Input

Input data using mouse or keyboard.

:::warning Input is a controlled component, it **always shows Vue binding value**.

Under normal circumstances, `input` event should be handled. Its handler should update component's binding value (or use `v-model`). Otherwise, input box's value will not change.

Do not support `v-model` modifiers. :::

Basic usage

:::demo

```
<el-input placeholder="Please input" v-model="input"></el-input>

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

:::

Disabled

:::demo Disable the Input with the `disabled` attribute.

```
<el-input
  placeholder="Please input"
  v-model="input"
  :disabled="true">
</el-input>

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

:::

Clearable

:::demo Make the Input clearable with the `clearable` attribute.

```
<el-input
  placeholder="Please input"
  v-model="input"
  clearable>
</el-input>

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

:::

Password box

:::demo Make a toggleable password Input with the `show-password` attribute.

```
<el-input placeholder="Please input password" v-model="input" show-password></el-
input>

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

:::

Input with icon

Add an icon to indicate input type.

:::demo To add icons in Input, you can simply use `prefix-icon` and `suffix-icon` attributes. Also, the `prefix` and `suffix` named slots works as well.

```
<div class="demo-input-suffix">
  <span class="demo-input-label">Using attributes</span>
  <el-input
    placeholder="Pick a date"
    suffix-icon="el-icon-date"
```

```
    v-model="input1">
  </el-input>
  <el-input
    placeholder="Type something"
    prefix-icon="el-icon-search"
    v-model="input2">
  </el-input>
</div>
<div class="demo-input-suffix">
  <span class="demo-input-label">Using slots</span>
  <el-input
    placeholder="Pick a date"
    v-model="input3">
    <i slot="suffix" class="el-input__icon el-icon-date"></i>
  </el-input>
  <el-input
    placeholder="Type something"
    v-model="input4">
    <i slot="prefix" class="el-input__icon el-icon-search"></i>
  </el-input>
</div>

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

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

:::

Textarea

Resizable for entering multiple lines of text information. Add attribute `type="textarea"` to change `input` into native `textarea`.

:::demo Control the height by setting the `rows` prop.

```
<el-input
  type="textarea"
  :rows="2"
  placeholder="Please input"
  v-model="textarea">
</el-input>

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

:::

Autosize Textarea

Setting the `autosize` prop for a textarea type of Input makes the height to automatically adjust based on the content. An options object can be provided to `autosize` to specify the minimum and maximum number of lines the textarea can automatically adjust.

:::demo

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

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

:::

Mixed input

Prepend or append an element, generally a label or a button.

:::demo Use `slot` to distribute elements that prepend or append to Input.

```
<div>
  <el-input placeholder="Please input" v-model="input1">
    <template slot="prepend">Http://</template>
  </el-input>
</div>
<div style="margin-top: 15px;">
  <el-input placeholder="Please input" v-model="input2">
    <template slot="append">.com</template>
  </el-input>
</div>
<div style="margin-top: 15px;">
  <el-input placeholder="Please input" v-model="input3" class="input-with-select">
    <el-select v-model="select" slot="prepend" placeholder="Select">
      <el-option label="Restaurant" value="1"></el-option>
      <el-option label="Order No." value="2"></el-option>
      <el-option label="Tel" value="3"></el-option>
    </el-select>
    <el-button slot="append" icon="el-icon-search"></el-button>
  </el-input>
</div>

<style>
  .el-select .el-input {
    width: 110px;
  }
  .input-with-select .el-input-group__prepend {
    background-color: #fff;
  }
</style>
<script>
export default {
  data() {
    return {
      input1: '',
      input2: '',
      input3: '',
      select: ''
    }
  }
}
</script>
```

:::

Sizes

:::demo Add `size` attribute to change the size of Input. In addition to the default size, there are three other options: `large`, `small` and `mini`.

```
<div class="demo-input-size">
  <el-input
    placeholder="Please Input"
    v-model="input1">
  </el-input>
  <el-input
    size="medium"
    placeholder="Please Input"
    v-model="input2">
  </el-input>
  <el-input
    size="small"
    placeholder="Please Input"
    v-model="input3">
  </el-input>
  <el-input
    size="mini"
    placeholder="Please Input"
    v-model="input4">
  </el-input>
</div>

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

:::

Autocomplete

You can get some recommended tips based on the current input.

:::demo Autocomplete component provides input suggestions. The `fetch-suggestions` attribute is a method that returns suggested input. In this example, `querySearch(queryString, cb)` returns suggestions to Autocomplete via `cb(data)` when suggestions are ready.

