64.       }
65.     },
66.   }
```

```
</script>
```

表尾合计行

若表格展示的是各类数字，可以在表尾显示各列的合计。

ID	姓名	数值 1	数值 2	数值 3
12987122	王小虎	234	3.2	10
12987123	王小虎	165	4.43	12
12987124	王小虎	324	1.9	9
12987125	王小虎	621	2.2	17
12987126	王小虎	539	4.1	15
合计		1883	15.83	63

ID	姓名	数值 1（元）	数值 2（元）	数值 3（元）
12987122	王小虎	234	3.2	10
12987123	王小虎	165	4.43	12
总价	N/A	1883 元	15.83 元	63 元

将 `show-summary` 设置为 `true` 就会在表格尾部展示合计行。默认情况下，对于合计行，第一列不进行数据求合操作，而是显示「合计」二字（可通过 `sum-text` 配置），其余列会将本列所有数值进行求合操作，并显示出来。当然，你也可以定义自己的合计逻辑。使用 `summary-method` 并传入一个方法，返回一个数组，这个数组中的各项就会显示在合计行的各列中，具体可以参考本例中的第二个表格。

```
<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="姓名">
    </el-table-column>
    <el-table-column
      prop="amount1"
      sortable
      label="数值 1">
    </el-table-column>
    <el-table-column
      prop="amount2"
      sortable
      label="数值 2">
    </el-table-column>
    <el-table-column
      prop="amount3"
      sortable
      label="数值 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="姓名">
    </el-table-column>
    <el-table-column
      prop="amount1"
```

```
51.       label="数值 1（元）">
52.     </el-table-column>
53.     <el-table-column
54.       prop="amount2"
55.       label="数值 2（元）">
56.     </el-table-column>
57.     <el-table-column
58.       prop="amount3"
59.       label="数值 3（元）">
60.     </el-table-column>
61.   </el-table>
62. </template>
63.
64. <script>
65.   export default {
66.     data() {
67.       return {
68.         tableData: [{
69.           id: '12987122',
70.           name: '王小虎',
71.           amount1: '234',
72.           amount2: '3.2',
73.           amount3: 10
74.         }, {
75.           id: '12987123',
76.           name: '王小虎',
77.           amount1: '165',
78.           amount2: '4.43',
79.           amount3: 12
80.         }, {
81.           id: '12987124',
82.           name: '王小虎',
83.           amount1: '324',
84.           amount2: '1.9',
85.           amount3: 9
86.         }, {
87.           id: '12987125',
88.           name: '王小虎',
89.           amount1: '621',
90.           amount2: '2.2',
91.           amount3: 17
92.         }, {
```

```
          id: '12987126',
          name: '王小虎',
          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] = '总价';
            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);
            sums[index] += ' 元';
          } else {
            sums[index] = 'N/A';
          }
        });

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

合并行或列

多行或多列共用一个数据时，可以合并行或列。

ID	姓名	数值 1	数值 2	数值 3
12987122		234	3.2	10
12987123	王小虎	165	4.43	12
12987124		324	1.9	9
12987125	王小虎	621	2.2	17
12987126		539	4.1	15

ID	姓名	数值 1（元）	数值 2（元）	数值 3（元）
12987122	王小虎	234	3.2	10
	王小虎	165	4.43	12
12987124	王小虎	324	1.9	9
	王小虎	621	2.2	17
12987126	王小虎	539	4.1	15

通过给 `table` 传入 `span-method` 方法可以实现合并行或列，方法的参数是一个对象，里面包含当前行 `row` 、当前列 `column` 、当前行号 `rowIndex` 、当前列号 `columnIndex` 四个属性。该函数可以返回一个包含两个元素的数组，第一个元素代表 `rowspan` ，第二个元素代表 `colspan` 。也可以返回一个键名为 `rowspan` 和 `colspan` 的对象。

```
<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"
```

```
15.         label="姓名">
16.       </el-table-column>
17.       <el-table-column
18.         prop="amount1"
19.         sortable
20.         label="数值 1">
21.       </el-table-column>
22.       <el-table-column
23.         prop="amount2"
24.         sortable
25.         label="数值 2">
26.       </el-table-column>
27.       <el-table-column
28.         prop="amount3"
29.         sortable
30.         label="数值 3">
31.       </el-table-column>
32.     </el-table>
33.
34.     <el-table
35.       :data="tableData"
36.       :span-method="objectSpanMethod"
37.       border
38.       style="width: 100%; margin-top: 20px">
39.       <el-table-column
40.         prop="id"
41.         label="ID"
42.         width="180">
43.       </el-table-column>
44.       <el-table-column
45.         prop="name"
46.         label="姓名">
47.       </el-table-column>
48.       <el-table-column
49.         prop="amount1"
50.         label="数值 1（元）">
51.       </el-table-column>
52.       <el-table-column
53.         prop="amount2"
54.         label="数值 2（元）">
55.       </el-table-column>
56.       <el-table-column
```

```
57.         prop="amount3"
58.         label="数值 3（元）">
59.       </el-table-column>
60.     </el-table>
61.   </div>
62. </template>
63.
64. <script>
65.   export default {
66.     data() {
67.       return {
68.         tableData: [{
69.           id: '12987122',
70.           name: '王小虎',
71.           amount1: '234',
72.           amount2: '3.2',
73.           amount3: 10
74.         }, {
75.           id: '12987123',
76.           name: '王小虎',
77.           amount1: '165',
78.           amount2: '4.43',
79.           amount3: 12
80.         }, {
81.           id: '12987124',
82.           name: '王小虎',
83.           amount1: '324',
84.           amount2: '1.9',
85.           amount3: 9
86.         }, {
87.           id: '12987125',
88.           name: '王小虎',
89.           amount1: '621',
90.           amount2: '2.2',
91.           amount3: 17
92.         }, {
93.           id: '12987126',
94.           name: '王小虎',
95.           amount1: '539',
96.           amount2: '4.1',
97.           amount3: 15
98.         }]
```

```
      };
    },
    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>
```

自定义索引

自定义 `type=index` 列的行号。

	日期	姓名	地址
0	2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2	2016-05-04	王小虎	上海市普陀区金沙江路 1517 弄
4	2016-05-01	王小虎	上海市普陀区金沙江路 1519 弄
6	2016-05-03	王小虎	上海市普陀区金沙江路 1516 弄

通过给 `type=index` 的列传入 `index` 属性，可以自定义索引。该属性传入数字时，将作为索引的起始值。也可以传入一个方法，它提供当前行的行号（从 `0` 开始）作为参数，返回值将作为索引展示。

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

<script>
  export default {
```

```
    data() {
      return {
        tableData: [{
          date: '2016-05-02',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1518 弄',
          zip: 200333,
          tag: '家'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1517 弄',
          zip: 200333,
          tag: '公司'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1519 弄',
          zip: 200333,
          tag: '家'
        }, {
          date: '2016-05-03',
          name: '王小虎',
          province: '上海',
          city: '普陀区',
          address: '上海市普陀区金沙江路 1516 弄',
          zip: 200333,
          tag: '公司'
        }],
      }
    },
    methods: {
      indexMethod(index) {
        return index * 2;
      }
    }
```

```
70.    };
71.  </script>
```

Table Attributes

参数	说明	类型	可选值	默认值
data	显示的数据	array	—	—
height	Table 的高度，默认为自动高度。如果 height 为 number 类型，单位 px；如果 height 为 string 类型，则这个高度会设置为 Table 的 style.height 的值，Table 的高度会受控于外部样式。	string/number	—	—
max-height	Table 的最大高度。合法的值为数字或者单位为 px 的高度。	string/number	—	—
stripe	是否为斑马纹 table	boolean	—	false
border	是否带有纵向边框	boolean	—	false
size	Table 的尺寸	string	medium / small / mini	—
fit	列的宽度是否自撑开	boolean	—	true
show-header	是否显示表头	boolean	—	true
highlight-current-row	是否要高亮当前行	boolean	—	false
current-row-key	当前行的 key，只写属性	String,Number	—	—
row-class-name	行的 className 的回调方法，也可以使用字符串为所有行设置一个固定的 className。	Function({row, rowIndex})/String	—	—
row-style	行的 style 的回调方法，也可以使用一个固定的 Object 为所有行设置一样的 Style。	Function({row, rowIndex})/Object	—	—
cell-class-name	单元格的 className 的回调方法，也可以使用字符串为所有单元格设置一个固定的 className。	Function({row, column, rowIndex, columnIndex})/String	—	—
cell-style	单元格的 style 的回调方法，也可以使用一个固定的 Object 为所有单元格设置一样的 Style。	Function({row, column, rowIndex, columnIndex})/Object	—	—

header-row-class-name	表头行的 className 的回调方法，也可以使用字符串为所有表头行设置一个固定的 className。	Function({row, rowIndex})/String	—	—
header-row-style	表头行的 style 的回调方法，也可以使用一个固定的 Object 为所有表头行设置一样的 Style。	Function({row, rowIndex})/Object	—	—
header-cell-class-name	表头单元格的 className 的回调方法，也可以使用字符串为所有表头单元格设置一个固定的 className。	Function({row, column, rowIndex, columnIndex})/String	—	—
header-cell-style	表头单元格的 style 的回调方法，也可以使用一个固定的 Object 为所有表头单元格设置一样的 Style。	Function({row, column, rowIndex, columnIndex})/Object	—	—
row-key	行数据的 Key，用来优化 Table 的渲染；在使用 reserve-selection 功能与显示树形数据时，该属性是必填的。类型为 String 时，支持多层访问：`user.info.id`，但不支持 `user.info[0].id`，此种情况请使用 `Function`。	Function(row)/String	—	—
empty-text	空数据时显示的文本内容，也可以通过 `#empty` 设置	String	—	暂无数据
default-expand-all	是否默认展开所有行，当 Table 包含展开行存在或者为树形表格时有效	Boolean	—	false
expand-row-keys	可以通过该属性设置 Table 目前的展开行，需要设置 row-key 属性才能使用，该属性为展开行的 keys 数组。	Array	—	
default-sort	默认的排序列的 prop 和顺序。它的 `prop` 属性指定默认的排序的列，`order` 指定默认排序的顺序	Object	`order`：ascending, descending	如果只指定了 `prop`，没有指定 `order`，默认顺序是 ascending
tooltip-effect	tooltip `effect` 属性	String	dark/light	
show-summary	是否在表尾显示合计行	Boolean	—	false
sum-text	合计行第一列的文本	String	—	合计
summary-		Function({ columns,		

method		data })		
span-method	合并行或列的计算方法	Function({ row, column, rowIndex, columnIndex })	—	—
select-on-indeterminate	在多选表格中，当仅有部分行被选中时，点击表头的多选框时的行为。若为 true，则选中所有行；若为 false，则取消选择所有行	Boolean	—	true
indent	展示树形数据时，树节点的缩进	Number	—	16
lazy	是否懒加载子节点数据	Boolean	—	—
load	加载子节点数据的函数，lazy 为 true 时生效，函数第二个参数包含了节点的层级信息	Function(row, treeNode, resolve)	—	—
tree-props	渲染嵌套数据的配置选项	Object	—	{ hasChildre ‘hasChildren children: ‘children’ }

Table Events

事件名	说明	参数
select	当用户手动勾选数据行的 Checkbox 时触发的事件	selection, row
select-all	当用户手动勾选全选 Checkbox 时触发的事件	selection
selection-change	当选择项发生变化时会触发该事件	selection
cell-mouse-enter	当单元格 hover 进入时会触发该事件	row, column, cell, event
cell-mouse-leave	当单元格 hover 退出时会触发该事件	row, column, cell, event
cell-click	当某个单元格被点击时会触发该事件	row, column, cell, event
cell-dblclick	当某个单元格被双击击时会触发该事件	row, column, cell, event
row-click	当某一行被点击时会触发该事件	row, column, event
row-contextmenu	当某一行被鼠标右键点击时会触发该事件	row, column, event
row-dblclick	当某一行被双击时会触发该事件	row, column, event
header-click	当某一列的表头被点击时会触发该事件	column, event
header-	当某一列的表头被鼠标右键点击时触发该事件	column, event

contextmenu	当某一列的表头被鼠标右键点击时触发该事件	column, event
sort-change	当表格的排序条件发生变化的时候会触发该事件	{ column, prop, order }
filter-change	当表格的筛选条件发生变化的时候会触发该事件，参数的值是一个对象，对象的 key 是 column 的 columnKey，对应的 value 为用户选择的筛选条件的数组。	filters
current-change	当表格的当前行发生变化的时候会触发该事件，如果要高亮当前行，请打开表格的 highlight-current-row 属性	currentRow, oldCurrentRow
header-dragend	当拖动表头改变了列的宽度的时候会触发该事件	newWidth, oldWidth, column, event
expand-change	当用户对某一行展开或者关闭的时候会触发该事件（展开行时，回调的第二个参数为 expandedRows；树形表格时第二参数为 expanded）	row, (expandedRows | expanded)

Table Methods

方法名	说明	参数
clearSelection	用于多选表格，清空用户的选择	—
toggleRowSelection	用于多选表格，切换某一行的选中状态，如果使用了第二个参数，则是设置这一行选中与否（selected 为 true 则选中）	row, selected
toggleAllSelection	用于多选表格，切换所有行的选中状态	-
toggleRowExpansion	用于可展开表格与树形表格，切换某一行的展开状态，如果使用了第二个参数，则是设置这一行展开与否（expanded 为 true 则展开）	row, expanded
setCurrentRow	用于单选表格，设定某一行为选中行，如果调用时不加参数，则会取消目前高亮行的选中状态。	row
clearSort	用于清空排序条件，数据会恢复成未排序的状态	—
clearFilter	不传入参数时用于清空所有过滤条件，数据会恢复成未过滤的状态，也可传入由columnKey组成的数组以清除指定列的过滤条件	columnKey
doLayout	对 Table 进行重新布局。当 Table 或其祖先元素由隐藏切换为显示时，可能需要调用此方法	—
sort	手动对 Table 进行排序。参数 `prop` 属性指定排序列，`order` 指定排序顺序。	prop: string, order: string

Table Slot

name	说明
append	插入至表格最后一行之后的内容，如果需要对表格的内容进行无限滚动操作，可能需要用到这个 slot。若表格有合计行，该 slot 会位于合计行之上。

Table-column Attributes

参数	说明	类型	可选值

参数	说明	类型	可选值
type	对应列的类型。如果设置了 `selection` 则显示多选框；如果设置了 `index` 则显示该行的索引（从 1 开始计算）；如果设置了 `expand` 则显示为一个可展开的按钮	string	selection/index/expand
index	如果设置了 `type=index` ，可以通过传递 `index` 属性来自定义索引	number, Function(index)	-
column-key	column 的 key，如果需要使用 filter-change 事件，则需要此属性标识是哪个 column 的筛选条件	string	—
label	显示的标题	string	—
prop	对应列内容的字段名，也可以使用 property 属性	string	—
width	对应列的宽度	string	—
min-width	对应列的最小宽度，与 width 的区别是 width 是固定的，min-width 会把剩余宽度按比例分配给设置了 min-width 的列	string	—
fixed	列是否固定在左侧或者右侧，true 表示固定在左侧	string, boolean	true, left, right
render-header	列标题 Label 区域渲染使用的 Function	Function(h, { column, $index })	—
sortable	对应列是否可以排序，如果设置为‘custom’，则代表用户希望远程排序，需要监听 Table 的 sort-change 事件	boolean, string	true, false, ‘custom’
sort-method	对数据进行排序的时候使用的方法，仅当 sortable 设置为 true 的时候有效，需返回一个数字，和 Array.sort 表现一致	Function(a, b)	—

sort-by	属性进行排序，仅当 sortable 设置为 true 且没有设置 sort-method 的时候有效。如果 sort-by 为数组，则先按照第 1 个属性排序，如果第 1 个相等，再按照第 2 个排序，以此类推	String/Array/Function(row, index)	—
sort-orders	数据在排序时所使用排序策略的轮转顺序，仅当 sortable 为 true 时有效。需传入一个数组，随着用户点击表头，该列依次按照数组中元素的顺序进行排序	array	数组中的元素需为以下三者之一：`ascending` 表示升序，`descending` 表示降序，`null` 表示还原为原始顺序
resizable	对应列是否可以通过拖动改变宽度（需要在 el-table 上设置 border 属性为真）	boolean	—
formatter	用来格式化内容	Function(row, column, cellValue, index)	—
show-overflow-tooltip	当内容过长被隐藏时显示 tooltip	Boolean	—
align	对齐方式	String	left/center/right
header-align	表头对齐方式，若不设置该项，则使用表格的对齐方式	String	left/center/right
class-name	列的 className	string	—
label-class-name	当前列标题的自定义类名	string	—
selectable	仅对 type=selection 的列有效，类型为 Function，Function 的返回值用来决定这一行的 CheckBox 是否可以勾选	Function(row, index)	—
reserve-selection	仅对 type=selection 的列有效，类型为 Boolean，为 true 则会在数据更新之后保留之前选中的数据（需指定 `row-key`）	Boolean	—
	数据过滤的选项，数组格式，数组中		

filters	数组格式，数组中的元素需要有 text 和 value 属性。	Array[{ text, value }]	—
filter-placement	过滤弹出框的定位	String	与 Tooltip 的 placement 属性相同
filter-multiple	数据过滤的选项是否多选	Boolean	—
filter-method	数据过滤使用的方法，如果是多选的筛选项，对每一条数据会执行多次，任意一次返回 true 就会显示。	Function(value, row, column)	—
filtered-value	选中的数据过滤项，如果需要自定义表头过滤的渲染方式，可能会需要此属性。	Array	—

Table-column Scoped Slot

name	说明
—	自定义列的内容，参数为 { row, column, $index }
header	自定义表头的内容. 参数为 { column, $index }

Tag 标签

用于标记和选择。

基础用法

由 `type` 属性来选择tag的类型，也可以通过 `color` 属性来自定义背景色。

```
<el-tag>标签一</el-tag>
<el-tag type="success">标签二</el-tag>
<el-tag type="info">标签三</el-tag>
<el-tag type="warning">标签四</el-tag>
<el-tag type="danger">标签五</el-tag>
```

可移除标签

设置 `closable` 属性可以定义一个标签是否可移除。默认的标签移除时会附带渐变动画，如果不想使用，可以设置 `disable-transitions` 属性，它接受一个 `Boolean` ，`true` 为关闭。

```
<el-tag
  v-for="tag in tags"
  :key="tag.name"
  closable
  :type="tag.type">
  {{tag.name}}
</el-tag>

<script>
  export default {
    data() {
      return {
        tags: [
```

```
          { name: '标签一', type: '' },
          { name: '标签二', type: 'success' },
          { name: '标签三', type: 'info' },
          { name: '标签四', type: 'warning' },
          { name: '标签五', type: 'danger' }
        ]
      };
    }
  }
</script>
```

动态编辑标签

动态编辑标签可以通过点击标签关闭按钮后触发的 `close` 事件来实现


```
<el-tag
  :key="tag"
  v-for="tag in dynamicTags"
  closable
  :disable-transitions="false"
  @close="handleClose(tag)">
  {{tag}}
</el-tag>
<el-input
  class="input-new-tag"
  v-if="inputVisible"
  v-model="inputValue"
  ref="saveTagInput"
  size="small"
  @keyup.enter.native="handleInputConfirm"
  @blur="handleInputConfirm"
>
</el-input>
<el-button v-else class="button-new-tag" size="small" @click="showInput">+ New Tag</el-button>

<style>
  .el-tag + .el-tag {
```

```
    margin-left: 10px;
  }
  .button-new-tag {
    margin-left: 10px;
    height: 32px;
    line-height: 30px;
    padding-top: 0;
    padding-bottom: 0;
  }
  .input-new-tag {
    width: 90px;
    margin-left: 10px;
    vertical-align: bottom;
  }
</style>

<script>
  export default {
    data() {
      return {
        dynamicTags: ['标签一', '标签二', '标签三'],
        inputVisible: false,
        inputValue: ''
      };
    },
    methods: {
      handleClose(tag) {
        this.dynamicTags.splice(this.dynamicTags.indexOf(tag), 1);
      },

      showInput() {
        this.inputVisible = true;
        this.$nextTick(_ => {
          this.$refs.saveTagInput.$refs.input.focus();
        });
      },

      handleInputConfirm() {
        let inputValue = this.inputValue;
        if (inputValue) {
          this.dynamicTags.push(inputValue);
        }
```

```
65.         this.inputVisible = false;
66.         this.inputValue = '';
67.       }
68.     }
69.   }
70. </script>
```

不同尺寸

Tag 组件提供除了默认值以外的三种尺寸，可以在不同场景下选择合适的按钮尺寸。

额外的尺寸： `medium` 、 `small` 、 `mini` ，通过设置 `size` 属性来配置它们。

```
1. <el-tag closable>默认标签</el-tag>
2. <el-tag size="medium" closable>中等标签</el-tag>
3. <el-tag size="small" closable>小型标签</el-tag>
4. <el-tag size="mini" closable>超小标签</el-tag>
```

不同主题

Tag 组件提供了三个不同的主题： `dark` 、 `light` 和 `plain`

通过设置 `effect` 属性来改变主题，默认为 `light`

```
1. <div class="tag-group">
2.   <span class="tag-group__title">Dark</span>
3.   <el-tag
4.     v-for="item in items"
5.     :key="item.label"
6.     :type="item.type"
7.     effect="dark">
8.     {{ item.label }}
9.   </el-tag>
```

```
</div>
<div class="tag-group">
  <span class="tag-group__title">Plain</span>
  <el-tag
    v-for="item in items"
    :key="item.label"
    :type="item.type"
    effect="plain">
    {{ item.label }}
  </el-tag>
</div>

<script>
  export default {
    data() {
      return {
        items: [
          { type: '', label: '标签一' },
          { type: 'success', label: '标签二' },
          { type: 'info', label: '标签三' },
          { type: 'danger', label: '标签四' },
          { type: 'warning', label: '标签五' }
        ]
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
type	类型	string	success/info/warning/danger	—
closable	是否可关闭	boolean	—	false
disable-transitions	是否禁用渐变动画	boolean	—	false
hit	是否有边框描边	boolean	—	false
color	背景色	string	—	—
size	尺寸	string	medium / small / mini	—
effect	主题	string	dark / light / plain	light

Events

事件名称	说明	回调参数
click	点击 Tag 时触发的事件	—
close	关闭 Tag 时触发的事件	—

Progress 进度条

用于展示操作进度，告知用户当前状态和预期。

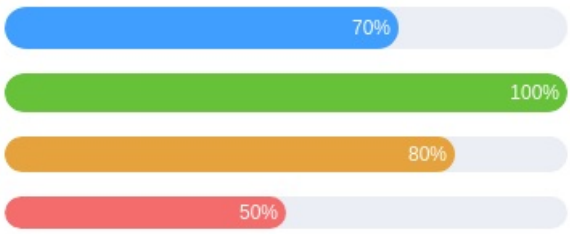
线形进度条

Progress 组件设置 `percentage` 属性即可，表示进度条对应的百分比，必填，必须在 0-100。通过 `format` 属性来指定进度条文字内容。

```
<el-progress :percentage="50"></el-progress>
<el-progress :percentage="100" :format="format"></el-progress>
<el-progress :percentage="100" status="success"></el-progress>
<el-progress :percentage="100" status="warning"></el-progress>
<el-progress :percentage="50" status="exception"></el-progress>

<script>
  export default {
    methods: {
      format(percentage) {
        return percentage === 100 ? '满' : `${percentage}%`;
      }
    }
  };
</script>
```

百分比内显

百分比不占用额外控件，适用于文件上传等场景。



Progress 组件可通过 `stroke-width` 属性更改进度条的高度，并可通过 `text-inside` 属性来将进度条描述置于进度条内部。

```
<el-progress :text-inside="true" :stroke-width="26" :percentage="70"></el-progress>
<el-progress :text-inside="true" :stroke-width="24" :percentage="100" status="success"></el-progress>
<el-progress :text-inside="true" :stroke-width="22" :percentage="80" status="warning"></el-progress>
<el-progress :text-inside="true" :stroke-width="20" :percentage="50" status="exception"></el-progress>
```

自定义颜色

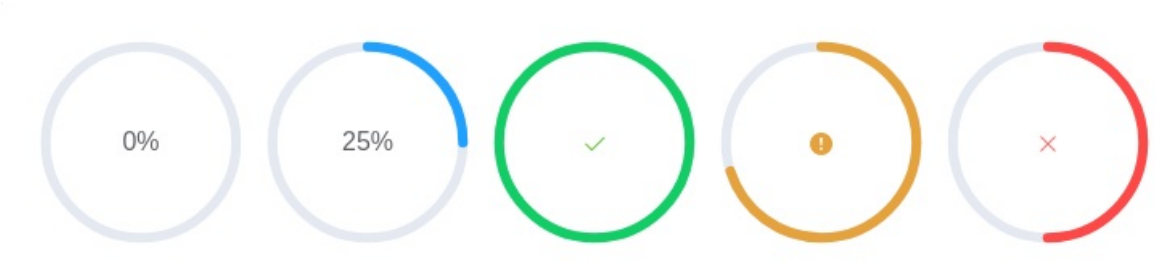
可以通过 `color` 设置进度条的颜色， `color` 可以接受颜色字符串，函数和数组。


```
<el-progress :percentage="percentage" :color="customColor"></el-progress>

<el-progress :percentage="percentage" :color="customColorMethod"></el-progress>

<el-progress :percentage="percentage" :color="customColors"></el-progress>
<div>
  <el-button-group>
    <el-button icon="el-icon-minus" @click="decrease"></el-button>
    <el-button icon="el-icon-plus" @click="increase"></el-button>
  </el-button-group>
</div>
```

```
12.
13. <script>
14.   export default {
15.     data() {
16.       return {
17.         percentage: 20,
18.         customColor: '#409eff',
19.         customColors: [
20.           {color: '#f56c6c', percentage: 20},
21.           {color: '#e6a23c', percentage: 40},
22.           {color: '#5cb87a', percentage: 60},
23.           {color: '#1989fa', percentage: 80},
24.           {color: '#6f7ad3', percentage: 100}
25.         ]
26.       };
27.     },
28.     methods: {
29.       customColorMethod(percentage) {
30.         if (percentage < 30) {
31.           return '#909399';
32.         } else if (percentage < 70) {
33.           return '#e6a23c';
34.         } else {
35.           return '#67c23a';
36.         }
37.       },
38.       increase() {
39.         this.percentage += 10;
40.         if (this.percentage > 100) {
41.           this.percentage = 100;
42.         }
43.       },
44.       decrease() {
45.         this.percentage -= 10;
46.         if (this.percentage < 0) {
47.           this.percentage = 0;
48.         }
49.       }
50.     }
51.   }
52. </script>
```

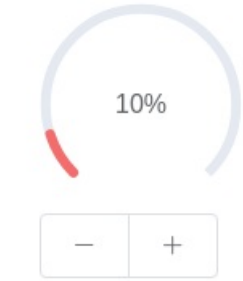
环形进度条

Progress 组件可通过 `type` 属性来指定使用环形进度条，在环形进度条中，还可以通过 `width` 属性来设置其大小。




```
<el-progress type="circle" :percentage="0"></el-progress>
<el-progress type="circle" :percentage="25"></el-progress>
<el-progress type="circle" :percentage="100" status="success"></el-progress>
<el-progress type="circle" :percentage="70" status="warning"></el-progress>
<el-progress type="circle" :percentage="50" status="exception"></el-progress>
```

仪表盘形进度条



通过 `type` 属性来指定使用仪表盘形进度条。

```

<el-progress type="dashboard" :percentage="percentage" :color="colors"></el-
progress>
<div>
  <el-button-group>
    <el-button icon="el-icon-minus" @click="decrease"></el-button>
    <el-button icon="el-icon-plus" @click="increase"></el-button>
  </el-button-group>
</div>

<script>
```

```
  export default {
    data() {
      return {
        percentage: 10,
        colors: [
          {color: '#f56c6c', percentage: 20},
          {color: '#e6a23c', percentage: 40},
          {color: '#5cb87a', percentage: 60},
          {color: '#1989fa', percentage: 80},
          {color: '#6f7ad3', percentage: 100}
        ]
      };
    },
    methods: {
      increase() {
        this.percentage += 10;
        if (this.percentage > 100) {
          this.percentage = 100;
        }
      },
      decrease() {
        this.percentage -= 10;
        if (this.percentage < 0) {
          this.percentage = 0;
        }
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值
percentage	百分比（必填）	number	0-100
type	进度条类型	string	line/circle/dashboard
stroke-width	进度条的宽度，单位 px	number	—
text-inside	进度条显示文字内置在进度条内（只在 type=line 时可用）	boolean	—
status	进度条当前状态	string	success/exception/warning
	进度条背景色（会覆		

color	盖 status 状态颜色）	string/function/array	—
width	环形进度条画布宽度（只在 type 为 circle 或 dashboard 时可用）	number	
show-text	是否显示进度条文字内容	boolean	—
stroke-linecap	circle/dashboard 类型路径两端的形状	string	butt/round/square

Tree 树形控件

用清晰的层级结构展示信息，可展开或折叠。

基础用法

基础的树形结构展示。


```
<el-tree :data="data" :props="defaultProps" @node-click="handleNodeClick"></el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          label: '一级 1',
          children: [{
            label: '二级 1-1',
            children: [{
              label: '三级 1-1-1'
            }]
          }]
        }, {
          label: '一级 2',
          children: [{
            label: '二级 2-1',
            children: [{
              label: '三级 2-1-1'
            }]
          }, {
            label: '二级 2-2',
            children: [{
              label: '三级 2-2-1'
            }]
```

```
          }]
        }, {
          label: '一级 3',
          children: [{
            label: '二级 3-1',
            children: [{
              label: '三级 3-1-1'
            }]
          }, {
            label: '二级 3-2',
            children: [{
              label: '三级 3-2-1'
            }]
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    },
    methods: {
      handleNodeClick(data) {
        console.log(data);
      }
    }
  };
</script>
```

可选择

适用于需要选择层级时使用。

本例还展示了动态加载节点数据的方法。

```
<el-tree
  :props="props"
  :load="loadNode"
```

```
    lazy
    show-checkbox
    @check-change="handleCheckChange">
  </el-tree>

  <script>
    export default {
      data() {
        return {
          props: {
            label: 'name',
            children: 'zones'
          },
          count: 1
        };
      },
      methods: {
        handleCheckChange(data, checked, indeterminate) {
          console.log(data, checked, indeterminate);
        },
        handleNodeClick(data) {
          console.log(data);
        },
        loadNode(node, resolve) {
          if (node.level === 0) {
            return resolve([{ name: 'region1' }, { name: 'region2' }]);
          }
          if (node.level > 3) return resolve([]);

          var hasChild;
          if (node.data.name === 'region1') {
            hasChild = true;
          } else if (node.data.name === 'region2') {
            hasChild = false;
          } else {
            hasChild = Math.random() > 0.5;
          }

          setTimeout(() => {
            var data;
            if (hasChild) {
              data = [{
```

```
              name: 'zone' + this.count++
            }, {
              name: 'zone' + this.count++
            }];
          } else {
            data = [];
          }

          resolve(data);
        }, 500);
      }
    }
  };
</script>
```

懒加载自定义叶子节点

由于在点击节点时才进行该层数据的获取，默认情况下 Tree 无法预知某个节点是否为叶子节点，所以会为每个节点添加一个下拉按钮，如果节点没有下层数据，则点击后下拉按钮会消失。同时，你也可以提前告知 Tree 某个节点是否为叶子节点，从而避免在叶子节点前渲染下拉按钮。

```
<el-tree
  :props="props"
  :load="loadNode"
  lazy
  show-checkbox>
</el-tree>

<script>
  export default {
    data() {
      return {
        props: {
          label: 'name',
          children: 'zones',
          isLeaf: 'leaf'
        },
      };
```

```
    },
    methods: {
      loadNode(node, resolve) {
        if (node.level === 0) {
          return resolve([{ name: 'region' }]);
        }
        if (node.level > 1) return resolve([]);

        setTimeout(() => {
          const data = [{
            name: 'leaf',
            leaf: true
          }, {
            name: 'zone'
          }];

          resolve(data);
        }, 500);
      }
    }
  };
</script>
```

默认展开和默认选中

可将 Tree 的某些节点设置为默认展开或默认选中

分别通过 `default-expanded-keys` 和 `default-checked-keys` 设置默认展开和默认选中的节点。需要注意的是，此时必须设置 `node-key` ，其值为节点数据中的一个字段名，该字段在整棵树中是唯一的。

```
<el-tree
  :data="data"
```

```
  show-checkbox
  node-key="id"
  :default-expanded-keys="[2, 3]"
  :default-checked-keys="[5]"
  :props="defaultProps">
</el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          id: 1,
          label: '一级 1',
          children: [{
            id: 4,
            label: '二级 1-1',
            children: [{
              id: 9,
              label: '三级 1-1-1'
            }, {
              id: 10,
              label: '三级 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: '一级 2',
          children: [{
            id: 5,
            label: '二级 2-1'
          }, {
            id: 6,
            label: '二级 2-2'
          }]
        }, {
          id: 3,
          label: '一级 3',
          children: [{
            id: 7,
            label: '二级 3-1'
          }, {
```

```
            id: 8,
            label: '二级 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

禁用状态

可将 Tree 的某些节点设置为禁用状态

通过 `disabled` 设置禁用状态。

```
<el-tree
  :data="data"
  show-checkbox
  node-key="id"
  :default-expanded-keys="[2, 3]"
  :default-checked-keys="[5]">
</el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          id: 1,
```

```
          label: '一级 2',
          children: [{
            id: 3,
            label: '二级 2-1',
            children: [{
              id: 4,
              label: '三级 3-1-1'
            }, {
              id: 5,
              label: '三级 3-1-2',
              disabled: true
            }]
          }, {
            id: 2,
            label: '二级 2-2',
            disabled: true,
            children: [{
              id: 6,
              label: '三级 3-2-1'
            }, {
              id: 7,
              label: '三级 3-2-2',
              disabled: true
            }]
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

树节点的选择

本例展示如何获取和设置选中节点。获取和设置各有两种方式：通过 node 或通过 key。如果需要通过 key 来获取或设置，则必须设置 `node-key` 。

```
<el-tree
  :data="data"
  show-checkbox
  default-expand-all
  node-key="id"
  ref="tree"
  highlight-current
  :props="defaultProps">
</el-tree>

<div class="buttons">
  <el-button @click="getCheckedNodes">通过 node 获取</el-button>
  <el-button @click="getCheckedKeys">通过 key 获取</el-button>
  <el-button @click="setCheckedNodes">通过 node 设置</el-button>
  <el-button @click="setCheckedKeys">通过 key 设置</el-button>
  <el-button @click="resetChecked">清空</el-button>
</div>

<script>
  export default {
    methods: {
      getCheckedNodes() {
        console.log(this.$refs.tree.getCheckedNodes());
      },
      getCheckedKeys() {
        console.log(this.$refs.tree.getCheckedKeys());
```

```
      },
      setCheckedNodes() {
        this.$refs.tree.setCheckedNodes([{
          id: 5,
          label: '二级 2-1'
        }, {
          id: 9,
          label: '三级 1-1-1'
        }]);
      },
      setCheckedKeys() {
        this.$refs.tree.setCheckedKeys([3]);
      },
      resetChecked() {
        this.$refs.tree.setCheckedKeys([]);
      }
    },

    data() {
      return {
        data: [{
          id: 1,
          label: '一级 1',
          children: [{
            id: 4,
            label: '二级 1-1',
            children: [{
              id: 9,
              label: '三级 1-1-1'
            }, {
              id: 10,
              label: '三级 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: '一级 2',
          children: [{
            id: 5,
            label: '二级 2-1'
          }, {
            id: 6,
```

```
            label: '二级 2-2'
          }]
        }, {
          id: 3,
          label: '一级 3',
          children: [{
            id: 7,
            label: '二级 3-1'
          }, {
            id: 8,
            label: '二级 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

自定义节点内容

节点的内容支持自定义，可以在节点区添加按钮或图标等内容

可以通过两种方法进行树节点内容的自定义： `render-content` 和 scoped slot。使用 `render-content` 指定渲染函数，该函数返回需要的节点区内容即可。渲染函数的用法请参考 Vue 文档。使用 scoped slot 会传入两个参数 `node` 和 `data` ，分别表示当前节点的 Node 对象和当前节点

的数据。注意：由于 jsfiddle 不支持 JSX 语法，所以 `render-content` 示例在 jsfiddle 中无法运行。但是在实际的项目中，只要正确地配置了相关依赖，就可以正常运行。

```
<div class="custom-tree-container">
  <div class="block">
    <p>使用 render-content</p>
    <el-tree
      :data="data"
      show-checkbox
      node-key="id"
      default-expand-all
      :expand-on-click-node="false"
      :render-content="renderContent">
    </el-tree>
  </div>
  <div class="block">
    <p>使用 scoped slot</p>
    <el-tree
      :data="data"
      show-checkbox
      node-key="id"
      default-expand-all
      :expand-on-click-node="false">
      <template #default="{ node, data }">
        <span class="custom-tree-node">
          <span>{{ node.label }}</span>
          <span>
            <a
              @click="append(data)">
              Append
            </a>
            <a
              @click="remove(node, data)">
              Delete
            </a>
          </span>
        </span>
      </template>
    </el-tree>
  </div>
</div>

