

# Autocomplete 自动补全组件

自动补全是一个普通文本输入框，它通过一组建议的选项来帮助用户输入。

该组件常用于以下两个场景中的单行文本框赋值：

1. 文本框必须取值于一组预设好的值，例如：一个地址项必须包含一个有效的地址：[组合框](#)。
2. 文本框也可以是任何值，但最好能够为用户提供可能的选项，譬如搜索框可以提供近似的或者曾搜索过的选项以节省用户时间：[灵活的单文本框](#)。

此组件旨在改进 “react-select” 和 “downshift” 这两个包。

```
{{"component": "modules/components/ComponentLinkHeader.js"}}
```

## Combo box 组合框

必须取值于一个预设的可选数据源。

```
{{"demo": "ComboBox.js"}}
```

### 选项结构

默认情况下，该组件接受和以下结构相同的选项：

```
interface AutocompleteOption {  
  label: string;  
}  
// 或者  
type AutocompleteOption = string;
```

例如：

```
const options = [  
  { label: 'The Godfather', id: 1 },  
  { label: 'Pulp Fiction', id: 2 },  
];  
// or  
const options = ['The Godfather', 'Pulp Fiction'];
```

然而，你也可以通过提供 `getOptionLabel` 属性来使用不同的结构。

### 练习

下面的每个示例都是自动完成组件的一个功能点的演示。

```
{{"demo": "Playground.js"}}
```

### 选择一个国家

从 248 个国家中选择一个。

```
{{"demo": "CountrySelect.js"}}
```

## 可控的状态

此组件有两种可控的状态：

1. “value” 状态 (state) 包含了 `value / onChange` 两种属性的组合。这个状态表示用户选择的值，如当按下 `Enter` 键时。
2. “input value” 状态 (state) 则包含了 `inputValue / onInputChange` 两种属性的组合。这个状态展示了在文本框中显示的值。

⚠️ 以上两种状态互不干涉，它们应该被单独控制着。

```
{{"demo": "ControllableStates.js"}}
```

## 任意输入

当将 `freeSolo` 设置为 `true` 时，用户可以文本框中输入任意值。

## 搜索输入栏

该属性的主要使用方式是创建一个带有搜索建议的 **输入文本框**，例如 Google 搜索 或 react-autowhatever。

```
{{"demo": "FreeSolo.js"}}
```

## 自由创造

如果您打算将此模块用于类似 [组合框](#) 的体验（一个选择控件元素的增强版），我们则建议如下的设置：

- `selectOnFocus` 可以帮助用户清除所选定的值。
- `clearOnBlur` 可以帮助用户输入一个新值。
- `handleHomeEndKeys` 使用 `Home` 和 `End` 键在弹出窗口内移动焦点。
- 最后一个选项，例如 加上 “你的搜索结果” 。

```
{{"demo": "FreeSoloCreateOption.js"}}
```

您也可以在用户想要加入一个新值的时候显示一个对话框。

```
{{"demo": "FreeSoloCreateOptionDialog.js"}}
```

## 分组

你可以使用 `groupBy` 属性对选项进行分组。如果你要这样做，请先确保选项也按照它们分组的相同维度进行排序，否则你将会注意到重复的标题。

```
{{"demo": "Grouped.js"}}
```

## 禁用选项

```
{{"demo": "DisabledOptions.js"}}
```

### useAutocomplete

对于需要高级自定义的场景，可以使用无界面的 `useAutocomplete()` hook。它接受几乎与 `Autocomplete` 组件相同的参数，除了所有与渲染 JSX 有关的参数。`Autocomplete` 组件就是用基于这个 hook 创建的。

```
import { useAutocomplete } from '@mui/base/AutocompleteUnstyled';
```

为了方便使用以及向后兼容，`useAutocomplete` hook 也在 `@mui/material` 中重新导出了。

```
import { createFilterOptions } from '@material-ui/core/Autocomplete';
```

