MessageBox

A set of modal boxes simulating system message box, mainly for alerting information, confirm operations and prompting messages. :::tip By design MessageBox provides simulations of system's `alert`, `confirm` and `prompt`, so it's content should be simple. For more complicated contents, please use Dialog. :::

Alert

Alert interrupts user operation until the user confirms.

:::demo Open an alert by calling the `$alert` method. It simulates the system's `alert`, and cannot be closed by pressing ESC or clicking outside the box. In this example, two parameters `message` and `title` are received. It is worth mentioning that when the box is closed, it returns a `Promise` object for further processing. If you are not sure if your target browsers support `Promise`, you should import a third party polyfill or use callbacks instead like this example.

```
<template>
  <el-button type="text" @click="open">Click to open the Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$alert('This is a message', 'Title', {
          confirmButtonText: 'OK',
          callback: action => {
            this.$message({
              type: 'info',
              message: `action: ${ action }`
            });
          }
        });
      }
    }
  }
</script>
```

:::

Confirm

Confirm is used to ask users' confirmation.

:::demo Call `$confirm` method to open a confirm, and it simulates the system's `confirm`. We can also highly customize Message Box by passing a third attribute `options` which is a literal object. The attribute `type` indicates the message type, and it's value can be `success`, `error`, `info` and `warning`. Note that the second attribute `title` must be a `string`, and if it is an `object`, it will be handled as the attribute `options`. Here we use `Promise` to handle further processing.

```
<template>
  <el-button type="text" @click="open">Click to open the Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$confirm('This will permanently delete the file. Continue?', 'Warning',
{
          confirmButtonText: 'OK',
          cancelButtonText: 'Cancel',
          type: 'warning'
        }).then(() => {
          this.$message({
            type: 'success',
            message: 'Delete completed'
          });
        }).catch(() => {
          this.$message({
            type: 'info',
            message: 'Delete canceled'
          });
        });
      }
    }
  }
</script>
```

:::

Prompt

Prompt is used when user input is required.

:::demo Call `$prompt` method to open a prompt, and it simulates the system's `prompt`. You can use `inputPattern` parameter to specify your own RegExp pattern. Use `inputValidator` to specify validation method, and it should return `Boolean` or `String`. Returning `false` or `String` means the validation has failed, and the string returned will be used as the `inputErrorMessage`. In addition, you can customize the placeholder of the input box with `inputPlaceholder` parameter.

```
<template>
  <el-button type="text" @click="open">Click to open Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$prompt('Please input your e-mail', 'Tip', {
          confirmButtonText: 'OK',
```

```
          cancelButtonText: 'Cancel',
          inputPattern: /[\w!#$%&'*+/=?^_`{|}~-]+(?:\.[\w!#$%&'*+/=?^_`{|}~-]+)*@(?:[\w](?:[\w-]*[\w])?\.)+[\w](?:[\w-]*[\w])?/,
          inputErrorMessage: 'Invalid Email'
        }).then(({ value }) => {
          this.$message({
            type: 'success',
            message: 'Your email is:' + value
          });
        }).catch(() => {
          this.$message({
            type: 'info',
            message: 'Input canceled'
          });
        });
      }
    }
  }
</script>
```

:::

Customization

Can be customized to show various content.

:::demo The three methods mentioned above are repackagings of the `$msgbox` method. This example calls `$msgbox` method directly using the `showCancelButton` attribute, which is used to indicate if a cancel button is displayed. Besides we can use `cancelButtonClass` to add a custom style and `cancelButtonText` to customize the button text (the confirm button also has these fields, and a complete list of fields can be found at the end of this documentation). This example also uses the `beforeClose` attribute. It is a method and will be triggered when the MessageBox instance will be closed, and its execution will stop the instance from closing. It has three parameters: `action`, `instance` and `done`. Using it enables you to manipulate the instance before it closes, e.g. activating `loading` for confirm button; you can invoke the `done` method to close the MessageBox instance (if `done` is not called inside `beforeClose`, the instance will not be closed).

```
<template>
  <el-button type="text" @click="open">Click to open Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        const h = this.$createElement;
        this.$msgbox({
          title: 'Message',
          message: h('p', null, [
            h('span', null, 'Message can be '),
            h('i', { style: 'color: teal' }, 'VNode')
          ]),
```

```
          showCancelButton: true,
          confirmButtonText: 'OK',
          cancelButtonText: 'Cancel',
          beforeClose: (action, instance, done) => {
            if (action === 'confirm') {
              instance.confirmButtonLoading = true;
              instance.confirmButtonText = 'Loading...';
              setTimeout(() => {
                done();
                setTimeout(() => {
                  instance.confirmButtonLoading = false;
                }, 300);
              }, 3000);
            } else {
              done();
            }
          }
        }).then(action => {
          this.$message({
            type: 'info',
            message: 'action: ' + action
          });
        });
      },
    }
  }
</script>
```

:::

:::tip The content of MessageBox can be `VNode`, allowing us to pass custom components. When opening the MessageBox, Vue compares new `VNode` with old `VNode`, then figures out how to efficiently update the UI, so the components may not be completely re-rendered (#8931). In this case, you can add a unique key to `VNode` each time MessageBox opens: example. :::

Use HTML String

`message` supports HTML string.