```

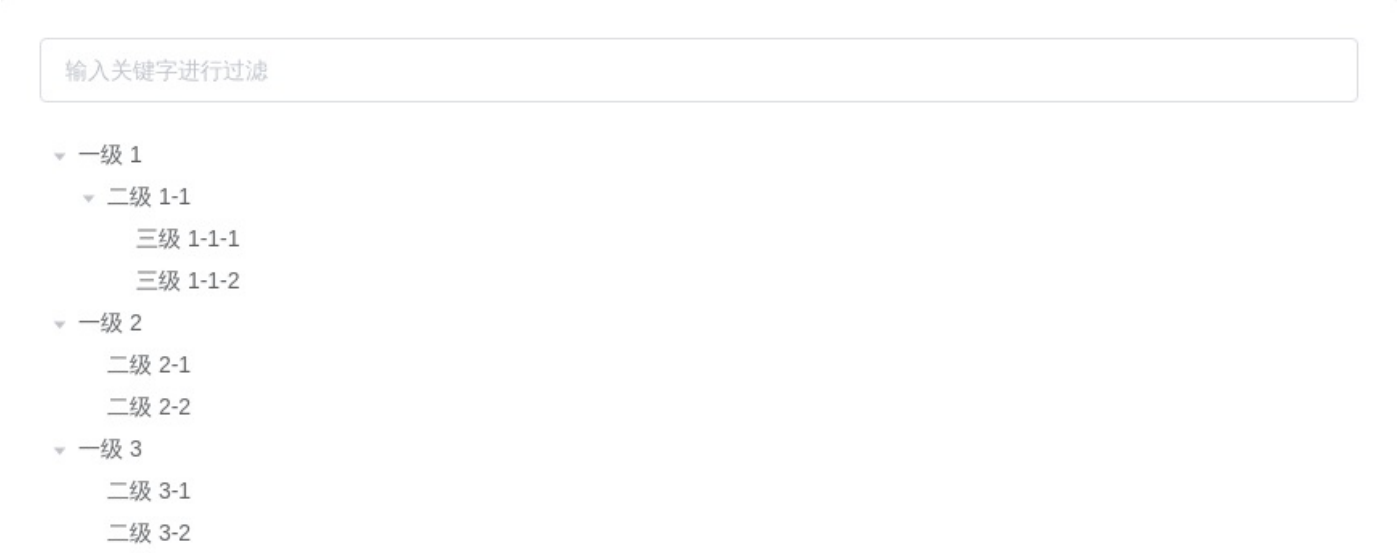
```
<script>
  let id = 1000;

  export default {
    data() {
      const data = [{
        id: 1,
        label: '一级 1',
        children: [{
          id: 4,
          label: '二级 1-1',
          children: [{
            id: 9,
            label: '三级 1-1-1'
          }, {
            id: 10,
            label: '三级 1-1-2'
          }]
        }]
      }, {
        id: 2,
        label: '一级 2',
        children: [{
          id: 5,
          label: '二级 2-1'
        }, {
          id: 6,
          label: '二级 2-2'
        }]
      }, {
        id: 3,
        label: '一级 3',
        children: [{
          id: 7,
          label: '二级 3-1'
        }, {
          id: 8,
          label: '二级 3-2'
        }]
      }];
      return {
        data: JSON.parse(JSON.stringify(data)),
```

```
        data: JSON.parse(JSON.stringify(data))
      }
    },

    methods: {
      append(data) {
        const newChild = { id: id++, label: 'testtest', children: [] };
        if (!data.children) {
          data.children = []
        }
        data.children.push(newChild);
        this.data = [...this.data]
      },

      remove(node, data) {
        const parent = node.parent;
        const children = parent.data.children || parent.data;
        const index = children.findIndex(d => d.id === data.id);
        children.splice(index, 1);
        this.data = [...this.data]
      },

      renderContent(h, { node, data, store }) {
        return h("span", {
          class: "custom-tree-node"
        }, h("span", null, node.label), h("span", null, h("a", {
          onClick: () => this.append(data)
        }, "Append "), h("a", {
          onClick: () => this.remove(node, data)
        }, "Delete")));
      }
    }
  };
</script>

<style>
  .custom-tree-node {
    flex: 1;
    display: flex;
    align-items: center;
    justify-content: space-between;
    font-size: 14px;
```

```
    padding-right: 8px;
  }
</style>
```

节点过滤

通过关键字过滤树节点



在需要对节点进行过滤时，调用 Tree 实例的 `filter` 方法，参数为关键字。需要注意的是，此时需要设置 `filter-node-method` ，值为过滤函数。

```
<el-input
  placeholder="输入关键字进行过滤"
  v-model="filterText">
</el-input>

<el-tree
  class="filter-tree"
  :data="data"
  :props="defaultProps"
  default-expand-all
  :filter-node-method="filterNode"
  ref="tree">
</el-tree>

<script>
  export default {
    watch: {
```

```
18.       filterText(val) {
19.         this.$refs.tree.filter(val);
20.       }
21.     },
22.
23.     methods: {
24.       filterNode(value, data) {
25.         if (!value) return true;
26.         return data.label.indexOf(value) !== -1;
27.       }
28.     },
29.
30.     data() {
31.       return {
32.         filterText: '',
33.         data: [{
34.           id: 1,
35.           label: '一级 1',
36.           children: [{
37.             id: 4,
38.             label: '二级 1-1',
39.             children: [{
40.               id: 9,
41.               label: '三级 1-1-1'
42.             }, {
43.               id: 10,
44.               label: '三级 1-1-2'
45.             }]
46.           }]
47.         }, {
48.           id: 2,
49.           label: '一级 2',
50.           children: [{
51.             id: 5,
52.             label: '二级 2-1'
53.           }, {
54.             id: 6,
55.             label: '二级 2-2'
56.           }]
57.         }, {
58.           id: 3,
59.           label: '一级 3',
```

```
          children: [{
            id: 7,
            label: '二级 3-1'
          }, {
            id: 8,
            label: '二级 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

手风琴模式

对于同一级的节点，每次只能展开一个


```
<el-tree
  :data="data"
  :props="defaultProps"
  accordion
  @node-click="handleNodeClick">
</el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          label: '一级 1',
          children: [{
            label: '二级 1-1',
```

```
            children: [{
              label: '三级 1-1-1'
            }]
          }]
        }, {
          label: '一级 2',
          children: [{
            label: '二级 2-1',
            children: [{
              label: '三级 2-1-1'
            }]
          }, {
            label: '二级 2-2',
            children: [{
              label: '三级 2-2-1'
            }]
          }]
        }, {
          label: '一级 3',
          children: [{
            label: '二级 3-1',
            children: [{
              label: '三级 3-1-1'
            }]
          }, {
            label: '二级 3-2',
            children: [{
              label: '三级 3-2-1'
            }]
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    },
    methods: {
      handleNodeClick(data) {
        console.log(data);
      }
    }
```

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

可拖拽节点

通过 draggable 属性可让节点变为可拖拽。


```
<el-tree
  :data="data"
  node-key="id"
  default-expand-all
  @node-drag-start="handleDragStart"
  @node-drag-enter="handleDragEnter"
  @node-drag-leave="handleDragLeave"
  @node-drag-over="handleDragOver"
  @node-drag-end="handleDragEnd"
  @node-drop="handleDrop"
  draggable
  :allow-drop="allowDrop"
  :allow-drag="allowDrag">
</el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          id: 1,
```

```
          label: '一级 1',
          children: [{
            id: 4,
            label: '二级 1-1',
            children: [{
              id: 9,
              label: '三级 1-1-1'
            }, {
              id: 10,
              label: '三级 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: '一级 2',
          children: [{
            id: 5,
            label: '二级 2-1'
          }, {
            id: 6,
            label: '二级 2-2'
          }]
        }, {
          id: 3,
          label: '一级 3',
          children: [{
            id: 7,
            label: '二级 3-1'
          }, {
            id: 8,
            label: '二级 3-2',
            children: [{
             id: 11,
              label: '三级 3-2-1'
            }, {
              id: 12,
              label: '三级 3-2-2'
            }, {
              id: 13,
              label: '三级 3-2-3'
            }]
          }]
```

```
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    },
    methods: {
      handleDragStart(node, ev) {
        console.log('drag start', node);
      },
      handleDragEnter(draggingNode, dropNode, ev) {
        console.log('tree drag enter: ', dropNode.label);
      },
      handleDragLeave(draggingNode, dropNode, ev) {
        console.log('tree drag leave: ', dropNode.label);
      },
      handleDragOver(draggingNode, dropNode, ev) {
        console.log('tree drag over: ', dropNode.label);
      },
      handleDragEnd(draggingNode, dropNode, dropType, ev) {
        console.log('tree drag end: ', dropNode && dropNode.label, dropType);
      },
      handleDrop(draggingNode, dropNode, dropType, ev) {
        console.log('tree drop: ', dropNode.label, dropType);
      },
      allowDrop(draggingNode, dropNode, type) {
        if (dropNode.data.label === '二级 3-1') {
          return type !== 'inner';
        } else {
          return true;
        }
      },
      allowDrag(draggingNode) {
        return draggingNode.data.label.indexOf('三级 3-2-2') === -1;
      }
    }
  };
</script>
```

Attributes

参数	说明	类型	可选值	默认值
data	展示数据	array	—	—
empty-text	内容为空的时候展示的文本	String	—	—
node-key	每个树节点用来作为唯一标识的属性，整棵树应该是唯一的	String	—	—
props	配置选项，具体看下表	object	—	—
render-after-expand	是否在第一次展开某个树节点后才渲染其子节点	boolean	—	true
load	加载子树数据的方法，仅当 lazy 属性为 true 时生效	function(node, resolve)	—	—
render-content	树节点的内容区的渲染 Function	Function(h, { node, data, store }	—	—
highlight-current	是否高亮当前选中节点，默认值是 false。	boolean	—	false
default-expand-all	是否默认展开所有节点	boolean	—	false
expand-on-click-node	是否在点击节点的时候展开或者收缩节点，默认值为 true，如果为 false，则只有点箭头图标的时候才会展开或者收缩节点。	boolean	—	true
check-on-click-node	是否在点击节点的时候选中节点，默认值为 false，即只有在点击复选框时才会选中节点。	boolean	—	false
auto-expand-parent	展开子节点的时候是否自动展开父节点	boolean	—	true
default-expanded-keys	默认展开的节点的 key 的数组	array	—	—
show-checkbox	节点是否可被选择	boolean	—	false
check-strictly	在显示复选框的情况下，是否严格的遵循父子不互相关联的做法，默认为 false	boolean	—	false
default-checked-keys	默认勾选的节点的 key 的数组	array	—	—
current-node-key	当前选中的节点	string, number	—	—
filter-node-method	对树节点进行筛选时执行的方法，返回 true 表示这个节点可以显示，返回 false 则表示这个节点会被隐藏	Function(value, data, node)	—	—
accordion	是否每次只打开一个同级树节点展开	boolean	—	false
indent	相邻级节点间的水平缩进，单位为像素	number	—	16
icon-class	自定义树节点的图标	string	-	-
lazy	是否懒加载子节点，需与 load 方法结合使用	boolean	—	false

draggable	是否开启拖拽节点功能	boolean	—	false
allow-drag	判断节点能否被拖拽	Function(node)	—	—
allow-drop	拖拽时判定目标节点能否被放置。type 参数有三种情况：'prev'、'inner' 和 'next'，分别表示放置在目标节点前、插入至目标节点和放置在目标节点后	Function(draggingNode, dropNode, type)	—	—

props

参数	说明	类型	可选值	默认值
label	指定节点标签为节点对象的某个属性值	string, function(data, node)	—	—
children	指定子树为节点对象的某个属性值	string	—	—
disabled	指定节点选择框是否禁用为节点对象的某个属性值	boolean, function(data, node)	—	—
isLeaf	指定节点是否为叶子节点，仅在指定了 lazy 属性的情况下生效	boolean, function(data, node)	—	—

方法

Tree 内部使用了 Node 类型的对象来包装用户传入的数据，用来保存目前节点的状态。
Tree 拥有如下方法：

方法名	说明	参数
filter	对树节点进行筛选操作	接收一个任意类型的参数，该参数会在 filter-node-method 中作为第一个参数
updateKeyChildren	通过 keys 设置节点子元素，使用此方法必须设置 node-key 属性	(key, data) 接收两个参数，1. 节点 key 2. 节点数据的数组
getCheckedNodes	若节点可被选择（即 show-checkbox 为 true），则返回目前被选中的节点所组成的数组	(leafOnly, includeHalfChecked) 接收两个 boolean 类型的参数，1. 是否只是叶子节点，默认值为 false 2. 是否包含半选节点，默认值为 false
setCheckedNodes	设置目前勾选的节点，使用此方法必须设置 node-key 属性	(nodes) 接收勾选节点数据的数组
getCheckedKeys	若节点可被选择（即 show-checkbox 为 true），则返回目前被选中的节点的 key 所组成的数组	(leafOnly) 接收一个 boolean 类型的参数，若为 true 则仅返回被选中的叶子节点的 keys，默认值为 false
setCheckedKeys	通过 keys 设置目前勾选的节点，使用此方法必须设置 node-key 属性	(keys, leafOnly) 接收两个参数，1. 勾选节点的 key 的数组 2. boolean 类型的参数，若为 true 则仅设置叶子节点的选中状态，默认值为 false
setChecked	通过 key / data 设置某个节点的勾选状态，使用此方法必须设置 node-key	(key/data, checked, deep) 接收三个参数，1. 勾选节点的 key 或者 data 2. boolean 类型，节点是否选中 3. boolean 类

	属性	型，是否设置子节点 ，默认为 false
getHalfCheckedNodes	若节点可被选择（即 show-checkbox 为 true ），则返回目前半选中的节点所组成的数组	-
getHalfCheckedKeys	若节点可被选择（即 show-checkbox 为 true ），则返回目前半选中的节点的 key 所组成的数组	-
getCurrentKey	获取当前被选中节点的 key，使用此方法必须设置 node-key 属性，若没有节点被选中则返回 null	—
getCurrentNode	获取当前被选中节点的 data，若没有节点被选中则返回 null	—
setCurrentKey	通过 key 设置某个节点的当前选中状态，使用此方法必须设置 node-key 属性	(key) 待被选节点的 key，若为 null 则取消当前高亮的节点
setCurrentNode	通过 node 设置某个节点的当前选中状态，使用此方法必须设置 node-key 属性	(node) 待被选节点的 node
getNode	根据 data 或者 key 拿到 Tree 组件中的 node	(data) 要获得 node 的 key 或者 data
remove	删除 Tree 中的一个节点，使用此方法必须设置 node-key 属性	(data) 要删除的节点的 data 或者 node
append	为 Tree 中的一个节点追加一个子节点	(data, parentNode) 接收两个参数，1. 要追加的子节点的 data 2. 子节点的 parent 的 data、key 或者 node
insertBefore	为 Tree 的一个节点的前面增加一个节点	(data, refNode) 接收两个参数，1. 要增加的节点的 data 2. 要增加的节点的后一个节点的 data、key 或者 node
insertAfter	为 Tree 的一个节点的后面增加一个节点	(data, refNode) 接收两个参数，1. 要增加的节点的 data 2. 要增加的节点的前一个节点的 data、key 或者 node

Events

事件名称	说明	回调参数
node-click	节点被点击时的回调	共三个参数，依次为：传递给 data 属性的数组中该节点所对应的对象、节点对应的 Node、节点组件本身。
node-contextmenu	当某一节点被鼠标右键点击时会触发该事件	共四个参数，依次为：event、传递给 data 属性的数组中该节点所对应的对象、节点对应的 Node、节点组件本身。
check-change	节点选中状态发生变化时的回调	共三个参数，依次为：传递给 data 属性的数组中该节点所对应的对象、节点本身是否被选中、节点的子树中是否有被选中的节点
check	当复选框被点击的时候触发	共两个参数，依次为：传递给 data 属性的数组中该节点所对应的对象、树目前的选中状态对象，包含 checkedNodes、checkedKeys、halfCheckedNodes、halfCheckedKeys 四个

		属性
current-change	当前选中节点变化时触发的事件	共两个参数，依次为：当前节点的数据，当前节点的 Node 对象
node-expand	节点被展开时触发的事件	共三个参数，依次为：传递给 `data` 属性的数组中该节点所对应的对象、节点对应的 Node、节点组件本身
node-collapse	节点被关闭时触发的事件	共三个参数，依次为：传递给 `data` 属性的数组中该节点所对应的对象、节点对应的 Node、节点组件本身
node-drag-start	节点开始拖拽时触发的事件	共两个参数，依次为：被拖拽节点对应的 Node、event
node-drag-enter	拖拽进入其他节点时触发的事件	共三个参数，依次为：被拖拽节点对应的 Node、所进入节点对应的 Node、event
node-drag-leave	拖拽离开某个节点时触发的事件	共三个参数，依次为：被拖拽节点对应的 Node、所离开节点对应的 Node、event
node-drag-over	在拖拽节点时触发的事件（类似浏览器的 mouseover 事件）	共三个参数，依次为：被拖拽节点对应的 Node、当前进入节点对应的 Node、event
node-drag-end	拖拽结束时（可能未成功）触发的事件	共四个参数，依次为：被拖拽节点对应的 Node、结束拖拽时最后进入的节点（可能为空）、被拖拽节点的放置位置（before、after、inner）、event
node-drop	拖拽成功完成时触发的事件	共四个参数，依次为：被拖拽节点对应的 Node、结束拖拽时最后进入的节点、被拖拽节点的放置位置（before、after、inner）、event

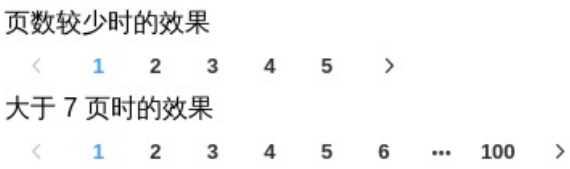
Scoped Slot

name	说明
—	自定义树节点的内容，参数为 { node, data }

Pagination 分页

当数据量过多时，使用分页分解数据。

基础用法



设置 `layout` ，表示需要显示的内容，用逗号分隔，布局元素会依次显示。 `prev` 表示上一页， `next` 为下一页， `pager` 表示页码列表，除此以外还提供了 `jumper` 和 `total` ， `size` 和特殊的布局符号 `->` ， `->` 后的元素会靠右显示， `jumper` 表示跳页元素， `total` 表示总条目数， `size` 用于设置每页显示的页码数量。

```
<div class="block">
  <span class="demonstration">页数较少时的效果</span>
  <el-pagination
    layout="prev, pager, next"
    :total="50">
  </el-pagination>
</div>
<div class="block">
  <span class="demonstration">大于 7 页时的效果</span>
  <el-pagination
    layout="prev, pager, next"
    :total="1000">
  </el-pagination>
</div>
```

设置最大页码按钮数

默认情况下，当总页数超过 7 页时，Pagination 会折叠多余的页码按钮。通过 `pager-count` 属性可以设置最大页码按钮数。

```
<el-pagination
  :page-size="20"
  :pager-count="11"
  layout="prev, pager, next"
  :total="1000">
</el-pagination>
```

带有背景色的分页

设置 `background` 属性可以为分页按钮添加背景色。

```
<el-pagination
  background
  layout="prev, pager, next"
  :total="1000">
</el-pagination>
```

小型分页

在空间有限的情况下，可以使用简单的小型分页。

只需要一个 `small` 属性，它接受一个 `Boolean` ，默认为 `false` ，设为 `true` 即可启用。

```
<el-pagination
  small
  layout="prev, pager, next"
  :total="50">
</el-pagination>
```

附加功能

根据场景需要，可以添加其他功能模块。

此例是一个完整的用例，使用了 `size-change` 和 `current-change` 事件来处理页码大小和当前页变动时候触发的事件。 `page-sizes` 接受一个整型数组，数组元素为展示的选择每页显示个数的选项， `[100, 200, 300, 400]` 表示四个选项，每页显示 100 个，200 个，300 个或者 400 个。

```
<template>
  <div class="block">
    <span class="demonstration">显示总数</span>
    <el-pagination
      @size-change="handleSizeChange"
      @current-change="handleCurrentChange"
      :current-page.sync="currentPage1"
      :page-size="100"
      layout="total, prev, pager, next"
      :total="1000">
    </el-pagination>
  </div>
  <div class="block">
    <span class="demonstration">调整每页显示条数</span>
    <el-pagination
      @size-change="handleSizeChange"
      @current-change="handleCurrentChange"
      :current-page.sync="currentPage2"
      :page-sizes="[100, 200, 300, 400]"
      :page-size="100"
      layout="sizes, prev, pager, next"
      :total="1000">
    </el-pagination>
  </div>
  <div class="block">
    <span class="demonstration">直接前往</span>
    <el-pagination
      @size-change="handleSizeChange"
```

```
      @current-change="handleCurrentChange"
      :current-page.sync="currentPage3"
      :page-size="100"
      layout="prev, pager, next, jumper"
      :total="1000">
    </el-pagination>
  </div>
  <div class="block">
    <span class="demonstration">完整功能</span>
    <el-pagination
      @size-change="handleSizeChange"
      @current-change="handleCurrentChange"
      :current-page="currentPage4"
      :page-sizes="[100, 200, 300, 400]"
      :page-size="100"
      layout="total, sizes, prev, pager, next, jumper"
      :total="400">
    </el-pagination>
  </div>
</template>
<script>
  export default {
    methods: {
      handleSizeChange(val) {
        console.log(`每页 ${val} 条`);
      },
      handleCurrentChange(val) {
        console.log(`当前页: ${val}`);
      }
    },
    data() {
      return {
        currentPage1: 5,
        currentPage2: 5,
        currentPage3: 5,
        currentPage4: 4
      };
    }
  }
</script>
```

当只有一页时隐藏分页

当只有一页时，通过设置 `hide-on-single-page` 属性来隐藏分页。

```
<div>
 <el-switch v-model="value">
 </el-switch>
 <el-pagination
  :hide-on-single-page="value"
  :total="5"
  layout="prev, pager, next">
</el-pagination>
</div>

<script>
  export default {
    data() {
      return {
        value: false
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
small	是否使用小型分页样式	boolean	—	false
background	是否为分页按钮添加背景色	boolean	—	false
page-size	每页显示条目个数，支持 .sync 修饰符	number	—	10
total	总条目数	number	—	—
page-count	总页数，total 和 page-count 设置任意一个就可以达到显示页码的功能；如果要支持 page-sizes 的更改，则需要使用 total 属性	Number	—	—
pager-count	页码按钮的数量，当总页数超过该值时会折叠	number	大于等于 5 且小于等于 21 的奇数	7
current-page	当前页数，支持 .sync 修饰符	number	—	1

layout	组件布局，子组件名用逗号分隔	String	sizes , prev , pager , next , jumper , -> . total , slot	‘prev, pager, next, jumper, ->, total’
page-sizes	每页显示个数选择器的选项设置	number[]	—	[10, 20, 30, 40, 50, 100]
popper-class	每页显示个数选择器的下拉框类名	string	—	—
prev-text	替代图标显示的上一页文字	string	—	—
next-text	替代图标显示的下一页文字	string	—	—
disabled	是否禁用	boolean	—	false
hide-on-single-page	只有一页时是否隐藏	boolean	—	-

Events

事件名称	说明	回调参数
size-change	pageSize 改变时会触发	每页条数
current-change	currentPage 改变时会触发	当前页
prev-click	用户点击上一页按钮改变当前页后触发	当前页
next-click	用户点击下一页按钮改变当前页后触发	当前页

Slot

name	说明
—	自定义内容，需要在 layout 中列出 slot

Badge 标记

出现在按钮、图标旁的数字或状态标记。

基础用法

展示新消息数量。

定义 `value` 属性，它接受 `Number` 或者 `String` 。

```
<el-badge :value="12" class="item">
  <el-button size="small">评论</el-button>
</el-badge>
<el-badge :value="3" class="item">
  <el-button size="small">回复</el-button>
</el-badge>
<el-badge :value="1" class="item" type="primary">
  <el-button size="small">评论</el-button>
</el-badge>
<el-badge :value="2" class="item" type="warning">
  <el-button size="small">回复</el-button>
</el-badge>

<el-dropdown trigger="click">
  <span class="el-dropdown-link">
    点我查看<i class="el-icon-caret-bottom el-icon--right"></i>
  </span>
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item class="clearfix">
        评论
        <el-badge class="mark" :value="12" />
      </el-dropdown-item>
      <el-dropdown-item class="clearfix">
        回复
        <el-badge class="mark" :value="3" />
      </el-dropdown-item>
```

```
    </el-dropdown-menu>
  </template>
</el-dropdown>

<style>
.item {
  margin-top: 10px;
  margin-right: 40px;
}
</style>
```

最大值

可自定义最大值。

由 `max` 属性定义，它接受一个 `Number` ，需要注意的是，只有当 `value` 为 `Number` 时，它才会生效。

```
<el-badge :value="200" :max="99" class="item">
  <el-button size="small">评论</el-button>
</el-badge>
<el-badge :value="100" :max="10" class="item">
  <el-button size="small">回复</el-button>
</el-badge>

<style>
.item {
  margin-top: 10px;
  margin-right: 40px;
}
</style>
```

自定义内容

可以显示数字以外的文本内容。

定义 `value` 为 `String` 类型是时可以用于显示自定义文本。

```
<el-badge value="new" class="item">
  <el-button size="small">评论</el-button>
</el-badge>
<el-badge value="hot" class="item">
  <el-button size="small">回复</el-button>
</el-badge>

<style>
.item {
  margin-top: 10px;
  margin-right: 40px;
}
</style>
```

小红点

以红点的形式标注需要关注的内容。

除了数字外，设置 `is-dot` 属性，它接受一个 `Boolean` 。

```
<el-badge is-dot class="item">数据查询</el-badge>
<el-badge is-dot class="item">
  <el-button class="share-button" icon="el-icon-share" type="primary"></el-
button>
</el-badge>

<style>
.item {
  margin-top: 10px;
  margin-right: 40px;
}
</style>
```

Attributes

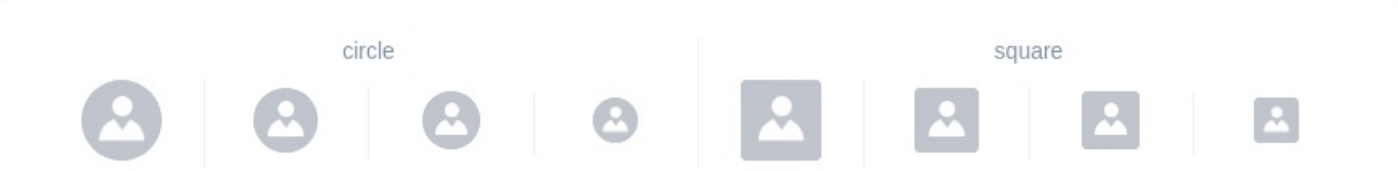
参数	说明	类型	可选值	默认值
value	显示值	string, number	—	—
max	最大值，超过最大值会显示 ‘{max}+’，要求 value 是 Number 类型	number	—	—
is-dot	小圆点	boolean	—	false
hidden	隐藏 badge	boolean	—	false
type	类型	string	primary / success / warning / danger / info	—

Avatar 头像

用图标、图片或者字符的形式展示用户或事物信息。

基本用法

通过 `shape` 和 `size` 设置头像的形状和大小。




```
<template>
  <el-row class="demo-avatar demo-basic">
    <el-col :span="12">
      <div class="sub-title">circle</div>
      <div class="demo-basic--circle">
        <div class="block"><el-avatar :size="50" :src="circleUrl"></el-avatar>
</div>
        <div class="block" v-for="size in sizeList" :key="size">
          <el-avatar :size="size" :src="circleUrl"></el-avatar>
        </div>
      </div>
    </el-col>
    <el-col :span="12">
      <div class="sub-title">square</div>
      <div class="demo-basic--circle">
        <div class="block"><el-avatar shape="square" :size="50"
:src="squareUrl"></el-avatar></div>
        <div class="block" v-for="size in sizeList" :key="size">
          <el-avatar shape="square" :size="size" :src="squareUrl"></el-avatar>
        </div>
      </div>
    </el-col>
  </el-row>
</template>
<script>
  export default {
    data () {
```

```
      return {
        circleUrl:
"https://cube.elemecdn.com/3/7c/3ea6beec64369c2642b92c6726f1epng.png",
        squareUrl:
"https://cube.elemecdn.com/9/c2/f0ee8a3c7c9638a54940382568c9dpng.png",
        sizeList: ["large", "medium", "small"]
      }
    }
  }
</script>
```

展示类型

支持三种类型：图标、图片和字符


```
<template>
  <div class="demo-type">
    <div>
      <el-avatar icon="el-icon-user-solid"></el-avatar>
    </div>
    <div>
      <el-avatar
src="https://cube.elemecdn.com/0/88/03b0d39583f48206768a7534e55bcpng.png"></el-
avatar>
    </div>
    <div>
      <el-avatar> user </el-avatar>
    </div>
  </div>
</template>
```

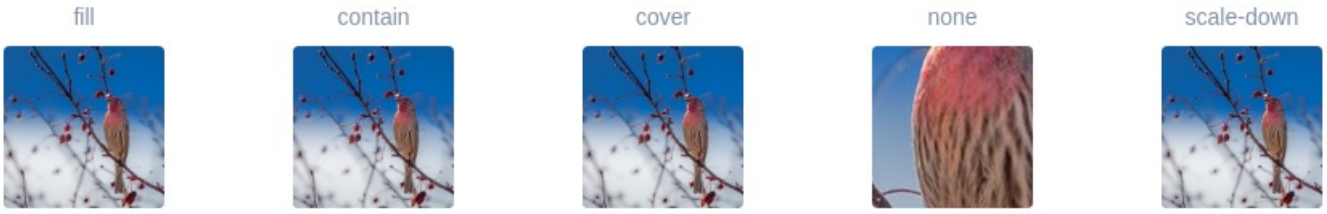
图片加载失败的 fallback 行为

当展示类型为图片的时候，图片加载失败的 fallback 行为

```
<template>
  <div class="demo-type">
    <el-avatar :size="60" src="https://empty" @error="errorHandler">
      <img
src="https://cube.elemecdn.com/e/fd/0fc7d20532fdaf769a25683617711png.png"/>
    </el-avatar>
  </div>
</template>
<script>
  export default {
    methods: {
      errorHandler() {
        return true
      }
    }
  }
</script>
```

图片如何适应容器框

当展示类型为图片的时候，使用 `fit` 属性定义图片如何适应容器框，同原生 `object-fit`。




```
<template>
  <div class="demo-fit">
    <div class="block" v-for="fit in fits" :key="fit">
        <span class="title">{{ fit }}</span>
        <el-avatar shape="square" :size="100" :fit="fit" :src="url"></el-
avatar>
    </div>
  </div>
```

```
</template>
<script>
  export default {
    data() {
      return {
        fits: ['fill', 'contain', 'cover', 'none', 'scale-down'],
        url:
'https://fuss10.elemecdn.com/e/5d/4a731a90594a4af544c0c25941171jpeg.jpeg'
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
icon	设置头像的图标类型，参考 Icon 组件	string		
size	设置头像的大小	number/string	number / large / medium / small	large
shape	设置头像的形状	string	circle / square	circle
src	图片头像的资源地址	string		
srcSet	以逗号分隔的一个或多个字符串列表表明一系列用户代理使用的可能的图像	string		
alt	描述图像的替换文本	string		
fit	当展示类型为图片的时候，设置图片如何适应容器框	string	fill / contain / cover / none / scale-down	cover

Events

事件名	说明	回调参数
error	图片类头像加载失败的回调， 返回 false 会关闭组件默认的 fallback 行为	(e: Event)

Slot

名称	说明
default	自定义头像展示内容

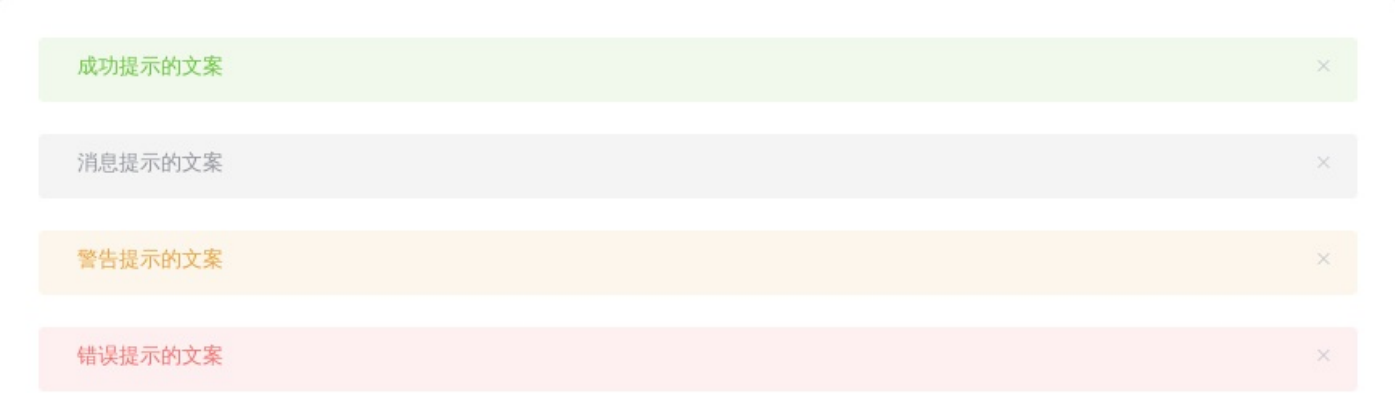
- Alert 警告
- Loading 加载
- Message 消息提示
- MessageBox 弹框
- Notification 通知

Alert 警告

用于页面中展示重要的提示信息。

基本用法

页面中的非浮层元素，不会自动消失。

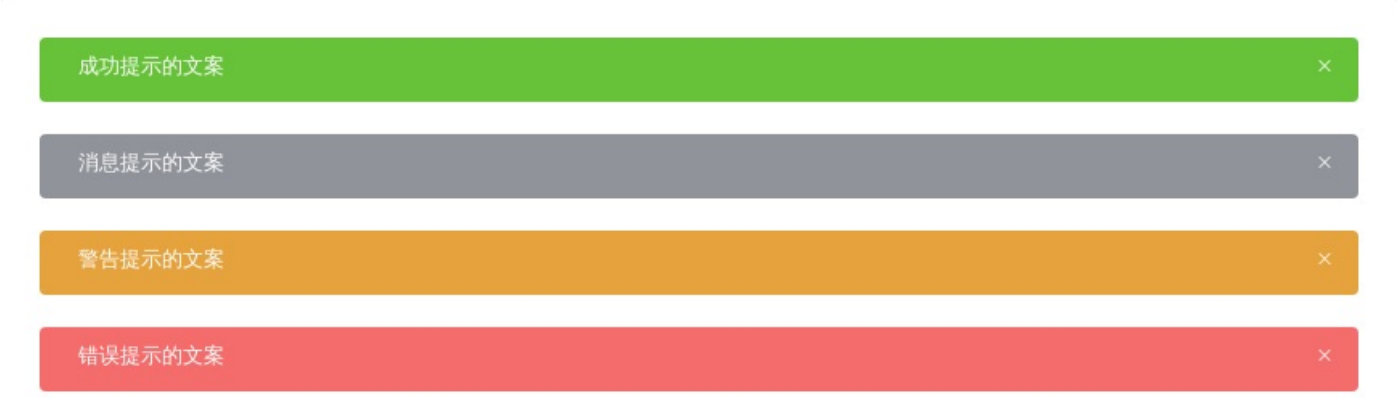


Alert 组件提供四种主题，由 `type` 属性指定，默认值为 `info` 。

```
<template>
  <el-alert
    title="成功提示的文案"
    type="success">
  </el-alert>
  <el-alert
    title="消息提示的文案"
    type="info">
  </el-alert>
  <el-alert
    title="警告提示的文案"
    type="warning">
  </el-alert>
  <el-alert
    title="错误提示的文案"
    type="error">
  </el-alert>
</template>
```

主题

Alert 组件提供了两个不同的主题： `light` 和 `dark` 。

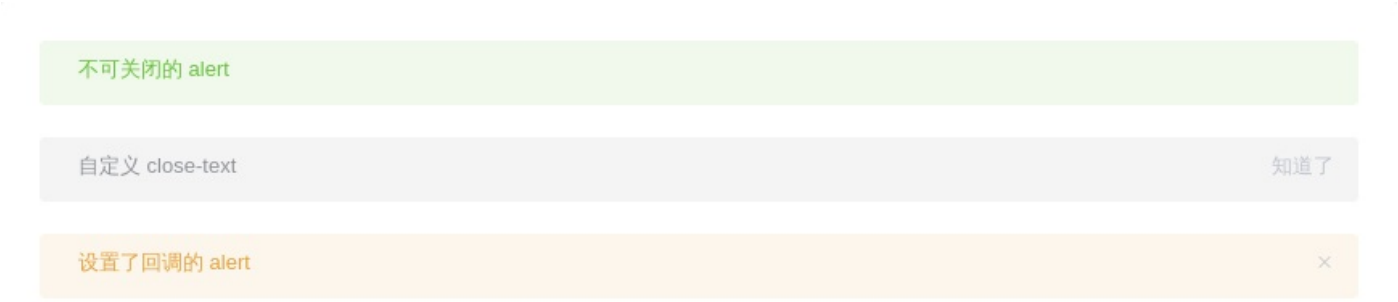


通过设置 `effect` 属性来改变主题，默认为 `light` 。

```
<template>
  <el-alert
    title="成功提示的文案"
    type="success"
    effect="dark">
  </el-alert>
  <el-alert
    title="消息提示的文案"
    type="info"
    effect="dark">
  </el-alert>
  <el-alert
    title="警告提示的文案"
    type="warning"
    effect="dark">
  </el-alert>
  <el-alert
    title="错误提示的文案"
    type="error"
    effect="dark">
  </el-alert>
</template>
```

自定义关闭按钮

自定义关闭按钮为文字或其他符号。



在 Alert 组件中，你可以设置是否可关闭，关闭按钮的文本以及关闭时的回调函数。 `closable` 属性决定是否可关闭，接受 `boolean` ，默认为 `true` 。你可以设置 `close-text` 属性来代替右侧的关闭图标，注意： `close-text` 必须为文本。设置 `close` 事件来设置关闭时的回调。

```
<template>
  <el-alert
    title="不可关闭的 alert"
    type="success"
    :closable="false">
  </el-alert>
  <el-alert
    title="自定义 close-text"
    type="info"
    close-text="知道了">
  </el-alert>
  <el-alert
    title="设置了回调的 alert"
    type="warning"
    @close="hello">
  </el-alert>
</template>

<script>
  import { defineComponent } from 'vue'
  export default defineComponent({
    setup() {
      const hello = () => {
        alert('Hello World!');
      }
      return {
        hello
      }
    }
  })
```

```
31.  </script>
```

带有 icon

表示某种状态时提升可读性。

通过设置 `show-icon` 属性来显示 Alert 的 icon，这能更有效地向用户展示你的显示意图。

```
1.  <template>
2.    <el-alert
3.      title="成功提示的文案"
4.      type="success"
5.      show-icon>
6.    </el-alert>
7.    <el-alert
8.      title="消息提示的文案"
9.      type="info"
10.     show-icon>
11.   </el-alert>
12.   <el-alert
13.     title="警告提示的文案"
14.     type="warning"
15.     show-icon>
16.   </el-alert>
17.   <el-alert
18.     title="错误提示的文案"
19.     type="error"
20.     show-icon>
21.   </el-alert>
22. </template>
```

文字居中

使用 `center` 属性让文字水平居中。