```
<el-row class="demo-autocomplete">
  <el-col :span="12">
    <div class="sub-title">list suggestions when activated</div>
    <el-autocomplete
      class="inline-input"
      v-model="state1"
      :fetch-suggestions="querySearch"
      placeholder="Please Input"
      @select="handleSelect"
    ></el-autocomplete>
  </el-col>
  <el-col :span="12">
    <div class="sub-title">list suggestions on input</div>
    <el-autocomplete
      class="inline-input"
      v-model="state2"
      :fetch-suggestions="querySearch"
      placeholder="Please Input"
      :trigger-on-focus="false"
      @select="handleSelect"
    ></el-autocomplete>
  </el-col>
</el-row>
<script>
  export default {
    data() {
      return {
        links: [],
        state1: '',
        state2: ''
      };
    },
    methods: {
      querySearch(queryString, cb) {
        var links = this.links;
        var results = queryString ? links.filter(this.createFilter(queryString)) :
links;
        // call callback function to return suggestions
        cb(results);
      },
      createFilter(queryString) {
        return (link) => {
          return (link.value.toLowerCase().indexOf(queryString.toLowerCase()) ===
0);
        };
      },
      loadAll() {
        return [
          { "value": "vue", "link": "https://github.com/vuejs/vue" },
          { "value": "element", "link": "https://github.com/ElemeFE/element" },
          { "value": "cooking", "link": "https://github.com/ElemeFE/cooking" },
```

```
          { "value": "mint-ui", "link": "https://github.com/ElemeFE/mint-ui" },
          { "value": "vuex", "link": "https://github.com/vuejs/vuex" },
          { "value": "vue-router", "link": "https://github.com/vuejs/vue-router" },
          { "value": "babel", "link": "https://github.com/babel/babel" }
        ];
      },
      handleSelect(item) {
        console.log(item);
      }
    },
    mounted() {
      this.links = this.loadAll();
    }
  }
</script>
```

:::

Custom template

Customize how suggestions are displayed.

:::demo Use `scoped slot` to customize suggestion items. In the scope, you can access the suggestion object via the `item` key.

```
<el-autocomplete
  popper-class="my-autocomplete"
  v-model="state"
  :fetch-suggestions="querySearch"
  placeholder="Please input"
  @select="handleSelect">
  <i
    class="el-icon-edit el-input__icon"
    slot="suffix"
    @click="handleIconClick">
  </i>
  <template slot-scope="{ item }">
    <div class="value">{{ item.value }}</div>
    <span class="link">{{ item.link }}</span>
  </template>
</el-autocomplete>

<style>
  .my-autocomplete {
    li {
      line-height: normal;
      padding: 7px;

      .value {
        text-overflow: ellipsis;
        overflow: hidden;
      }
```

```
      .link {
        font-size: 12px;
        color: #b4b4b4;
      }
    }
  }
</style>

<script>
  export default {
    data() {
      return {
        links: [],
        state: ''
      };
    },
    methods: {
      querySearch(queryString, cb) {
        var links = this.links;
        var results = queryString ? links.filter(this.createFilter(queryString)) :
links;
        // call callback function to return suggestion objects
        cb(results);
      },
      createFilter(queryString) {
        return (link) => {
          return (link.value.toLowerCase().indexOf(queryString.toLowerCase()) ===
0);
        };
      },
      loadAll() {
        return [
          { "value": "vue", "link": "https://github.com/vuejs/vue" },
          { "value": "element", "link": "https://github.com/ElemeFE/element" },
          { "value": "cooking", "link": "https://github.com/ElemeFE/cooking" },
          { "value": "mint-ui", "link": "https://github.com/ElemeFE/mint-ui" },
          { "value": "vuex", "link": "https://github.com/vuejs/vuex" },
          { "value": "vue-router", "link": "https://github.com/vuejs/vue-router" },
          { "value": "babel", "link": "https://github.com/babel/babel" }
         ];
      },
      handleSelect(item) {
        console.log(item);
      },
      handleIconClick(ev) {
        console.log(ev);
      }
    },
    mounted() {
      this.links = this.loadAll();
    }
```

```
  }
</script>
```

:::

Remote search

Search data from server-side.

:::demo

```
<el-autocomplete
  v-model="state"
  :fetch-suggestions="querySearchAsync"
  placeholder="Please input"
  @select="handleSelect"
></el-autocomplete>
<script>
  export default {
    data() {
      return {
        links: [],
        state: '',
        timeout:  null
      };
    },
    methods: {
      loadAll() {
        return [
          { "value": "vue", "link": "https://github.com/vuejs/vue" },
          { "value": "element", "link": "https://github.com/ElemeFE/element" },
          { "value": "cooking", "link": "https://github.com/ElemeFE/cooking" },
          { "value": "mint-ui", "link": "https://github.com/ElemeFE/mint-ui" },
          { "value": "vuex", "link": "https://github.com/vuejs/vuex" },
          { "value": "vue-router", "link": "https://github.com/vuejs/vue-router" },
          { "value": "babel", "link": "https://github.com/babel/babel" }
         ];
      },
      querySearchAsync(queryString, cb) {
        var links = this.links;
        var results = queryString ? links.filter(this.createFilter(queryString)) :
links;

        clearTimeout(this.timeout);
        this.timeout = setTimeout(() => {
          cb(results);
        }, 3000 * Math.random());
      },
      createFilter(queryString) {
        return (link) => {
          return (link.value.toLowerCase().indexOf(queryString.toLowerCase()) ===
0);
```