:::demo Set `dangerouslyUseHTMLString` to true and `message` will be treated as an HTML string.

```
<template>
  <el-button type="text" @click="open">Click to open Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$alert('<strong>This is <i>HTML</i> string</strong>', 'HTML String', {
          dangerouslyUseHTMLString: true
        });
```

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

:::

:::warning Although `message` property supports HTML strings, dynamically rendering arbitrary HTML on your website can be very dangerous because it can easily lead to XSS attacks. So when `dangerouslyUseHTMLString` is on, please make sure the content of `message` is trusted, and **never** assign `message` to user-provided content.
:::

Distinguishing cancel and close

In some cases, clicking the cancel button and close button may have different meanings.

:::demo By default, the parameters of Promise's reject callback and `callback` are 'cancel' when the user cancels (clicking the cancel button) and closes (clicking the close button or mask layer, pressing the ESC key) the MessageBox. If `distinguishCancelAndClose` is set to true, the parameters of the above two operations are 'cancel' and 'close' respectively.

```
<template>
  <el-button type="text" @click="open">Click to open Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$confirm('You have unsaved changes, save and proceed?', 'Confirm', {
          distinguishCancelAndClose: true,
          confirmButtonText: 'Save',
          cancelButtonText: 'Discard Changes'
        })
          .then(() => {
            this.$message({
              type: 'info',
              message: 'Changes saved. Proceeding to a new route.'
            });
          })
          .catch(action => {
            this.$message({
              type: 'info',
              message: action === 'cancel'
                ? 'Changes discarded. Proceeding to a new route.'
                : 'Stay in the current route'
            })
          });
      }
    }
  }
</script>
```

:::

Centered content

Content of MessageBox can be centered.

:::demo Setting `center` to `true` will center the content

```
<template>
  <el-button type="text" @click="open">Click to open Message Box</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$confirm('This will permanently delete the file. Continue?', 'Warning',
{
          confirmButtonText: 'OK',
          cancelButtonText: 'Cancel',
          type: 'warning',
          center: true
        }).then(() => {
          this.$message({
            type: 'success',
            message: 'Delete completed'
          });
        }).catch(() => {
          this.$message({
            type: 'info',
            message: 'Delete canceled'
          });
        });
      }
    }
  }
</script>
```

:::

Global method

If Element is fully imported, it will add the following global methods for Vue.prototype: `$msgbox`, `$alert`, `$confirm` and `$prompt`. So in a Vue instance you can call `MessageBox` like what we did in this page. The parameters are:

- `$msgbox(options)`
- `$alert(message, title, options)` or `$alert(message, options)`
- `$confirm(message, title, options)` or `$confirm(message, options)`
- `$prompt(message, title, options)` or `$prompt(message, options)`

Local import

If you prefer importing `MessageBox` on demand:

```
import { MessageBox } from 'element-ui';
```

The corresponding methods are: `MessageBox`, `MessageBox.alert`, `MessageBox.confirm` and `MessageBox.prompt`. The parameters are the same as above.

Options

Attribute	Description	Type	Accepted Values	Default
title	title of the MessageBox	string	—	—
message	content of the MessageBox	string	—	—
dangerouslyUseHTMLString	whether `message` is treated as HTML string	boolean	—	false
type	message type, used for icon display	string	success / info / warning / error	—
iconClass	custom icon's class, overrides `type`	string	—	—
customClass	custom class name for MessageBox	string	—	—
callback	MessageBox closing callback if you don't prefer Promise	function(action), where action can be 'confirm', 'cancel' or 'close', and `instance` is the MessageBox instance. You can access to that instance's attributes and methods	—	—
showClose	whether to show close icon of MessageBox	boolean	—	true
beforeClose	callback before MessageBox closes, and it will prevent MessageBox from closing	function(action, instance, done), where `action` can be 'confirm', 'cancel' or 'close'; `instance` is the MessageBox instance, and	—	—

		you can access to that instance's attributes and methods; `done` is for closing the instance		
distinguishCancelAndClose	whether to distinguish canceling and closing the MessageBox	boolean	—	false
lockScroll	whether to lock body scroll when MessageBox prompts	boolean	—	true
showCancelButton	whether to show a cancel button	boolean	—	false (true when called with confirm and prompt)
showConfirmButton	whether to show a confirm button	boolean	—	true
cancelButtonText	text content of cancel button	string	—	Cancel
confirmButtonText	text content of confirm button	string	—	OK
cancelButtonClass	custom class name of cancel button	string	—	—
confirmButtonClass	custom class name of confirm button	string	—	—
closeOnClickModal	whether MessageBox can be closed by clicking the mask	boolean	—	true (false when called with alert)
closeOnPressEscape	whether MessageBox can be closed by pressing the ESC	boolean	—	true (false when called with alert)
closeOnHashChange	whether to close	boolean	—	true

	MessageBox when hash changes			
showInput	whether to show an input	boolean	—	false (true when called with prompt)
inputPlaceholder	placeholder of input	string	—	—
inputType	type of input	string	—	text
inputValue	initial value of input	string	—	—
inputPattern	regexp for the input	regexp	—	—
inputValidator	validation function for the input. Should returns a boolean or string. If a string is returned, it will be assigned to inputErrorMessage	function	—	—
inputErrorMessage	error message when validation fails	string	—	Illegal input
center	whether to align the content in center	boolean	—	false
roundButton	whether to use round button	boolean	—	false