-  [4.5kB 的压缩包](#)。

```
{{"demo": "UseAutocomplete.js", "defaultCodeOpen": false}}
```

## 自定义的 hook

```
{{"demo": "CustomizedHook.js"}}
```

前往 [自定义](#) 章节，来查看如何使用 `自从完成` 组件来代替该 hook。

## 异步请求

该组件支持两种不同的异步用例：

- [打开时加载](#)：它将等待用户与组件进行交互以加载选项。
- [当你键入内容时进行搜索](#)：每一次键入都会提交一个新的请求。

### 打开时加载

只要正在处理网络请求，它就会显示一个进度状态。

```
{{"demo": "Asynchronous.js"}}
```

### 当你键入内容时进行搜索

如果您的逻辑在每次按键时都要获取新的选项，并使用文本框的当前值在服务器上进行过滤，那么您可能需要考虑对请求进行节流处理。


此外，您需要通过 `filterOptions` 属性来禁用内置的 `Autocomplete`（自动完成）组件的筛选功能：

```
<Autocomplete filterOptions={ (x) => x} />
```

## Google Maps Places

一个自定义 UI 来配合 Google 地图位置的自动完成功能。For this demo, we need to load the [Google Maps JavaScript](#) and [Google Places](#) API.

```
{{"demo": "GoogleMaps.js"}}
```

 在你开始使用 Google Maps JavaScript API 之前，你必须注册并且创建一个可支付的账户。

## 多个输入值

在这个演示中，我们需要加载 [Google Maps JavaScript](#) API。

```
{{"demo": "Tags.js"}}
```

## 固定的选项

有时候您需要锁定某个标签，这样他们不会被从界面中移除，那么这时您可以将 `chips` 设置为禁用来达到这个目的。

```
{{"demo": "FixedTags.js"}}
```

## 复选框

```
{{"demo": "CheckboxesTags.js"}}
```

## 限制标签数量

当没有聚焦时，您可以使用 `limitTags` 属性来限制显示选项的数量。

```
{{"demo": "LimitTags.js"}}
```

## 尺寸

想要使用外观看起来比较小的输入框吗？试着使用 `size` 属性吧。

```
{{"demo": "Sizes.js"}}
```

## 个性化

### 自定义输入

使用 `renderInput` 属性，您可以对输入内容进行自定义渲染。此 `render` 属性的第一个参数包含了您想要传递的那些属性。请特别注意 `ref` 和 `inputProps`。

```
{{"demo": "CustomInputAutocomplete.js"}}
```

### GitHub 标签选择器

该演示再现了 GitHub 的标签选择器：

```
{{"demo": "GitHubLabel.js"}}
```

Head to the [Customized hook](#) section for a customization example with the `useAutocomplete` hook instead of the component.

## 高亮显示

在该示例中使用 `useAutocomplete` hook 而不是组件，您也可以查看 [Customized hook](#) 部分来了解自定义示例。

```
{{"demo": "Highlights.js"}}
```

## 自定义筛选

此组件提供了一个 factory 来构建一个筛选的方法，来供给 `filterOptions` 属性使用。您可以使用该方法来更改默认的筛选行为。

```
import matchSorter from 'match-sorter';

const filterOptions = (options, { inputValue }) => matchSorter(options, inputValue);
```

```
<Autocomplete filterOptions={filterOptions} />;
```

```
createFilterOptions(config) => filterOptions
```

### 参数

1. `config` (*object* [optional]):
  - `config.ignoreAccents` (*bool* [optional]): 默认为 `true`。移除字母的变音符号。
  - `config.ignoreCase` (*bool* [optional]): 默认为 `true`。所有字母都小写。
  - `config.limit` (*number* [optional]): 默认值为 `null`。显示限定数量的建议选项。例如，如果 `config.limit` 是 `100`，那么只显示前 `100` 个匹配的选项。如果存在很多选项匹配，并且虚拟化设置还没建立成时，这样的限制是非常有效的。
  - `config.matchFrom` (*'any' | 'start'* [optional]): 默认值为 `'any'`。
  - `config.stringify` (*func* [optional]): 控制如何将一个选项转换成一个字符串，这样，选项就能够和输入文本的片段相匹配。
  - `config.trim` (*bool* [optional]): 默认为 `false`。删除尾随空格。

### 返回结果

`filterOptions`: the returned filter method can be provided directly to the `filterOptions` prop of the `Autocomplete` component, or the parameter of the same name for the hook.