```
<template>
  <el-alert
    title="成功提示的文案"
    type="success"
    center
    show-icon>
  </el-alert>
  <el-alert
    title="消息提示的文案"
    type="info"
    center
    show-icon>
  </el-alert>
  <el-alert
    title="警告提示的文案"
    type="warning"
    center
    show-icon>
  </el-alert>
  <el-alert
    title="错误提示的文案"
    type="error"
    center
    show-icon>
  </el-alert>
</template>
```

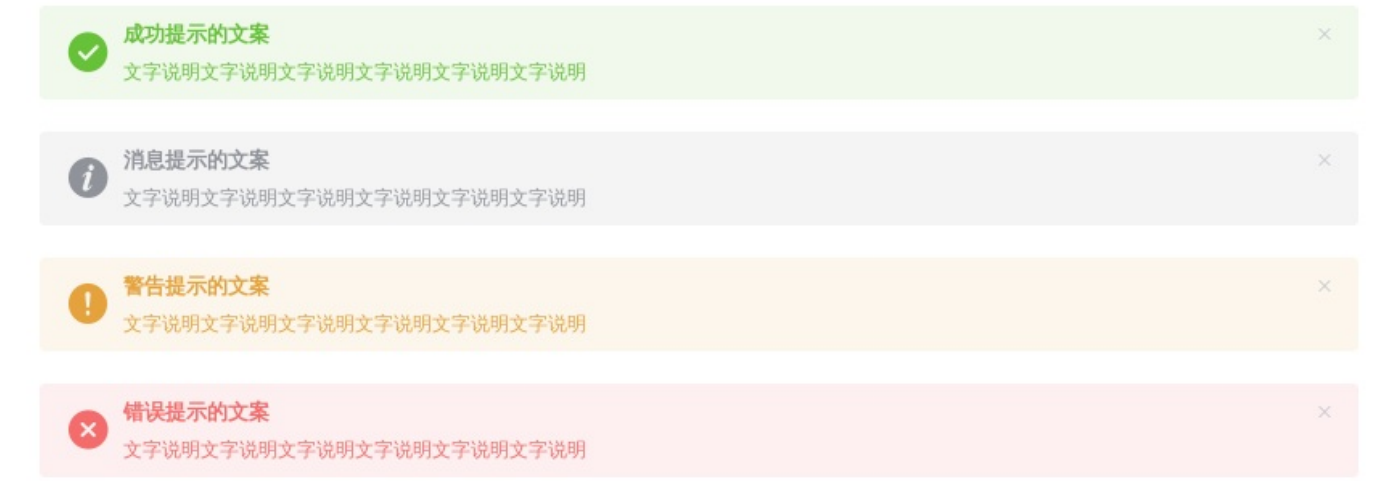
带有辅助性文字介绍

包含标题和内容，解释更详细的警告。

除了必填的 `title` 属性外，你可以设置 `description` 属性来帮助你更好地介绍，我们称之为辅助性文字。辅助性文字只能存放单行文本，会自动换行显示。

```
<template>
  <el-alert
    title="带辅助性文字介绍"
    type="success"
    description="这是一句绕口令：黑灰化肥会挥发发灰黑化肥挥发；灰黑化肥会挥发发黑灰化肥发挥。 黑灰化肥会挥发发灰黑化肥黑灰挥发化为灰……">
  </el-alert>
</template>
```

带有 icon 和辅助性文字介绍



最后，这是一个同时具有 icon 和辅助性文字的样例。

```
<template>
  <el-alert
    title="成功提示的文案"
    type="success"
    description="文字说明文字说明文字说明文字说明文字说明文字说明"
```

```
    show-icon>
  </el-alert>
  <el-alert
    title="消息提示的文案"
    type="info"
    description="文字说明文字说明文字说明文字说明文字说明文字说明"
    show-icon>
  </el-alert>
  <el-alert
    title="警告提示的文案"
    type="warning"
    description="文字说明文字说明文字说明文字说明文字说明文字说明"
    show-icon>
  </el-alert>
  <el-alert
    title="错误提示的文案"
    type="error"
    description="文字说明文字说明文字说明文字说明文字说明文字说明"
    show-icon>
  </el-alert>
</template>
```

Attributes

参数	说明	类型	可选值	默认值
title	标题	string	—	—
type	主题	string	success/warning/info/error	info
description	辅助性文字。也可通过默认 slot 传入	string	—	—
closable	是否可关闭	boolean	—	true
center	文字是否居中	boolean	—	true
close-text	关闭按钮自定义文本	string	—	—
show-icon	是否显示图标	boolean	—	false
effect	选择提供的主题	string	light/dark	light

Slot

Name	Description
—	描述
title	标题的内容

Events

事件名称	说明	回调参数
close	关闭alert时触发的事件	—

Loading 加载

加载数据时显示动效。

区域加载

在表格等容器中加载数据时显示。

Element Plus 提供了两种调用 Loading 的方法：指令和服务。对于自定义指令 `v-loading` ，只需要绑定 `Boolean` 即可。默认状况下，Loading 遮罩会插入到绑定元素的子节点，通过添加 `body` 修饰符，可以使遮罩插入至 DOM 中的 body 上。

```
<template>
  <el-table
    v-loading="loading"
    :data="tableData"
    style="width: 100%">
    <el-table-column
      prop="date"
      label="日期"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址">
    </el-table-column>
  </el-table>
```

```
</template>

<style>
  body {
    margin: 0;
  }
</style>

<script>
  export default {
    data() {
      return {
        tableData: [{
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }],
        loading: true
      };
    }
  };
</script>
```

自定义

可自定义加载文案、图标和背景色。

在绑定了 `v-loading` 指令的元素上添加 `element-loading-text` 属性，其值会被渲染为加载文案，并显示在加载图标的下方。类似地， `element-loading-spinner` 和 `element-loading-background` 属性分别用来设定图标类名和背景色值。

```
<template>
  <el-table
    v-loading="loading"
    element-loading-text="拼命加载中"
    element-loading-spinner="el-icon-loading"
    element-loading-background="rgba(0, 0, 0, 0.8)"
    :data="tableData"
    style="width: 100%">
    <el-table-column
      prop="date"
      label="日期"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址">
    </el-table-column>
  </el-table>
</template>

<script>
  export default {
    data() {
      return {
```

```
        tableData: [{
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }],
        loading: true
      };
    }
  };
</script>
```

整页加载

页面数据加载时显示。

当使用指令方式时，全屏遮罩需要添加 `fullscreen` 修饰符（遮罩会插入至 body 上），此时若需要锁定屏幕的滚动，可以使用 `lock` 修饰符；当使用服务方式时，遮罩默认即为全屏，无需额外设置。

```
<template>
  <el-button
    type="primary"
    @click="openFullScreen1"
    v-loading.fullscreen.lock="fullscreenLoading">
    指令方式
  </el-button>
  <el-button
    type="primary"
    @click="openFullScreen2">
    服务方式
```

```
  </el-button>
</template>

<script>
  export default {
    data() {
      return {
        fullscreenLoading: false
      }
    },
    methods: {
      openFullScreen1() {
        this.fullscreenLoading = true;
        setTimeout(() => {
          this.fullscreenLoading = false;
        }, 2000);
      },
      openFullScreen2() {
        const loading = this.$loading({
          lock: true,
          text: 'Loading',
          spinner: 'el-icon-loading',
          background: 'rgba(0, 0, 0, 0.7)'
        });
        setTimeout(() => {
          loading.close();
        }, 2000);
      }
    }
  }
</script>
```

服务

Loading 还可以以服务的方式调用。引入 Loading 服务:

```
import { ElLoading } from 'element-plus';
```

在需要调用时:

```
ElLoading.service(options);
```

其中 `options` 参数为 Loading 的配置项，具体见下表。`LoadingService` 会返回一个 Loading 实例，可通过调用该实例的 `close` 方法来关闭它：

```
let loadingInstance = ElLoading.service(options);
this.$nextTick(() => { // 以服务的方式调用的 Loading 需要异步关闭
  loadingInstance.close();
});
```

需要注意的是，以服务的方式调用的全屏 Loading 是单例的：若在前一个全屏 Loading 关闭前再次调用全屏 Loading，并不会创建一个新的 Loading 实例，而是返回现有全屏 Loading 的实例：

```
let loadingInstance1 = ElLoading.service({ fullscreen: true });
let loadingInstance2 = ElLoading.service({ fullscreen: true });
console.log(loadingInstance1 === loadingInstance2); // true
```

此时调用它们中任意一个的 `close` 方法都能关闭这个全屏 Loading。

如果完整引入了 Element，那么 `app.config.globalProperties` 上会有一个全局方法 `$loading`，它的调用方式为：`this.$loading(options)`，同样会返回一个 Loading 实例。

Options

参数	说明	类型	可选值	默认值
target	Loading 需要覆盖的 DOM 节点。可传入一个 DOM 对象或字符串；若传入字符串，则会将其作为参数传入 `document.querySelector` 以获取到对应 DOM 节点	object/string	—	document.body
body	同 `v-loading` 指令中的 `body` 修饰符	boolean	—	false
fullscreen	同 `v-loading` 指令中的 `fullscreen` 修饰符	boolean	—	true
lock	同 `v-loading` 指令中的 `lock` 修饰符	boolean	—	false
text	显示在加载图标下方的加载文案	string	—	—
spinner	自定义加载图标类名	string	—	—
background	遮罩背景色	string	—	—
customClass	Loading 的自定义类名	string	—	—

Message 消息提示

常用于主动操作后的反馈提示。与 Notification 的区别是后者更多用于系统级通知的被动提醒。

基础用法

从顶部出现，3 秒后自动消失。

Message 在配置上与 Notification 非常类似，所以部分 options 在此不做详尽解释，文末有 options 列表，可以结合 Notification 的文档理解它们。Element Plus 注册了一个 `$message` 方法用于调用，Message 可以接收一个字符串或一个 VNode 作为参数，它会被显示为正文内容。

```
<template>
  <el-button :plain="true" @click="open">打开消息提示</el-button>
  <el-button :plain="true" @click="openVn">VNode</el-button>
</template>

<script>
  import { h } from 'vue';

  export default {
    methods: {
      open() {
        this.$message('这是一条消息提示');
      },

      openVn() {
        this.$message({
          message: h('p', null, [
            h('span', null, '内容可以是 '),
            h('i', { style: 'color: teal' }, 'VNode')
          ])
        });
      }
    }
  }
```

```
25.  </script>
```

不同状态

用来显示「成功、警告、消息、错误」类的操作反馈。

当需要自定义更多属性时，Message 也可以接收一个对象为参数。比如，设置 `type` 字段可以定义不同的状态，默认为 `info` 。此时正文内容以 `message` 的值传入。同时，我们也为 Message 的各种 type 注册了方法，可以在不传入 `type` 字段的情况下像 `open4` 那样直接调用。

```
1.  <template>
2.    <el-button :plain="true" @click="open2">成功</el-button>
3.    <el-button :plain="true" @click="open3">警告</el-button>
4.    <el-button :plain="true" @click="open1">消息</el-button>
5.    <el-button :plain="true" @click="open4">错误</el-button>
6.  </template>
7.
8.  <script>
9.    export default {
10.     methods: {
11.       open1() {
12.         this.$message('这是一条消息提示');
13.       },
14.       open2() {
15.         this.$message({
16.           message: '恭喜你，这是一条成功消息',
17.           type: 'success'
18.         });
19.       },
20.
21.       open3() {
22.         this.$message({
23.           message: '警告哦，这是一条警告消息',
24.           type: 'warning'
25.         });
26.       },
27.
28.       open4() {
```

```
        this.$message.error('错了哦，这是一条错误消息');
      }
    }
  }
</script>
```

可关闭

可以添加关闭按钮。

默认的 Message 是不可以被人工关闭的，如果需要可手动关闭的 Message，可以使用 `showClose` 字段。此外，和 Notification 一样，Message 拥有可控的 `duration` ，设置 `0` 为不会被自动关闭，默认为 3000 毫秒。

```
<template>
  <el-button :plain="true" @click="open1">消息</el-button>
  <el-button :plain="true" @click="open2">成功</el-button>
  <el-button :plain="true" @click="open3">警告</el-button>
  <el-button :plain="true" @click="open4">错误</el-button>
</template>

<script>
  export default {
    methods: {
      open1() {
        this.$message({
          showClose: true,
          message: '这是一条消息提示'
        });
      },

      open2() {
        this.$message({
          showClose: true,
          message: '恭喜你，这是一条成功消息',
          type: 'success'
        });
      },
```

```
25.
26.       open3() {
27.         this.$message({
28.           showClose: true,
29.           message: '警告哦，这是一条警告消息',
30.           type: 'warning'
31.         });
32.       },
33.
34.       open4() {
35.         this.$message({
36.           showClose: true,
37.           message: '错了哦，这是一条错误消息',
38.           type: 'error'
39.         });
40.       }
41.     }
42.   }
43. </script>
```

文字居中

使用 `center` 属性让文字水平居中。


```
1.  <template>
2.    <el-button :plain="true" @click="openCenter">文字居中</el-button>
3.  </template>
4.
5.  <script>
6.    export default {
7.      methods: {
8.        openCenter() {
9.          this.$message({
10.           message: '居中的文字',
11.           center: true
12.         });
13.       }
14.     }
```

```
  }
</script>
```

使用 HTML 片段

`message` 属性支持传入 HTML 片段

将 `dangerouslyUseHTMLString` 属性设置为 true， `message` 就会被当作 HTML 片段处理。

```
<template>
  <el-button :plain="true" @click="openHTML">使用 HTML 片段</el-button>
</template>

<script>
  export default {
    methods: {
      openHTML() {
        this.$message({
          dangerouslyUseHTMLString: true,
          message: '<strong>这是 <i>HTML</i> 片段</strong>'
        });
      }
    }
  }
</script>
```

`message` 属性虽然支持传入 HTML 片段，但是在网站上动态渲染任意 HTML 是非常危险的，因为容易导致 XSS 攻击。因此在 `dangerouslyUseHTMLString` 打开的情况下，请确保 `message` 的内容是可信的，永远不要将用户提交的内容赋值给 `message` 属性。

全局方法

Element Plus 为 `app.config.globalProperties` 添加了全局方法 $message。因此在 vue instance 中可以采用本页面中的方式调用 `Message` 。

单独引用

```
1.  import { ElMessage } from 'element-plus';
```

此时调用方法为 `ElMessage(options)` 。我们也为每个 type 定义了各自的方法，如 `ElMessage.success(options)` 。并且可以调用 `ElMessage.closeAll()` 手动关闭所有实例。

Options

参数	说明	类型	可选值	默认值
message	消息文字	string / VNode	—	—
type	主题	string	success/warning/info/error	info
iconClass	自定义图标的类名，会覆盖 `type`	string	—	—
dangerouslyUseHTMLString	是否将 message 属性作为 HTML 片段处理	boolean	—	false
customClass	自定义类名	string	—	—
duration	显示时间，毫秒。设为 0 则不会自动关闭	number	—	3000
showClose	是否显示关闭按钮	boolean	—	false
center	文字是否居中	boolean	—	false
onClose	关闭时的回调函数，参数为被关闭的 message 实例	function	—	—
offset	Message 距离窗口顶部的偏移量	number	—	20

方法

调用 `Message` 或 `this.$message` 会返回当前 Message 的实例。如果需要手动关闭实例，可以调用它的 `close` 方法。

方法名	说明

close	关闭当前的 Message

MessageBox 弹框

模拟系统的消息提示框而实现的一套模态对话框组件，用于消息提示、确认消息和提交内容。

从场景上说，MessageBox 的作用是美化系统自带的 `alert`、`confirm` 和 `prompt`，因此适合展示较为简单的内容。如果需要弹出较为复杂的内容，请使用 Dialog。

消息提示

当用户进行操作时会被触发，该对话框中断用户操作，直到用户确认知晓后才可关闭。

调用 `$alert` 方法即可打开消息提示，它模拟了系统的 `alert`，无法通过按下 ESC 或点击框外关闭。此例中接收了两个参数，`message` 和 `title`。值得一提的是，窗口被关闭后，它默认会返回一个 `Promise` 对象便于进行后续操作的处理。若不确定浏览器是否支持 `Promise`，可自行引入第三方 polyfill 或像本例一样使用回调进行后续处理。

```
<template>
  <el-button type="text" @click="open">点击打开 Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$alert('这是一段内容', '标题名称', {
          confirmButtonText: '确定',
          callback: action => {
            this.$message({
              type: 'info',
              message: `action: ${ action }`
            });
          }
        });
      }
    }
  }
</script>
```

确认消息

提示用户确认其已经触发的动作，并询问是否进行此操作时会用到此对话框。

调用 `$confirm` 方法即可打开消息提示，它模拟了系统的 `confirm` 。Message Box 组件也拥有极高的定制性，我们可以传入 `options` 作为第三个参数，它是一个字面量对象。 `type` 字段表明消息类型，可以为 `success` ， `error` ， `info` 和 `warning` ，无效的设置将会被忽略。注意，第二个参数 `title` 必须定义为 `String` 类型，如果是 `Object` ，会被理解为 `options` 。在这里我们用了 Promise 来处理后续响应。

```
<template>
  <el-button type="text" @click="open">点击打开 Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$confirm('此操作将永久删除该文件, 是否继续?', '提示', {
          confirmButtonText: '确定',
          cancelButtonText: '取消',
          type: 'warning'
        }).then(() => {
          this.$message({
            type: 'success',
            message: '删除成功!'
          });
        }).catch(() => {
          this.$message({
            type: 'info',
            message: '已取消删除'
          });
        });
      }
    }
  }
</script>
```

提交内容

当用户进行操作时会被触发，中断用户操作，提示用户进行输入的对话框。

调用 `$prompt` 方法即可打开消息提示，它模拟了系统的 `prompt` 。可以用 `inputPattern` 字段自己规定匹配模式，或者用 `inputValidator` 规定校验函数，可以返回 `Boolean` 或 `String` ，返回 `false` 或字符串时均表示校验未通过，同时返回的字符串相当于定义了 `inputErrorMessage` 字段。此外，可以用 `inputPlaceholder` 字段来定义输入框的占位符。

```
<template>
  <el-button type="text" @click="open">点击打开 Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$prompt('请输入邮箱', '提示', {
          confirmButtonText: '确定',
          cancelButtonText: '取消',
          inputPattern: /[\w!#$%&'*+/=?^_`{|}~-]+(?:\.[\w!#$%&'*+/=?^_`{|}~-
]+)*@(?:[\w](?:[\w-]*[\w])?\.)+[\w](?:[\w-]*[\w])?/,
          inputErrorMessage: '邮箱格式不正确'
        }).then(({ value }) => {
          this.$message({
            type: 'success',
            message: '你的邮箱是: ' + value
          });
        }).catch(() => {
          this.$message({
            type: 'info',
            message: '取消输入'
          });
        });
      }
    }
  }
</script>
```

自定义

可自定义配置不同内容。

以上三个方法都是对 `$msgbox` 方法的再包装。本例直接调用 `$msgbox` 方法，使用了 `showCancelButton` 字段，用于显示取消按钮。另外可使用 `cancelButtonClass` 为其添加自定义样式，使用 `cancelButtonText` 来自定义按钮文本（Confirm 按钮也具有相同的字段，在文末的字段说明中有完整的字段列表）。此例还使用了 `beforeClose` 属性，它的值是一个方法，会在 MessageBox 的实例关闭前被调用，同时暂停实例的关闭。它有三个参数： `action` 、实例本身和 `done` 方法。使用它能够在关闭前对实例进行一些操作，比如为确定按钮添加 `loading` 状态等；此时若需要关闭实例，可以调用 `done` 方法（若在 `beforeClose` 中没有调用 `done` ，则实例不会关闭）。

```
<template>
  <el-button type="text" @click="open">点击打开 Message Box</el-button>
</template>

<script>
  import { h } from 'vue';

  export default {
    methods: {
      open() {
        this.$msgbox({
          title: '消息',
          message: h('p', null, [
            h('span', null, '内容可以是 '),
            h('i', { style: 'color: teal' }, 'VNode')
          ]),
          showCancelButton: true,
          confirmButtonText: '确定',
          cancelButtonText: '取消',
          beforeClose: (action, instance, done) => {
            if (action === 'confirm') {
              instance.confirmButtonLoading = true;
              instance.confirmButtonText = '执行中...';
              setTimeout(() => {
                done();
```

```
                setTimeout(() => {
                  instance.confirmButtonLoading = false;
                }, 300);
              }, 3000);
            } else {
              done();
            }
          }
        }).then(action => {
          this.$message({
            type: 'info',
            message: 'action: ' + action
          });
        });
      }
    }
  }
</script>
```

弹出层的内容可以是 `VNode`，所以我们能把一些自定义组件传入其中。每次弹出层打开后，Vue 会对新老 `VNode` 节点进行比对，然后将根据比较结果进行最小单位地修改视图。这也许会造成弹出层内容区域的组件没有重新渲染，例如 #8931。当这类问题出现时，解决方案是给 `VNode` 加上一个不相同的 key，参考这里。

使用 HTML 片段

`message` 属性支持传入 HTML 片段。

点击打开 Message Box

将 `dangerouslyUseHTMLString` 属性设置为 true，`message` 就会被当作 HTML 片段处理。

```
<template>
  <el-button type="text" @click="open">点击打开 Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
```

```
        this.$alert('<strong>这是 <i>HTML</i> 片段</strong>', 'HTML 片段', {
          dangerouslyUseHTMLString: true
        });
      }
    }
  }
</script>
```

`message` 属性虽然支持传入 HTML 片段，但是在网站上动态渲染任意 HTML 是非常危险的，因为容易导致 XSS 攻击。因此在 `dangerouslyUseHTMLString` 打开的情况下，请确保 `message` 的内容是可信的，永远不要将用户提交的内容赋值给 `message` 属性。

区分取消与关闭

有些场景下，点击取消按钮与点击关闭按钮有着不同的含义。

点击打开 Message Box

默认情况下，当用户触发取消（点击取消按钮）和触发关闭（点击关闭按钮或遮罩层、按下 ESC 键）时，Promise 的 reject 回调和 `callback` 回调的参数均为 ‘cancel’。如果将 `distinguishCancelAndClose` 属性设置为 true，则上述两种行为的参数分别为 ‘cancel’ 和 ‘close’。

```
<template>
  <el-button type="text" @click="open">点击打开 Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$confirm('检测到未保存的内容，是否在离开页面前保存修改？', '确认信息', {
          distinguishCancelAndClose: true,
          confirmButtonText: '保存',
          cancelButtonText: '放弃修改'
        })
          .then(() => {
            this.$message({
              type: 'info',
              message: '保存修改'
```

```
            });
          })
          .catch(action => {
            this.$message({
              type: 'info',
              message: action === 'cancel'
                ? '放弃保存并离开页面'
                : '停留在当前页面'
            })
          });
      }
    }
  }
</script>
```

居中布局

内容支持居中布局

将 `center` 设置为 `true` 即可开启居中布局

```
<template>
  <el-button type="text" @click="open">点击打开 Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$confirm('此操作将永久删除该文件, 是否继续?', '提示', {
          confirmButtonText: '确定',
          cancelButtonText: '取消',
          type: 'warning',
          center: true
        }).then(() => {
          this.$message({
            type: 'success',
            message: '删除成功!'
```

```
          });
        }).catch(() => {
          this.$message({
            type: 'info',
            message: '已取消删除'
          });
        });
      }
    }
  }
</script>
```

全局方法

如果你完整引入了 Element，它会为 `app.config.globalProperties` 添加如下全局方法：$msgbox, $alert, $confirm 和 $prompt。因此在 Vue instance 中可以采用本页面中的方式调用 `MessageBox` 。调用参数为：

- `$msgbox(options)`
- `$alert(message, title, options)` 或 `$alert(message, options)`
- `$confirm(message, title, options)` 或 `$confirm(message, options)`
- `$prompt(message, title, options)` 或 `$prompt(message, options)`

单独引用

如果单独引入 `MessageBox` ：

```
import { ElMessageBox } from 'element-plus';
```

那么对应于上述四个全局方法的调用方法依次为：ElMessageBox, ElMessageBox.alert, ElMessageBox.confirm 和 ElMessageBox.prompt，调用参数与全局方法相同。

Options

参数	说明	类型	可选值	默认
title	MessageBox 标题	string	—	—
message	MessageBox 消息正文内容	string / VNode	—	—
dangerouslyUseHTMLString	是否将 message 属性作为 HTML 片段处理	boolean	—	false
			success / info	

type	标	string	/ warning / error	—
iconClass	自定义图标的类名，会覆盖 `type`	string	—	—
customClass	MessageBox 的自定义类名	string	—	—
callback	若不使用 Promise，可以使用此参数指定 MessageBox 关闭后的回调	function(action, instance)，action 的值为'confirm'，'cancel'或'close'，instance 为 MessageBox 实例，可以通过它访问实例上的属性和方法	—	—
showClose	MessageBox 是否显示右上角关闭按钮	boolean	—	true
beforeClose	MessageBox 关闭前的回调，会暂停实例的关闭	function(action, instance, done)，action 的值为'confirm'，'cancel'或'close'；instance 为 MessageBox 实例，可以通过它访问实例上的属性和方法；done 用于关闭 MessageBox 实例	—	—
distinguishCancelAndClose	是否将取消（点击取消按钮）与关闭（点击关闭按钮或遮罩层、按下 ESC 键）进行区分	boolean	—	false
lockScroll	是否在 MessageBox 出现时将 body 滚动锁定	boolean	—	true
showCancelButton	是否显示取消按钮	boolean	—	false confi 和 promp 方式调 为 tr
showConfirmButton	是否显示确定按钮	boolean	—	true
cancelButtonText	取消按钮的文本内容	string	—	取消
confirmButtonText	确定按钮的文本内容	string	—	确定
cancelButtonClass	取消按钮的自定义类名	string	—	—
confirmButtonClass	确定按钮的自定义类名	string	—	—
closeOnClickModal	是否可通过点击遮罩关闭 MessageBox	boolean	—	true alert 式调用 false
closeOnPressEscape	是否可通过按下 ESC 键关闭 MessageBox	boolean	—	true alert 式调用 false

closeOnHashChange	是否在 hashchange 时关闭 MessageBox	boolean	—	true
showInput	是否显示输入框	boolean	—	false prompt 方式调为 tr
inputPlaceholder	输入框的占位符	string	—	—
inputType	输入框的类型	string	—	text
inputValue	输入框的初始文本	string	—	—
inputPattern	输入框的校验表达式	regexp	—	—
inputValidator	输入框的校验函数。可以返回布尔值或字符串，若返回一个字符串，则返回结果会被赋值给 inputErrorMessage	function	—	—
inputErrorMessage	校验未通过时的提示文本	string	—	输入的不合法
center	是否居中布局	boolean	—	false
roundButton	是否使用圆角按钮	boolean	—	false

Notification 通知

悬浮出现在页面角落，显示全局的通知提醒消息。

基本用法

适用性广泛的通知栏

Notification 组件提供通知功能，Element Plus 注册了 `$notify` 方法，接收一个 `options` 字面量参数，在最简单的情况下，你可以设置 `title` 字段和 `message` 字段，用于设置通知的标题和正文。默认情况下，经过一段时间后 Notification 组件会自动关闭，但是通过设置 `duration` ，可以控制关闭的时间间隔，特别的是，如果设置为 `0` ，则不会自动关闭。注意： `duration` 接收一个 `Number` ，单位为毫秒，默认为 `4500` 。

```
<template>
  <el-button
    plain
    @click="open1">
    可自动关闭
  </el-button>
  <el-button
    plain
    @click="open2">
    不会自动关闭
    </el-button>
</template>

<script>
  import { h } from 'vue';

  export default {
    methods: {
      open1() {
        this.$notify({
          title: '标题名称',
          message: h('i', { style: 'color: teal'}, '这是提示文案这是提示文案这是提示文案这是提示文案这是提示文案这是提示文案这是提示文案这是提示文案')
```

```
        });
      },

      open2() {
        this.$notify({
          title: '提示',
          message: '这是一条不会自动关闭的消息',
          duration: 0
        });
      }
    }
  }
</script>
```

带有倾向性

带有 icon，常用来显示「成功、警告、消息、错误」类的系统消息

Element Plus 为 Notification 组件准备了四种通知类型： `success` ， `warning` ， `info` ， `error` 。通过 `type` 字段来设置，除此以外的值将被忽略。同时，我们也为 Notification 的各种 type 注册了方法，可以在不传入 `type` 字段的情况下像 `open3` 和 `open4` 那样直接调用。

```
<template>
  <el-button
    plain
    @click="open1">
    成功
  </el-button>
  <el-button
    plain
    @click="open2">
    警告
  </el-button>
  <el-button
    plain
    @click="open3">
    消息
```

```
16.     </el-button>
17.     <el-button
18.       plain
19.       @click="open4">
20.       错误
21.     </el-button>
22.   </template>
23.
24.   <script>
25.     export default {
26.       methods: {
27.         open1() {
28.           this.$notify({
29.             title: '成功',
30.             message: '这是一条成功的提示消息',
31.             type: 'success'
32.           });
33.         },
34.
35.         open2() {
36.           this.$notify({
37.             title: '警告',
38.             message: '这是一条警告的提示消息',
39.             type: 'warning'
40.           });
41.         },
42.
43.         open3() {
44.           this.$notify.info({
45.             title: '消息',
46.             message: '这是一条消息的提示消息'
47.           });
48.         },
49.
50.         open4() {
51.           this.$notify.error({
52.             title: '错误',
53.             message: '这是一条错误的提示消息'
54.           });
55.         }
56.       }
57.     }
```

```
</script>
```

自定义弹出位置

可以让 Notification 从屏幕四角中的任意一角弹出

使用 `position` 属性定义 Notification 的弹出位置，支持四个选项： `top-right` 、 `top-left` 、 `bottom-right` 、 `bottom-left` ，默认为 `top-right` 。

```
<template>
  <el-button
    plain
    @click="open1">
    右上角
  </el-button>
  <el-button
    plain
    @click="open2">
    右下角
  </el-button>
  <el-button
    plain
    @click="open3">
    左下角
  </el-button>
  <el-button
    plain
    @click="open4">
    左上角
  </el-button>
</template>

<script>
  export default {
    methods: {
      open1() {
        this.$notify({
          title: '自定义位置',
```

```
          message: '右上角弹出的消息'
        });
      },

      open2() {
        this.$notify({
          title: '自定义位置',
          message: '右下角弹出的消息',
          position: 'bottom-right'
        });
      },

      open3() {
        this.$notify({
          title: '自定义位置',
          message: '左下角弹出的消息',
          position: 'bottom-left'
        });
      },

      open4() {
        this.$notify({
          title: '自定义位置',
          message: '左上角弹出的消息',
          position: 'top-left'
        });
      }
    }
  }
</script>
```

带有偏移

让 Notification 偏移一些位置

Notification 提供设置偏移量的功能，通过设置 `offset` 字段，可以使弹出的消息距屏幕边缘偏移一段距离。注意在同一时刻，所有的 Notification 实例应当具有一个相同的偏移量。

```
<template>
  <el-button
    plain
    @click="open">
    偏移的消息
  </el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$notify({
          title: '偏移',
          message: '这是一条带有偏移的提示消息',
          offset: 100
        });
      }
    }
  }
</script>
```

使用 HTML 片段

`message` 属性支持传入 HTML 片段

将 `dangerouslyUseHTMLString` 属性设置为 true，`message` 就会被当作 HTML 片段处理。

```
<template>
  <el-button
    plain
    @click="open">
    使用 HTML 片段
  </el-button>
</template>

<script>
  export default {
```

```
    methods: {
      open() {
        this.$notify({
          title: 'HTML 片段',
          dangerouslyUseHTMLString: true,
          message: '<strong>这是 <i>HTML</i> 片段</strong>'
        });
      }
    }
  }
</script>
```

`message` 属性虽然支持传入 HTML 片段，但是在网站上动态渲染任意 HTML 是非常危险的，因为容易导致 XSS 攻击。因此在 `dangerouslyUseHTMLString` 打开的情况下，请确保 `message` 的内容是可信的，永远不要将用户提交的内容赋值给 `message` 属性。

隐藏关闭按钮

可以不显示关闭按钮

将 `showClose` 属性设置为 `false` 即可隐藏关闭按钮。

```
<template>
  <el-button
    plain
    @click="open">
    隐藏关闭按钮
  </el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$notify.success({
          title: 'Info',
          message: '这是一条没有关闭按钮的消息',
          showClose: false
```

```
        });
      }
    }
  }
</script>
```

全局方法

Element Plus 为 `app.config.globalProperties` 添加了全局方法 `$notify` 。因此在 vue instance 中可以采用本页面中的方式调用 Notification。

单独引用

```
import { ElNotification } from 'element-plus';
```

此时调用方法为 `ElNotification(options)` 。我们也为每个 type 定义了各自的方法，如 `ElNotification.success(options)` 。并且可以调用 `ElNotification.closeAll()` 手动关闭所有实例。

Options

参数	说明	类型	可选值
title	标题	string	—
message	说明文字	string/Vue.VNode	—
dangerouslyUseHTMLString	是否将 message 属性作为 HTML 片段处理	boolean	—
type	主题样式，如果不在可选值内将被忽略	string	success/warning/info/er
iconClass	自定义图标的类名。若设置了 `type` ，则 `iconClass` 会被覆盖	string	—
customClass	自定义类名	string	—
duration	显示时间，毫秒。设为 0 则不会自动关闭	number	—
position	自定义弹出位置	string	top-right/top-left/bott right/bottom-left
showClose	是否显示关闭按钮	boolean	—
onClose	关闭时的回调函	function	—

onClose	数	function	—
onClick	点击 Notification 时的回调函数	function	—
offset	偏移的距离，在同一时刻，所有的 Notification 实例应当具有一个相同的偏移量	number	—

方法

调用 `Notification` 或 `this.$notify` 会返回当前 Notification 的实例。如果需要手动关闭实例，可以调用它的 `close` 方法。

方法名	说明
close	关闭当前的 Notification

- NavMenu 导航菜单
- Tabs 标签页
- Breadcrumb 面包屑
- PageHeader 页头
- Dropdown 下拉菜单
- Steps 步骤条

NavMenu 导航菜单

为网站提供导航功能的菜单。

顶栏

适用广泛的基础用法。

导航菜单默认为垂直模式，通过 `mode` 属性可以使导航菜单变更为水平模式。另外，在菜单中通过 `submenu` 组件可以生成二级菜单。Menu 还提供了 `background-color` 、 `text-color` 和 `active-text-color` ，分别用于设置菜单的背景色、菜单的文字颜色和当前激活菜单的文字颜色。

```
    <el-menu :default-active="activeIndex" class="el-menu-demo" mode="horizontal"
1.  @select="handleSelect">
2.    <el-menu-item index="1">处理中心</el-menu-item>
3.    <el-submenu index="2">
4.      <template #title>我的工作台</template>
5.      <el-menu-item index="2-1">选项1</el-menu-item>
6.      <el-menu-item index="2-2">选项2</el-menu-item>
7.      <el-menu-item index="2-3">选项3</el-menu-item>
8.      <el-submenu index="2-4">
9.        <template #title>选项4</template>
10.       <el-menu-item index="2-4-1">选项1</el-menu-item>
11.       <el-menu-item index="2-4-2">选项2</el-menu-item>
12.       <el-menu-item index="2-4-3">选项3</el-menu-item>
13.     </el-submenu>
14.   </el-submenu>
15.   <el-menu-item index="3" disabled>消息中心</el-menu-item>
      <el-menu-item index="4"><a href="https://www.ele.me" target="_blank">订单管
16. 理</a></el-menu-item>
17. </el-menu>
```

```
<div class="line"></div>
<el-menu
  :default-active="activeIndex2"
  class="el-menu-demo"
  mode="horizontal"
  @select="handleSelect"
  background-color="#545c64"
  text-color="#fff"
  active-text-color="#ffd04b">
  <el-menu-item index="1">处理中心</el-menu-item>
  <el-submenu index="2">
    <template #title>我的工作台</template>
    <el-menu-item index="2-1">选项1</el-menu-item>
    <el-menu-item index="2-2">选项2</el-menu-item>
    <el-menu-item index="2-3">选项3</el-menu-item>
    <el-submenu index="2-4">
      <template #title>选项4</template>
      <el-menu-item index="2-4-1">选项1</el-menu-item>
      <el-menu-item index="2-4-2">选项2</el-menu-item>
      <el-menu-item index="2-4-3">选项3</el-menu-item>
    </el-submenu>
  </el-submenu>
  <el-menu-item index="3" disabled>消息中心</el-menu-item>
  <el-menu-item index="4"><a href="https://www.ele.me" target="_blank">订单管
理</a></el-menu-item>
</el-menu>

<script>
  export default {
    data() {
      return {
        activeIndex: '1',
        activeIndex2: '1'
      };
    },
    methods: {
      handleSelect(key, keyPath) {
        console.log(key, keyPath);
      }
    }
  }
</script>
```

侧栏

垂直菜单，可内嵌子菜单。

通过 `el-menu-item-group` 组件可以实现菜单进行分组，分组名可以通过 `title` 属性直接设定，也可以通过具名 slot 来设定。

```
<el-row class="tac">
  <el-col :span="12">
    <h5>默认颜色</h5>
    <el-menu
      default-active="2"
      class="el-menu-vertical-demo"
      @open="handleOpen"
      @close="handleClose">
      <el-submenu index="1">
        <template #title>
          <i class="el-icon-location"></i>
          <span>导航一</span>
        </template>
        <el-menu-item-group>
          <template #title>分组一</template>
          <el-menu-item index="1-1">选项1</el-menu-item>
          <el-menu-item index="1-2">选项2</el-menu-item>
        </el-menu-item-group>
```

```
19.         <el-menu-item-group title="分组2">
20.           <el-menu-item index="1-3">选项3</el-menu-item>
21.         </el-menu-item-group>
22.         <el-submenu index="1-4">
23.           <template #title>选项4</template>
24.           <el-menu-item index="1-4-1">选项1</el-menu-item>
25.         </el-submenu>
26.       </el-submenu>
27.       <el-menu-item index="2">
28.         <i class="el-icon-menu"></i>
29.         <template #title>导航二</template>
30.       </el-menu-item>
31.       <el-menu-item index="3" disabled>
32.         <i class="el-icon-document"></i>
33.         <template #title>导航三</template>
34.       </el-menu-item>
35.       <el-menu-item index="4">
36.         <i class="el-icon-setting"></i>
37.         <template #title>导航四</template>
38.       </el-menu-item>
39.     </el-menu>
40.   </el-col>
41.   <el-col :span="12">
42.     <h5>自定义颜色</h5>
43.     <el-menu
44.       default-active="2"
45.       class="el-menu-vertical-demo"
46.       @open="handleOpen"
47.       @close="handleClose"
48.       background-color="#545c64"
49.       text-color="#fff"
50.       active-text-color="#ffd04b">
51.       <el-submenu index="1">
52.         <template #title>
53.           <i class="el-icon-location"></i>
54.           <span>导航一</span>
55.         </template>
56.         <el-menu-item-group>
57.           <template #title>分组一</template>
58.           <el-menu-item index="1-1">选项1</el-menu-item>
59.           <el-menu-item index="1-2">选项2</el-menu-item>
60.         </el-menu-item-group>
```

