MessageBox

Un conjunto de cajas modales simulando un sistema de message box, principalmente para alertar información, confirmar operaciones y mostrar mensajes de aviso. :::tip Por diseño los message box nos proveen de simulaciones de sistemas como los componentes `alert`, `confirm` y `prompt`, entonces su contenido debería ser simple. para contenido mas complejo, por favor utilice el componente Dialog. :::

Alert

Alert interrumpe las operaciones realizadas hasta que el usuario confirme la alerta.

:::demo Desplegar una alerta utilizando el método `$alert`. Simula el sistema `alert`, y no puede ser cerrado al presionar la tecla ESC o al dar clic fuera de la caja. En este ejemplo, dos parámetros son recibidos `message` y `title`. Vale la pena mencionar que cuando la caja es cerrada, regresa un objeto `Promise` para su procesamiento posteriormente. Si no estas seguro si el navegador soporta `Promise`, deberías importar una librería de terceros de polyfill o utilizar callbacks.

```
<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 es utilizado para preguntar al usuario y recibir una confirmación.

:::demo Llamando al método `$confirm` para abrir el componente confirm, y simula el sistema `confirm`. También podemos personalizar a gran medida el componente Message Box al mandar un tercer atributo llamado `options` que es literalmente un objeto. El atributo `type` indica el tipo de mensaje, y su valor puede ser `success`, `error`, `info` y `warning`. Se debe tener en cuenta que el segundo atributo `title` debe ser de tipo `string`, y si es de tipo `object`, sera manejado como el atributo `options`. Aquí utilizamos `Promise` para manejar posteriormente el proceso.

```
<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 es utilizado cuando se requiere entrada de información del usuario.

:::demo Llamando al método `$prompt` desplegamos el componente prompt, y simula el sistema `prompt` .Puedes utilizar el parámetro `inputPattern` para especificar tu propio patrón RegExp. Utiliza el parámetro `inputValidator` para especificar el método de validación, y debería regresar un valor de tipo `Boolean` o `String` . Al regresar `false` o `String` significa que la validación a fallado, y la cadena regresada se usara como `inputErrorMessage` . Ademas, puedes personalizar el atributo placeholder del input box con el parámetro `inputPlaceholder` .

```
<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>
```

:::

Personalización

Puede ser personalizado para mostrar diversos contenidos.

:::demo Los tres métodos mencionados anteriormente son un rempaquetado del método `$msgbox`. En este ejemplo se realiza una llamada al método `$msgbox` directamente utilizando el atributo `showCancelButton`, el cual es utilizado para indicar si el botón cancelar es mostrado en pantalla. Además podemos utilizar el atributo `cancelButtonClass` para agregar un estilo personalizado y el atributo `cancelButtonText` para personalizar el texto del botón (el botón de confirmación también cuenta con estos campos, y podrá encontrar una lista completa de estos atributos al final de esta documentación). Este ejemplo también utiliza el atributo `beforeClose`. Es un método que es disparado cuando una instancia del componente MessageBox es cerrada, y su ejecución detendrá el cierre de la instancia. Tiene tres parámetros: `action`, `instance` y `done`. Al utilizarla te permite manipular la instancia antes de que sea cerrada, e.g. activando `loading` para el botón de confirmación; puede invocar el método `done` para cerrar la instancia del componente MessageBox (si el método `done` no es llamado dentro del atributo `beforeClose`, la instancia no podrá cerrarse).

```
<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

El contenido de MessageBox puede ser `VNode`, permitiéndonos pasar componentes personalizados. Al abrir el MessageBox, Vue compara el nuevo `VNode` con el viejo `VNode`, y luego averigua cómo actualizar eficientemente la interfaz de usuario, de modo que es posible que los componentes no se vuelvan a procesar completamente (#8931). En este caso, se puede añadir una clave única a `VNode` cada vez que se abre MessageBox: ejemplo.

:::

Utiliza cadenas HTML

`message` soporta cadenas HTML.

:::demo Establezca el valor de `dangerouslyUseHTMLString` a true y `message` sera tratado como una cadena HTML.

```
<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 Aunque la propiedad `message` soporta cadenas HTML, realizar arbitrariamente render dinámico de HTML en nuestro sitio web puede ser muy peligroso ya que puede conducir fácilmente a XSS attacks. Entonces cuando `dangerouslyUseHTMLString` esta activada, asegúrese que el contenido de `message` sea de confianza, y **nunca** asignar `message` a contenido generado por el usuario. :::

Distinguir entre cancelar y cerrar

En algunos casos, hacer clic en el botón Cancelar y en el botón Cerrar puede tener diferentes significados.