`filterOptions`: 返回的 `filter` (过滤) 方法可以直接提供给带有 `filterOptions` 属性的 `Autocomplete` 组件，或者和 `hooks` 同名的参数。

```
const filterOptions = createFilterOptions({
  matchFrom: 'start',
  stringify: (option) => option.title,
});

<Autocomplete filterOptions={filterOptions} />;
```

```
{{"demo": "Filter.js", "defaultCodeOpen": false}}
```

### 进阶使用

对于更复杂的过滤机制，譬如模糊匹配 (fuzzy matching)，我们推荐您看一下 [match-sorter](#)。就像这样：

```
import { matchSorter } from 'match-sorter';

const filterOptions = (options, { inputValue }) => matchSorter(options, inputValue);

<Autocomplete filterOptions={filterOptions} />;
```

### 虚拟滚动

在 10000 个随机生成的选项中搜索。多亏了 [react-window](#)，这个列表得以实现虚拟滚动。

```
{{"demo": "Virtualize.js"}}
```

## 事件

如果您想要阻止默认的按键行为，您可以将事件的 `defaultMuiPrevented` 属性设置为 `true`：

```
<Autocomplete
  onKeyDown={ (event) => {
    if (event.key === 'Enter') {
      // Prevent's default 'Enter' behavior.
      event.defaultMuiPrevented = true;
      // your handler code
    }
  }}
/>
```

## 设计局限

### autocomplete/autofill

Browsers have heuristics to help the user fill in form inputs. However, this can harm the UX of the component. 然而，这样的功能会削弱用户的组件体验。 However, this can harm the UX of the component.

By default, the component disables the input **autocomplete** feature (remembering what the user has typed for a given field in a previous session) with the `autoComplete="off"` attribute. Google Chrome 浏览器目前不支持此属性设置 ([Issue 587466](#))。Google Chrome 浏览器目前不支持此属性设置 ([Issue 587466](#))。要解决这个问题，可以采用的变通方法是删除 `id`，让组件自行随机生成。

除了记住过去输入的值，浏览器还可能发出 **自动填写 (autofill)** 建议（保存的登录名、地址或支付详情）。若您不需要自动填充，您可以尝试以下方式：

- 给输入框一个不同的名字，这样不会给浏览器泄露任何可以滥用的信息。例如：`id="field1"` 而不是 `id="country"`。若您不填写 `id` 的话，该组件则会使用一个随机的 `id`。
- 设置 `autoComplete="new-password"`（当设置此属性时，有些浏览器会建议输入高复杂度的密码）。

```
<TextField
  {...params}
  inputProps={{
    ...params.inputProps,
    autoComplete: 'new-password',
  }}
/>
```

请阅读 [这篇 MDN 指南](#) 来寻求更多解决方案。

### iOS VoiceOver 辅助功能

iOS Safari 中的 VoiceOver 对 `aria-owns` 属性的支持并不是很到位。你可以用 `disablePortal` 属性来解决这个问题。

### ListboxComponent

若你提供一共自定义的 `ListboxComponent` 属性，请保证需要滚动功能的容器将 `role` 属性设置为 `listbox`。这能保证滚动功能在一些情况下，例如当用键盘切换的时候，仍然能够正常显示。

## 无障碍设计

(WAI-ARIA: <https://www.w3.org/TR/wai-aria-practices/#combobox>)

我们鼓励用户在 `textbox` 中使用标签。组件带入了 WAI-ARIA 授权的一些标准。