```
        <el-menu-item-group title="分组2">
          <el-menu-item index="1-3">选项3</el-menu-item>
        </el-menu-item-group>
        <el-submenu index="1-4">
          <template #title>选项4</template>
          <el-menu-item index="1-4-1">选项1</el-menu-item>
        </el-submenu>
      </el-submenu>
      <el-menu-item index="2">
        <i class="el-icon-menu"></i>
        <template #title>导航二</template>
      </el-menu-item>
      <el-menu-item index="3" disabled>
        <i class="el-icon-document"></i>
        <template #title>导航三</template>
      </el-menu-item>
      <el-menu-item index="4">
        <i class="el-icon-setting"></i>
        <template #title>导航四</template>
      </el-menu-item>
    </el-menu>
  </el-col>
</el-row>

<script>
  export default {
    methods: {
      handleOpen(key, keyPath) {
        console.log(key, keyPath);
      },
      handleClose(key, keyPath) {
        console.log(key, keyPath);
      }
    }
  }
</script>
```

折叠


```
<el-radio-group v-model="isCollapse" style="margin-bottom: 20px;">
  <el-radio-button :label="false">展开</el-radio-button>
  <el-radio-button :label="true">收起</el-radio-button>
</el-radio-group>
<el-menu default-active="1-4-1" class="el-menu-vertical-demo"
@open="handleOpen" @close="handleClose" :collapse="isCollapse">
  <el-submenu index="1">
    <template #title>
      <i class="el-icon-location"></i>
      <span>导航一</span>
    </template>
    <el-menu-item-group>
      <template #title>分组一</template>
      <el-menu-item index="1-1">选项1</el-menu-item>
      <el-menu-item index="1-2">选项2</el-menu-item>
    </el-menu-item-group>
    <el-menu-item-group title="分组2">
      <el-menu-item index="1-3">选项3</el-menu-item>
    </el-menu-item-group>
    <el-submenu index="1-4">
      <template #title>选项4</template>
      <el-menu-item index="1-4-1">选项1</el-menu-item>
    </el-submenu>
  </el-submenu>
  <el-menu-item index="2">
    <i class="el-icon-menu"></i>
    <template #title>导航二</template>
  </el-menu-item>
  <el-menu-item index="3" disabled>
    <i class="el-icon-document"></i>
```

```
      <template #title>导航三</template>
    </el-menu-item>
    <el-menu-item index="4">
      <i class="el-icon-setting"></i>
      <template #title>导航四</template>
    </el-menu-item>
  </el-menu>

  <style>
    .el-menu-vertical-demo:not(.el-menu--collapse) {
      width: 200px;
      min-height: 400px;
    }
  </style>

  <script>
    export default {
      data() {
        return {
          isCollapse: true
        };
      },
      methods: {
        handleOpen(key, keyPath) {
          console.log(key, keyPath);
        },
        handleClose(key, keyPath) {
          console.log(key, keyPath);
        }
      }
    }
  </script>
```

Menu Attribute

参数	说明	类型	可选值	默认值
mode	模式	string	horizontal / vertical	vertical
collapse	是否水平折叠收起菜单（仅在 mode 为 vertical 时可用）	boolean	—	false
background-color	菜单的背景色（仅支持 hex 格式）	string	—	#ffffff
text-color	菜单的文字颜色（仅支持 hex 格式）	string	—	#303133

active-text-color	当前激活菜单的文字颜色（仅支持 hex 格式）	string	—	#409EFF
default-active	当前激活菜单的 index	string	—	—
default-openeds	当前打开的 sub-menu 的 index 的数组	Array	—	—
unique-opened	是否只保持一个子菜单的展开	boolean	—	false
menu-trigger	子菜单打开的触发方式(只在 mode 为 horizontal 时有效)	string	hover / click	hover
router	是否使用 vue-router 的模式，启用该模式会在激活导航时以 index 作为 path 进行路由跳转	boolean	—	false
collapse-transition	是否开启折叠动画	boolean	—	true

Menu Methods

方法名称	说明	参数
open	展开指定的 sub-menu	index: 需要打开的 sub-menu 的 index
close	收起指定的 sub-menu	index: 需要收起的 sub-menu 的 index

Menu Events

事件名称	说明	回调参数
select	菜单激活回调	index: 选中菜单项的 index, indexPath: 选中菜单项的 index path
open	sub-menu 展开的回调	index: 打开的 sub-menu 的 index, indexPath: 打开的 sub-menu 的 index path
close	sub-menu 收起的回调	index: 收起的 sub-menu 的 index, indexPath: 收起的 sub-menu 的 index path

SubMenu Attribute

参数	说明	类型	可选值	默认值
index	唯一标志	string/null	—	null
popper-class	弹出菜单的自定义类名	string	—	—
show-timeout	展开 sub-menu 的延时	number	—	300
hide-timeout	收起 sub-menu 的延时	number	—	300
disabled	是否禁用	boolean	—	false

popper-append-to-body	是否将弹出菜单插入至 body 元素。在菜单的定位出现问题时，可尝试修改该属性	boolean	—	一级子菜单: true / 非一级子菜单: false

Menu-Item Attribute

参数	说明	类型	可选值	默认值
index	唯一标志	string	—	—
route	Vue Router 路径对象	Object	—	—
disabled	是否禁用	boolean	—	false

Menu-Group Attribute

参数	说明	类型	可选值	默认值
title	分组标题	string	—	—

Tabs 标签页

分隔内容上有关联但属于不同类别的数据集合。

基础用法

基础的、简洁的标签页。

Tabs 组件提供了选项卡功能，默认选中第一个标签页，你也可以通过 `value` 属性来指定当前选中的标签页。

```
<template>
  <el-tabs v-model="activeName" @tab-click="handleClick">
    <el-tab-pane label="用户管理" name="first">用户管理</el-tab-pane>
    <el-tab-pane label="配置管理" name="second">配置管理</el-tab-pane>
    <el-tab-pane label="角色管理" name="third">角色管理</el-tab-pane>
    <el-tab-pane label="定时任务补偿" name="fourth">定时任务补偿</el-tab-pane>
  </el-tabs>
</template>
<script>
  export default {
    data() {
      return {
        activeName: 'second'
      };
    },
    methods: {
      handleClick(tab, event) {
        console.log(tab, event);
      }
    }
  };
</script>
```

选项卡样式

选项卡样式的标签页。

只需要设置 `type` 属性为 `card` 就可以使选项卡改变为标签风格。

```
<template>
  <el-tabs v-model="activeName" type="card" @tab-click="handleClick">
    <el-tab-pane label="用户管理" name="first">用户管理</el-tab-pane>
    <el-tab-pane label="配置管理" name="second">配置管理</el-tab-pane>
    <el-tab-pane label="角色管理" name="third">角色管理</el-tab-pane>
    <el-tab-pane label="定时任务补偿" name="fourth">定时任务补偿</el-tab-pane>
  </el-tabs>
</template>
<script>
  export default {
    data() {
      return {
        activeName: 'first'
      };
    },
    methods: {
      handleClick(tab, event) {
        console.log(tab, event);
      }
    }
  };
</script>
```

卡片化

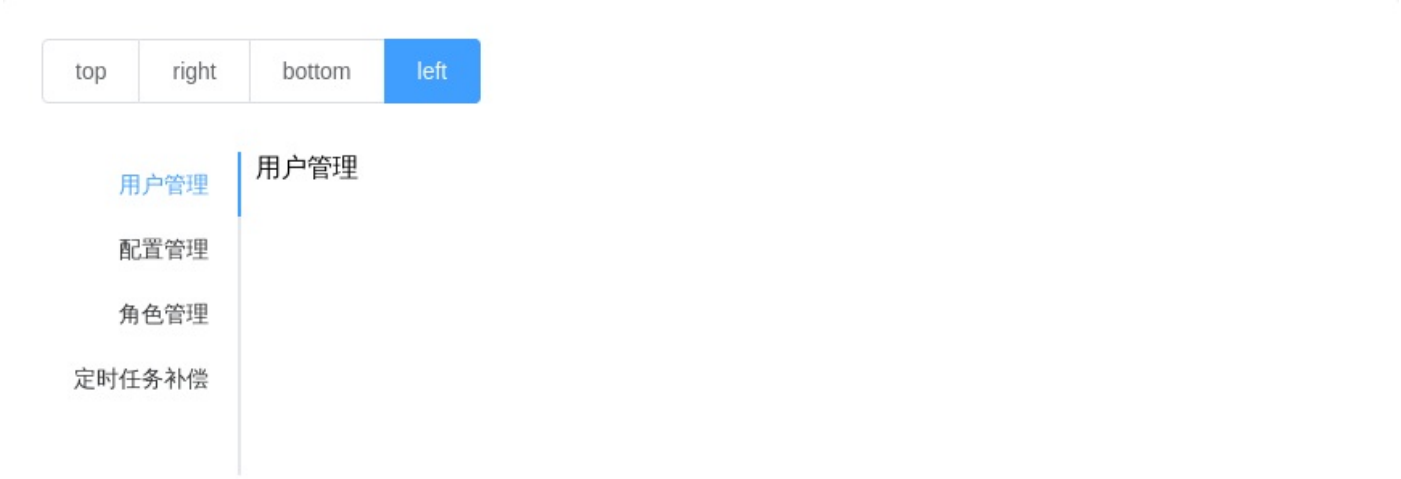
卡片化的标签页。

将 `type` 设置为 `border-card` 。

```
<el-tabs type="border-card">
  <el-tab-pane label="用户管理">用户管理</el-tab-pane>
  <el-tab-pane label="配置管理">配置管理</el-tab-pane>
  <el-tab-pane label="角色管理">角色管理</el-tab-pane>
  <el-tab-pane label="定时任务补偿">定时任务补偿</el-tab-pane>
</el-tabs>
```

位置

可以通过 `tab-position` 设置标签的位置



标签一共有四个方向的设置 `tabPosition="left|right|top|bottom"`

```
<template>
  <el-radio-group v-model="tabPosition" style="margin-bottom: 30px;">
    <el-radio-button label="top">top</el-radio-button>
    <el-radio-button label="right">right</el-radio-button>
    <el-radio-button label="bottom">bottom</el-radio-button>
    <el-radio-button label="left">left</el-radio-button>
  </el-radio-group>

  <el-tabs :tab-position="tabPosition" style="height: 200px;">
    <el-tab-pane label="用户管理">用户管理</el-tab-pane>
    <el-tab-pane label="配置管理">配置管理</el-tab-pane>
    <el-tab-pane label="角色管理">角色管理</el-tab-pane>
    <el-tab-pane label="定时任务补偿">定时任务补偿</el-tab-pane>
  </el-tabs>
</template>
```

```
<script>
  export default {
    data() {
      return {
        tabPosition: 'left'
      };
    }
  };
</script>
```

自定义标签页

可以通过具名 `slot` 来实现自定义标签页的内容


```
<el-tabs type="border-card">
  <el-tab-pane>
    <template #label>
      <span><i class="el-icon-date"></i> 我的行程</span>
    </template>
    我的行程
  </el-tab-pane>
  <el-tab-pane label="消息中心">消息中心</el-tab-pane>
  <el-tab-pane label="角色管理">角色管理</el-tab-pane>
  <el-tab-pane label="定时任务补偿">定时任务补偿</el-tab-pane>
</el-tabs>
```

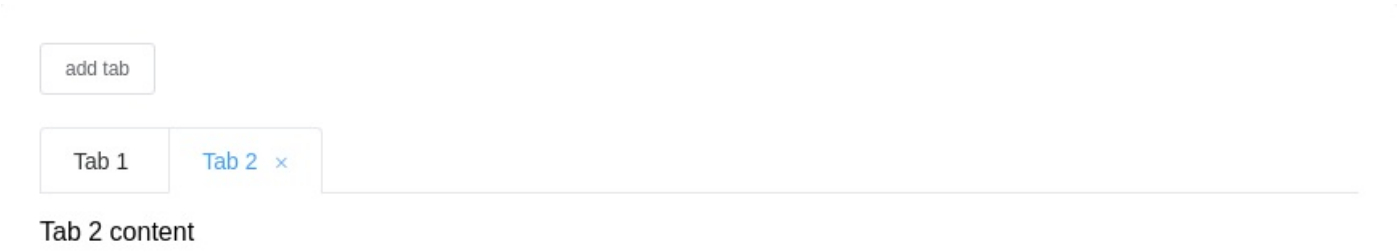
动态增减标签页

增减标签页按钮只能在选项卡样式的标签页下使用

```
<el-tabs v-model="editableTabsValue" type="card" editable
@edit="handleTabsEdit">
  <el-tab-pane
    :key="item.name"
    v-for="(item, index) in editableTabs"
    :label="item.title"
    :name="item.name"
  >
    {{item.content}}
  </el-tab-pane>
</el-tabs>
<script>
  export default {
    data() {
      return {
        editableTabsValue: '2',
        editableTabs: [{
          title: 'Tab 1',
          name: '1',
          content: 'Tab 1 content'
        }, {
          title: 'Tab 2',
          name: '2',
          content: 'Tab 2 content'
        }],
        tabIndex: 2
      }
    },
    methods: {
      handleTabsEdit(targetName, action) {
        if (action === 'add') {
          let newTabName = ++this.tabIndex + '';
          this.editableTabs.push({
            title: 'New Tab',
            name: newTabName,
            content: 'New Tab content'
          });
          this.editableTabsValue = newTabName;
        }
        if (action === 'remove') {
          let tabs = this.editableTabs;
          let activeName = this.editableTabsValue;
```

```
          if (activeName === targetName) {
            tabs.forEach((tab, index) => {
              if (tab.name === targetName) {
                let nextTab = tabs[index + 1] || tabs[index - 1];
                if (nextTab) {
                  activeName = nextTab.name;
                }
              }
            });
          }

          this.editableTabsValue = activeName;
          this.editableTabs = tabs.filter(tab => tab.name !== targetName);
        }
      }
    }
  }
</script>
```

自定义增加标签页触发器




```
<div style="margin-bottom: 20px;">
  <el-button
    size="small"
    @click="addTab(editableTabsValue)"
  >
    add tab
  </el-button>
</div>
<el-tabs v-model="editableTabsValue" type="card" closable @tab-
remove="removeTab">
  <el-tab-pane
    v-for="(item, index) in editableTabs"
    :key="item.name"
    :label="item.title"
```

```
    :name="item.name"
  >
    {{item.content}}
  </el-tab-pane>
</el-tabs>
<script>
  export default {
    data() {
      return {
        editableTabsValue: '2',
        editableTabs: [{
          title: 'Tab 1',
          name: '1',
          content: 'Tab 1 content'
        }, {
          title: 'Tab 2',
          name: '2',
          content: 'Tab 2 content'
        }],
        tabIndex: 2
      }
    },
    methods: {
      addTab(targetName) {
        let newTabName = ++this.tabIndex + '';
        this.editableTabs.push({
          title: 'New Tab',
          name: newTabName,
          content: 'New Tab content'
        });
        this.editableTabsValue = newTabName;
      },
      removeTab(targetName) {
        let tabs = this.editableTabs;
        let activeName = this.editableTabsValue;
        if (activeName === targetName) {
          tabs.forEach((tab, index) => {
            if (tab.name === targetName) {
              let nextTab = tabs[index + 1] || tabs[index - 1];
              if (nextTab) {
                activeName = nextTab.name;
              }
```

```
            }
          });
        }

        this.editableTabsValue = activeName;
        this.editableTabs = tabs.filter(tab => tab.name !== targetName);
      }
    }
  }
</script>
```

Tabs Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值，选中选项卡的 name	string	—	第一个选项卡的 name
type	风格类型	string	card/border-card	—
closable	标签是否可关闭	boolean	—	false
addable	标签是否可增加	boolean	—	false
editable	标签是否同时可增加和关闭	boolean	—	false
tab-position	选项卡所在位置	string	top/right/bottom/left	top
stretch	标签的宽度是否自撑开	boolean	-	false
before-leave	切换标签之前的钩子，若返回 false 或者返回 Promise 且被 reject，则阻止切换。	Function(activeName, oldActiveName)	—	—

Tabs Events

事件名称	说明	回调参数
tab-click	tab 被选中时触发	被选中的标签 tab 实例
tab-remove	点击 tab 移除按钮后触发	被删除的标签的 name
tab-add	点击 tabs 的新增按钮后触发	—
edit	点击 tabs 的新增按钮或 tab 被关闭后触发	(targetName, action)

Tab-pane Attributes

			可	

参数	说明	类型	选值	默认值
label	选项卡标题	string	—	—
disabled	是否禁用	boolean	—	false
name	与选项卡绑定值 value 对应的标识符，表示选项卡别名	string	—	该选项卡在选项卡列表中的顺序值，如第一个选项卡则为'1'
closable	标签是否可关闭	boolean	—	false
lazy	标签是否延迟渲染	boolean	—	false

Breadcrumb 面包屑

显示当前页面的路径，快速返回之前的任意页面。

基础用法

适用广泛的基础用法。

在 `el-breadcrumb` 中使用 `el-breadcrumb-item` 标签表示从首页开始的每一级。Element Plus 提供了一个 `separator` 属性，在 `el-breadcrumb` 标签中设置它来决定分隔符，它只能是字符串，默认为斜杠 `/` 。

```
<el-breadcrumb separator="/">
  <el-breadcrumb-item :to="{ path: '/' }">首页</el-breadcrumb-item>
  <el-breadcrumb-item><a href="/">活动管理</a></el-breadcrumb-item>
  <el-breadcrumb-item>活动列表</el-breadcrumb-item>
  <el-breadcrumb-item>活动详情</el-breadcrumb-item>
</el-breadcrumb>
```

图标分隔符

通过设置 `separator-class` 可使用相应的 `iconfont` 作为分隔符，注意这将使 `separator` 设置失效

```
<el-breadcrumb separator-class="el-icon-arrow-right">
  <el-breadcrumb-item :to="{ path: '/' }">首页</el-breadcrumb-item>
  <el-breadcrumb-item>活动管理</el-breadcrumb-item>
  <el-breadcrumb-item>活动列表</el-breadcrumb-item>
  <el-breadcrumb-item>活动详情</el-breadcrumb-item>
</el-breadcrumb>
```

Breadcrumb Attributes

参数	说明	类型	可选值	默认值

separator	分隔符	string	—	斜杠'/'
separator-class	图标分隔符 class	string	—	-

Breadcrumb Item Attributes

参数	说明	类型	可选值	默认值
to	路由跳转对象，同 `vue-router` 的 `to`	string/object	—	—
replace	在使用 to 进行路由跳转时，启用 replace 将不会向 history 添加新记录	boolean	—	false

PageHeader 页头

如果页面的路径比较简单，推荐使用页头组件而非面包屑组件。

基础


```
<el-page-header @back="goBack" content="详情页面">
</el-page-header>

<script>
  export default {
    methods: {
      goBack() {
        console.log('go back');
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
title	标题	string	—	返回
content	内容	string	—	—

Events

事件名称	说明	回调参数
back	点击左侧区域触发	—

Slots

事件名称	说明
title	标题内容
content	内容

Dropdown 下拉菜单

将动作或菜单折叠到下拉菜单中。

基础用法

移动到下拉菜单上，展开更多操作。

通过组件 `slot` 来设置下拉触发的元素以及需要通过具名 `slot` 为 `dropdown` 来设置下拉菜单。默认情况下，下拉按钮只要 `hover` 即可，无需点击也会显示下拉菜单。

```
<el-dropdown>
  <span class="el-dropdown-link">
    下拉菜单<i class="el-icon-arrow-down el-icon--right"></i>
  </span>
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item>黄金糕</el-dropdown-item>
      <el-dropdown-item>狮子头</el-dropdown-item>
      <el-dropdown-item>螺蛳粉</el-dropdown-item>
      <el-dropdown-item disabled>双皮奶</el-dropdown-item>
      <el-dropdown-item divided>蚵仔煎</el-dropdown-item>
    </el-dropdown-menu>
  </template>
</el-dropdown>

<style>
  .el-dropdown-link {
    cursor: pointer;
    color: #409EFF;
  }
  .el-icon-arrow-down {
    font-size: 12px;
  }
</style>
```

触发对象

可使用按钮触发下拉菜单。

设置 `split-button` 属性来让触发下拉元素呈现为按钮组，左边是功能按钮，右边是触发下拉菜单的按钮，设置为 `true` 即可。

```
<el-dropdown>
  <el-button type="primary">
    更多菜单<i class="el-icon-arrow-down el-icon--right"></i>
  </el-button>
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item>黄金糕</el-dropdown-item>
      <el-dropdown-item>狮子头</el-dropdown-item>
      <el-dropdown-item>螺蛳粉</el-dropdown-item>
      <el-dropdown-item>双皮奶</el-dropdown-item>
      <el-dropdown-item>蚵仔煎</el-dropdown-item>
    </el-dropdown-menu>
  </template>
</el-dropdown>
<el-dropdown split-button type="primary" @click="handleClick">
  更多菜单
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item>黄金糕</el-dropdown-item>
      <el-dropdown-item>狮子头</el-dropdown-item>
      <el-dropdown-item>螺蛳粉</el-dropdown-item>
      <el-dropdown-item>双皮奶</el-dropdown-item>
      <el-dropdown-item>蚵仔煎</el-dropdown-item>
    </el-dropdown-menu>
  </template>
</el-dropdown>

<style>
  .el-dropdown {
    vertical-align: top;
  }
  .el-dropdown + .el-dropdown {
    margin-left: 15px;
```

```
  }
  .el-icon-arrow-down {
    font-size: 12px;
  }
</style>

<script>
  export default {
    methods: {
      handleClick() {
        alert('button click');
      }
    }
  }
</script>
```

触发方式

可以配置 `click` 激活或者 `hover` 激活。

在 `trigger` 属性设置为 `click` 即可。

```
<el-row class="block-col-2">
  <el-col :span="12">
    <span class="demonstration">hover 激活</span>
    <el-dropdown>
      <span class="el-dropdown-link">
        下拉菜单<i class="el-icon-arrow-down el-icon--right"></i>
      </span>
      <template #dropdown>
        <el-dropdown-menu>
          <el-dropdown-item icon="el-icon-plus">黄金糕</el-dropdown-item>
          <el-dropdown-item icon="el-icon-circle-plus">狮子头</el-dropdown-item>
          <el-dropdown-item icon="el-icon-circle-plus-outline">螺蛳粉</el-
dropdown-item>
          <el-dropdown-item icon="el-icon-check">双皮奶</el-dropdown-item>
```

```
            <el-dropdown-item icon="el-icon-circle-check">蚵仔煎</el-dropdown-
item>
        </el-dropdown-menu>
      </template>
    </el-dropdown>
  </el-col>
  <el-col :span="12">
    <span class="demonstration">click 激活</span>
    <el-dropdown trigger="click">
      <span class="el-dropdown-link">
        下拉菜单<i class="el-icon-arrow-down el-icon--right"></i>
      </span>
      <template #dropdown>
        <el-dropdown-menu>
          <el-dropdown-item icon="el-icon-plus">黄金糕</el-dropdown-item>
          <el-dropdown-item icon="el-icon-circle-plus">狮子头</el-dropdown-item>
          <el-dropdown-item icon="el-icon-circle-plus-outline">螺蛳粉</el-
dropdown-item>
          <el-dropdown-item icon="el-icon-check">双皮奶</el-dropdown-item>
          <el-dropdown-item icon="el-icon-circle-check">蚵仔煎</el-dropdown-
item>
        </el-dropdown-menu>
      </template>
    </el-dropdown>
  </el-col>
</el-row>

<style>
  .el-dropdown-link {
    cursor: pointer;
    color: #409EFF;
  }
  .el-icon-arrow-down {
    font-size: 12px;
  }
  .demonstration {
    display: block;
    color: #8492a6;
    font-size: 14px;
    margin-bottom: 20px;
  }
</style>
```

菜单隐藏方式

可以 `hide-on-click` 属性来配置。

下拉菜单默认在点击菜单项后会被隐藏，将 `hide-on-click` 属性默认为 `false` 可以关闭此功能。

```
<el-dropdown :hide-on-click="false">
  <span class="el-dropdown-link">
    下拉菜单<i class="el-icon-arrow-down el-icon--right"></i>
  </span>
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item>黄金糕</el-dropdown-item>
      <el-dropdown-item>狮子头</el-dropdown-item>
      <el-dropdown-item>螺蛳粉</el-dropdown-item>
      <el-dropdown-item disabled>双皮奶</el-dropdown-item>
      <el-dropdown-item divided>蚵仔煎</el-dropdown-item>
    </el-dropdown-menu>
  </template>
</el-dropdown>

<style>
  .el-dropdown-link {
    cursor: pointer;
    color: #409EFF;
  }
  .el-icon-arrow-down {
    font-size: 12px;
  }
</style>
```

指令事件

点击菜单项后会触发事件，用户可以通过相应的菜单项 key 进行不同的操作

```
<el-dropdown @command="handleCommand">
  <span class="el-dropdown-link">
    下拉菜单<i class="el-icon-arrow-down el-icon--right"></i>
  </span>
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item command="a">黄金糕</el-dropdown-item>
      <el-dropdown-item command="b">狮子头</el-dropdown-item>
      <el-dropdown-item command="c">螺蛳粉</el-dropdown-item>
      <el-dropdown-item command="d" disabled>双皮奶</el-dropdown-item>
      <el-dropdown-item command="e" divided>蚵仔煎</el-dropdown-item>
    </el-dropdown-menu>
  </template>
</el-dropdown>

<style>
  .el-dropdown-link {
    cursor: pointer;
    color: #409EFF;
  }
  .el-icon-arrow-down {
    font-size: 12px;
  }
</style>

<script>
  export default {
    methods: {
      handleCommand(command) {
        this.$message('click on item ' + command);
      }
    }
  }
</script>
```

不同尺寸

Dropdown 组件提供除了默认值以外的三种尺寸，可以在不同场景下选择合适的尺寸。

额外的尺寸： `medium` 、 `small` 、 `mini` ，通过设置 `size` 属性来配置它们。

```
<el-dropdown split-button type="primary">
  默认尺寸
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item>黄金糕</el-dropdown-item>
      <el-dropdown-item>狮子头</el-dropdown-item>
      <el-dropdown-item>螺蛳粉</el-dropdown-item>
      <el-dropdown-item>双皮奶</el-dropdown-item>
      <el-dropdown-item>蚵仔煎</el-dropdown-item>
    </el-dropdown-menu>
  </template>
</el-dropdown>

<el-dropdown size="medium" split-button type="primary">
  中等尺寸
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item>黄金糕</el-dropdown-item>
      <el-dropdown-item>狮子头</el-dropdown-item>
      <el-dropdown-item>螺蛳粉</el-dropdown-item>
      <el-dropdown-item>双皮奶</el-dropdown-item>
      <el-dropdown-item>蚵仔煎</el-dropdown-item>
    </el-dropdown-menu>
  </template>
</el-dropdown>

<el-dropdown size="small" split-button type="primary">
  小型尺寸
  <template #dropdown>
    <el-dropdown-menu>
      <el-dropdown-item>黄金糕</el-dropdown-item>
      <el-dropdown-item>狮子头</el-dropdown-item>
      <el-dropdown-item>螺蛳粉</el-dropdown-item>
      <el-dropdown-item>双皮奶</el-dropdown-item>
      <el-dropdown-item>蚵仔煎</el-dropdown-item>
    </el-dropdown-menu>
```

```
    </template>
  </el-dropdown>

  <el-dropdown size="mini" split-button type="primary">
    超小尺寸
    <template #dropdown>
      <el-dropdown-menu>
        <el-dropdown-item>黄金糕</el-dropdown-item>
        <el-dropdown-item>狮子头</el-dropdown-item>
        <el-dropdown-item>螺蛳粉</el-dropdown-item>
        <el-dropdown-item>双皮奶</el-dropdown-item>
        <el-dropdown-item>蚵仔煎</el-dropdown-item>
      </el-dropdown-menu>
    </template>
  </el-dropdown>
```

Dropdown Attributes

参数	说明	类型	可选值	默认值
type	菜单按钮类型，同 Button 组件（只在 split-button 为 true 的情况下有效）	string	—	—
size	菜单尺寸，在 split-button 为 true 的情况下也对触发按钮生效	string	medium / small / mini	—
split-button	下拉触发元素呈现为按钮组	boolean	—	false
placement	菜单弹出位置	string	top/top-start/top-end/bottom/bottom-start/bottom-end	bottom-end
trigger	触发下拉的行为	string	hover, click	hover
hide-on-click	是否在点击菜单项后隐藏菜单	boolean	—	true
show-timeout	展开下拉菜单的延时（仅在 trigger 为 hover 时有效）	number	—	250
hide-timeout	收起下拉菜单的延时（仅在 trigger 为 hover 时有效）	number	—	150
tabindex	Dropdown 组件的 tabindex	number	—	0

Dropdown Slots

Name	说明
—	触发下拉列表显示的元素。 注意： 必须是一个元素或者或者组件
dropdown	下拉列表，通常是 <el-dropdown-menu> 组件

Dropdown Events

事件名称	说明	回调参数
click	`split-button` 为 true 时，点击左侧按钮的回调	—
command	点击菜单项触发的事件回调	dropdown-item 的指令
visible-change	下拉框出现/隐藏时触发	出现则为 true，隐藏则为 false

Dropdown Menu Item Attributes

参数	说明	类型	可选值	默认值
command	指令	string/number/object	—	—
disabled	禁用	boolean	—	false
divided	显示分割线	boolean	—	false
icon	图标类名	string	—	—

Steps 步骤条

引导用户按照流程完成任务的分步导航条，可根据实际应用场景设定步骤，步骤不得少于 2 步。

基础用法

简单的步骤条。

设置 `active` 属性，接受一个 `Number` ，表明步骤的 index，从 0 开始。需要定宽的步骤条时，设置 `space` 属性即可，它接受 `Number` ，单位为 `px` ，如果不设置，则为自适应。设置 `finish-status` 属性可以改变已经完成的步骤的状态。

```
<el-steps :active="active" finish-status="success">
  <el-step title="步骤 1"></el-step>
  <el-step title="步骤 2"></el-step>
  <el-step title="步骤 3"></el-step>
</el-steps>

<el-button style="margin-top: 12px;" @click="next">下一步</el-button>

<script>
  export default {
    data() {
      return {
        active: 0
      };
    },

    methods: {
      next() {
        if (this.active++ > 2) this.active = 0;
      }
    }
  }
```

```
</script>
```

含状态步骤条

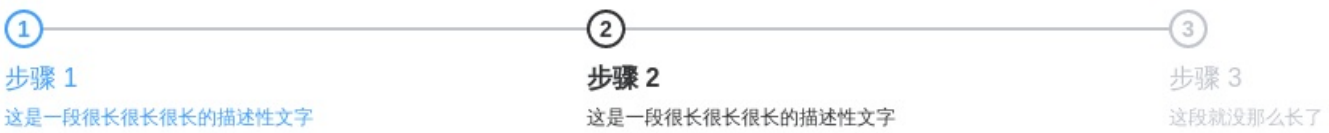
每一步骤显示出该步骤的状态。

也可以使用 `title` 具名分发，可以用 `slot` 的方式来取代属性的设置，在本文档最后的列表中有所有的 `slot name` 可供参考。

```
<el-steps :space="200" :active="1" finish-status="success">
  <el-step title="已完成"></el-step>
  <el-step title="进行中"></el-step>
  <el-step title="步骤 3"></el-step>
</el-steps>
```

有描述的步骤条

每个步骤有其对应的步骤状态描述。




```
<el-steps :active="1">
  <el-step title="步骤 1" description="这是一段很长很长很长的描述性文字"></el-step>
  <el-step title="步骤 2" description="这是一段很长很长很长的描述性文字"></el-step>
  <el-step title="步骤 3" description="这段就没那么长了"></el-step>
</el-steps>
```

居中的步骤条

标题和描述都将居中。


```
<el-steps :active="2" align-center>
  <el-step title="步骤1" description="这是一段很长很长很长的描述性文字"></el-step>
  <el-step title="步骤2" description="这是一段很长很长很长的描述性文字"></el-step>
  <el-step title="步骤3" description="这是一段很长很长很长的描述性文字"></el-step>
  <el-step title="步骤4" description="这是一段很长很长很长的描述性文字"></el-step>
</el-steps>
```

带图标的步骤条

步骤条内可以启用各种自定义的图标。

通过 `icon` 属性来设置图标，图标的类型可以参考 Icon 组件的文档，除此以外，还能通过具名 `slot` 来使用自定义的图标。

```

<el-steps :active="1">
  <el-step title="步骤 1" icon="el-icon-edit"></el-step>
  <el-step title="步骤 2" icon="el-icon-upload"></el-step>
  <el-step title="步骤 3" icon="el-icon-picture"></el-step>
</el-steps>
```

竖式步骤条

竖直方向的步骤条。

只需要在 `el-steps` 元素中设置 `direction` 属性为 `vertical` 即可。

```
<div style="height: 300px;">
  <el-steps direction="vertical" :active="1">
    <el-step title="步骤 1"></el-step>
    <el-step title="步骤 2"></el-step>
    <el-step title="步骤 3" description="这是一段很长很长很长的描述性文字"></el-step>
  </el-steps>
</div>
```

简洁风格的步骤条

设置 `simple` 可应用简洁风格，该条件下 `align-center` / `description` / `direction` / `space` 都将失效。


```

<el-steps :active="1" simple>
  <el-step title="步骤 1" icon="el-icon-edit"></el-step>
  <el-step title="步骤 2" icon="el-icon-upload"></el-step>
  <el-step title="步骤 3" icon="el-icon-picture"></el-step>
</el-steps>

```

```
<el-steps :active="1" finish-status="success" simple style="margin-top: 20px">
  <el-step title="步骤 1" ></el-step>
  <el-step title="步骤 2" ></el-step>
  <el-step title="步骤 3" ></el-step>
</el-steps>
```

Steps Attributes

参数	说明	类型	可选值	默认值
space	每个 step 的间距，不填写将自适应间距。支持百分比。	number / string	—	—
direction	显示方向	string	vertical/horizontal	horizontal
active	设置当前激活步骤	number	—	0
process-status	设置当前步骤的状态	string	wait / process / finish / error / success	process
finish-status	设置结束步骤的状态	string	wait / process / finish / error / success	finish
align-center	进行居中对齐	boolean	-	false
simple	是否应用简洁风格	boolean	-	false

Step Attributes

参数	说明	类型	可选值	默认值
title	标题	string	—	—
description	描述性文字	string	—	—
icon	图标	传入 icon 的 class 全名来自定义 icon，也支持 slot 方式写入	string	—
status	设置当前步骤的状态，不设置则根据 steps 确定状态	wait / process / finish / error / success	-	

Step Slot

name	说明
icon	自定义图标
title	自定义标题
description	自定义描述性文字

- Dialog 对话框
- Tooltip 文字提示
- Popover 弹出框
- Popconfirm 气泡确认框
- Card 卡片
- Carousel 走马灯
- Collapse 折叠面板
- Timeline 时间线
- Divider 分割线
- Calendar 日历
- Image 图片
- Backtop 回到顶部
- InfiniteScroll 无限滚动
- Drawer 抽屉

Dialog 对话框

在保留当前页面状态的情况下，告知用户并承载相关操作。

基本用法

Dialog 弹出一个对话框，适合需要定制性更大的场景。

需要设置 `visible` 属性，它接收 `Boolean` ，当为 `true` 时显示 Dialog。Dialog 分为两个部分： `body` 和 `footer` ， `footer` 需要具名为 `footer` 的 `slot` 。 `title` 属性用于定义标题，它是可选的，默认值为空。最后，本例还展示了 `before-close` 的用法。

```
<el-button type="text" @click="dialogVisible = true">点击打开 Dialog</el-button>

<el-dialog
  title="提示"
  v-model="dialogVisible"
  width="30%"
  :before-close="handleClose">
  <span>这是一段信息</span>
  <template #footer>
    <span class="dialog-footer">
      <el-button @click="dialogVisible = false">取 消</el-button>
      <el-button type="primary" @click="dialogVisible = false">确 定</el-
button>
    </span>
  </template>
</el-dialog>

<script>
  export default {
    data() {
      return {
        dialogVisible: false
      };
    },
    methods: {
```

```
      handleClose(done) {
        this.$confirm('确认关闭？')
          .then(_ => {
            done();
          })
          .catch(_ => {});
      }
    }
  };
</script>
```

`before-close` 仅当用户通过点击关闭图标或遮罩关闭 Dialog 时起效。如果你在 `footer` 具名 slot 里添加了用于关闭 Dialog 的按钮，那么可以在按钮的点击回调函数里加入 `before-close` 的相关逻辑。

自定义内容

Dialog 组件的内容可以是任意的，甚至可以是表格或表单，下面是应用了 Element Plus Table 和 Form 组件的两个样例。


```
<el-button type="text" @click="dialogTableVisible = true">打开嵌套表格的
Dialog</el-button>

<el-dialog title="收货地址" v-model="dialogTableVisible">
  <el-table :data="gridData">
    <el-table-column property="date" label="日期" width="150"></el-table-
column>
    <el-table-column property="name" label="姓名" width="200"></el-table-
column>
    <el-table-column property="address" label="地址"></el-table-column>
  </el-table>
</el-dialog>

<!-- Form -->
<el-button type="text" @click="dialogFormVisible = true">打开嵌套表单的
Dialog</el-button>

<el-dialog title="收货地址" v-model="dialogFormVisible">
```

```
    <el-form :model="form">
      <el-form-item label="活动名称" :label-width="formLabelWidth">
        <el-input v-model="form.name" autocomplete="off"></el-input>
      </el-form-item>
      <el-form-item label="活动区域" :label-width="formLabelWidth">
        <el-select v-model="form.region" placeholder="请选择活动区域">
          <el-option label="区域一" value="shanghai"></el-option>
          <el-option label="区域二" value="beijing"></el-option>
        </el-select>
      </el-form-item>
    </el-form>
    <template #footer>
      <span class="dialog-footer">
        <el-button @click="dialogFormVisible = false">取 消</el-button>
        <el-button type="primary" @click="dialogFormVisible = false">确 定</el-
button>
      </span>
    </template>
</el-dialog>

<script>
  export default {
    data() {
      return {
        gridData: [{
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-03',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }],
        dialogTableVisible: false,
```

```
        dialogFormVisible: false,
        form: {
          name: '',
          region: '',
          date1: '',
          date2: '',
          delivery: false,
          type: [],
          resource: '',
          desc: ''
        },
        formLabelWidth: '120px'
      };
    }
  };
</script>
```