:::demo Por defecto, los parámetros de `Promise's reject callback` y `callback` son `cancel` cuando el usuario cancela (haciendo clic en el botón de cancelación) y cierra (haciendo clic en el botón de cerrar o en la capa de máscara, pulsando la tecla ESC) el MessageBox. Si `distinguishCancelAndClose` está ajustado a `true`, los parámetros de las dos operaciones anteriores son `cancel` y `close` respectivamente.

```
<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>
```

:::

Contenido centrado

El contenido del componente MessageBox puede ser centrado.

:::demo Establecer `center` a `true` centrara el contenido

```
<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>
```

:::

Métodos Globales

Si Element fue importado completamente, agregara los siguientes métodos globales para Vue.prototype: `$msgbox`, `$alert`, `$confirm` y `$prompt`. Así que en una instancia de Vue puedes llamar el método `MessageBox` como lo que hicimos en esta pagina. Los parámetros son:

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

Importación local

Si prefieres importar `MessageBox` cuando lo necesites (on demand):

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

Los métodos correspondientes: `MessageBox`, `MessageBox.alert`, `MessageBox.confirm` y `MessageBox.prompt`. Los parámetros son los mismos que los anteriores.

Opciones

Atributo	Descripción	Tipo	Valores Permitidos	Por defecto
title	titulo del componente MessageBox	string	—	—
message	contenido del componente MessageBox	string	—	—
dangerouslyUseHTMLString	utilizado para que `message` sea tratado como una cadena HTML	boolean	—	false
type	tipo de mensaje , utilizado para mostrar el icono	string	success / info / warning / error	—
iconClass	clase personalizada para el icono, sobrescribe `type`	string	—	—
customClass	nombre de la clase personalizada para el componente MessageBox	string	—	—
callback	MessageBox callback al cerrar si	function(action), donde la accion puede ser 'confirm', 'cancel' o	—	—

	no desea utilizar Promise	'close', e `instance` es la instancia del componente MessageBox. Puedes acceder a los metodos y atributos de esa instancia		
beforeClose	callback llamado antes de cerrar el componente MessageBox, y previene que el componente MessageBox se cierre	function(action, instance, done), donde `action` pueden ser 'confirm', 'cancel' o 'close'; `instance` es la instancia del componente MessageBox, Puedes acceder a los metodos y atributos de esa instancia; `done` es para cerrar la instancia	—	—
distinguishCancelAndClose	si se debe distinguir entre cancelar y cerrar	boolean	—	false
lockScroll	utilizado para bloquear el desplazamiento del contenido del MessageBox prompts	boolean	—	true
showCancelButton	utilizado para mostrar un botón cancelar	boolean	—	false (true cuando es llamado con confirm y prompt)
showConfirmButton	utilizado para mostrar un botón confirmar	boolean	—	true
cancelButtonText	contenido de texto del botón cancelar	string	—	Cancel
confirmButtonText	contenido de texto del botón confirmar	string	—	OK
cancelButtonClass	nombre de la clase personalizada del	string	—	—

	botón cancelar			
confirmButtonClass	nombre de la clase personalizada del botón confirmar	string	—	—
closeOnClickModal	utilizado para que que el componente MessageBox pueda ser cerrado al dar clic en la mascara	boolean	—	true (false cuando es llamado con alert)
closeOnPressEscape	utilizado para que que el componente MessageBox pueda ser cerrado al presionar la tecla ESC	boolean	—	true (false cuando es llamado con alert)
closeOnHashChange	utilizado para cerra el componente MessageBox cuando hash cambie	boolean	—	true
showInput	utilizado para mostrar el componente input	boolean	—	false (true cuando es llamado con prompt)
inputPlaceholder	placeholder para el componente input	string	—	—
inputType	tipo del componente input	string	—	text
inputValue	valor inicial del componente input	string	—	—
inputPattern	regexp del componente input	regexp	—	—
inputValidator	función de validación del componente input. Debe regresar un valor de tipo boolean o string. Si regresa un valor tipo	function	—	—

	string, sera asignado a inputErrorMessage			
inputErrorMessage	mensaje de error cuando la validación falla	string	—	Illegal input
center	utilizado para alinear el contenido al centro	boolean	—	false
roundButton	utilizado para redondear el botón	boolean	—	false