

MessageBox

Un ensemble de fenêtres modales pour afficher des messages système, tels que des alertes, des demandes de confirmation ou des informations importantes.

:::tip MessageBox est avant tout conçue pour émuler des `alert`, `confirm` ou `prompt`, son contenu devrait donc être simple. Pour afficher un contenu plus riche, utilisez plutôt Dialog. :::

Alert

Alert interrompt l'action de l'utilisateur jusqu'à ce qu'il confirme.

:::demo Ouvrez un alert en appelant la méthode `$alert`. Elle simule une `alert` système et ne peut être fermée en pressant ESC ou en cliquant hors de la boîte. Dans cet exemple, deux paramètres `message` et `title` sont reçus. Notez que lorsque la boîte est fermée, elle retourne un objet `Promise`. Si vous n'êtes pas certains que tous vos navigateurs cibles supportent `Promise`, utilisez un polyfill ou utilisez des callbacks comme dans l'exemple qui suit.

```
<template>
  <el-button type="text" @click="open">Cliquez pour ouvrir la MessageBox</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$alert('Ceci est un message', 'Titre', {
          confirmButtonText: 'OK',
          callback: action => {
            this.$message({
              type: 'info',
              message: `action: ${action}`
            });
          }
        });
      }
    }
  }
</script>
```

...

Confirm

Confirm est utilisé pour demander une confirmation à l'utilisateur.

:::demo Appelez la méthode `$confirm` pour ouvrir une confirm, qui simule le `confirm` système. Vous pouvez aussi personnaliser MessageBox en passant un objet à l'attribut `options`. L'attribut `type` indique le type de message (voir en bas de la page pour la liste des valeurs possibles). Notez que l'attribut `title` doit être de type `string`. S'il se trouve être un `object`, il sera considéré comme étant l'attribut `options`. Cet exemple utilise une `Promise` pour gérer la suite du processus.

```

<template>
  <el-button type="text" @click="open">Cliquez pour ouvrir la MessageBox</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$confirm('Ceci effacera le fichier. Continuer?', 'Warning', {
          confirmButtonText: 'OK',
          cancelButtonText: 'Annuler',
          type: 'warning'
        }).then(() => {
          this.$message({
            type: 'success',
            message: 'Fichier supprimé'
          });
        }).catch(() => {
          this.$message({
            type: 'info',
            message: 'Suppression annulée'
          });
        });
      }
    }
  }
</script>

```

...

Prompt

Prompt est utilisé lorsqu'un utilisateur.

:::demo Appelez la méthode `$prompt` pour ouvrir un prompt simulant le `prompt` système. Vous pouvez utiliser le paramètre `inputPattern` pour spécifier un pattern à l'aide d'une RegExp. Utilisez `inputValidator` pour indiquer la méthode de validation, qui devra retourner un `Boolean` ou un `String`. Retourner `false` ou un `String` veut dire que la validation a échoué, et la string ainsi fournie sera le `inputErrorMessage`. De plus, vous pouvez personnaliser le placeholder du champs avec le paramètre `inputPlaceholder`.

```

<template>
  <el-button type="text" @click="open">Cliquez pour ouvrir la MessageBox</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$prompt('Entrez votre e-mail', 'Astuce', {
          confirmButtonText: 'OK',
          cancelButtonText: 'Annuler',

```

```

        inputPattern: /[\\w!#$%&'*/+=?^_`{|}~-]+(?:\\. [\\w!#$%&'*/+=?^_`{|}~-]+)*@(?:[\\w](?:[\\w-]*[\\w])?\\.)+[\\w](?:[\\w-]*[\\w])?/,
        inputErrorMessage: 'E-mail invalide'
    }).then(({ value }) => {
        this.$message({
            type: 'success',
            message: 'Votre e-mail est: ' + value
        });
    }).catch(() => {
        this.$message({
            type: 'info',
            message: 'Annulé'
        });
    });
}
}
</script>

```

...

Personnalisation

Il est possible d'afficher du contenu un peu plus varié et personnalisé.