嵌套的 Dialog

如果需要在一个 Dialog 内部嵌套另一个 Dialog，需要使用 `append-to-body` 属性。

正常情况下，我们不建议使用嵌套的 Dialog，如果需要在页面上同时显示多个 Dialog，可以将它们平级放置。对于确实需要嵌套 Dialog 的场景，我们提供了 `append-to-body` 属性。将内层 Dialog 的该属性设置为 true，它就会插入至 body 元素上，从而保证内外层 Dialog 和遮罩层级关系的正确。

```
<template>
  <el-button type="text" @click="outerVisible = true">点击打开外层 Dialog</el-
button>

  <el-dialog title="外层 Dialog" v-model="outerVisible">
    <el-dialog
      width="30%"
      title="内层 Dialog"
      v-model="innerVisible"
      append-to-body>
    </el-dialog>
    <template #footer>
```

```
      <div class="dialog-footer">
        <el-button @click="outerVisible = false">取 消</el-button>
        <el-button type="primary" @click="innerVisible = true">打开内层
Dialog</el-button>
      </div>
    </template>
  </el-dialog>
</template>

<script>
  export default {
    data() {
      return {
        outerVisible: false,
        innerVisible: false
      };
    }
  }
</script>
```

居中布局

标题和底部可水平居中

将 `center` 设置为 `true` 即可使标题和底部居中。 `center` 仅影响标题和底部区域。Dialog 的内容是任意的，在一些情况下，内容并不适合居中布局。如果需要内容也水平居中，请自行为其添加 CSS。

```
    <el-button type="text" @click="centerDialogVisible = true">点击打开 Dialog</el-
button>

<el-dialog
  title="提示"
  v-model="centerDialogVisible"
  width="30%"
  center>
  <span>需要注意的是内容是默认不居中的</span>
  <template #footer>
```

```
      <span class="dialog-footer">
        <el-button @click="centerDialogVisible = false">取 消</el-button>
        <el-button type="primary" @click="centerDialogVisible = false">确 定</el-
button>
      </span>
    </template>

</el-dialog>

<script>
  export default {
    data() {
      return {
        centerDialogVisible: false
      };
    }
  };
</script>
```

Dialog 的内容是懒渲染的，即在第一次被打开之前，传入的默认 slot 不会被渲染到 DOM 上。因此，如果需要执行 DOM 操作，或通过 `ref` 获取相应组件，请在 `open` 事件回调中进行。

如果 `visible` 属性绑定的变量位于 Vuex 的 store 内，那么 `.sync` 不会正常工作。此时需要去除 `.sync` 修饰符，同时监听 Dialog 的 `open` 和 `close` 事件，在事件回调中执行 Vuex 中对应的 mutation 更新 `visible` 属性绑定的变量的值。

Attributes

参数	说明	类型	可选值	默认值
model-value / v-model	是否显示 Dialog	boolean	—	—
title	Dialog 的标题，也可通过具名 slot （见下表）传入	string	—	—
width	Dialog 的宽度	string	—	50%
fullscreen	是否为全屏 Dialog	boolean	—	false
top	Dialog CSS 中的 margin-top 值	string	—	15vh
modal	是否需要遮罩层	boolean	—	true
append-to-body	Dialog 自身是否插入至 body 元素上。嵌套的 Dialog 必须指定该属性并赋值为 true	boolean	—	false
lock-				

scroll				
custom-class	Dialog 的自定义类名	string	—	—
open-delay	Dialog 打开的延时时间，单位毫秒	number	—	0
close-delay	Dialog 关闭的延时时间，单位毫秒	number	—	0
close-on-click-modal	是否可以通过点击 modal 关闭 Dialog	boolean	—	true
close-on-press-escape	是否可以通过按下 ESC 关闭 Dialog	boolean	—	true
show-close	是否显示关闭按钮	boolean	—	true
before-close	关闭前的回调，会暂停 Dialog 的关闭	function(done)，done 用于关闭 Dialog	—	—
center	是否对头部和底部采用居中布局	boolean	—	false
destroy-on-close	关闭时销毁 Dialog 中的元素	boolean	—	false

Slot

name	说明
—	Dialog 的内容
title	Dialog 标题区的内容
footer	Dialog 按钮操作区的内容

Events

事件名称	说明	回调参数
open	Dialog 打开的回调	—
opened	Dialog 打开动画结束时的回调	—
close	Dialog 关闭的回调	—
closed	Dialog 关闭动画结束时的回调	—

Tooltip 文字提示

常用于展示鼠标 hover 时的提示信息。

基础用法

在这里我们提供 9 种不同方向的展示方式，可以通过以下完整示例来理解，选择你要的效果。

使用 `content` 属性来决定 `hover` 时的提示信息。由 `placement` 属性决定展示效果：`placement` 属性值为：`方向-对齐位置`；四个方向：`top`、`left`、`right`、`bottom`；三种对齐位置：`start`，`end`，默认为空。如 `placement="left-end"`，则提示信息出现在目标元素的左侧，且提示信息的底部与目标元素的底部对齐。

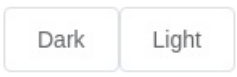
```
<div class="box">
  <div class="top">
    <el-tooltip class="item" effect="dark" content="Top Left 提示文字"
placement="top-start">
      <el-button>上左</el-button>
    </el-tooltip>
    <el-tooltip class="item" effect="dark" content="Top Center 提示文字"
placement="top">
      <el-button>上边</el-button>
    </el-tooltip>
    <el-tooltip class="item" effect="dark" content="Top Right 提示文字"
placement="top-end">
      <el-button>上右</el-button>
    </el-tooltip>
  </div>
  <div class="left">
```

```
      <el-tooltip class="item" effect="dark" content="Left Top 提示文字"
14.  placement="left-start">
15.       <el-button>左上</el-button>
16.     </el-tooltip>
      <el-tooltip class="item" effect="dark" content="Left Center 提示文字"
17.  placement="left">
18.       <el-button>左边</el-button>
19.     </el-tooltip>
      <el-tooltip class="item" effect="dark" content="Left Bottom 提示文字"
20.  placement="left-end">
21.       <el-button>左下</el-button>
22.     </el-tooltip>
23.   </div>
24.
25.   <div class="right">
      <el-tooltip class="item" effect="dark" content="Right Top 提示文字"
26.  placement="right-start">
27.       <el-button>右上</el-button>
28.     </el-tooltip>
      <el-tooltip class="item" effect="dark" content="Right Center 提示文字"
29.  placement="right">
30.       <el-button>右边</el-button>
31.     </el-tooltip>
      <el-tooltip class="item" effect="dark" content="Right Bottom 提示文字"
32.  placement="right-end">
33.       <el-button>右下</el-button>
34.     </el-tooltip>
35.   </div>
36.   <div class="bottom">
      <el-tooltip class="item" effect="dark" content="Bottom Left 提示文字"
37.  placement="bottom-start">
38.       <el-button>下左</el-button>
39.     </el-tooltip>
      <el-tooltip class="item" effect="dark" content="Bottom Center 提示文字"
40.  placement="bottom">
41.       <el-button>下边</el-button>
42.     </el-tooltip>
      <el-tooltip class="item" effect="dark" content="Bottom Right 提示文字"
43.  placement="bottom-end">
44.       <el-button>下右</el-button>
45.     </el-tooltip>
46.   </div>
47. </div>
```

```
<style>
  .box {
    width: 400px;

    .top {
      text-align: center;
    }

    .left {
      float: left;
      width: 60px;
    }

    .right {
      float: right;
      width: 60px;
    }

    .bottom {
      clear: both;
      text-align: center;
    }

    .item {
      margin: 4px;
    }

    .left .el-tooltip__popper,
    .right .el-tooltip__popper {
      padding: 8px 10px;
    }
  }
</style>
```

主题

Tooltip 组件提供了两个不同的主题： `dark` 和 `light` 。

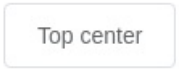


通过设置 `effect` 属性来改变主题，默认为 `dark` 。

```
<el-tooltip content="Top center" placement="top">
  <el-button>Dark</el-button>
</el-tooltip>
<el-tooltip content="Bottom center" placement="bottom" effect="light">
  <el-button>Light</el-button>
</el-tooltip>
```

更多 Content

展示多行文本或者是设置文本内容的格式



用具名 slot 分发 `content` ，替代 `tooltip` 中的 `content` 属性。

```
<el-tooltip placement="top">
  <template #content>
    多行信息<br/>第二行信息
  </template>
  <el-button>Top center</el-button>
</el-tooltip>
```

高级扩展

除了这些基本设置外，还有一些属性可以让使用者更好的定制自己的效果：

`transition` 属性可以定制显隐的动画效果，默认为 `fade-in-linear` 。 如果需要关闭 `tooltip` 功能， `disabled` 属性可以满足这个需求，它接受一个 `Boolean` ，设置为 `true` 即可。

事实上，这是基于 Vue-popper 的扩展，你可以自定义任意 Vue-popper 中允许定义的字段。 当然 Tooltip 组件实际上十分强大，文末的API文档会做一一说明。

```
<template>
  <el-tooltip :disabled="disabled" content="点击关闭 tooltip 功能"
placement="bottom" effect="light">
    <el-button @click="disabled = !disabled">点击{{disabled ? '开启' : '关闭'}}
tooltip 功能</el-button>
  </el-tooltip>
</template>
<script>
  export default {
    data() {
      return {
        disabled: false
      };
    }
  };
</script>
```

tooltip 内不支持 `router-link` 组件，请使用 `vm.$router.push` 代替。

tooltip 内不支持 disabled form 元素，参考MDN，请在 disabled form 元素外层添加一层包裹元素。

Attributes

参数	说明	类型	可选值	默认值
effect	默认提供的主题	String	dark/light	dark
content	显示的内容，也可以通过 `slot#content` 传入 DOM	String	—	—
placement	Tooltip 的出现位置	String	top/top-start/top-end/bottom/bottom-start/bottom-end/left/left-start/left-end/right/right-start/right-end	bottom
value / v-model	状态是否可见	Boolean	—	false
disabled	Tooltip 是否可用	Boolean	—	false
offset	出现位置的偏移量	Number	—	0
transition	定义渐变动画	String	—	el-fade-in-linear
visible-arrow	是否显示 Tooltip 箭头，更多参数可见Vue-	Boolean	—	true

	popper			
popper-options	popper.js 的参数	Object	参考 popper.js 文档	{ boundariesElement: 'body', gpuAcceleration: false }
open-delay	延迟出现，单位毫秒	Number	—	0
manual	手动控制模式，设置为 true 后，mouseenter 和 mouseleave 事件将不会生效	Boolean	—	false
popper-class	为 Tooltip 的 popper 添加类名	String	—	—
enterable	鼠标是否可进入到 tooltip 中	Boolean	—	true
hide-after	Tooltip 出现后自动隐藏延时，单位毫秒，为 0 则不会自动隐藏	number	—	0
tabindex	Tooltip 组件的 tabindex	number	—	0

Popover 弹出框

基础用法

Popover 的属性与 Tooltip 很类似，它们都是基于 `Vue-popper` 开发的，因此对于重复属性，请参考 Tooltip 的文档，在此文档中不做详尽解释。

`trigger` 属性用于设置何时触发 Popover，支持四种触发方式： `hover` ， `click` ， `focus` 和 `manual` 。对于触发 Popover 的元素，有两种写法：使用 `#reference` 的具名插槽，或使用自定义指令 `v-popover` 指向 Popover 的索引 `ref` 。

```
<template>
  <el-popover
    placement="top-start"
    title="标题"
    :width="200"
    trigger="hover"
    content="这是一段内容,这是一段内容,这是一段内容,这是一段内容。"
  >
    <template #reference>
      <el-button>hover 激活</el-button>
    </template>
  </el-popover>

  <el-popover
    placement="bottom"
    title="标题"
    :width="200"
    trigger="click"
    content="这是一段内容,这是一段内容,这是一段内容,这是一段内容。"
  >
    <template #reference>
      <el-button>click 激活</el-button>
    </template>
  </el-popover>

```

```
  <el-popover
    ref="popover"
    placement="right"
    title="标题"
    :width="200"
    trigger="focus"
    content="这是一段内容,这是一段内容,这是一段内容,这是一段内容。"
  >
    <template #reference>
      <el-button>focus 激活</el-button>
    </template>
  </el-popover>


  <el-popover
    placement="bottom"
    title="标题"
    :width="200"
    trigger="manual"
    content="这是一段内容,这是一段内容,这是一段内容,这是一段内容。"
    v-model:visible="visible"
  >
    <template #reference>
      <el-button @click="visible = !visible">手动激活</el-button>
    </template>
  </el-popover>
</template>

<script>
  export default {
    data() {
      return {
        visible: false
      };
    }
  };
</script>
```

嵌套信息

可以在 Popover 中嵌套多种类型信息，以下为嵌套表格的例子。

利用分发取代 `content` 属性

```
<el-popover
  placement="right"
  :width="400"
  trigger="click"
>
  <template #reference>
    <el-button>click 激活</el-button>
  </template>
  <el-table :data="gridData">
    <el-table-column width="150" property="date" label="日期"></el-table-
column>
    <el-table-column width="100" property="name" label="姓名"></el-table-
column>
    <el-table-column width="300" property="address" label="地址"></el-table-
column>
  </el-table>
</el-popover>

<script>
  export default {
    data() {
      return {
        gridData: [{
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-03',
```

```
            name: '王小虎',
            address: '上海市普陀区金沙江路 1518 弄'
          }]
        };
      }
    };
</script>
```

嵌套操作

当然，你还可以嵌套操作，这相比 Dialog 更为轻量：


```
<el-popover
  placement="top"
  :width="160"
  v-model:visible="visible"
>
  <p>这是一段内容这是一段内容确定删除吗？</p>
  <div style="text-align: right; margin: 0">
    <el-button size="mini" type="text" @click="visible = false">取消</el-
button>
    <el-button type="primary" size="mini" @click="visible = false">确定</el-
button>
  </div>
  <template #reference>
    <el-button @click="visible = true">删除</el-button>
  </template>
</el-popover>

<script>
  export default {
    data() {
      return {
        visible: false,
      };
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
trigger	触发方式	String	click/focus/hover/manual	click
title	标题	String	—	—
content	显示的内容，也可以通过 `slot` 传入 DOM	String	—	—
width	宽度	String, Number	—	最小宽度 150px
placement	出现位置	String	top/top-start/top-end/bottom/bottom-start/bottom-end/left/left-start/left-end/right/right-start/right-end	bottom
disabled	Popover 是否可用	Boolean	—	false
value / v-model	状态是否可见	Boolean	—	false
offset	出现位置的偏移量	Number	—	0
transition	定义渐变动画	String	—	fade-in-linear
visible-arrow	是否显示 Tooltip 箭头，更多参数可见Vue-popper	Boolean	—	true
popper-options	popper.js 的参数	Object	参考 popper.js 文档	`{ boundariesElement: 'body', gpuAcceleration: false }`
popper-class	为 popper 添加类名	String	—	—
open-delay	触发方式为 hover 时的显示延迟，单位为毫秒	Number	—	—
close-delay	触发方式为 hover 时的隐藏延迟，单位为毫秒	number	—	200
tabindex	Popover 组件的 tabindex	number	—	0

Slot

参数	说明

—	Popover 内嵌 HTML 文本
reference	触发 Popover 显示的 HTML 元素

Events

事件名称	说明	回调参数
show	显示时触发	—
after-enter	显示动画播放完毕后触发	—
hide	隐藏时触发	—
after-leave	隐藏动画播放完毕后触发	—

Popconfirm 气泡确认框

点击元素，弹出气泡确认框。

基础用法

Popconfirm 的属性与 Popover 很类似，因此对于重复属性，请参考 Popover 的文档，在此文档中不做详尽解释。

在 Popconfirm 中，只有 `title` 属性可用，`content` 属性不会被展示。

```
<template>
<el-popconfirm
  title="这是一段内容确定删除吗？"
>
<template #reference>
  <el-button>删除</el-button>
  </template>
</el-popconfirm>
</template>
```

自定义

可以在 Popconfirm 中自定义内容。


```
<template>
<el-popconfirm
  confirmButtonText='好的'
  cancelButtonText='不用了'
  icon="el-icon-info"
  iconColor="red"
  title="这是一段内容确定删除吗？"
```

```
 8.  >
 9.  <template #reference>
10.    <el-button>删除</el-button>
11.    </template>
12.  </el-popconfirm>
13.  </template>
```

Attributes

参数	说明	类型	可选值	默认值
title	标题	String	—	—
confirmButtonText	确认按钮文字	String	—	—
cancelButtonText	取消按钮文字	String	—	—
confirmButtonType	确认按钮类型	String	—	Primary
cancelButtonType	取消按钮类型	String	—	Text
icon	Icon	String	—	el-icon-question
iconColor	Icon 颜色	String	—	#f90
hideIcon	是否隐藏 Icon	Boolean	—	false

Slot

参数	说明
reference	触发 Popconfirm 显示的 HTML 元素

Events

事件名称	说明	回调参数
confirm	点击确认按钮时触发	—
cancel	点击取消按钮时触发	—

Card 卡片

将信息聚合在卡片容器中展示。

基础用法

包含标题，内容和操作。

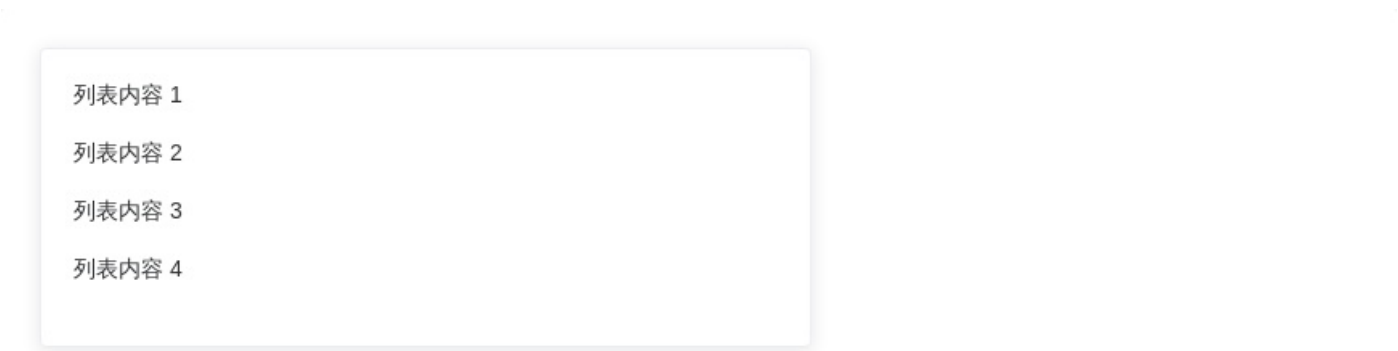
Card 组件包括 `header` 和 `body` 部分， `header` 部分需要有显式具名 slot 分发，同时也是可选的。

```
<el-card class="box-card">
  <template #header>
    <div class="clearfix">
      <span>卡片名称</span>
      <el-button style="float: right; padding: 3px 0" type="text">操作按钮</el-
button>
    </div>
  </template>
  <div v-for="o in 4" :key="o" class="text item">
    {{'列表内容 ' + o }}
  </div>
</el-card>

<style>
  .text {
    font-size: 14px;
  }

  .item {
```

```
    margin-bottom: 18px;
  }

  .clearfix:before,
  .clearfix:after {
    display: table;
    content: "";
  }
  .clearfix:after {
    clear: both
  }

  .box-card {
    width: 480px;
  }
</style>
```

简单卡片

卡片可以只有内容区域。




```
<el-card class="box-card">
  <div v-for="o in 4" :key="o" class="text item">
    {{'列表内容 ' + o }}
  </div>
</el-card>

<style>
  .text {
    font-size: 14px;
  }

  .item {
```

```
    padding: 18px 0;
  }

  .box-card {
    width: 480px;
  }
</style>
```

带图片

可配置定义更丰富的内容展示。

配置 `body-style` 属性来自定义 `body` 部分的 `style` ，我们还使用了布局组件。

```
<el-row>
  <el-col :span="8" v-for="(o, index) in 2" :key="o" :offset="index > 0 ? 2 :
0">
    <el-card :body-style="{ padding: '0px' }">
      <img src="https://shadow.elemecdn.com/app/element/hamburger.9cf7b091-
55e9-11e9-a976-7f4d0b07eef6.png" class="image">
      <div style="padding: 14px;">
        <span>好吃的汉堡</span>
        <div class="bottom clearfix">
          <time class="time">{{ currentDate }}</time>
          <el-button type="text" class="button">操作按钮</el-button>
        </div>
      </div>
    </el-card>
```

```
  </el-col>
</el-row>

<style>
  .time {
    font-size: 13px;
    color: #999;
  }

  .bottom {
    margin-top: 13px;
    line-height: 12px;
  }

  .button {
    padding: 0;
    float: right;
  }

  .image {
    width: 100%;
    display: block;
  }

  .clearfix:before,
  .clearfix:after {
      display: table;
      content: "";
  }

  .clearfix:after {
      clear: both
  }
</style>

<script>
export default {
  data() {
    return {
      currentDate: new Date()
    };
  }
```

```
55.  }
56.  </script>
```

卡片阴影

可对阴影的显示进行配置。

通过 `shadow` 属性设置卡片阴影出现的时机： `always` 、 `hover` 或 `never` 。

```
1.  <el-row :gutter="12">
2.    <el-col :span="8">
3.      <el-card shadow="always">
4.        总是显示
5.      </el-card>
6.    </el-col>
7.    <el-col :span="8">
8.      <el-card shadow="hover">
9.        鼠标悬浮时显示
10.     </el-card>
11.   </el-col>
12.   <el-col :span="8">
13.     <el-card shadow="never">
14.       从不显示
15.     </el-card>
16.   </el-col>
17. </el-row>
```

Attributes

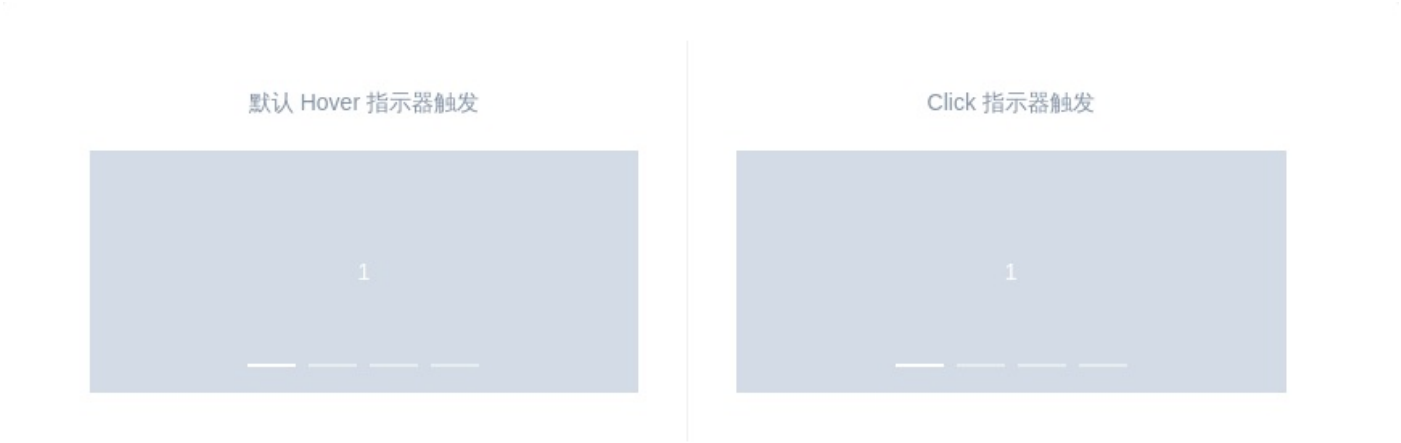
参数	说明	类型	可选值	默认值
header	设置 header，也可以通过 `slot#header` 传入 DOM	string	—	—
body-style	设置 body 的样式	object	—	{ padding: '20px' }
shadow	设置阴影显示时机	string	always / hover / never	always

Carousel 走马灯

在有限空间内，循环播放同一类型的图片、文字等内容

基础用法

适用广泛的基础用法



结合使用 `el-carousel` 和 `el-carousel-item` 标签就得到了一个走马灯。幻灯片的内容是任意的，需要放在 `el-carousel-item` 标签中。默认情况下，在鼠标 hover 底部的指示器时就会触发切换。通过设置 `trigger` 属性为 `click` ，可以达到点击触发的效果。

```
<template>
  <div class="block">
    <span class="demonstration">默认 Hover 指示器触发</span>
    <el-carousel height="150px">
      <el-carousel-item v-for="item in 4" :key="item">
        <h3 class="small">{{ item }}</h3>
      </el-carousel-item>
    </el-carousel>
  </div>
  <div class="block">
    <span class="demonstration">Click 指示器触发</span>
    <el-carousel trigger="click" height="150px">
      <el-carousel-item v-for="item in 4" :key="item">
        <h3 class="small">{{ item }}</h3>
      </el-carousel-item>
    </el-carousel>
  </div>
</template>
```

```
19.
20.  <style>
21.    .el-carousel__item h3 {
22.      color: #475669;
23.      font-size: 14px;
24.      opacity: 0.75;
25.      line-height: 150px;
26.      margin: 0;
27.    }
28.
29.    .el-carousel__item:nth-child(2n) {
30.       background-color: #99a9bf;
31.    }
32.
33.    .el-carousel__item:nth-child(2n+1) {
34.       background-color: #d3dce6;
35.    }
36.  </style>
```

指示器

可以将指示器的显示位置设置在容器外部

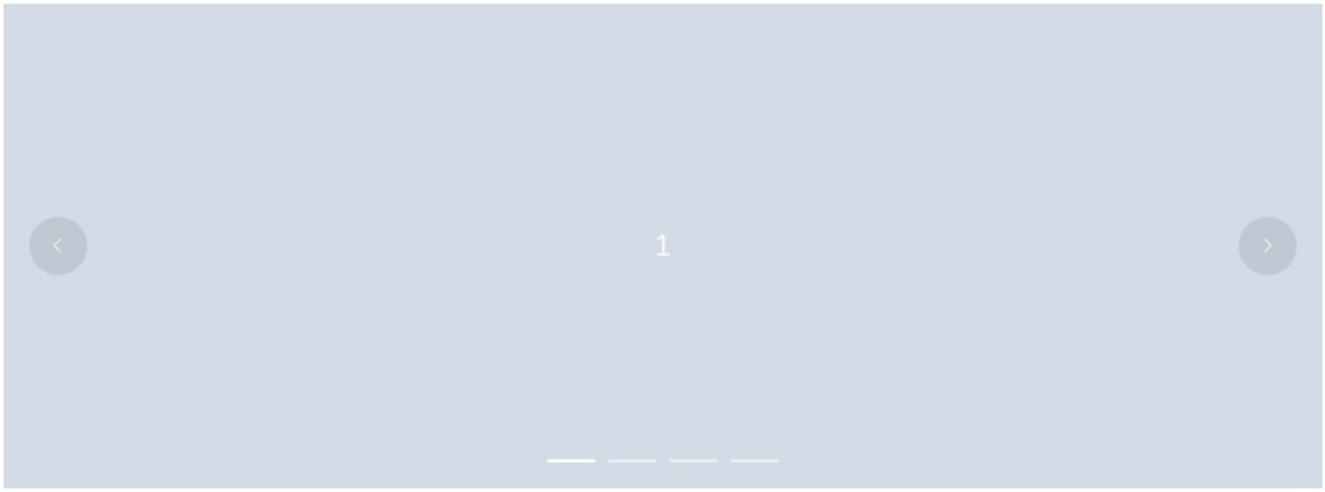
`indicator-position` 属性定义了指示器的位置。默认情况下，它会显示在走马灯内部，设置为 `outside` 则会显示在外部；设置为 `none` 则不会显示指示器。

```
1.  <template>
2.    <el-carousel indicator-position="outside">
```

```
    <el-carousel-item v-for="item in 4" :key="item">
      <h3>{{ item }}</h3>
    </el-carousel-item>
  </el-carousel>
</template>

<style>
  .el-carousel__item h3 {
    color: #475669;
    font-size: 18px;
    opacity: 0.75;
    line-height: 300px;
    margin: 0;
  }

  .el-carousel__item:nth-child(2n) {
    background-color: #99a9bf;
  }

  .el-carousel__item:nth-child(2n+1) {
    background-color: #d3dce6;
  }
</style>
```

切换箭头

可以设置切换箭头的显示时机



`arrow` 属性定义了切换箭头的显示时机。默认情况下，切换箭头只有在鼠标 hover 到走马灯上时

才会显示；若将 `arrow` 设置为 `always` ，则会一直显示；设置为 `never` ，则会一直隐藏。

```
<template>
  <el-carousel :interval="5000" arrow="always">
    <el-carousel-item v-for="item in 4" :key="item">
      <h3>{{ item }}</h3>
    </el-carousel-item>
  </el-carousel>
</template>

<style>
  .el-carousel__item h3 {
    color: #475669;
    font-size: 18px;
    opacity: 0.75;
    line-height: 300px;
    margin: 0;
  }

  .el-carousel__item:nth-child(2n) {
    background-color: #99a9bf;
  }

  .el-carousel__item:nth-child(2n+1) {
    background-color: #d3dce6;
  }
</style>
```

卡片化

当页面宽度方向空间空余，但高度方向空间匮乏时，可使用卡片风格

将 `type` 属性设置为 `card` 即可启用卡片模式。从交互上来说，卡片模式和一般模式的最大区别在于，可以通过直接点击两侧的幻灯片进行切换。

```
<template>
  <el-carousel :interval="4000" type="card" height="200px">
    <el-carousel-item v-for="item in 6" :key="item">
      <h3 class="medium">{{ item }}</h3>
    </el-carousel-item>
  </el-carousel>
</template>

<style>
  .el-carousel__item h3 {
    color: #475669;
    font-size: 14px;
    opacity: 0.75;
    line-height: 200px;
    margin: 0;
  }

  .el-carousel__item:nth-child(2n) {
    background-color: #99a9bf;
  }

  .el-carousel__item:nth-child(2n+1) {
    background-color: #d3dce6;
  }
</style>
```

方向

默认情况下， `direction` 为 `horizontal` 。通过设置 `direction` 为 `vertical` 来让走马灯在垂直方向上显示。

```vue
<template>
  <el-carousel height="200px" direction="vertical" :autoplay="false">
    <el-carousel-item v-for="item in 3" :key="item">
      <h3 class="medium">{{ item }}</h3>
    </el-carousel-item>
  </el-carousel>
</template>

<style>
  .el-carousel__item h3 {
    color: #475669;
    font-size: 14px;
    opacity: 0.75;
    line-height: 200px;
    margin: 0;
  }

  .el-carousel__item:nth-child(2n) {
    background-color: #99a9bf;
  }

  .el-carousel__item:nth-child(2n+1) {
    background-color: #d3dce6;
  }
</style>
```

Carousel Attributes

参数	说明	类型	可选值	默认值
height	走马灯的高度	string	—	—
initial-index	初始状态激活的幻灯片的索引，从 0 开始	number	—	0

trigger	指示器的触发方式	string	click	—
autoplay	是否自动切换	boolean	—	true
interval	自动切换的时间间隔，单位为毫秒	number	—	3000
indicator-position	指示器的位置	string	outside/none	—
arrow	切换箭头的显示时机	string	always/hover/never	hover
type	走马灯的类型	string	card	—
loop	是否循环显示	boolean	-	true
direction	走马灯展示的方向	string	horizontal/vertical	horizontal

Carousel Events

事件名称	说明	回调参数
change	幻灯片切换时触发	目前激活的幻灯片的索引，原幻灯片的索引

Carousel Methods

方法名	说明	参数
setActiveItem	手动切换幻灯片	需要切换的幻灯片的索引，从 0 开始；或相应的 `name` 属性值 `el-carousel-item`
prev	切换至上一张幻灯片	—
next	切换至下一张幻灯片	—

Carousel-Item Attributes

参数	说明	类型	可选值	默认值
name	幻灯片的名字，可用作 `setActiveItem` 的参数	string	—	—
label	该幻灯片所对应指示器的文本	string	—	—

Collapse 折叠面板

通过折叠面板收纳内容区域

基础用法

可同时展开多个面板，面板之间不影响


```
<el-collapse v-model="activeNames" @change="handleChange">
  <el-collapse-item title="一致性 Consistency" name="1">
    <div>与现实生活一致：与现实生活的流程、逻辑保持一致，遵循用户习惯的语言和概念；</div>
    <div>在界面中一致：所有的元素和结构需保持一致，比如：设计样式、图标和文本、元素的位置等。</div>
  </el-collapse-item>
  <el-collapse-item title="反馈 Feedback" name="2">
    <div>控制反馈：通过界面样式和交互动效让用户可以清晰的感知自己的操作；</div>
    <div>页面反馈：操作后，通过页面元素的变化清晰地展现当前状态。</div>
  </el-collapse-item>
  <el-collapse-item title="效率 Efficiency" name="3">
    <div>简化流程：设计简洁直观的操作流程；</div>
    <div>清晰明确：语言表达清晰且表意明确，让用户快速理解进而作出决策；</div>
    <div>帮助用户识别：界面简单直白，让用户快速识别而非回忆，减少用户记忆负担。</div>
  </el-collapse-item>
  <el-collapse-item title="可控 Controllability" name="4">
    <div>用户决策：根据场景可给予用户操作建议或安全提示，但不能代替用户进行决策；</div>
    <div>结果可控：用户可以自由的进行操作，包括撤销、回退和终止当前操作等。</div>
  </el-collapse-item>
</el-collapse>
<script>
```

```
21.   export default {
22.     data() {
23.       return {
24.         activeNames: ['1']
25.       };
26.     },
27.     methods: {
28.       handleChange(val) {
29.         console.log(val);
30.       }
31.     }
32.   }
33. </script>
```

手风琴效果

每次只能展开一个面板

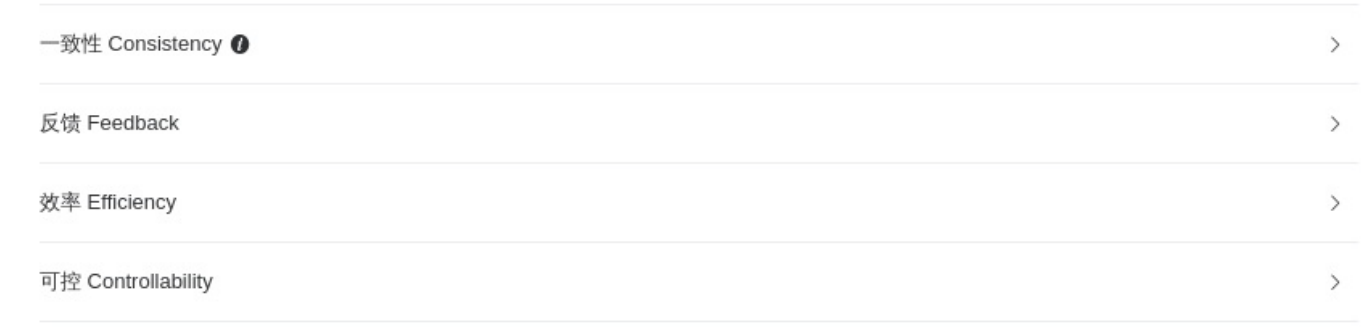
通过 `accordion` 属性来设置是否以手风琴模式显示。

```
1. <el-collapse v-model="activeName" accordion>
2.   <el-collapse-item title="一致性 Consistency" name="1">
3.     <div>与现实生活一致：与现实生活的流程、逻辑保持一致，遵循用户习惯的语言和概念；</div>
     <div>在界面中一致：所有的元素和结构需保持一致，比如：设计样式、图标和文本、元素的位置
4. 等。</div>
5.   </el-collapse-item>
6.   <el-collapse-item title="反馈 Feedback" name="2">
7.     <div>控制反馈：通过界面样式和交互动效让用户可以清晰的感知自己的操作；</div>
8.     <div>页面反馈：操作后，通过页面元素的变化清晰地展现当前状态。</div>
9.   </el-collapse-item>
```

```
    <el-collapse-item title="效率 Efficiency" name="3">
      <div>简化流程：设计简洁直观的操作流程；</div>
      <div>清晰明确：语言表达清晰且表意明确，让用户快速理解进而作出决策；</div>
      <div>帮助用户识别：界面简单直白，让用户快速识别而非回忆，减少用户记忆负担。</div>
    </el-collapse-item>
    <el-collapse-item title="可控 Controllability" name="4">
      <div>用户决策：根据场景可给予用户操作建议或安全提示，但不能代替用户进行决策；</div>
      <div>结果可控：用户可以自由的进行操作，包括撤销、回退和终止当前操作等。</div>
    </el-collapse-item>
  </el-collapse>
  <script>
    export default {
      data() {
        return {
          activeName: '1'
        };
      }
    }
  </script>
```

自定义面板标题

除了可以通过 `title` 属性以外，还可以通过具名 `slot` 来实现自定义面板的标题内容，以实现增加图标等效果。




```
  <el-collapse accordion>
    <el-collapse-item>
      <template #title>
        一致性 Consistency<i class="header-icon el-icon-info"></i>
      </template>
      <div>与现实生活一致：与现实生活的流程、逻辑保持一致，遵循用户习惯的语言和概念；</div>
```

```
      <div>在界面中一致：所有的元素和结构需保持一致，比如：设计样式、图标和文本、元素的位置
7.  等。</div>
8.    </el-collapse-item>
9.    <el-collapse-item title="反馈 Feedback">
10.     <div>控制反馈：通过界面样式和交互动效让用户可以清晰的感知自己的操作；</div>
11.     <div>页面反馈：操作后，通过页面元素的变化清晰地展现当前状态。</div>
12.   </el-collapse-item>
13.   <el-collapse-item title="效率 Efficiency">
14.     <div>简化流程：设计简洁直观的操作流程；</div>
15.     <div>清晰明确：语言表达清晰且表意明确，让用户快速理解进而作出决策；</div>
16.     <div>帮助用户识别：界面简单直白，让用户快速识别而非回忆，减少用户记忆负担。</div>
17.   </el-collapse-item>
18.   <el-collapse-item title="可控 Controllability">
19.     <div>用户决策：根据场景可给予用户操作建议或安全提示，但不能代替用户进行决策；</div>
20.     <div>结果可控：用户可以自由的进行操作，包括撤销、回退和终止当前操作等。</div>
21.   </el-collapse-item>
22. </el-collapse>
```

Collapse Attributes

参数	说明	类型	可选值	默认值
value / v-model	当前激活的面板(如果是手风琴模式，绑定值类型需要为 `string` ，否则为 `array`)	string / array	—	—
accordion	是否手风琴模式	boolean	—	false

Collapse Events

事件名称	说明	回调参数
change	当前激活面板改变时触发(如果是手风琴模式，参数 `activeNames` 类型为 `string` ，否则为 `array`)	(activeNames: array / string)

Collapse Item Attributes

参数	说明	类型	可选值	默认值
name	唯一标志符	string/number	—	—
title	面板标题	string	—	—
disabled	是否禁用	boolean	—	—

Timeline 时间线

可视化地呈现时间流信息。

基础用法

Timeline 可拆分成多个按照时间戳排列的 activity，时间戳是其区分于其他控件的重要特征，使用时注意与 Steps 步骤条等区分。