```
        };
      },
      handleSelect(item) {
        console.log(item);
      }
    },
    mounted() {
      this.links = this.loadAll();
    }
  };
</script>
```

:::

Limit length

:::demo `maxlength` and `minlength` are attributes of native input, they declare a limit on the number of characters a user can input. The "number of characters" is measured using JavaScript string length.Setting the `maxlength` prop for a text or textarea type of Input can limit the length of input value, allows you to show word count by setting `show-word-limit` to `true` at the same time.

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

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

:::

Input Attributes

Attribute	Description	Type	Accepted Values	Default
type	type of input	string	text, textarea and other native input types	text
value / v-model	binding value	string / number	—	—
maxlength	same as `maxlength` in native input	number	—	—
minlength	same as `minlength` in native input	number	—	—
show-word-limit	whether show word count， only works when `type` is 'text' or 'textarea'	boolean	—	false
placeholder	placeholder of Input	string	—	—
clearable	whether to show clear button	boolean	—	false
show-password	whether to show toggleable password input	boolean	—	false
disabled	whether Input is disabled	boolean	—	false
size	size of Input, works when `type` is not 'textarea'	string	medium / small / mini	—
prefix-icon	prefix icon class	string	—	—
suffix-icon	suffix icon class	string	—	—
rows	number of rows of textarea, only works when `type` is 'textarea'	number	—	2
autosize	whether textarea has an adaptive height, only works when `type` is 'textarea'. Can accept an object, e.g. { minRows: 2, maxRows: 6 }	boolean / object	—	false
autocomplete	same as `autocomplete` in native input	string	on/off	off
auto-complete	@DEPRECATED in next major version	string	on/off	off
name	same as `name` in native input	string	—	—
readonly	same as `readonly` in native input	boolean	—	false
max	same as `max` in native input	—	—	—
min	same as `min` in native input	—	—	—
step	same as `step` in native input	—	—	—

resize	control the resizability	string	none, both, horizontal, vertical	—
autofocus	same as `autofocus` in native input	boolean	—	false
form	same as `form` in native input	string	—	—
label	label text	string	—	—
tabindex	input tabindex	string	-	-
validate-event	whether to trigger form validation	boolean	-	true

Input slots

Name	Description
prefix	content as Input prefix, only works when `type` is 'text'
suffix	content as Input suffix, only works when `type` is 'text'
prepend	content to prepend before Input, only works when `type` is 'text'
append	content to append after Input, only works when `type` is 'text'

Input Events

Event Name	Description	Parameters
blur	triggers when Input blurs	(event: Event)
focus	triggers when Input focuses	(event: Event)
change	triggers only when the input box loses focus or the user presses Enter	(value: string | number)
input	triggers when the Input value change	(value: string | number)
clear	triggers when the Input is cleared by clicking the clear button	—

Input Methods

Method	Description	Parameters
focus	focus the input element	—
blur	blur the input element	—
select	select the text in input element	—

Autocomplete Attributes

Attribute	Description	Type	Options	Default
placeholder	the placeholder of Autocomplete	string	—	—
clearable	whether to show clear button	boolean	—	false
disabled	whether Autocomplete is disabled	boolean	—	false
value-key	key name of the input suggestion object for display	string	—	value
icon	icon name	string	—	—
value	binding value	string	—	—
debounce	debounce delay when typing, in milliseconds	number	—	300
placement	placement of the popup menu	string	top / top-start / top-end / bottom / bottom-start / bottom-end	bottom-start
fetch-suggestions	a method to fetch input suggestions. When suggestions are ready, invoke `callback(data:[])` to return them to Autocomplete	Function(queryString, callback)	—	—
popper-class	custom class name for autocomplete's dropdown	string	—	—
trigger-on-focus	whether show suggestions when input focus	boolean	—	true
name	same as `name` in native input	string	—	—
select-when-unmatched	whether to emit a `select` event on enter when there is no autocomplete match	boolean	—	false
label	label text	string	—	—
prefix-icon	prefix icon class	string	—	—
suffix-icon	suffix icon class	string	—	—
hide-loading	whether to hide the loading icon in remote search	boolean	—	false
popper-append-to-body	whether to append the dropdown to body. If the positioning of the dropdown is wrong, you can try to set this prop to false	boolean	-	true

highlight-first-item	whether to highlight first item in remote search suggestions by default	boolean	—	false

Autocomplete Slots

Name	Description
prefix	content as Input prefix
suffix	content as Input suffix
prepend	content to prepend before Input
append	content to append after Input

Autocomplete Scoped Slot

Name	Description
—	Custom content for input suggestions. The scope parameter is { item }

Autocomplete Events

Event Name	Description	Parameters
select	triggers when a suggestion is clicked	suggestion being clicked
change	triggers when the icon inside Input value change	(value: string | number)

Autocomplete Methods

Method	Description	Parameters
focus	focus the input element	—