Les trois précédentes méthodes sont des repackagings de la méthode `$msgbox`. Cet exemple appelle directement `$msgbox` en utilisant l'attribut `showCancelButton` pour indiquer si un bouton annuler doit être affiché. De plus, vous pouvez utiliser `cancelButtonClass` pour ajouter du style et `cancelButtonText` pour personnaliser le bouton texte (voir la liste complète en fin de page). La méthode `beforeClose` est appelée quand la `MessageBox` va être fermée et prévient sa fermeture. Elle prend trois paramètres: `action`, `instance` et `done`. Elle vous permet ainsi d'effectuer des actions avant la fermeture, e.g. activer `loading` pour le bouton de confirmation. Appelez `done` pour fermer la `MessageBox`, sinon l'instance ne sera jamais fermée.

```

<template>
  <el-button type="text" @click="open">Cliquez pour ouvrir la MessageBox</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        const h = this.$createElement;
        this.$msgbox({
          title: 'Message',
          message: h('p', null, [
            h('span', null, 'Le message peut être '),
            h('i', { style: 'color: teal' }, 'VNode')
          ]),
          showCancelButton: true,
          confirmButtonText: 'OK',
          cancelButtonText: 'Annuler',

```

```

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

```

...

:::tip Le contenu de MessageBox peut être `VNode` , Vous permettant de passer des composants personnalisés.

Lorsque vous ouvrez MessageBox, Vue compare le nouveau `VNode` avec l'ancien `VNode` , puis détermine comment rafraîchir efficacement l'UI, le composant peut donc ne pas être totalement re-rendu ([#8931](#)). Dans ce cas, Vous pouvez ajouter une clé unique à `VNode` à chaque fois que MessageBox s'ouvre: [exemple](#). ...

Utiliser du HTML

`message` supporte le HTML.

:::demo Mettez `dangerouslyUseHTMLString` à `true` et `message` sera traité comme du HTML.

```

<template>
  <el-button type="text" @click="open">Cliquez pour ouvrir la MessageBox</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$alert('<strong>Ceci est du <i>HTML</i></strong>', 'HTML', {
          dangerouslyUseHTMLString: true
        });
      }
    }
  }
}

```

```
}  
</script>
```

...

:::warning Bien que la propriété `message` supporte le HTML, générer du contenu HTML dynamiquement peut être très dangereux, car cela permet des [attaques XSS](#). Donc lorsque `dangerouslyUseHTMLString` est présent, soyez certain de sécuriser le contenu de `message`, et n'assignez **jamais** à `message` du contenu fournit par l'utilisateur.

...

Distinguer 'annuler' de 'fermer'

Dans certains cas, les boutons fermer et annuler peuvent avoir des sens différents.

:::demo Par défaut, le paramètre de la `callback` et de la promesse rejetée est 'cancel' lorsque l'utilisateur annule (clique sur le bouton annuler) ou lorsqu'il ferme la fenêtre (clique sur le bouton fermer, hors de la fenêtre, ou presse ESC). Si `distinguishCancelAndClose` est à `true`, le dernier cas précédent renvoie 'close' au lieu de 'cancel' afin de pouvoir gérer les deux actions différemment.

```
<template>  
  <el-button type="text" @click="open">Cliquez pour ouvrir la MessageBox</el-button>  
</template>  
  
<script>  
  export default {  
    methods: {  
      open() {  
        this.$confirm('Vous avez du travail non enregistré, enregistrer et  
quitter?', 'Confirm', {  
          distinguishCancelAndClose: true,  
          confirmButtonText: 'Enregistrer',  
          cancelButtonText: 'Ne pas enregistrer'  
        })  
        .then(() => {  
          this.$message({  
            type: 'info',  
            message: 'Enregistré. Passage a une nouvelle route.'  
          });  
        })  
        .catch(action => {  
          this.$message({  
            type: 'info',  
            message: action === 'cancel'  
              ? 'Changements annulés. Passage sur une nouvelle route.'  
              : 'Reste sur la même route'  
          });  
        })  
      };  
    }  
  }  
</script>
```

...

Contenu centré

le contenu de MessageBox peut être centré.