```
<div class="block">
  <el-timeline>
    <el-timeline-item
      v-for="(activity, index) in activities"
      :key="index"
      :timestamp="activity.timestamp">
      {{activity.content}}
    </el-timeline-item>
  </el-timeline>
</div>

<script>
  export default {
    data() {
      return {
        activities: [{
          content: '活动按期开始',
          timestamp: '2018-04-15'
        }, {
          content: '通过审核',
          timestamp: '2018-04-13'
        }, {
          content: '创建成功',
```

```
          timestamp: '2018-04-11'
        }]
      };
    }
  };
</script>
```

自定义节点样式

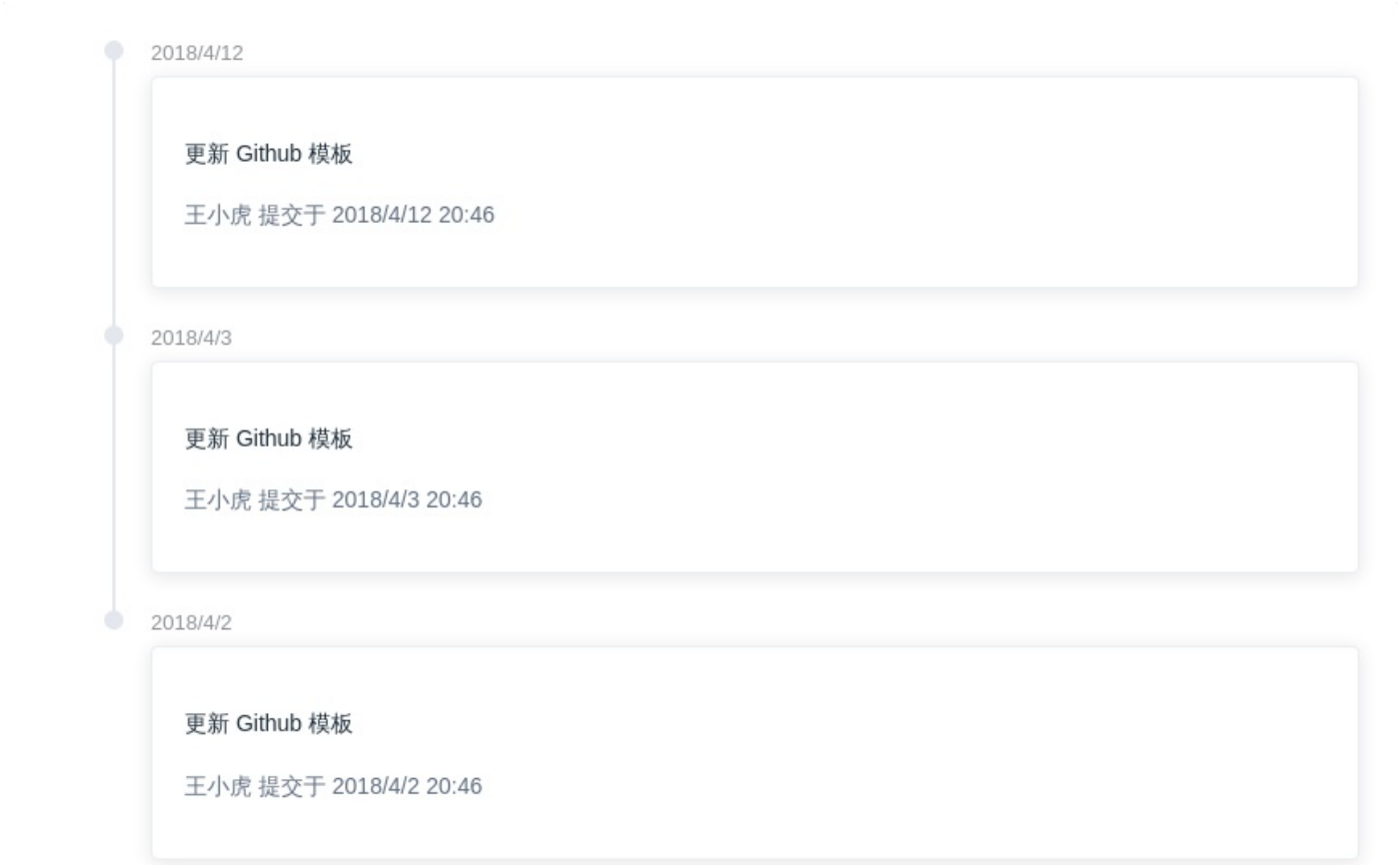
可根据实际场景自定义节点尺寸、颜色，或直接使用图标。


```
<div class="block">
  <el-timeline>
    <el-timeline-item
      v-for="(activity, index) in activities"
      :key="index"
      :icon="activity.icon"
      :type="activity.type"
      :color="activity.color"
      :size="activity.size"
      :timestamp="activity.timestamp">
      {{activity.content}}
    </el-timeline-item>
  </el-timeline>
</div>

<script>
  export default {
    data() {
      return {
        activities: [{
```

```
            content: '支持使用图标',
            timestamp: '2018-04-12 20:46',
            size: 'large',
            type: 'primary',
            icon: 'el-icon-more'
          }, {
            content: '支持自定义颜色',
            timestamp: '2018-04-03 20:46',
            color: '#0bbd87'
          }, {
            content: '支持自定义尺寸',
            timestamp: '2018-04-03 20:46',
            size: 'large'
          }, {
            content: '默认样式的节点',
            timestamp: '2018-04-03 20:46'
          }]
        };
      }
    };
</script>
```

自定义时间戳

当内容在垂直方向上过高时，可将时间戳置于内容之上。




```
<div class="block">
  <el-timeline>
    <el-timeline-item timestamp="2018/4/12" placement="top">
      <el-card>
        <h4>更新 Github 模板</h4>
        <p>王小虎 提交于 2018/4/12 20:46</p>
      </el-card>
    </el-timeline-item>
    <el-timeline-item timestamp="2018/4/3" placement="top">
      <el-card>
        <h4>更新 Github 模板</h4>
        <p>王小虎 提交于 2018/4/3 20:46</p>
      </el-card>
    </el-timeline-item>
    <el-timeline-item timestamp="2018/4/2" placement="top">
      <el-card>
        <h4>更新 Github 模板</h4>
        <p>王小虎 提交于 2018/4/2 20:46</p>
      </el-card>
    </el-timeline-item>
  </el-timeline>
```

```
22.  </div>
```

Timeline-item Attributes

参数	说明	类型	可选值	默认值
timestamp	时间戳	string	-	—
hide-timestamp	是否隐藏时间戳	boolean	—	false
placement	时间戳位置	string	top / bottom	bottom
type	节点类型	string	primary / success / warning / danger / info	-
color	节点颜色	string	hsl / hsv / hex / rgb	-
size	节点尺寸	string	normal / large	normal
icon	节点图标	string	—	-

Timeline-Item Slot

name	说明
—	Timeline-Item 的内容
dot	自定义节点

Divider 分割线

区隔内容的分割线。

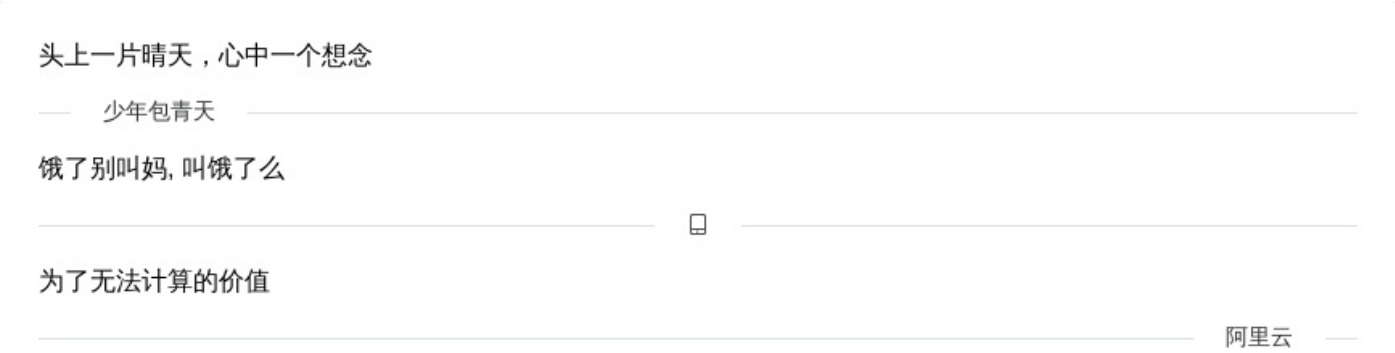
基础用法

对不同章节的文本段落进行分割。


```
<template>
  <div>
    <span>青春是一个短暂的美梦， 当你醒来时， 它早已消失无踪</span>
    <el-divider></el-divider>
    <span>少量的邪恶足以抵消全部高贵的品质， 害得人声名狼藉</span>
  </div>
</template>
```

设置文案

可以在分割线上自定义文案内容。




```
<template>
  <div>
    <span>头上一片晴天，心中一个想念</span>
    <el-divider content-position="left">少年包青天</el-divider>
```

```
      <span>饿了别叫妈，叫饿了么</span>
      <el-divider><i class="el-icon-mobile-phone"></i></el-divider>
      <span>为了无法计算的价值</span>
      <el-divider content-position="right">阿里云</el-divider>
    </div>
  </template>
```

垂直分割

雨纷纷 | 旧故里 | 草木深

```
  <template>
    <div>
      <span>雨纷纷</span>
      <el-divider direction="vertical"></el-divider>
      <span>旧故里</span>
      <el-divider direction="vertical"></el-divider>
      <span>草木深</span>
    </div>
  </template>
```

Divider Attributes

参数	说明	类型	可选值	默认值
direction	设置分割线方向	string	horizontal / vertical	horizontal
content-position	设置分割线文案的位置	string	left / right / center	center

Calendar calendar

显示日期

基本

设置 `value` 来指定当前显示的月份。如果 `value` 未指定，则显示当月。 `value` 支持 `v-model` 双向绑定。

```
<el-calendar v-model="value">
</el-calendar>

<script>
  export default {
    data() {
      return {
```

```
        value: new Date()
      }
    }
  }
</script>
```

自定义内容

通过设置名为 `dateCell` 的 `scoped-slot` 来自定义日历单元格中显示的内容。在 `scoped-slot` 可以获取到 date（当前单元格的日期），data（包括 type，isSelected，day 属性）。详情解释参考下方的 API 文档。

```
<el-calendar>
  <template
    #dateCell="{data}"
  >
    <p :class="data.isSelected ? 'is-selected' : ''">
```

```
          {{ data.day.split('-').slice(1).join('-') }} {{ data.isSelected ? '✔' :
'' }}
      </p>
    </template>
</el-calendar>
<style>
    .is-selected {
      color: #1989FA;
    }
</style>
```

自定义范围

设置 `range` 属性指定日历的显示范围。开始时间必须是周起始日，结束时间必须是周结束日，且时间跨度不能超过两个月。

```
<el-calendar :range="[new Date(2019, 2, 4), new Date(2019, 2, 24)]">
</el-calendar>
```

Attributes

参数	说明	类型	可选值	默认值
value / v-model	绑定值	Date	—	—

range	时间范围，包括开始时间与结束时间。开始时间必须是周起始日，结束时间必须是周结束日，且时间跨度不能超过两个月。	[Date]Array	—	—

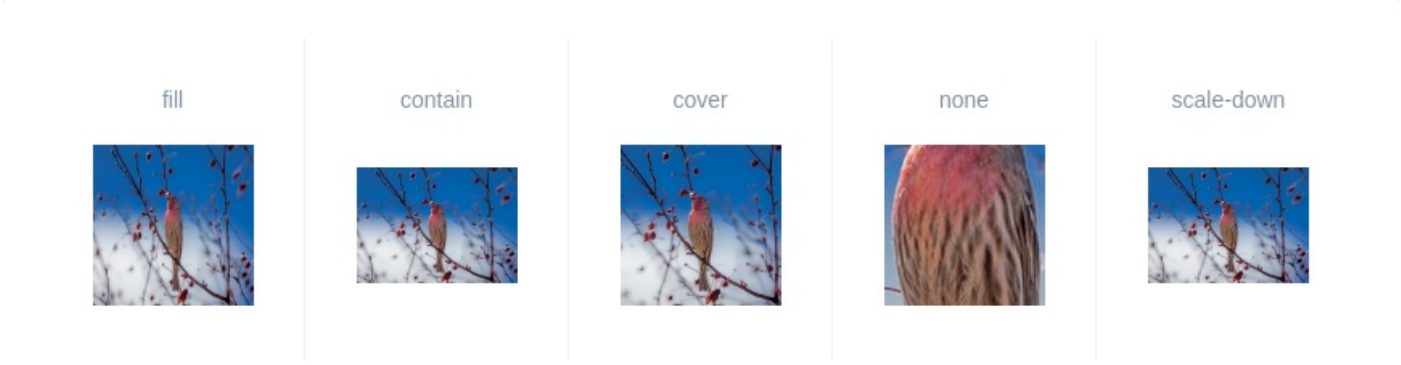
dateCell scoped slot 参数

参数	说明	类型	可选值	默认值
date	单元格代表的日期	Date	—	—
data	{ type, isSelected, day}, `type` 表示该日期的所属月份，可选值有 prev-month, current-month, next-month; `isSelected` 标明该日期是否被选中; `day` 是格式化的日期，格式为 yyyy-MM-dd	Object	—	—

Image 图片

图片容器，在保留原生img的特性下，支持懒加载，自定义占位、加载失败等

基础用法



可通过 `fit` 确定图片如何适应到容器框，同原生 `object-fit`。

```
<div class="demo-image">
  <div class="block" v-for="fit in fits" :key="fit">
    <span class="demonstration">{{ fit }}</span>
    <el-image
      style="width: 100px; height: 100px"
      :src="url"
      :fit="fit"></el-image>
  </div>
</div>

<script>
  export default {
    data() {
      return {
        fits: ['fill', 'contain', 'cover', 'none', 'scale-down'],
        url:
'https://fuss10.elemecdn.com/e/5d/4a731a90594a4af544c0c25941171jpeg.jpeg'
      }
    }
  }
</script>
```

占位内容

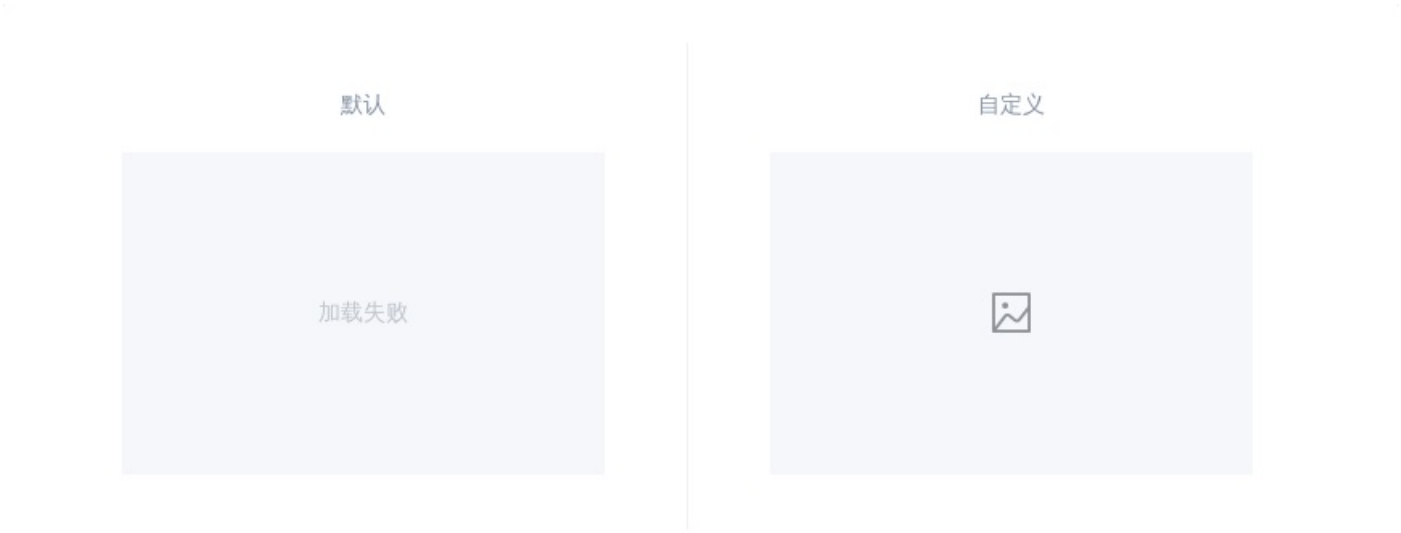
默认

自定义

可通过 `slot = placeholder` 可自定义占位内容

```
<div class="demo-image__placeholder">
  <div class="block">
    <span class="demonstration">默认</span>
    <el-image :src="src"></el-image>
  </div>
  <div class="block">
    <span class="demonstration">自定义</span>
    <el-image :src="src">
      <template #placeholder>
        <div class="image-slot">
          加载中<span class="dot">...</span>
        </div>
      </template>
    </el-image>
  </div>
</div>

<script>
  export default {
    data() {
      return {
        src:
'https://cube.elemecdn.com/6/94/4d3ea53c084bad6931a56d5158a48jpeg.jpeg'
      }
    }
  }
</script>
```

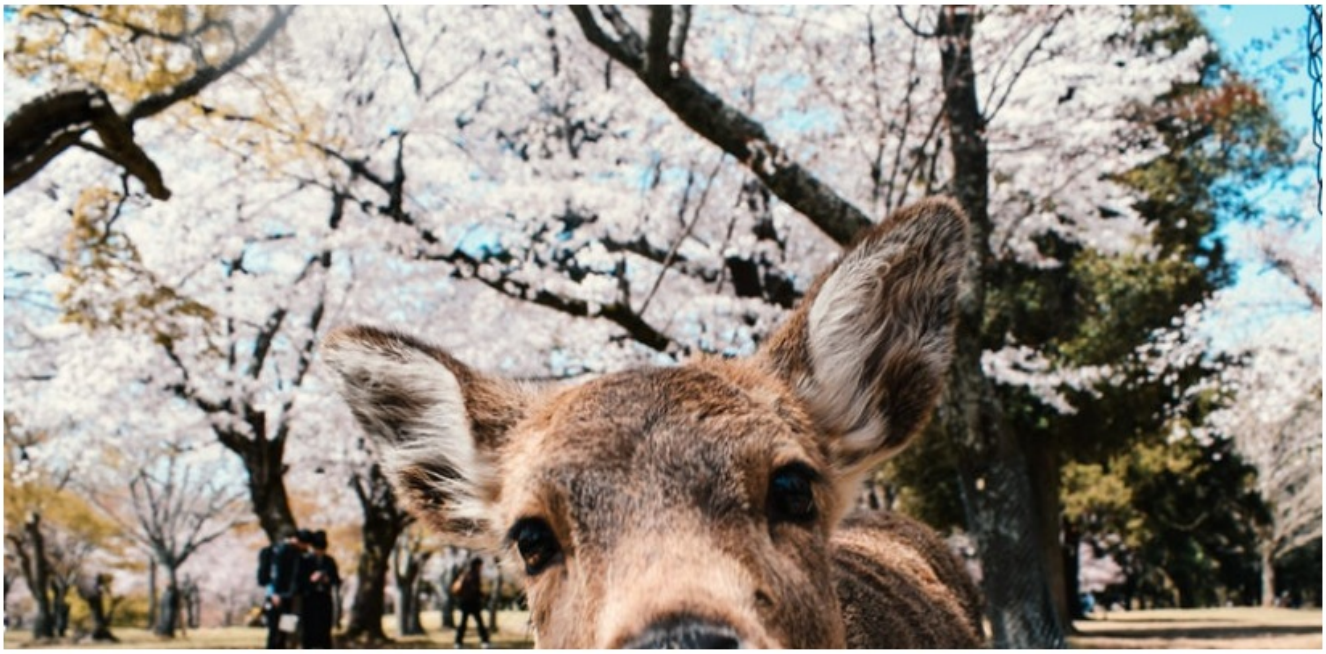
加载失败



可通过 `slot = error` 可自定义加载失败内容

```
<div class="demo-image__error">
  <div class="block">
    <span class="demonstration">默认</span>
    <el-image></el-image>
  </div>
  <div class="block">
    <span class="demonstration">自定义</span>
    <el-image>
      <template #error>
        <div class="image-slot">
          <i class="el-icon-picture-outline"></i>
        </div>
      </template>
    </el-image>
  </div>
</div>
```

懒加载



可通过 `lazy` 开启懒加载功能，当图片滚动到可视范围内才会加载。可通过 `scroll-container` 来设置滚动容器，若未定义，则为最近一个 `overflow` 值为 `auto` 或 `scroll` 的父元素。

```
<div class="demo-image__lazy">
  <el-image v-for="url in urls" :key="url" :src="url" lazy></el-image>
</div>

<script>
  export default {
    data() {
      return {
        urls: [
'https://fuss10.elemecdn.com/a/3f/3302e58f9a181d2509f3dc0fa68b0jpeg.jpeg',
'https://fuss10.elemecdn.com/1/34/19aa98b1fcb2781c4fba33d850549jpeg.jpeg',
'https://fuss10.elemecdn.com/0/6f/e35ff375812e6b0020b6b4e8f9583jpeg.jpeg',
'https://fuss10.elemecdn.com/9/bb/e27858e973f5d7d3904835f46abbdjpeg.jpeg',
'https://fuss10.elemecdn.com/d/e6/c4d93a3805b3ce3f323f7974e6f78jpeg.jpeg',
'https://fuss10.elemecdn.com/3/28/bbf893f792f03a54408b3b7a7ebf0jpeg.jpeg',
'https://fuss10.elemecdn.com/2/11/6535bcfb26e4c79b48ddde44f4b6fjpeg.jpeg'
        ]
```

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

大图预览

可通过 `previewSrcList` 开启预览大图的功能。

```
<div class="demo-image__preview">
  <el-image
    style="width: 100px; height: 100px"
    :src="url"
    :preview-src-list="srcList">
  </el-image>
</div>

<script>
  export default {
    data() {
      return {
        url:
'https://fuss10.elemecdn.com/e/5d/4a731a90594a4af544c0c25941171jpeg.jpeg',
        srcList: [

'https://fuss10.elemecdn.com/8/27/f01c15bb73e1ef3793e64e6b7bbccjpeg.jpeg',

'https://fuss10.elemecdn.com/1/8e/aeffeb4de74e2fde4bd74fc7b4486jpeg.jpeg'
        ]
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
src	图片源，同原生	string	—	-
fit	确定图片如何适应容器框，同原生 object-fit	string	fill / contain / cover / none / scale-down	-
alt	原生 alt	string	-	-
referrer-policy	原生 referrerPolicy	string	-	-
lazy	是否开启懒加载	boolean	—	false
scroll-container	开启懒加载后，监听 scroll 事件的容器	string / HTMLElement	—	最近一个 overflow 值为 auto 或 scroll 的父元素
preview-src-list	开启图片预览功能	Array	—	-
z-index	设置图片预览的 z-index	Number	—	2000

Events

事件名称	说明	回调参数
load	图片加载成功触发	(e: Event)
error	图片加载失败触发	(e: Error)

Slots

名称	说明
placeholder	图片未加载的占位内容
error	加载失败的内容

Backtop 回到顶部

返回页面顶部的操作按钮

基础用法

滑动页面即可看到右下方的按钮。

Scroll down to see the bottom-right button.

```
<template>
  Scroll down to see the bottom-right button.
  <el-backtop target=".page-component__scroll .el-scrollbar__wrap"></el-
backtop>
</template>
```

自定义显示内容

显示区域被固定为 40px * 40px 的区域, 其中的内容可支持自定义。

Scroll down to see the bottom-right button.

```
<template>
  Scroll down to see the bottom-right button.
  <el-backtop target=".page-component__scroll .el-scrollbar__wrap"
:bottom="100">
    <div
      style="{
        height: 100%;
        width: 100%;
        background-color: #f2f5f6;
        box-shadow: 0 0 6px rgba(0,0,0, .12);
        text-align: center;
        line-height: 40px;
        color: #1989fa;
      }"
    >
      UP
```

```
16.      </div>
17.    </el-backtop>
18.  </template>
```

Attributes

参数	说明	类型	可选值	默认值
target	触发滚动的对象	string		
visibility-height	滚动高度达到此参数值才出现	number		200
right	控制其显示位置，距离页面右边距	number		40
bottom	控制其显示位置，距离页面底部距离	number		40

Events

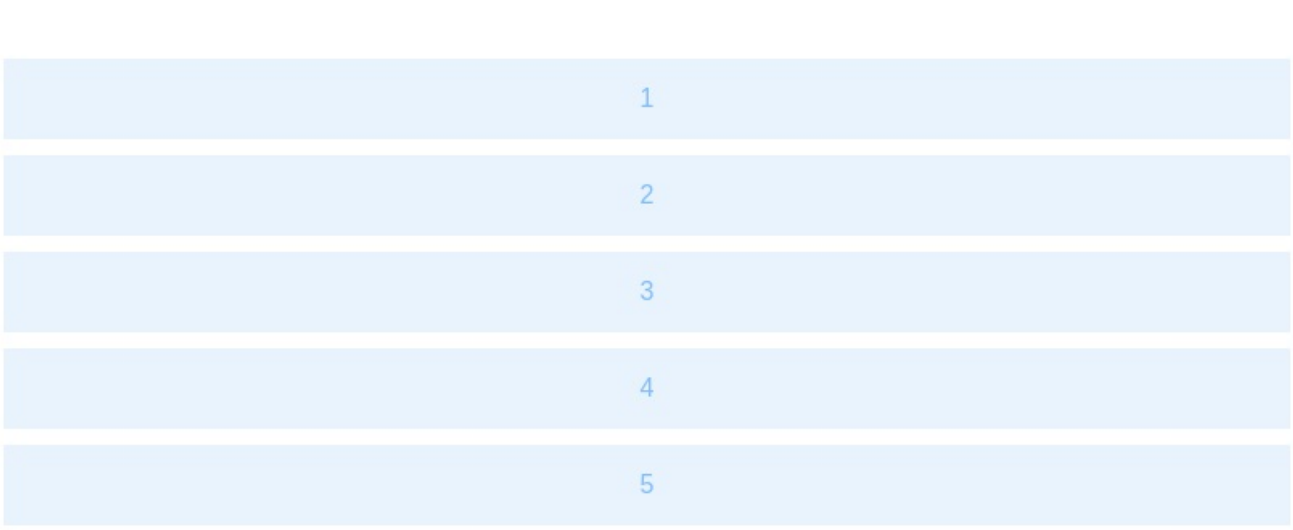
事件名	说明	回调参数
click	点击按钮触发的事件	点击事件

InfiniteScroll 无限滚动

滚动至底部时，加载更多数据。

基础用法

在要实现滚动加载的列表上上添加 `v-infinite-scroll` ，并赋值相应的加载方法，可实现滚动到底部时自动执行加载方法。




```
<template>
  <ul class="infinite-list" v-infinite-scroll="load" style="overflow:auto">
    <li v-for="i in count" class="infinite-list-item">{{ i }}</li>
  </ul>
</template>

<script>
  export default {
    data () {
      return {
        count: 0
      }
    },
    methods: {
      load () {
        this.count += 2
      }
    }
  }
```

```
</script>
```

禁用加载

1

2

3

4

5

```
<template>
  <div class="infinite-list-wrapper" style="overflow:auto">
    <ul
      class="list"
      v-infinite-scroll="load"
      infinite-scroll-disabled="disabled">
      <li v-for="i in count" class="list-item">{{ i }}</li>
    </ul>
    <p v-if="loading">加载中...</p>
    <p v-if="noMore">没有更多了</p>
  </div>
</template>

<script>
  export default {
    data () {
      return {
        count: 10,
        loading: false
      }
    },
    computed: {
      noMore () {
        return this.count >= 20
      },
```

```
      disabled () {
        return this.loading || this.noMore
      }
    },
    methods: {
      load () {
        this.loading = true
        setTimeout(() => {
          this.count += 2
          this.loading = false
        }, 2000)
      }
    }
  }
</script>
```

Attributes

参数	说明	类型	可选值	默认值
infinite-scroll-disabled	是否禁用	boolean	-	false
infinite-scroll-delay	节流时延，单位为ms	number	-	200
infinite-scroll-distance	触发加载的距离阈值，单位为px	number	-	0
infinite-scroll-immediate	是否立即执行加载方法，以防初始状态下内容无法撑满容器。	boolean	-	true

Drawer 抽屉

有些时候， `Dialog` 组件并不满足我们的需求，比如你的表单很长，亦或是你需要临时展示一些文档， `Drawer` 拥有和 `Dialog` 几乎相同的 API，在 UI 上带来不一样的体验.

基本用法

呼出一个临时的侧边栏，可以从多个方向呼出

需要设置 `model-value` 属性，它的类型是 `boolean` ，当为 **true** 时显示 Drawer。Drawer 分为两个部分： `title` 和 `body` ， `title` 需要具名为 **title** 的 `slot` ， 也可以通过 `title` 属性来定义，默认值为空。需要注意的是，Drawer 默认是从右往左打开，当然可以设置对应的 `direction` ，详细请参考 `direction` 用法 最后，本例还展示了 `before-close` 的用法

```
<el-radio-group v-model="direction">
  <el-radio label="ltr">从左往右开</el-radio>
  <el-radio label="rtl">从右往左开</el-radio>
  <el-radio label="ttb">从上往下开</el-radio>
  <el-radio label="btt">从下往上开</el-radio>
</el-radio-group>

<el-button @click="drawer = true" type="primary" style="margin-left: 16px;">
  点我打开
</el-button>

<el-drawer
  title="我是标题"
  v-model="drawer"
  :direction="direction"
  :before-close="handleClose" destroy-on-close>
  <span>我来啦!</span>
</el-drawer>

<script>
  export default {
```

```
    data() {
      return {
        drawer: false,
        direction: 'rtl',
      };
    },
    methods: {
      handleClose(done) {
        this.$confirm('确认关闭？')
          .then(_ => {
            done();
          })
          .catch(_ => {});
      }
    }
  };
</script>
```

不添加 Title

当你不需要标题到时候，你还可以去掉标题

当遇到不需要 title 的场景时，可以通过 `withHeader` 这个属性来关闭掉 title 的显示，这样可以留出更大的空间给到用户，为了用户的可访问性，请务必设定 `title` 的值

```
<el-button @click="drawer = true" type="primary" style="margin-left: 16px;">
  点我打开
</el-button>

<el-drawer
  title="我是标题"
  v-model="drawer"
  :with-header="false">
  <span>我来啦!</span>
</el-drawer>

<script>
  export default {
```

```
    data() {
      return {
        drawer: false,
      };
    }
  };
</script>
```

自定义内容

和 `Dialog` 组件一样， `Drawer` 同样可以在其内部嵌套各种丰富的操作


```
<el-button type="text" @click="table = true">打开嵌套表格的 Drawer</el-button>
<el-button type="text" @click="dialog = true">打开嵌套 Form 的 Drawer</el-button>
<el-drawer
  title="我嵌套了表格!"
  v-model="table"
  direction="rtl"
  size="50%">
   <el-table :data="gridData">
      <el-table-column property="date" label="日期" width="150"></el-table-column>
      <el-table-column property="name" label="姓名" width="200"></el-table-column>
      <el-table-column property="address" label="地址"></el-table-column>
    </el-table>
</el-drawer>

<el-drawer
  title="我嵌套了 Form !"
  :before-close="handleClose"
  v-model="dialog"
  direction="ltr"
  custom-class="demo-drawer"
  ref="drawer"
  >
  <div class="demo-drawer__content">
```

```
    <el-form :model="form">
      <el-form-item label="活动名称" :label-width="formLabelWidth">
        <el-input v-model="form.name" autocomplete="off"></el-input>
      </el-form-item>
      <el-form-item label="活动区域" :label-width="formLabelWidth">
        <el-select v-model="form.region" placeholder="请选择活动区域">
          <el-option label="区域一" value="shanghai"></el-option>
          <el-option label="区域二" value="beijing"></el-option>
        </el-select>
      </el-form-item>
    </el-form>
    <div class="demo-drawer__footer">
      <el-button @click="cancelForm">取 消</el-button>
      <el-button type="primary" @click="$refs.drawer.closeDrawer()"
:loading="loading">{{ loading ? '提交中 ...' : '确 定' }}</el-button>
    </div>
  </div>
</el-drawer>

<script>
export default {
  data() {
    return {
      table: false,
      dialog: false,
      loading: false,
      gridData: [{
        date: '2016-05-02',
        name: '王小虎',
        address: '上海市普陀区金沙江路 1518 弄'
      }, {
        date: '2016-05-04',
        name: '王小虎',
        address: '上海市普陀区金沙江路 1518 弄'
      }, {
        date: '2016-05-01',
        name: '王小虎',
        address: '上海市普陀区金沙江路 1518 弄'
      }, {
        date: '2016-05-03',
        name: '王小虎',
        address: '上海市普陀区金沙江路 1518 弄'
```

```
      }],
      form: {
        name: '',
        region: '',
        date1: '',
        date2: '',
        delivery: false,
        type: [],
        resource: '',
        desc: ''
      },
      formLabelWidth: '80px',
      timer: null,
    };
  },
  methods: {
    handleClose(done) {
      if (this.loading) {
        return;
      }
      this.$confirm('确定要提交表单吗？')
        .then(_ => {
          this.loading = true;
          this.timer = setTimeout(() => {
            done();
            // 动画关闭需要一定的时间
            setTimeout(() => {
              this.loading = false;
            }, 400);
          }, 2000);
        })
        .catch(_ => {});
    },
    cancelForm() {
      this.loading = false;
      this.dialog = false;
      clearTimeout(this.timer);
    }
  }
}
</script>
```

多层嵌套

`Drawer` 组件也拥有多层嵌套的方法

同样，如果你需要嵌套多层 `Drawer` 请一定要设置 `append-to-body` 属性为 **true**

```
1.
2.  <el-button @click="drawer = true" type="primary" style="margin-left: 16px;">
3.    点我打开
4.  </el-button>
5.
6.  <el-drawer
7.    title="我是外面的 Drawer"
8.    v-model="drawer"
9.    size="50%">
10.   <div>
11.    <el-button @click="innerDrawer = true">打开里面的!</el-button>
12.    <el-drawer
13.      title="我是里面的"
14.      :append-to-body="true"
15.      :before-close="handleClose"
16.      v-model="innerDrawer">
17.      <p>_(:3  ∠)_</p>
18.    </el-drawer>
19.   </div>
20. </el-drawer>
21.
22. <script>
23.   export default {
24.     data() {
25.       return {
26.         drawer: false,
27.         innerDrawer: false,
28.       };
29.     },
30.     methods: {
31.       handleClose(done) {
32.         this.$confirm('还有未保存的工作哦确定关闭吗？')
```

```
33.             .then(_ => {
34.               done();
35.             })
36.             .catch(_ => {});
37.         }
38.       }
39.     };
40.   </script>
```

Drawer 的内容是懒渲染的，即在第一次被打开之前，传入的默认 slot 不会被渲染到 DOM 上。因此，如果需要执行 DOM 操作，或通过 `ref` 获取相应组件，请在 `open` 事件回调中进行。

Drawer 提供一个 `destroyOnClose` API，用来在关闭 Drawer 时销毁子组件内容，例如清理表单内的状态，在必要时可以将该属性设置为 **true** 用来保证初始状态的一致性

Drawer Attributes

参数	说明	类型	可选值	默认值
append-to-body	Drawer 自身是否插入至 body 元素上。嵌套的 Drawer 必须指定该属性并赋值为 true	boolean	—	false
before-close	关闭前的回调，会暂停 Drawer 的关闭	function(done)，done 用于关闭 Drawer	—	—
close-on-press-escape	是否可以通过按下 ESC 关闭 Drawer	boolean	—	true
custom-class	Drawer 的自定义类名	string	—	—
destroy-on-close	控制是否在关闭 Drawer 之后将子元素全部销毁	boolean	-	false
modal	是否需要遮罩层	boolean	—	true
modal-append-to-body	遮罩层是否插入至 body 元素上，若为 false，则遮罩层会插入至 Drawer 的父元素上	boolean	—	true
direction	Drawer 打开的方向	Direction	rtl / ltr / ttb / btt	rtl
show-close	是否显示关闭按钮	boolean	—	true
size	Drawer 窗体的大小，当使用 `number` 类型时，以像素为单位，当使用 `string` 类型时，请传入 ‘x%’，否则便会以 `number` 类型解释	number / string	-	‘30%’
	Drawer 的标题，也可通过具名 slot			

	（见下表）传入			
model-value	是否显示 Drawer，支持 .sync 修饰符	boolean	—	false
wrapperClosable	点击遮罩层是否可以关闭 Drawer	boolean	-	true
withHeader	控制是否显示 header 栏，默认为 true，当此项为 false 时，title attribute 和 title slot 均不生效	boolean	-	true

Drawer Slot

name	说明
—	Drawer 的内容
title	Drawer 标题区的内容

Drawer Methods

name	说明
closeDrawer	用于关闭 Drawer，该方法会调用传入的 `before-close` 方法

Drawer Events

事件名称	说明	回调参数
open	Drawer 打开的回调	—
opened	Drawer 打开动画结束时的回调	—
close	Drawer 关闭的回调	—
closed	Drawer 关闭动画结束时的回调	—

更新日志

- ## Element Plus 更新进度

  Element Plus 更新进度 请查看 https://github.com/element-plus/element-plus

- ## 2.13.2

  2020-05-18

  ### Bug 修复

  - Autocomplete
    - 修复 ‘change event’ 错误 (#19200 by @sxzz)
  - Image
    - 更新错误状态 (#19194 by @lhx6538665)

  ### Optimization

  - I18n
    - 更新 ru-RU popconfirm 翻译 (#19220 by @Opppex)
    - 更新 vi 翻译 (#19244 by @quangln2810)
    - 更新 Catalan 和 Spanish 翻译 (#19296 by @Ismaaa)
    - 更新 Indonesia 翻译 (#19320) by @therour)
    - 更新 Brazilian Portuguese 翻译 (#19374 by @diegomengarda)

- ## 2.13.1

  2020-04-13

  ### 新特性

  - Autocomplete
    - 添加 change 事件 (#17913 by @sxzz)

  ### Bug 修复

  - Autocomplete
    - 修复类型为 textarea 时建议错误问题 (#18478 by @Roojay)
  - Carousel
    - 修复 console.warn 文案拼写错误 (#18264 by @IceFox)
  - Image
    - 修复当 preview-src-list 属性不包含 src 时图片预览大图展示为空的问题

(#18975) (#19130 by @luckyCao)
        - 修复第二次图片预览时快捷键失效问题 (#18983) (#19156 by @luckyCao)
        - 修复 preview-src-list 为空时点击图片会给 body 添加 overflow: scroll 的问题 (#18967 by @inooNgt)
    - Transfer
        - 修复和 Form 组件一起使用时错误的行高问题 (#18917 by @Hanx)
    - InputNumber
        - 正确计算 inputNumberDisabled (#18439 by @ashuser-pendo)
    - Chore
        - 更新首页文案 (#19155 by @iamkun)
    - Doc
        - 更新 Popconfirm 文档 (#18324 by @iamkun)
        - 修复 step-strictly 文档拼写问题 (#18705 by @dream2023)
        - 修复 Steps 组件文档问题 (#17555 by @haoranyu)

- 2.13.0

2019-11-26

新特性

- Popconfirm
    - 新增 Popconfirm 组件 (#17548 by @iamkun)

Bug fixes

- BackTop
    - 平滑过渡动画 (by @lon)
- DatePicker
    - 修复选择最小日期的 bug (#17191 by @smk0621)
- Select
    - 修复测试用例 (by @msidolphin)
- Tree
    - 增加 font-size 样式 (#17094 by @spengjie)
- Table
    - 头部可自定义 (#17291 by @ziyoung)
    - 更新头部样式 (#17284 by @ziyoung)
    - 修复时候 filter 之后高度问题 (#17348 by @ziyoung)
    - 修复 row-style 失效的 bug (#17002 by @a631807682)
    - 修复头部消失的 bug (#17341 by @ziyoung)
- Calendar

        - 导入 el-button 和 el-button-group (#17376 by @masongzhi)
    - MessageBox
        - 修复图表位置 (#17410 by @nullptru)
    - TimePicker
        - 滚动后设置正确的位置 (#16868 by @mattheyan)
    - Message
        - 修复关闭的 offsetHeight (#17564) (#17852 by @gzwgq222)
    - Form
        - ValidateField 的回调应为可选项 (#17314 by @CarterLi)
    - Cascader
        - 修复 TypeScript 3.7 的兼容问题 (#17881 by @CarterLi)
    - Menu
        - 修复 NavigationDuplicated 在 vue-router@^3.1.0 的报错 (#17269 by @iamkun)
    - Dropdown
        - 更新类型文件 (#17550 by @iamkun)
    - Progress
        - 增加 strokeLinecap 属性 (#17552 by @iamkun)
    - InfiniteScroll
        - 跳过不可见元素触发 (#17553 by @iamkun)
    - Image
        - 优化用户体验 (#16985 by @luckyCao)
        - 优化大图片展示问题 (#16796 by @luckyCao)
    - Drawer
        - 修复 drawer-append-to-body 失效的 bug (#16953 by @JeremyWuuuuu)
    - Select
        - 修复空 tag 的 bug (17199 by @luckyCao)
    - Scrollbar
        - 修复 FireFox 双滚动条的 bug (#18091 by @iamkun)

Optimization

- I18n
    - 更新 sv-SE.js (#17926 by @FOLLGAD)
    - 更新 avatar 组件法语文档 (#17762 by @blombard)
- Docs
    - 修复 time-select 文档错误 (#17250 by @wacky6)
    - 修复 Drawer 文档错误 (#17122 by @haoranyu)
    - 更新 Spanish changelog 2.12.0 (#17364 by @Gonzalo2310)
    - 修复 Changelog 文档错误 (#17874 by @renlixin)

        - 修复 Loading 示例 (#17862 by @MBearo)
        - 增加 input event 相关文档 (#18061 by @zhouxinyong)
        - 移除 Input repeat change event 相关文档 (#18085 by @zhouxinyong)

2.12.0

2019-08-29

新特性

- Popover
    - 添加 close-delay 属性 (#16671 by @LachlanStuart)
- Theme
    - 增加 Chrome 插件: Element Theme Extension (#16686 by @iamkun)
- Icon
    - 支持 font-display 属性的配置 (#16805 by @iamfaizalandyka)

Bug fixes

- Table
    - 在表头拖拽后阻止 click 事件的触发 (#16850 by @ziyoung)
    - 修复表头 display 为 none 造成浏览器崩溃的问题 (#16956 by @luckyCao)
    - 修复没有数据的时表格高度问题 (#16861 by @ziyoung)
    - 调用 toggleExpansion 不再抛出异常 (#16304 by @yyjjqq94)
    - 挂载时不再触发 sort-change 事件 (#17113 by @a631807682)
    - 修复 setCurrentRow 方法不生效的问题 (#16879 by @ziyoung)
    - 修复当数据异步加载时，expand-row-keys 不生效的问题 (#16899 by @ziyoung)
    - 把 toggleAllSelection 设置为 Table 示例的属性 (#17137 by @ziyoung)
- Tree
    - 修复文字与复选框之间的距离 (#16799 by @Hazlank)
- Tabs
    - 修复 TabItem 位置不正确的问题 (#16520 by @victorting)
    - 修复高亮的 Tab 不在可视区的问题 (#17033 by @nullptru)
- Calendar
    - 修复日期的显示问题 (#16772 by @ubitoffee)
    - 修复在夏令时的显示问题 (#17208 by @iamkun)
- Cascader
    - 修复 CascaderPanel 的显示问题 (#16716 by @zhangHongEn)
    - 禁用状态下，关闭按钮不显示 (#16224 by @yyjjqq94)
- Input

        - 修复韩语输入问题 (#15069 by @MoonHyuk)
        - 触发清除按钮的点击事件 (#16576 by @a631807682)
    - Select
        - 过滤时，不收起下拉框 (#17205 by @luckyCao)
    - Transfer
        - 修复样式问题 (#17206 by @iamkun)
    - Dialog
        - 添加 SCSS 变量 (#16365 by @haoranyu)
    - RadioGroup
        - is 指定时，不产生非法的 HTML 片段 (#17070 by @nullptru)
    - Divider
        - 支持自定义类 (#17078 by @island205)
    - Carousel
        - 修复 change 的触发时机 (#16705 by @iamkun)
    - Notification
        - 不修改传入的 option (#16704 by @iamkun)
    - DatePicker
        - 给 picker-option 添加 className 属性 (#16632 by @iamkun)
    - DateTimePicker
        - 修复时间选择滚动条的问题 (#16854 by @jesse-li)

Optimization

- Checkbox
    - 提高可访问性 (#16575 by @tylertrotter)
- Docs
    - 更新 changelog (#16773 by @SimonaliaChen)
    - 更新贡献指南 (#14800 by @sinchang)
    - 修复 Drawer 文档中的拼写错误 (#16848 by @winkay)
    - 更新自定义主题 (#16983 by @iamkun)
    - 新增 Esperanto 翻译 (#16955 by @maxkoryukov)
    - 更新 input-number 文档 (#16316 by @luckyCao)
    - 更新 Spanish 文档 (#16961 #16548 by @Gonzalo2310)
- I18n
    - 更新加泰罗尼亚语翻译 (#14722 by @oscaralbareda)
    - 更新阿拉伯语翻译 (#16653 by @l3op)
- Test
    - 修复拼写错误 (#16672 by @boomler)
    - 优化 image 的单元测试 (#16847 by @a631807682)
- Types

        - 修复 httprequest 的类型 (#16633 by @luckyCao)

2.11.1

2019-07-26

Bug 修复

- Image
    - 修复 Image 组件 SSR 兼容性 (#16737 by @luckyCao)
- Chore
    - 更新 dart-sass 的兼容性 (#16744 by @LewisChennnnn)

2.11.0

2019-07-25

新特性

- Drawer
    - 新增抽屉组件 (#16577 by @JeremyWuuuuu)

Bug 修复

- Checkbox
    - 修复 CSS 样式问题 (#16006 by @Hazlank)
- Tree
    - 更新类型定义为泛类型 (#15934 by @JeremyWuuuuu)
    - 修复设置节点 isCurrent 的为 false 的问题 (#15870 by @kkkisme)
- Dropdown
    - 更新 split-button 默认颜色 (#15931 by @JuniorTour)
- Cascader
    - 修复一级菜单更新问题 (#16399 by @luckyCao)
    - 懒加载时设默认值 (#16420 by @luckyCao)
    - 修复当节点值重复时的显示问题 (#15935 by @junyiz)
    - 对外暴露 getCheckedNodes 和修复 options 改变会影响选中的问题 (#16709 by @SimonaliaChen)
- Calendar
    - 更新显示正确的 header 的逻辑 (#16354 by @ziyoung)
- Submenu
    - 修复 append-to-body 问题 (#16289 by @a631807682)
- Table

        - 修复 tree table 数据更新问题 (#16481 by @island205)
    - Select
        - 修复内存泄漏问题 (#16463 by @island205)
    - InfiniteScroll
        - 更新命名和说明 (#16698 by @iamkun)
    - Avatar
        - 修复图片不居中的问题 (#16489 by @luckyCao)
    - Dialog
        - 增加 destroyOnClose 属性 (#16455 by @ziyoung)
    - Image
        - 增加大图预览 (#16333 by @luckyCao)

优化

- Docs
    - 修复 dropdown 示例 (#16193 by @webxmsj)
    - 修正笔误 (#15971 by @howiefh)
- I18n
    - 更新泰文翻译 (#16689 by @ponkrit)
- Chore
    - 更新基础 API 地址 (#16607 by @iamkun)
    - 增加 Form 的主题 token (#16699 by @iamkun)
    - 更新统计 (#16609 by @iamkun)
    - 修复文档锚点问题 (#16692 by @iamkun)

2.10.1

2019-07-02

Bug 修复

- Table
    - 排序 icon 问题修复 (#15439 by @bezany)
    - 修复 `append` slot 存在时布局错位问题 (#16332 by @ziyoung)
    - 修复 `showOverflowTooltip` 更新无效的问题 (#16295 by @a631807682)
    - 修复 `FilterPanel` 中 `Scrollbar` 未注册问题 (#16246 by @ziyoung)
- Chore
    - 更新版本号，修复文档问题 (#16233 by @ziyoung)
    - 修复英文首页样式问题 (#16254 by @iamkun)

优化

- Tag
  - 兼容 IE (#16334 by @ziyoung)
- Chore
  - 更新钉钉3群二维码 (#16236 by @iamkun)
- Doc
  - 更新主题编辑器文档 (#16244 by @iamkun)

2.10.0

2019-06-25

新特性

- I18n
  - 支持乌兹别克语 (#15796 by @ogabek96)
- Calendar
  - 支持 `first-day-of-week` 配置 (#16047 by @ziyoung)
- Avatar
  - 新增 `Avatar` 组件 (#16144 by @luckyCao)
- Upload:
  - 支持自定义缩略图模版 (#13192 by @victorzhuk)

Bug 修复

- Tree
  - 当 `currentKey` 为 `null` 时取消对树节点的高亮 (#15668 by @yyjjqq94)
  - 修复多实例共享数据的问题 #15538 (#15615 by @VanMess)
- Upload
  - 更新 `fileList` 的类型定义 (#15716 by @underfin)
- Table
  - 修复加载 icon 不显示的问题 (#15868 by @ziyoung)
  - 修复复杂表格中 hover 行背景色问题 (#15504 by @cnlon)
  - 修复 `current-row-key` 和选择事件的问题 (#15983 by @ziyoung)
  - `height` 属性接受更多单位 (#16013 by @ziyoung)
  - 修复 `reserve-selection` 无效的问题 (#16135 by @ziyoung)
- Menu
  - 修复 `popper-append-to-body` 设置后，子菜单无法收起的问题 (#15391 by @PanJiaChen)
- Select
  - 修复 `initialInputHeight` 问题 (#15989 by @yyjjqq94)
  - 修复输入中文时 `default-first-option` 的行为问题 (#15431 by @VanMess)

        - `import` 重复 (#16215 by @lengband)
    - Message
        - 类型定义中添加 `offset` 属性 (#16027 by @matjaz)
    - Timeline
        - 修复逆序问题 (#16091 by @ziyoung)
    - Slider
        - 补充 `input` 事件文档 (#15588 by @VanMess)
    - InfiniteScroll
        - 更新包名 (#16125 by @iamkun)
    - MessageBox
        - 修复 `distinguishCancelAndClose` 行为与文档不符的问题 (#15438 by @qingdengyue)
    - PopupManager
        - 修复无法复写 `z-index` 的问题 (#15738 by @luckyCao)
    - Docs
        - 删除不必要的内容 (#16194 by @Alexeykhr)
        - 更正 `Divider` 属性类型 (#15889 by @haoranyu)
    - Chore
        - 更新测试 API 地址 (#15807 by @iamkun)

优化

- Tree -优化循环性能 (#15699 by @KingJeason)
- Theme
    - 更新 GA 打点，修改页底地址链接到主题编辑器 (#16007 by @island205)
- Badge
    - 更新类型定义 (#16198 by @iamkun)
- Avatar
    - 更新主题变量配置 (#16202 by @luckyCao)
- I18n
    - 更新葡萄牙语 (#15776 by @gigioSouza)
    - 更新波斯语 (#15881 by @pamenary)
- Docs
    - 补充入门文档中的组件列表 (#16063 by @pape2016)
    - 更新法语文档 (#16208 by @blombard)
    - 为 `Alert` 添加 默认插槽文档 (#15444 by @Alexeykhr)
    - 更新西班牙语文档 (#15840 by @Gonzalo2310)
    - 更新法语文档中的拼写错误 (#15837 by @blombard)
    - 更新 2.9.2 西班牙文档 (#16185 by @Gonzalo2310)

非兼容性更新

- Form
  - 移除输入框的成功状态 (#16159 by @ziyoung)

2.9.2

2019-06-21

Bug 修复

- Chore
  - 修复 TS 定义文件 (#15805 by @NateScarlet)

2.9.1

2019-05-30

新特性

- Table
  - Tree Table 支持 tree-props, default-expand-all, expand-row-keys 属性, toggle-row-expansion 方法, expand-change 事件 (#15709 by @ziyoung)

Bug 修复

- Table
  - 修复了一些问题 (#15709 by @ziyoung)
- Theme
  - 更新接口域名 (#15784 by @iamkun)

优化

- Chore
  - 更新 InfiniteScroll 组件定义文件 (#15794 by @iamkun)

2.9.0

2019-05-30

新特性

- Backtop

        - 新增 Backtop 组件 (#15541 by @iamkun)
    - PageHeader
        - 新增 PageHeader 组件 (#15714 by @ziyoung)
    - InfiniteScroll
        - 新增 InfiniteScroll 指令 (#15567 by @iamkun)
    - Cascader
        - 新增多选模式和 filter-method 方法 (#15611 by @SimonaliaChen)
    - Message
        - 信息依次展示 (#15639 by @island205)
    - Tag
        - 新增 effect 属性 (#15725 by @SimonaliaChen)
    - Tabs
        - 卡片模式下标题左对齐 (#15695 by @luckyCao)
    - DatePicker
        - 支持字符串常量 (#15525 by island205)
    - Image
        - 支持 attrs 和 listeners (#15578 by @VanMess)
    - Theme
        - 新增 popup 背景配置 (#15412 by @iamkun)
    - Chore
        - 更新文档首页 (#15682 by @iamkun)

Bug 修复

- Table
    - 修复排序条件为空时的排序问题 (#15012 by @joelxr)
- Image
    - 修复 ssr 问题和 object-fit 的兼容性 (#15346 by @SimonaliaChen)
- Input
    - 修复 show-word-count 样式问题 (#15359 by @lvjiaxuan)
    - 修复删除图标样式 (#15354 by @YiiGuxing)
- Calendar
    - 修复星期展示错误 (#15399 by @qingdengyue)
    - 修复十月展示问题 (#15394 by @qingdengyue)
- Tabs
    - 修复 padding 问题 (#15461 by @SimonaliaChen)
- Tag
    - 修复阻止冒泡问题 (#15150 by @infjer)
- Form
    - 修复 form-item 的高度错误 (#15457 by @SimonaliaChen)

        - 修复 resetFields 问题 (15181 by @luckyCao)
    - Tooltip
        - 修复自定义 tabindex 不生效问题 (#15619 by @SimonaliaChen)
    - Link
        - 修复图标 class 问题 (#15752 by @iamkun)
    - Select
        - 回滚清除时，设置 value 为 null 的修改 (#15447 by @iamkun)
    - Loading
        - 修复 Dom 不更新的问题 (#15123 by @FAKER-A)
    - Switch
        - 修复事件重复触发问题 (#15178 by @FAKER-A)
    - Slider
        - 修复点击时样式问题 (#15561 by @luckyCao)
    - Radio
        - 修复 value 不更新的问题 (#14809 by @OverTree)
    - Form
        - 修复 resetFields 问题 (15181 by @luckyCao)
    - Chore
        - 更新依赖 (#15324 by ziyoung)
    - Type
        - 修复 Loading 定义文件 (#15635 by @iamkun)
        - 修复 Icon 定义文件 (#15634 by @iamkun)
        - 修复 Link 定义文件 (#15402 by @iamkun)

优化

- Cascader
    - 重构 (#15611 by @SimonaliaChen)
- Chore
    - 更新新建组件的脚本 (by @iamkun)
- Docs
    - 重新命名文档变量 (#15185 by @liupl)
    - 更新 Image 组件文档 (#15423 by @haoranyu)
    - 修复 Form 组件文档错误 (#15228 by @SHERlocked93)

2.8.2

2019-04-25

Bug 修复

- Icon
  - 更新 icon (#15272 by @iamkun)
- 文档
  - 修复 Form 与 Input 文档样式 (#15273 by @ziyoung)

2.8.1

2019-04-25

Bug 修复

- Icon
  - 更新 Select 与 Cascader 的 icon (#15264 by @SimonaliaChen)
  - 更新 icon (#15258 by @iamkun)

优化

- Chore
  - 更新构建脚本 (#15267 by @ziyoung)
- Docs
  - 修复 link 的样式 (#15265 by @iamkun)
- 其他
  - migrating 配置兼容驼峰名称 (#15260 by @SimonaliaChen)

2.8.0

2019-04-25

新特性

- Divider
  - 新增 Divider 组件 (#15055 by @island205)
- Rate
  - 支持通过对象自定义 colors 与 icon-classes 属性 (#15051 by @SimonaliaChen)
- Link
  - 新增 Link 组件 (#15052 by @iamkun)
- Calendar
  - 新增 Calendar 组件 (#14908 by @ziyoung)
- Icon
  - 新增图标 (#15214 by @iamkun)
- Alert

        - 新增高饱和度主题 (#15041 by @island205)
    - Image
        - 新增 Image 组件 (#15117 by @SimonaliaChen)
    - Collapse
        - CollapseItem 支持禁用 (#15076 by @ziyoung)
    - Carousel
        - 新增 direction 属性，支持垂直方向切换 (#15122 by @ziyoung)
    - Pagination
        - 新增 hide-on-single-page 属性 (#15096 by @ziyoung)
    - Slider
        - 新增 marks 属性 (#15133 by @luckyCao)
    - Input
        - 新增 show-word-count 属性 (#15075 by @luckyCao)
    - InputNumber
        - 新增 step-strictly 属性 (#15050 by @luckyCao)
    - Tooltip, Dropdown, Popover
        - 新增 tabindex 属性 (#15167 by @ziyoung)

Bug 修复

- Notification
    - 修复标题不换行的问题 (#15008 by @iamkun)
- Form
    - 修复动态表单校验规则不生效的问题 (#14985 by @luckyCao)
    - 修复 label 的样式 (#14969 by @ziyoung)
    - 当 required 为 true 时，显示星号 (#15144 by @ziyoung)
- Pagination
    - 修复 slot 未更新的问题 (#14711 by @lucyhao)
- Table
    - 修复懒加载时加载数据的 bug (#15101 by @ziyoung)
    - 在合并单元格时，修复单元格的宽度计算不正确的问题 (#15196 by @ziyoung)
    - 提升表格的性能 (#14868 by @ziyoung)
    - 初始化时不再触发 sort-change 事件 (#14625 by @PeanutWatson)
    - 让 height 与 max-height 属性的行为保持一致 (#14660 by @arthurdenner)
- Dialog
    - 修复内容不换行的问题 (#15027 by @iamkun)
- Alert
    - 更新 typescript 定义文件 (#15186 by @ziyoung)
- Tabs

        - Fix issue where Promise rejection was hitting application (#14816 by @ffxsam)
        - slot 改变时，重新渲染 (#15238 by @ziyoung)
    - Message
        - 修复 typescript 定义文件 (#14968 by @agoni1212)
    - Select
        - 修复当 value 为 undefined 或者 null 的报错 (#15022 by @luckyCao)
    - Tree
        - 当前节点被删除后，选中的节点也应该删除 (#14604 by @sinchang)
        - 提升性能 (#14881 by @ChenZhuoSteve)
    - Dropdown
        - 修复样式 (#14907 by @doing123)
    - Slider
        - 修复可访问性问题 (#14792 by @erezsob)
    - Menu
        - 如果 defaultIndex 不存在，activeIndex 应该为空 (#14074 by @hoythan)
    - Directive
        - RepeatClick: 使用 Date.now 提升性能 (#14776 by @pavelmash)
    - Upload
        - 修复 Upload 的背景颜色 (#15039 by @iamkun)
    - Theme
        - 添加无圆角变量 (#15256 by @iamkun)

优化

- Chore
    - 更新中文 changelog (#14965 by @iamkun)
    - 当 demo 描述为空时，不再显示 (#15014 by @ziyoung)
    - 显示 DevServer 的信息 (#15028 by @iamkun)
    - 修复 2.6 changelog 的 bug (#15026 by @iamkun)
    - 更新构建脚本 (#14821 by @abc3660170)
    - 本次开发时支持热更新 (#15221 by @SimonaliaChen)
    - 本地开发时，加载 sourcemap (#15087 by @ibufu) Docs
    - 重命名 demo 中的变量 (#14602 #15003 #15094 #15105 by @liupl)
    - 修复 upload 文档中的错误 (#15023 by @iamkun)
    - 更新 Form 文档 (#15040 by @iamkun)
    - 更新 Tabs 文档 (#15053 by @iamkun)
    - 使用 eleme.cn 作为新域名 (#15139 by @ziyoung)
    - 修复 Image 组件的路由名 (#15194 by @iamkun)

        - 删除多余的法语翻译 (#15207 by @iamkun)

非兼容性更新

- Rate
    - 禁用情况下，显示小位数 (#15089 by @haoranyu)
- Select
    - 过滤情况下，placeholder 为选中选项的 label (#14989 by @ibufu)

2.7.2

2019-04-03

修复

- Form
    - 修复 `label-width` 为 `auto` 的样式 (#14955 by @ziyoung)

优化

- Docs
    - 修复文档内图片链接错误 (#14957 by @iamkun)
- Chore
    - 修复发布时 mkdir 异常 (#14952 by @iamkun)

2.7.1

2019-04-03

修复

- Select
    - 清空时设置 value 为 null (#14322 by @aaronfulkerson)
- Input
    - 当类型改变时更新 DOM (#14889 by @wacky6)
- Table
    - 修复当有展开列时 `defaultExpandAll` 的行为 (#14935 by @ziyoung)
- Dialog
    - 可以设置背景色 (#14939 by @ziyoung)
- Form
    - `label-width` 支持自动宽度 (#14944 by @ziyoung)

优化

- Docs
  - 更新西班牙语文档 (#14913 by @Gonzalo2310)
  - 新增组件自动生成法语文档 (#14924 by @ziyoung)
  - 更新 Tabs 文档 (#14938 by @ziyoung)

2.7.0

2019-03-28

新特性

- Table
  - 增加对树形结构数据的支持 (#14632 by @ziyoung)

修复

- Tabs
  - 阴影样式使用全局主颜色 (#14558 by @Richard-Choooou)
  - 当 label 改变时触发更新 (#14496 by @akki-jat)
- Table
  - Table footer 与 body 的对齐一致 (#14730 by @ziyoung)
- NavMenu
  - 修复点击 el-submenu 多次触发 childMenu 问题 (#14443 by @PanJiaChen)
- Dropdown
  - 兼容 Vue 2.6 新 v-slot 语法 (#14832 by @ziyoung)
- ColorPicker
  - 修复十六进制颜色字符串解析问题 (#14793 by @iamkun)
- Tree
  - 恢复 pr #13349 (#14847 by @ziyoung)
- Tooltip
  - 当初始值为 true 时默认显示 (#14826 by @ziyoung)
- Docs
  - 更新 Cascader 文档 (#14442 by @panhezeng)
- Style
  - 修复媒体查询 sm-only, md-only, lg-only 问题 (#14611 by @sinchang)

优化

- Chore
  - 增加网页描述信息 (#14802 by @iamkun)

- ## 2.6.3

  2019-03-21

  ### 修复

  - 修复 Cascader 文档页的样式 (#14789 by @ziyoung)
  - 移除 Cascader 中多余的 DOM 操作 (#14788 by @ziyoung)
  - DateRange 支持夏令时 (#14562 by @wacky6)

- ## 2.6.2

  2019-03-21

  ### 新特性

  - DatePicker
    - 支持 monthrange 类型 (#14487 by @zxyRealm)
  - i18n
    - 添加 Croatian 语言包 (#14360 by @danijelh)
  - Docs
    - 更新 2.6.1 法语文档，修复笔误 (#14555 by @smalesys)
    - 更新法语翻译 (#14643 by @smalesys)

  ### 修复

  - Input
    - Fix regression (#14572 by @wacky6)
  - DatePicker
    - 修复 first-day-of-week 的计算 (#14523 by @sinchang)
    - 修复 WeekPicker value-format 的问题 (#13754 by @wacky6)
  - Steps
    - 修复 #14502 (#14596 by @sinchang)
    - 修复简单模式下的样式 (#14610 by @sinchang)
  - Docs
    - 重命名 Table 文档中的变量 (#14587 by @likwotsing)
    - 添加法语文档索引 (#14565 by @iamkun)
    - 修复 TimePicker 文档页的样式 (#14579 by @ziyoung)
    - 重命名 Upload 文档中的变量 (#14593 by @liupl)
    - 在 Form 文档中 添加的 async-validator 文档 (#14694 by @iamkun)
    - 修复 Tooltip 文档的 bug (#14748 by @iamkun)
    - 修复笔误 (#14751 by @2bj)

        - 修复 Switch 在移动端 Webkit 浏览器的高亮问题 (#14703 by @VladG0r)

优化

- Chore:
    - 更新 ci 构建脚本 (#14600 by @ziyoung)
    - 更新谷歌统计 (#14560 by @iamkun)
    - 添加更多谷歌统计事件 (#14633 by @iamkun)
    - 更新聊天组信息 (#14741 by @iamkun)
    - 升级测试依赖 (#14735 by @wacky6)
    - 升级 gulp (#14745 by @ziyoung)
    - 使用 codepen 显示 demo，修复文档中的错误 (#14747 by @ziyoung)

2.6.1

2019-03-03

修复

- 不再指定 **node** 版本 (by @iamkun in #14546)
- 调整 `deloy-faas.sh` 中的文档目录 (by @ziyoung in #14553)
- 调整 2.6.0 中 changelog 日期样式 (by @island205 in #14547)
- 修复拼写错误 (by @wack6 in #14552)

2.6.0

2019-03-01

- 新特性
  - Timeline
    - 添加 Timeline 组件 (by @jikkai in #14248)
  - DropdownItem
    - `el-dropdown-item` 支持添加 icon (by @gabrielboliveira in #14088)
  - Input
    - 添加 `show-password` 属性，支持配置显示密码按钮 (by @phshy0607 in #13966)
  - Select
    - 添加 slot `empty` (by @elfman in #13785)
  - Autocomplete
    - 添加 `highlight-first-item` 属性，控制是否默认突出显示远程搜索建议中的第一项 (by @YamenSharaf in #14269)
  - I18n
    - 添加亚美尼亚语支持 (by @hamletbarsamyan in #14214)
  - Docs
    - 新增法语文档 (by @smalesys in #12153, #14418, #14434)

- 优化
  - Alert
    - 组件对通过 slot 传入的 description 也应用默认样式类 (by @iamkun in #14488)
  - InputNumber -移除多余的 `parseFloat` (by @JuniorTour in #14172)
  - Menu
    - 支持 `el-menu-item` 不添加 index (by @georgyfarniev in #13298)
  - Table
    - 移除无用的 DOM 操作 (by @elfman in #13643)
  - Upload
    - 代码优化 (by @elfman in #13973)
  - Popup
    - 移除无用代码 (by @KAionro in #14413)
  - Docs
    - 添加更多文档说明如何贡献代码 (by @island205 in #14355)
    - 添加 `el-input` 是受控组件的警示 (by @wacky6 in #14463)
    - 优化 Table 的文档 (by @luguokong in #14329)
    - 更新 Input 文档 (by @iamkun in #14437)
    - 优化自定义主题文档 (by @wangguohao in #14297)

        - 为 Icon 文档添加 hover 效果 (by @tuxinghuan in #14295)
    - Build
        - 压缩 Element 文档站的 JS 和 CSS 文件 (by @iamkun in #14430)
        - 优化 Webpack 打包速度，从6分钟优化到1分多 (by @hetech in #14484)
        - 添加 CLI 工具，选择版本号 (by @hetech in #14354)
    - 使用 Stale 来管理过时（暂定1年）的 Issue 和 PR (by @island205 in #14392)

问题修复

- Menu
    - 修复浏览器标签切换引起的 focus 问题 (by @liupl in #13976)
- MessageBox
    - 修复 TS 定义 (by @NateScarlet in #14278)
- ScrollBar
    - 修复点击鼠标右键导致拖动的问题 (by @xifeiwu in #14196)
- Switch
    - 添加 `validate-event` 属性，设置改变 Switch 状态时是否触发表单的校验 (by @hetech in #14426)
- Table
    - 修复多 Table 实例共享 `toggleAllSelection` 方法，造成无法切换问题 (by @letanure in #14075)
- Tabs & Dropdown
    - 修复样式问题 (by @hetech in #14452)
- Tree
    - 与 Table 统一占位文样式 (by @ColinCll in #14331)
- Docs
    - 修复 DatetimePicker 文档问题 (by @iamkun in #14290)
    - 修复 DatePicker 文档拼写问题 (by @helmut in #14481)
    - 修复分页组件文档样式问题 (by @liuchuzhang in #14451)

非兼容性更新

- Table
    - 修复 row 事件的参数顺序 (by @jikkai in #12086)

2.5.4

2019-02-01

修复

- 构建: 修复 `.babelrc` 配置问题—导致 Tree 等组件没有动画 (by @island205 in #14282)

2.5.3

2019-01-31

优化

- 优化 Message 的代码 (by @vok123 in #14029)
- 移除 gh-pages (by @ziyoung in #14266)
- 添加 IssueHunt 的链接 (by @island205 in #14261)

修复

- 修复 UMD 包在服务器端运行出错的问题 (by @island205 in #14242)
- 修复 Tabbar 高亮时的样式 (by @iamkun in #14240)
- 修复 Table 示例代码的错误 (by @xunmeng in #14253)

2.5.2

2019-01-27

优化

- 文档:
  - 2.5.1 版本西班牙语文档更新 (by @Gonzalo2310 in #14231)

修复

- 构建:
  - 删除 umd 模块 `lib/index.js` 中本没有的注释 (by @island205 in #14233)
  - 修复 nuxt.js 中关于 `export` 关键字的报错 (by @island205 in #14232)
  - 修复发布 2.5.1 过程中的错误 (by @iamkun in #14228)

2.5.1

2019-01-26

优化

- DatePicker: 添加月、年高亮的样式 (by @Debiancc in #14211)

- 更新 2.5.0 changelog （by @wacky6 in #14217）

修复

- 修复升级 Webpack 4 产生的问题，无法具名 `import` 组件，`ELEMENT.locale()` 调用报错。（by @island205 in #14220）
- 恢复 2.4.11 文档 (by @iamkun in #14222)

2.5.0

2019-01-25

新特性

- DatePicker
  - 新增 `validate-event` 属性 (by @ziyoung in #13531)
- DateTimePicker
  - `pickerOptions` 支持 `selectableRange` 选项 (by @eeeeeeeason)
- Tag
  - 新增 `click` 事件 (by @licdream in #14106)
- I18n
  - 新增 柯尔克孜语 (Kyrgyz) (by @zzjframework in #14174)

优化

- 升级到 webpack@4 (by @jikkai in #14173)
- Input
  - 简化内部实现，遵循单向数据流；修复若干相关 Bug (by @wacky6 in #13471)
- 更新 Axure 文件，增加新组件 (by @ziyoung in #13773)

修复

- Autocomplete
  - 修正下拉框最后一行显示不完整的问题 (by @ziyoung in #13597)
  - 修正下拉框箭头 (by @liuchuzhang in #13762)
- Carousel
  - 组件销毁时释放内部 Timer (by @elfman in #13820)
- Cascader
  - 移除已废弃的计算属性的 cache 属性 (by @iamkun in #13737)
  - 修正 TypeScript 中 CascaderOption 类型定义 (by @NateScarlet in #13613)
  - 修正图标覆盖文字的问题 (by @ziyoung in #13596)

- Checkbox
  - 改进显示样式 (by @PanJiaChen)
- DatePicker
  - 修正 TimeSpinner 中缺失的 v-for `key` 属性 (by @Ende93 in #13547)
  - 修正周选择器在跨年时的高亮行为 (by @suyi91 in #13883)
- Input
  - 修复 textarea 时的 DOM 节点引用 (by @laomu1988 @island205 in #13803)
- Pagination
  - 输入框的值不会小于 1 (by @elfman in #13727)
- Popover
  - 修正 hover 的触发行为 (by @goldengecko in #13104)
  - 修正弹出框的内存泄漏 (by @qpxtWhite in #13988)
- Radio
  - 改进显示样式 (by @ohhoney1)
- Table
  - 改进点击排序箭头时的行为 (by @ohhoney1 in #12890)
  - 修正 IE10+ 中 “暂无数据” 提示的垂直布局 (by @imzjy in #13638)
  - 修正文档中 `index` 的类型说明 (by @ilovefafa in #13628)
  - 修正多级表头使用 `fixed` 属性时，表尾合计行的显示样式 (by @luckyCao in #13914)
- Tabs
  - 修正自动滚动 (by @iamkun in #13696)
  - 通过面板名称查找面板 (by @iamkun in #13705)
  - 使用 `paneName` 计算面板样式 (by @iamkun in #13733)
- Tree
  - 修正 `showCheckbox` 不能影响子节点的问题 (by @KidneyFlower)
  - 更新文档和 TypeScript 定义 (by @ziyoung in #13540)
- Upload
  - `list-type` 改变时，保留 `url` 属性 (by @elfman in #13771)
- Slider
  - 修正源代码缩进 (by @wacky6 in #13955)
- I18n
  - 补充加泰罗尼亚语 (Catalan) 翻译 (by @jaumesala)
  - 补充俄语 (Russian) 翻译 (by @justlp in #13658)
  - 补充芬兰语 (Finnish) 翻译 (by @jenkrisu in #14137)
- Doc
  - 更新西班牙语文档至 2.4.11 (by @Gonzalo2310 in #13522)
- 其它

- 移除多余的构建脚本 (by @ziyoung)
- 修正文档超链接 (by @iamkun in #13753)
- 修正文档中不一致的大小写 (by @wonderjar)
- 增加钉钉群的二维码 (by @iamkun in #13957)
- .gitignore 增加 yarn 日志文件 (by @mimimi in #13922)
- 移除赞助商 多态 (by @island205 in #14156)
- Update readme qr code src (by @iamkun in #13960)
- 更新 CDN 链接，修正错别字 (by @ziyoung)

2.4.11

2018-11-21

- 撤销 pr #13296，修复点击 Menu 外部导致 Submenu 收起的问题，#13478
- 调整小屏幕（xs）媒体查询断点，#13468 （by @alekoshen712）

2.4.10

2018-11-16

- 修复多次点击 Select 才显示下拉列表的问题，#13268
- Form 禁用时不显示 Input 的 clear 图标，#13208
- 调整 Select，Progress，Autocomplete，Tooltip，Collaspe，TimePicker 的样式，#13188 (by @porcelainHeart) #13210 #13266 #13257 #13290 #13347 (by @PanJiaChen)
- Carousel 组件新增 `loop` 属性，#13217
- Table 的 data 改变时，高亮行会继续保留，#13200
- Table 的 header slot 可以接收参数，#13263
- Table 的 `clearFilter` 方法支持参数，#13176
- Table 单元格内没有内容时不再创建 Tooltip，#13152 (by @rongxingsun)
- ColorPicker 面板的输入框内容可以正常显示了，#13278
- 在拖拽时，ColorPicker 不再触发表单校验，#13299
- InputNumber 新增 `select` 方法，#13286 (by @st-sloth)
- Autocomplete 新增 `clear` 事件，#12171(by arthurdenner) #13326
- 可以通过点击 Menu 外部来关闭 Menu，#13296
- Form 的 `validateField` 方法可以接收参数，#13319
- Cascader 新增 `visible-change` 事件，#13415
- DatePicker 新增 range-separator slot， #13272 (by @milworm)
- Tree 新增 `iconClass` 与 `currentNodeKey` 属性，#13337 #13197 (by @isnifer)
- Progress 的 `status` 添加了 text #13198 (by @ali-master)

- 修复 Tree 的 `defaultCheckedKeys` 导致显示的错误，#13349 (by @dive2Pro)