:::demo Mettre `center` à `true` centrera le contenu

```
<template>
  <el-button type="text" @click="open">Cliquez pour ouvrir la MessageBox</el-button>
</template>

<script>
  export default {
    methods: {
      open() {
        this.$confirm('Ceci effacera le fichier, continuer?' , 'Warning', {
          confirmButtonText: 'OK',
          cancelButtonText: 'Annuler',
          type: 'warning',
          center: true
        }).then(() => {
          this.$message({
            type: 'success',
            message: 'Fichier supprimé'
          });
        }).catch(() => {
          this.$message({
            type: 'info',
            message: 'Annulé'
          });
        });
      }
    }
  }
</script>
```

...

Méthode globale

Si Element est importé entièrement, il ajoutera les méthodes suivantes à Vue.prototype: `$msgbox` , `$alert` , `$confirm` et `$prompt` . Dans ce cas vous pouvez appeler `MessageBox` comme nous l'avons fait dans cette page. Les paramètres sont:

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

Import à la demande

Si vous préférez importer `MessageBox` à la demande:

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

Les méthodes correspondantes sont: `MessageBox`, `MessageBox.alert`, `MessageBox.confirm` et `MessageBox.prompt`. Les paramètres sont les mêmes que précédemment.

Options

Attribut	Description	Type	Valeurs acceptées	Défaut
title	Titre de la MessageBox.	string	—	—
message	Contenu de la MessageBox.	string	—	—
dangerouslyUseHTMLString	Si <code>message</code> doit être traité comme du HTML.	boolean	—	false
type	Type du message, utilisé pour le choix d'icône.	string	success / info / warning / error	—
iconClass	Classe d'icône personnalisée, écrase <code>type</code> .	string	—	—
customClass	Nom de classe pour MessageBox.	string	—	—
callback	La callback de fermeture de MessageBox si vous n'utilisez pas les promesses.	function(action, instance), ou <code>action</code> peut être 'confirm', 'cancel' ou 'close', et <code>instance</code> est l'instance MessageBox.	—	—
showClose	Si l'icône de fermeture doit être affichée.	boolean	—	true
beforeClose	La callback de pré-fermeture qui empêchera MessageBox de se fermer.	function(action, instance, done), ou <code>action</code> peut-être 'confirm', 'cancel' ou 'close'; <code>instance</code> est l'instance de MessageBox; <code>done</code> est la méthode pour fermer l'instance.	—	—

distinguishCancelAndClose	S'il doit y avoir une différence entre l'annulation et la fermeture de la MessageBox.	boolean	—	false
lockScroll	Si le défilement de la page doit être bloqué lorsque la MessageBox est active.	boolean	—	true
showCancelButton	Si le bouton annuler doit être affiché.	boolean	—	false (true dans le cas de confirm ou prompt).
showConfirmButton	Si le bouton confirmer doit être affiché.	boolean	—	true
cancelButtonText	Le texte du bouton annuler.	string	—	Cancel
confirmButtonText	Le texte du bouton confirmer.	string	—	OK
cancelButtonClass	Classe du bouton annuler.	string	—	—
confirmButtonClass	Classe du bouton confirmer.	string	—	—
closeOnClickModal	Si MessageBox peut être fermée en cliquant en dehors.	boolean	—	true (false dans le cas de alert).
closeOnPressEscape	Si MessageBox peut être fermée en pressant ESC.	boolean	—	true (false dans le cas de alert)
closeOnHashChange	Si MessageBox doit être fermée quand le hash change.	boolean	—	true
showInput	Si un champs d'input doit être affiché.	boolean	—	false (true dans le

				cas de prompt).
inputPlaceholder	Placeholder du champs d'input.	string	—	—
inputType	Type du champs d'input.	string	—	text
inputValue	Valeur initiale du champs d'input.	string	—	—
inputPattern	RegExp du champs d'input.	regexp	—	—
inputValidator	Fonction de validation du champs d'input. Doit retourner un boolean ou un string. Si c'est un string, il sera assigné à inputErrorMessage.	function	—	—
inputErrorMessage	Message d'erreur lorsque la validation échoue.	string	—	Illegal input
center	Si le contenu doit être centré.	boolean	—	false
roundButton	Si le bouton doit être rond.	boolean	—	false