2.4.9

2018-10-26

- Form 组件 clearValidate 方法参数支持字符串，#12990 (by @codinglobster)
- Badge 新增 type 属性，#12991
- 用户可以使用 scoped-slot 来自定义表头，#13012 (by @ivanseidel)
- 修复 IE 下 Select 输入框不能输入的问题，#13034 (by @GaliMU)
- Select 多选时，选项不换行，#12329 (by @akki-jat)
- Select 下拉列表展开后，箭头图标也可以正确显示，#12353 (by @firesh)
- 修复 Select 的 size 属性不生效的问题，#13070
- 多选时可以清除 Select 已选中的值，#13049 (by @ZSkycat)
- 修复最后一个 TabNav 不能删除的问题，#13039
- 修复 TabNav 中 label 显示不正确的问题，#13178
- Alert 新增 title slot，#13082 (by @Kingwl)
- 修复 Table 中的 tooltip 内容不正确的问题，#13159 (by @elfman)
- 优化 Upload 文件列表删除时的动画，#12987
- 当 InputNumber 控制按钮不显示时，调整了边距，#13052

2.4.8

- Switch 聚焦时不显示轮廓，#12771
- 修复 Dropdown 在 ButtonGroup 中样式问题，#12819 (by @bluejfox)
- Dialog 新增 opened 事件，#12828
- 修复 TabNav 显示顺序不正确的问题，#12846
- 修复 Tabs 没有滑动到选中 tab 的问题，#12948
- 修复 Tree 节点在拖拽时标识符不显示的问题，#12854
- Form 的 validate 事件参数中包含了校验的信息，#12860 (by @YamenSharaf)
- 修复 DatePicker 没有校验用户输入时间的合法性问题，#12898
- 修复 Table 表头的 `render-header` 属性不生效的问题，#12914

2.4.7

2018-09-14

- 修复 DatePicker 未触发表单检验的问题，#12328，#12348
- 修复 DatePicker 多选时报错的问题，#12347
- 修复 DatePicker 选择时间时 spinner 位置不正确的问题，#12415 (by @rang-ali)

- 修复 Datepicker 输入框自动填充的问题，#12521 （by @abdallanayer）
- 修复 Cascader 中 Input 未高亮的问题，#12341
- 修复 Tabpane 顺序不正确的问题，#12346
- 修复 ColorPicker 取色光标位置不正确的问题，#12376 （by @cnwhy）
- 调整 Submenu 的样式，#12457
- 修复 Submenu 选中后没有高亮的问题，#12479
- 修复 Cascader 选择值不正确的问题，#12508 （by @huangjinqiang）
- 修复 Pagination 输入框值不正确的问题，#12525
- 调整 Pagination 触发事件的顺序，#12530
- 修复 Table 的 filter 不显示的问题，#12539
- 修复 Tree 无法删除节点的问题，#12684
- 修复 Select 在单选时 Input 高度变化的问题，#12719
- 修复 Form 在嵌套时 label 显示不正确的问题，#12748
- 新增 Input 的 autocomplete 属性，废弃 auto-complete 属性，#12514 （by @axetroy）
- 新增 Form 的 slot-scope 展示表单校验信息，#12715 （by @YamenSharaf）

2.4.6

2018-08-09

- 修复 Table 的 filter 初始值为空数组时不显示筛选图标的问题，#12165
- 修复 Menu 在更改 `collapse` 时不保存菜单激活状态的问题，#12178 (by @elfman)
- 修复 Cascader 未转义特殊字符的问题，#12248
- 修复禁用的 RadioButton 在点击时显示 box-shadow 的问题，#12262
- 修复 Select 初始值为 `undefined` 时方向键失效的问题，#12322
- 修复 Select 多选时输入的关键字消失的问题，#12304
- 修复 Select 多选时查询函数没有去抖的问题，#12181
- 修复 Dialog 在全屏显示时宽度不正确的问题，#12203
- 修复 Main 在 IE 下的显示不正确的问题，#12237
- 修复 Input 触发两次表单校验的问题，#12260
- 修复 Tree 在懒加载时添加节点导致节点消失的问题，#12256
- 修复 Tree 节点在拖拽后无法删除的问题，#12279
- 修复 Popover 在 InputNumber 聚焦时不显示的问题，#12284
- 添加 Autocomplete 的 popper-append-to-body 属性，#12241
- 添加 Pagination 的 `page-size` 属性 `sync` 修饰符的支持，#12281

2.4.5

2018-07-26

- 修复 Table 设置 `class-name` 对 `expand` 列不生效的问题，#12006
- 新增 Table 的 `toggleAllSelection` 方法，#12047
- 修复 Input 包含 Select 时，suffix 插槽位置显示不正确的问题，#12108
- 修复 Option 的 `line-height` 无法设置的问题，#12120
- 修复初始值为 `null` 的 TimeSelect 在执行 `resetField` 后无法再赋值的问题，#12010
- 修复 Tree 组件中不响应方向键以外 keydown 事件的问题，#12008
- 修复 Tree 在懒加载情况下选中父节点的问题，#12106
- Tree 的 `getCheckedNodes` 方法新增 `includeHalfChecked` 参数，#12014

2.4.4

2018-07-13

- 修复重置表单后触发 Select 组件校验问题，#11837
- 修复 Input 组件 `suffix` 与 `append` 共存时样式错乱问题，#11951
- 修复可清空的只读 Input 仍会显示清空图标的问题，#11967
- 修复 Tree 节点禁用时仍可以选中的问题，#11847
- 修复 Tree `default-checked-keys` 属性不生效的问题，#11971
- 修复 Tree 在过滤节点时下 `empty-text` 不显示的问题，#11971
- 修复 Table 的 `empty-text` 过长时的位置样式问题，#11965
- 修复 Table 的 `current-row-key` 设置为 `null` 时高亮行不清除的问题，#11866
- 修复当 `filters` 为空数组时显示过滤器下拉列表的问题，#11864
- 修复 Radio 的 label 不阻止事件冒泡的问题，#11912

2.4.3

2018-07-03

- 修复当自定义 Tree 节点高度时，`allow-drop` 不能正常工作的问题，#11797
- 现在 Form 的 `clearValidate` 方法支持传入参数，指定需要清空校验结果的 FormItem，#11821
- 新增 MessageBox 的 `distinguishCancelAndClose` 属性，#11831

2.4.2

2018-06-26

- 修复 Table 的 `class-name` 和 `label-class-name` 属性不支持动态更新的问题，#11626

- 修复 Table 在 `highlight-current-row` 为 `false` 时点击行也会触发高亮的问题，#11691 #11563
- 修复 ButtonGroup 中只有一个 `round` 或 `circle` 的 Button 时的样式错误，#11605
- 修复在某些情况下 Pagination 的条目数选择器的样式错误，#11622
- 修复 Menu 的 `collapse` 属性变化后无法使用 `open` 方法的问题，#11646
- Tabs 的 `before-leave` 钩子添加了 `activeName` 和 `oldActiveName` 参数，#11713
- 修复 Cascader 关闭后的聚焦问题，#11588
- 修复 Cascader 在 `change-on-select` 状态下点击选项不关闭的问题，#11623
- 现在通过代码改变 Select 的值后会触发表单校验，与 Input 行为一致，#11672

2.4.1

2018-06-08

- 移除 Autocomplete 的重复类型声明，#11388
- 修复嵌套在 Form 内的 Select 在 FireFox 浏览器中下拉箭头错位的问题，#11427
- 修复 Select 的初始值为 `null` 时仍然显示清除图标的问题，#11460
- 修复禁用的 Radio 在点击时显示 box-shadow 的问题，#11462
- 新增 MessageBox 的 `iconClass` 属性，#11499
- 新增 Tabs 的 `stretch` 属性，#11476
- 修复 Tabs 开启 `lazy` 时渲染顺序异常的问题，#11461
- 修复 Table 展开行时无法保留选中行样式的问题，#11464
- 修复 Tabs 调用 `before-leave` 并返回 Promise 的时候，Tabs 会存在 focus 状态的问题，#11386
- 修复 Popover 禁用状态下创建弹出框的问题，#11426
- 修复 Tree 在懒加载状态下添加新节点造成无限循环的问题，#11430 （by @wangjingf）
- 新增 Dialog 的 `closed` 事件，#11490

2.4.0 Fullerene

2018-05-28

新特性

- 综合
  - 使用原生 webpack 作为构建和打包工具，#11216
  - 可以全局配置弹出层的初始 z-index，#11257
- Autocomplete

    - 新增 `hide-loading` 属性，#11260
- Button
    - 现在圆形按钮也支持通过 `size` 属性改变其尺寸了，#11275
- InputNumber
    - 新增 `precision` 属性，#11281
- Tabs
    - 新增 `before-leave` 钩子，#11259
    - 新增 `lazy` 属性，#11167（by @Kingwl）
- Table
    - 新增 `sort` 方法，支持手动排序，#11311

修复

- Input
    - 修复使用中文输入法快速输入文字时会导致视图重新渲染的问题，#11235（by @STLighter）
- Popover
    - 修复当触发元素为 Radio 或 Checkbox 时控制台报错的问题，#11265
- Breadcrumb
    - 修复 `to` 属性不支持动态更新的问题，#11286
- Upload
    - 修复在 `beforeUpload` 方法返回的 Promise 中 resolve 一个 File 时控制台报错的问题，#11297（by @qusiba）
- Tooltip
    - 修复内容为空时箭头错位的问题，#11335
- Autocomplete
    - 修复在快速删除搜索内容后输入建议不正确的问题，#11323
- ColorPicker
    - 修复关闭选色器时触发 `active-change` 事件的问题，#11304
- Table
    - 修复筛选列表过长导致样式超出的问题，#11314
    - 修复排序后导致无法正常显示选中行样式的问题，#11348
- Checkbox
    - 修复单个 Checkbox 不支持表单验证的问题，#11271
- Radio
    - 修复通过空格可以选中被禁用的 Radio 的问题，#11303
- MessageBox
    - 修复连续打开两个 MessageBox 时 `el-popup-parent--hidden` 无法移除的问题，#11371

- ## 2.3.9

  2018-05-18

  - 修复当 TableColumn 的 `prop` 属性指定的字段在数据源中不存在时，鼠标移入该列单元格会报错的问题，#11137
  - 弹出类组件的 `lockScroll` 属性不再为父元素添加内联样式，而是添加相应类名，#11114
  - 修复 Progress 在 `status` 为 exception 时图标不显示的问题，#11172
  - 修复可搜索的 Cascader 在输入关键词后，选项的 `disabled` 属性失效的问题，#11185
  - 修复可展开的 Table 在展开某一行后更新数据源会造成该行无法收起的问题，#11186
  - Tree 的 `setCurrentKey` 方法支持传入 `null`，可取消当前高亮的节点，#11205

- ## 2.3.8

  2018-05-11

  - 修复 `type` 为 dates 的 DatePicker 在选择非当前月的日期后，面板会跳转至当前月的问题，#10973
  - 修复可清空的只读 Input 仍会显示清空图标的问题，#10912
  - 修复范围选择的 DatePicker 在未改变值的情况下关闭下拉面板仍会触发 `change` 事件的问题，#11017
  - 修复 Select 在有分组选项时不能正确通过键盘导航的问题，#11058
  - 新增 Select 的 `prefix` 具名 slot，#11063
  - 新增 FormItem 的 `clearValidate` 方法，#11076
  - 新增 Tree 的 `checkOnClickNode` 属性，#11111

- ## 2.3.7

  2018-04-29

  - 修复 Table 在由于筛选而使原有的滚动条消失后表头各列宽度未及时更新的问题，#10834
  - 修复可清空的 Input 在初始值为 `null` 时仍然显示清空图标的问题，#10912
  - 修复在通过代码改变 ColorPicker 的绑定值后错误地触发 `active-change` 事件的问题，#10903（by @zhangbobell）
  - 修复可搜索的 Select 在备选项均被禁用时，通过键盘导航会造成无限循环的问题，#10945

- ## 2.3.6

2018-04-21

- 修复 Tree 的 `allow-drop` 回调在使用 `type` 参数后的错误行为，#10821
- 修复可搜索的单选 Select 在 IE11 中无法输入搜索关键词的问题，#10822
- 修复单选 Select 在使用鼠标选中某个选项后错误地触发 `blur` 事件的问题，#10822

2.3.5

2018-04-20

- 修复 DatePicker 的 `type` 为 week 时面板错误高亮的问题，#10712
- 修复 InputNumber 初始值为 0 时输入框为空的问题，#10714
- 新增 Select 的 `automatic-dropdown` 属性，#10042（by @Seebiscuit）
- 修复 `disabled` 的 Rate 仍能通过键盘左右键改变组件值的问题，#10726（by @Richard-Choooou）
- 现在 DatePicker 的 `type` 属性可以接收 `'dates'`，用于选择多个日期，#10650（by @Mini256）
- 新增 Pagination 的 `prev-click` 和 `next-click` 事件，#10755
- 新增 Pagination 的 `pager-count` 属性，#10493（by @chongjohn716）
- 新增 `type` 作为 Tree 的 `allow-drop` 属性回调的第三个参数，#10792
- 改用 ResizeObserver 对元素的尺寸变化进行监测，#10779

2.3.4

2018-04-12

- 删除 SubMenu 在 TypeScript 类型声明中重复的 `showTimeout` 属性，#10566（by @kimond）
- 现在 Transfer 数据项的渲染支持通过 scoped slot 自定义，#10577
- 修复点击 Pagination 禁用的上一页、下一页按钮仍会触发 `current-change` 事件的问题，#10628
- 修复未绑定值的 Textarea 在 SSR 中会显示 `undefined` 的问题，#10630
- 修复 `type` 为 border-card 的 Tabs 中被禁用标签项的样式，#10640
- 新增 `$index` 作为 Table 的 `formatter` 属性回调的第四个参数，#10645
- 修复 TypeScript 类型声明未导出 CheckboxButton 的问题，#10666

2.3.3

2018-04-04

- 新增 Card 的 `shadow` 属性，#10418（by @YunYouJun）
- 修复 Badge 在 `value` 属性为 `0` 时不显示上标的问题，#10470

- 修复 Tree 节点拖拽相关的问题，#10474 #10494
- 新增 Autocomplete 的 `placement` 属性，#10475
- 现在 `default-time` 属性也可用于非范围选择的 DateTimePicker 了，#10321（by @RickMacTurk）
- 修复 TabItem 在浏览器失焦和隐藏后出现蓝色边框的问题，#10503
- 新增 SubMenu 的 `popper-append-to-body` 属性，#10515
- 现在非链接的 BreadcrumbItem 在 hover 时不再具有视觉反馈，#10551
- 调整 InputNumber `change` 事件的触发时机，使得在回调中能够取得最新的组件绑定值，#10553

2.3.2

2018-03-29

- 修复 Autocomplete 报错的问题，#10442

2.3.1

2018-03-29

- 修复 Input 的 `type` 属性未传递至原生 input 元素的问题，#10415
- 新增 Select 的 `blur` 方法，#10416

2.3.0 Diamond

2018-03-28

新特性

- Table
  - 现在 TableColumn 的 `formatter` 属性可以是动态的，#10184（by @elfman）
  - 新增 `select-on-indeterminate` 属性，#9924（by @syn-zeta）
- Menu
  - 新增 `collapse-transition` 属性，#8809（by @limichange）
- Input
  - 新增 `select` 方法，#10229
  - 新增 `blur` 方法，#10356
- ColorPicker
  - 新增 `predefine` 属性，#10170（by @elfman）
- Tree
  - 新增 `draggable`、`allow-drop` 和 `allow-drag` 属性，以及 `node-drag-`

`start` 、 `node-drag-enter` 、 `node-drag-leave` 、 `node-drag-over` 、 `node-drag-end` 和 `node-drop` 事件，#9251 #10372（by @elfman）

- Form
  - `validate` 方法新增第二个参数，包含未通过本次校验的表单项信息，#10279
  - 新增 `validate` 事件，#10351
- Progress
  - 新增 `color` 属性，#10352（by @YunYouJun）
- Button
  - 新增 `circle` 属性，#10359（by @YunYouJun）

修复

- Form
  - 修复嵌套复合型 Input 时，FormItem 标签与输入框未对齐的问题，#10189
- Menu
  - 现在折叠状态的菜单项仅在传入 `title` slot 时才显示 Tooltip，#10193（by @PanJiaChen）
- Pagination
  - 修复 `current-change` 在未发生用户交互时错误触发的问题，#10247
- DatePicker
  - 现在时间日期选择器下拉面板中的值能够正确地从 `format` 属性中获取对应格式了，#10174（by @remizovvv）
- Upload
  - 现在拖拽上传会拦截不在 `accept` 属性范围内的文件，#10278

2.2.2

2018-03-14

- 新增 Input 的 `clear` 事件，#9988（by @blackmiaool）
- 现在 ColorPicker 的手动输入支持 `hsl` 、 `hsv` 和 `rgb` 格式了，#9991
- 修复 DatePicker 在清除初始值时不触发 `change` 事件的问题，#9986
- 现在 Rate 的图标类相关属性支持动态更新了，#10003
- 修复含有固定列的 Table 在设置 `max-height` 属性后有时不能及时更新布局高度的问题，#10034
- 现在 DatePicker 的范围选择支持先点选结束日期，再点选开始日期了，#8156（by @earlymeme）
- 新增 Pagination 的 `disabled` 属性，#10006
- 新增 Popover 的 `after-enter` 和 `after-leave` 事件，#10047
- 修复重置表单后，用户第一次改变 Select 的值时不触发校验的问题，#10105
- 修复 Table 的固定列在某些情况下宽度不正确的问题，#10130

- 修复调用 MessageBox 未传入 `title` 时，打开的 MessageBox 会继承上一个实例的 `title` 属性的问题，#10126（by @Pochodaydayup）
- 新增 Slider 的 `input-size` 属性，#10154
- 新增 Transfer 的 `left-check-change` 和 `right-check-change` 事件，#10156

2.2.1

2018-03-02

- 修复 Aside、Header 和 Footer 在某些布局中被压缩的问题，#9812
- 修复设置了 `height` 属性的 Table 在服务端渲染时无法加载的问题，#9876
- 修复可展开的 Table 在展开某一行后高度未重新计算的问题，#9848
- 修复在 DateTimePicker 中手动输入日期后不能正确触发 `change` 事件的问题，#9913
- 修复鼠标右键点击 Select 的输入框会展开选项的问题，#9894（by @openks）
- 新增 Slider 的 `tooltip-class` 属性，#9957
- 现在的 Select 在选中选项后仍然处于 focus 状态，#9857（by @Seebiscuit）
- 新增 Transfer 的 `target-order` 属性，#9960

2.2.0 Graphite

2018-02-12

新特性

- Menu
  - SubMenu 新增 `popper-class` 和 `disabled` 属性，#9604 #9771
  - 现在水平模式下的 Menu 支持多级 SubMenu 了，#9741
- Tree
  - 新增 `node-contextmenu` 事件，#9678
  - 现在可以使用 scoped slot 自定义树节点的模板了，#9686
  - 新增 `getNode`、`remove`、`append`、`insertBefore`、`insertAfter`、`getCheckedKeys`、`getHalfCheckedNodes`、`getHalfCheckedKeys` 方法和 `check` 事件，#9718 #9730
- Transfer
  - 新增 `clearQuery` 方法，#9753
- Select
  - 新增 `popper-append-to-body` 属性，#9782

修复

- Table
  - 修复点击可展开行的展开图标会触发 `row-click` 事件的问题，#9654
  - 修复某些情况下通过拖动改变列宽后，布局没有同步更新的问题，#9668
  - 修复合计行与固定列并存时的样式问题，#9667
- Container
  - 修复布局组件在 IE11 中无法自动填充可用空间的问题，#9655
- Loading
  - 修复在 `mounted` 中修改 `v-loading` 的值为 true 时不能正确显示 Loading 的问题，#9722
- Switch
  - 修复点击时会触发两次原生 click 事件的问题，#9760

2.1.0 Charcoal

2018-01-31

新特性

- Cascader
  - 新增 `focus` 和 `blur` 事件，#9184（by @viewweiwu）
- Table
  - `filter-method` 方法加入第三个参数 `column`，#9196（by @liyanlong）
- DatePicker
  - 新增 `prefix-icon` 和 `clear-icon` 属性，#9237（by @AdamSGit）
  - 新增 `default-time` 属性，#9094（by @nighca）
  - `value-format` 属性增加对 `timestamp` 的支持，#9319（by @wacky6）
- InputNumber
  - 组件绑定变量的值支持 `undefined`，#9361
- Select
  - 新增 `auto-complete` 属性，#9388
- Form
  - 新增 `disabled` 属性，#9529
  - 新增 `validateOnRuleChange` 属性，#8141
- Notificaition
  - 新增 `closeAll` 方法，#9514

修复

- InputNumber
  - 修复初始输入小数点时被重置的问题，#9116
- Dropdown

        - 修复当页面仅有水平滚动条时，某些浏览器下拉菜单定位错误的问题，#9138（by @banzhuanmei）
    - Table
        - 修复带有固定列的 Table 在列数据变化后固定列的个数计算错误的问题，#9188（by @kolesoffac）
        - 修复多级表头最后一列的边框不能正确显示的问题，#9326
        - 修复在 Safari 浏览器中表头错位的问题，#9327
        - 修复带有展开行的表格在展开某一行后，当表格数据更新但 `row-key` 值不变时，该行会自动收起的问题，#9462
        - 修复在一些情况下不必要的多次渲染问题，#9426
        - 修复动态改变 TableColumn 的 `width` 属性时，其宽度计算错误的问题，#9426
    - Loading
        - 修复某些情况下 Loading 不能被正确隐藏的问题，#9313
    - DatePicker
        - 修复 `focus` 方法在范围选择时无效的问题，#9437
        - 修复当目前时刻处于不可选择的范围内时，点击面板上的「此刻」按钮仍能选中目前时刻的问题，#9470（by @wacky6）
        - 修复当在月选择面板中选中天数较少的月份时，日期面板呈现下一个月的问题，#9577（by @wacky6）
    - Steps
        - 修复在 IE 11 中的样式问题，#9454

非兼容性更新

- Menu
    - `collapse` 状态下的弹出菜单现在会插入至 body 元素，修复其位于 Aside 内时弹出菜单不可见的问题，#9263
- Table
    - 勾选多选表格的 checkbox 时不再同时触发 `row-click` 事件，#9467
- Loading
    - 非全屏 Loading 遮罩层的 `z-index` 修改为 2000；全屏 Loading 遮罩层的 `z-index` 值会随页面上的弹出组件动态更新，#9522
- Dropdown
    - `show-timeout` 和 `hide-timeout` 属性现在仅在 trigger 为 `hover` 时生效，#9573

2.0.11

2018-01-08

- 修复 Input 的 `prepend` 或 `append` slot 中 Select 的边框颜色错误，#9089

- 修复 Select 的 `remove-tag` 事件参数与文档不符的问题，#9090
- 新增 SubMenu 的 `show-timeout` 和 `hide-timeout` 属性，#8934（by @HugoLew）
- 修复按需引入 Table 时 `show-overflow-tooltip` 的 Tooltip 样式丢失的问题，#9130
- 修复 Table 在执行 `clearSort` 后点击对应列的排序图标无法正常排序的问题，#9100（by @zEmily）
- 捷克语的 i18n 配置文件由 `cz` 重命名为 `cs-CZ`，#9164

2.0.10

2017-12-29

- 修复了 Table 在固定列和合计行并存时的高度计算错误的问题，#9026
- 修复了 Table 样式 SCSS 文件错误编译的问题，#9028
- 现在 DatePicker 的 `change` 事件只会在 `value` 真正改变的时候触发，#9029（by @remizovvv）
- 新增 Input 的 `tabindex` 属性，#9041（by @dicklwm）

2.0.9

2017-12-24

- 新增 Upload 的 `before-remove` 钩子方法，#8788（by @firesh）
- 修复 FormItem 的 `error` 属性初始值无效的问题，#8840
- 通过指令调用的 Loading 现在支持以 `element-loading-custom-class` 属性的方式设置自定义类名，#8826（by @earlymeme）
- 修复 CarouselItem 为异步获取时被隐藏的问题，#8921
- 新增 Tree 的 `renderAfterExpand` 属性，#8972

2.0.8

2017-12-12

- 新增西班牙语文档
- 修复 `show-timeout` 对点击触发的 Dropdown 无效的问题，#8734（by @presidenten）
- 修复 Form 对于 `trigger` 为 blur 的校验规则触发时机有误的问题，#8776
- 修复 DatePicker 在范围选择时 blur 事件触发时机有误的问题，#8784
- TimePicker 的 `format` 新增对 AM/PM 的支持，#8620（by @firesh）

- ## 2.0.7

  2017-11-29

  - 修复禁用文字按钮的样式问题，#8570

- ## 2.0.6

  2017-11-29

  - 修复 Table 排序图标的样式问题，#8405
  - 修复 `trigger` 为 manual 的 Popover 的触发问题，#8467
  - 新增 Autocomplete 的 `prefix-icon` 和 `suffix-icon` 属性，#8446（by @liyanlong）
  - 新增 Cascader 的 `separator` 属性，#8501
  - 新增 Input 的 `clearable` 属性，#8509（by @lbogdan）
  - 新增 Pagination 的 `background` 属性，#8553

- ## 2.0.5

  2017-11-17

  - 修复上个版本引入的 Popover、Tree、Breadcrumb、Cascader 的 bug，#8188 #8217 #8283
  - 修复 clickoutside 指令的内存泄露问题，#8168 #8225（by @badpunman @STLighter）
  - 修复默认尺寸的多选 Select 在清空选项后输入框高度不随之更新的问题，#8317（by @luciy）
  - 新增 Select 的 `collapse-tags` 属性，用于在多选时以文字代替 Tag，避免组件高度的增大，#8190
  - 修复被隐藏的 Table 会造成 CPU 占用持续增加的问题，#8351
  - 开放 Table 的 `doLayout` 方法，用于重新计算 Table 的布局，#8351

- ## 2.0.4

  2017-11-10

  - 提升 Cascader、Dropdown、Message、Notification、Popover、Tooltip、Tree 的可访问性
  - 修复当视口变窄时 Container 无法同步更新其宽度的问题，#8042
  - 修复 Tree 的 `updateKeyChildren` 在删除子节点时的行为错误，#8100
  - 修复带有边框的 CheckboxButton 在 Form 中高度错误的问题，#8100

- 修复 Menu 在解析自定义颜色时的错误，#8153（by @zhouyixiang）

2.0.3

2017-11-03

- 修复范围选择的 DatePicker `editable` 和 `readonly` 属性无法正常工作的问题，#7922
- 修复嵌套的 Tabs 的样式错误，#7941
- 修复纵向 Steps 中最后一个 Step 的样式错误，#7980
- 修复 Pagination 的 `current-change` 事件触发时机错误的问题，#7995
- 修复由于 Menu 使用了未注册的 Tooltip 造成其在按需引入时报错的问题，#7995

2.0.2

2017-10-31

- 在 InputNumber 的加减按钮上单击鼠标右键不再触发值的改变，#7817
- Form 的 `validate` 方法现在能够正确地在异步校验完成后执行回调了，#7774（by @Allenice）
- 修复 DatePicker 的范围选择在内核为 Chromium 53-57 的浏览器中无法使用的问题，#7838
- 修复 `list-type` 为 picture-card 的 Upload 预览和删除图标丢失的问题，#7857
- 新增 TableColumn 的 `sort-by` 属性，#7828（by @wangfengming）
- 修复周模式下的 DatePicker 在选择某年第一周可能会显示为前一年第一周的问题，#7860（by @hh23485）
- 修复垂直模式的 Steps 中图标宽度的样式错误，#7891
- 增大了 Tree 中展开箭头的点击热区，#7891

2.0.1

2017-10-28

- 修复 RadioButton 和 CheckboxButton 的样式问题，#7793
- 修复 TimePicker 在某些情况下无法滚动的问题，#7811
- 修复部分组件在按需引入时样式不完整的问题，#7811

2.0.0 Carbon

2017-10-27

新特性

- 综合
  - 新增 `theme-chalk` 主题
  - 增强以下组件的可访问性：Alert、AutoComplete、Breadcrumb、Button、Checkbox、Collapse、Input、InputNumber、Menu、Progress、Radio、Rate、Slider、Switch 和 Upload
  - 新增布局组件 Container、Header、Aside、Main 和 Footer
  - 新增 TypeScript 类型声明
  - 重绘了全部图标，并新增了部分图标
  - 新增了一系列基于断点的工具类，用于当视口尺寸满足一定条件时隐藏元素
  - 新增全局配置组件尺寸的功能。在引入 Element 时，配置 `size` 字段可以改变所有组件的默认尺寸
- Button
  - 新增 `round` 属性，用于圆角按钮 #6643
- TimeSelect
  - 可以用 `Up` 、 `Down` 导航，用 `Enter` 选中时间 #6023
- TimePicker
  - 可以用方向键导航，用 `Enter` 选中时间 #6050
  - 新增 `start-placeholder` 和 `end-placeholder` ，用于设置范围选择时两个输入框的占位符 #7169
  - 新增 `arrow-control` 属性，提供另一种交互形式，#7438
- Tree
  - 子节点在首次被展开之前不进行渲染 #6257
  - 新增 `check-descendants` 属性，设置 `lazy` 模式下勾选节点时，是否完全展开整个子树 #6235
- Tag
  - 新增 `size` 属性 #7203
- Datepicker
  - type 为 `datetimerange` 时可以使用 `timeFormat` 格式化时间选择器 #6052
  - 新增 `start-placeholder` 和 `end-placeholder` ，用于设置范围选择时两个输入框的占位符 #7169
  - 新增 `value-format` 属性，支持对绑定值的格式进行自定义，#7367
  - 新增 `unlink-panels` 属性，用于在选择日期范围时取消两个日期面板之间的联动
- MessageBox
  - 新增 `closeOnHashChange` 属性 #6043
  - 新增 `center` 属性，提供居中布局 #7029
  - 新增 `roundButton` 属性，使得内部按钮为圆角按钮 #7029
  - 新增 `dangerouslyUseHTMLString` 属性，使得 `message` 支持传入 HTML 字符串* #6043
  - 新增 `inputType` 属性，用户指定内部输入框的类型，#7651

- Dialog
  - 新增 `width`、`fullscreen`、`append-to-body` 属性，支持嵌套使用
  - 新增 `center` 属性，提供居中布局 #7042
  - 新增 `focus-after-closed`、`focus-after-open` 属性，支持无障碍访问 #6511
- ColorPicker
  - 增加手动输入色值的支持 #6167
  - 新增 `size` 属性，用于控制组件的大小 #7026
  - 新增 `disabled` 属性，用于禁用组件 #7026
  - 新增 `popper-class` 属性，#7351
- Message
  - 图标部分使用 icon 代替图片，从而支持通过 CSS 修改图标背景色 #6207
  - 新增 `dangerouslyUseHTMLString` 属性，使得 `message` 属性支持传入 HTML 字符串* #6207
  - 新增 `center` 属性，提供居中布局 #6875
- Notification
  - 新增 `position` 属性，用于配置 Notification 出现的位置 #6231
  - 新增 `dangerouslyUseHTMLString` 属性，使得 `message` 属性支持传入 HTML 字符串* #6231
  - 新增 `showClose` 属性，用于隐藏关闭按钮 #6402
- Rate
  - 新增 `show-score` 属性，控制是否在右侧显示当前分数 #6295
- Tabs
  - 新增 `tab-position` 属性，控制选项面板内容显示的上、下、左、右四个方向 #6096
- Radio
  - 增加 `border` 属性和 `size` 属性 #6690
- Checkbox
  - 增加 `border` 属性和 `size` 属性 #6690
- Alert
  - 新增 `center` 属性，提供居中布局 #6876
- Menu
  - 新增 `background-color`、`text-color` 和 `active-text-color` 属性，分别用于设置菜单的背景色、菜单的文字颜色和当前激活菜单的文字颜色 #7064
  - 新增 `open` 和 `close` 方法，支持手动打开和关闭 SubMenu，#7412
- Form
  - 新增 `inline-message` 属性，设置后校验信息会以行内样式显示 #7032
  - 新增 `status-icon` 属性，用于在输入框中显示校验结果反馈图标 #7032
  - Form 和 FormItem 新增 `size` 属性，用于控制表单内组件的尺寸，#7428
  - `validate` 方法在不传入 callback 的情况下返回 promise，#7405

    - 新增 `clearValidate` 方法，用于清空所有表单项的验证信息，#7623
- Input
    - 新增 `suffix` 、 `prefix` 的 slot，以及 `suffixIcon` 、 `prefixIcon` 属性，用于给输入框内部增加前置和后置内容 #7032
- Breadcrumb
    - 新增 `separator-class` 属性，可使用图标作为分隔符 #7203
- Steps
    - 新增 `simple` 属性，用于开启简洁风格的步骤条 #7274
- Pagination
    - 新增 `prev-text` 和 `next-text` 属性，用于自定义上一页和下一页的文本 #7005
- Loading
    - 配置对象新增 `spinner` 和 `background` 字段，支持自定义加载图标和背景色，#7390
- Autocomplete
    - 新增 `debounce` 属性，#7413
- Upload
    - 新增 `limit` 和 `on-exceed` 属性，支持对上传文件的个数进行限制，#7405
- DateTimePicker
    - 新增 `time-arrow-control` 属性，用于开启时间选择器的 `arrow-control` ，#7438
- Layout
    - 新增断点 `xl` ，适用于宽度大于 1920px 的视口
- Table
    - 新增 `span-method` 属性，用于合并行或列
    - 新增 `clearSort` 方法，用于清空排序状态
    - 新增 `clearFilter` 方法，用于清空过滤状态
    - 对于可展开行，当该行展开时会获得一个 `.expanded` 类名，方便自定义样式
    - 新增 `size` 属性，用于控制表格尺寸
    - 新增 `toggleRowExpansion` 方法，用于手动展开或关闭行
    - 新增 `cell-class-name` 属性，用于指定单元格的类名
    - 新增 `cell-style` 属性，用于指定单元格的样式
    - 新增 `header-row-class-name` 属性，用于指定表头行的类名
    - 新增 `header-row-style` 属性，用于指定表头行的样式
    - 新增 `header-cell-class-name` 属性，用于指定表头单元格的类名
    - 新增 `header-cell-style` 属性，用于指定表头单元格的样式
    - TableColumn 的 `prop` 属性支持 `object[key]` 格式
    - TableColumn 新增 `index` 属性，用于自定义索引值
- Select

- 新增 `reserve-keyword` 属性，用于在选择某个选项后保留当前的搜索关键词

修复

- DatePicker
  - 选择周数时，`v-model` 结果返回该周第二天的问题 #6038
  - 在 `daterange` 类型中，第一次的输入会被清空的问题 #6021
- DateTimePicker
  - 和 TimePicker 相互影响的问题 #6090
  - 选择时间小时和秒可超出限制的问题 #6076
- TimePicker
  - 失去焦点时无法正确改变 `v-model` 值的问题 #6023
- Dialog
  - 当含有下拉框时，下拉框的打开和关闭会造成文字虚晃的问题 #6088
- Select
  - 提升性能，修复组件销毁时可能导致 Vue dev-tool 卡死的问题 #6151
- Table
  - 修复 Table 在父元素从 `display: none` 变成其他状态时会隐藏的问题
  - 修复 Table 在父元素为 `display: flex` 时可能出现的宽度逐渐变大的问题
  - 修复 `append` 具名 slot 和固定列并存时，动态获取表格数据会导致固定列消失的问题
  - 修复 `expand-row-keys` 属性初始化无效的问题
  - 修复 `data` 改变时过滤条件失效的问题
  - 修复多级表头时固定列隐藏情况计算错误的问题
  - 修复 `max-height` 变更后无法恢复的问题
  - 修复一些样式上的计算错误

非兼容性更新

- 综合
  - 移除 `theme-default`
  - 最低兼容 Vue 2.5.2 和 IE 10
  - 表单组件的 `change` 事件和 Pagination 的 `current-change` 事件现在仅响应用户交互
  - Button 和表单组件的 `size` 属性现在可接受 `medium`、`small` 和 `mini`
  - 为了方便使用第三方图标，Button 的 `icon` 属性、Input 的 `prefix-icon` 和 `suffix-icon` 属性、Steps 的 `icon` 属性现在需要传入完整的图标类名
- Dialog
  - 移除 `size` 属性。现在 Dialog 的尺寸由 `width` 和 `fullscreen` 控制
  - 移除通过 `v-model` 控制 Dialog 显示和隐藏的功能

- Rate
  - `text-template` 属性更名为 `score-template`
- Dropdown
  - `menu-align` 属性变更为 `placement`，增加更多方位属性
- Transfer
  - `footer-format` 属性更名为 `format`
- Switch
  - 由于 `on-*` 属性在 JSX 中会被识别为事件，导致 Switch 所有 `on-*` 属性在 JSX 中无法正常工作，所以 `on-*` 属性更名为 `active-*`，对应地，`off-*` 属性更名为 `inactive-*`。受到影响的属性有：`on-icon-class`、`off-icon-class`、`on-text`、`off-text`、`on-color`、`off-color`、`on-value`、`off-value`
  - `active-text` 和 `inactive-text` 属性不再有默认值
- Tag
  - `type` 属性现在支持 `success`、`info`、`warning` 和 `danger` 四个值
- Menu
  - 移除 `theme` 属性。现在通过 `background-color`、`text-color` 和 `active-text-color` 属性进行颜色的自定义
- Input
  - 移除 `icon` 属性。现在通过 `suffix-icon` 属性或者 `suffix` 具名 slot 来加入尾部图标
  - 移除 `on-icon-click` 属性和 `click` 事件。现在如果需要为输入框中的图标添加点击事件，请以具名 slot 的方式添加图标
  - `change` 事件现在仅在输入框失去焦点或用户按下回车时触发，与原生 input 元素一致。如果需要实时响应用户的输入，可以使用 `input` 事件
- Autocomplete
  - 移除 `custom-item` 属性。现在通过 `scoped slot` 自定义输入建议列表项的内容
  - 移除 `props` 属性，现在使用 `value-key` 属性指定输入建议对象中用于显示的键名
- Steps
  - 移除 `center` 属性
  - 现在步骤条将默认充满父容器
- DatePicker
  - `change` 事件参数现在为组件的绑定值，格式由 `value-format` 控制
- Table
  - 移除通过 `inline-template` 自定义列模板的功能
  - `sort-method` 现在和 `Array.sort` 保持一致的逻辑，要求返回一个数字
  - 将 `append` slot 移至 `tbody` 元素以外，以保证其只被渲染一次

- `expand` 事件更名为 `expand-change`，以保证 API 的命名一致性
- `row-class-name` 和 `row-style` 的函数参数改为对象，以保证 API 的一致性

\ 在网站上动态渲染任意 HTML 是非常危险的，因为容易导致 XSS 攻击。因此请在 `dangerouslyUseHTMLString` 打开的情况下，确保 `message` 的内容是可信的，永远不要将用户提交的内容赋值给 `message` 属性。*