Tree

Affiche un ensemble de données possédant plusieurs niveaux de hiérarchie ou d'imbrication.

Usage

Voici la structure basique.

:::demo

```
<el-tree :data="data" :props="defaultProps" @node-click="handleNodeClick"></el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          label: 'Niveau un 1',
          children: [{
            label: 'Niveau deux 1-1',
            children: [{
              label: 'Niveau trois 1-1-1'
            }]
          }]
        }, {
          label: 'Niveau un 2',
          children: [{
            label: 'Niveau deux 2-1',
            children: [{
              label: 'Niveau trois 2-1-1'
            }]
          }, {
            label: 'Niveau deux 2-2',
            children: [{
              label: 'Niveau trois 2-2-1'
            }]
          }]
        }, {
          label: 'Niveau un 3',
          children: [{
            label: 'Niveau deux 3-1',
            children: [{
              label: 'Niveau trois 3-1-1'
            }]
          }, {
            label: 'Niveau deux 3-2',
            children: [{
              label: 'Niveau trois 3-2-1'
            }]
          }]
        }],
        defaultProps: {
```

```
          children: 'children',
          label: 'label'
        }
      };
    },
    methods: {
      handleNodeClick(data) {
        console.log(data);
      }
    }
  };
</script>
```

:::

Sélection

Vous pouvez activer la sélection des noeuds.

:::demo cet exemple montre également comment charger des données de manière asynchrone.

```
<el-tree
  :props="props"
  :load="loadNode"
  lazy
  show-checkbox
  @check-change="handleCheckChange">
</el-tree>

<script>
  export default {
    data() {
      return {
        props: {
          label: 'name',
          children: 'zones'
        },
        count: 1
      };
    },
    methods: {
      handleCheckChange(data, checked, indeterminate) {
        console.log(data, checked, indeterminate);
      },
      handleNodeClick(data) {
        console.log(data);
      },
      loadNode(node, resolve) {
        if (node.level === 0) {
          return resolve([{ name: 'Root1' }, { name: 'Root2' }]);
        }
        if (node.level > 3) return resolve([]);
```

```
        var hasChild;
        if (node.data.name === 'region1') {
          hasChild = true;
        } else if (node.data.name === 'region2') {
          hasChild = false;
        } else {
          hasChild = Math.random() > 0.5;
        }

        setTimeout(() => {
          var data;
          if (hasChild) {
            data = [{
              name: 'zone' + this.count++
            }, {
              name: 'zone' + this.count++
            }];
          } else {
            data = [];
          }

          resolve(data);
        }, 500);
      }
    }
  };
</script>
```

:::

Noeud-feuille personnalisés en mode lazy

:::demo Les données d'un noeud ne sont pas accessibles tant que la noeud n'est pas cliqué, l'arbre ne peut donc pas prédire si un noeud sera une feuille. C'est pourquoi un bouton de menu est ajouté à chaque noeud, et si c'est une feuille il disparaîtra après le clic. Vous pouvez également dire par avance à l'arbre si un noeud est une feuille, pour éviter l'apparition du bouton de menu.

```
<el-tree
  :props="props"
  :load="loadNode"
  lazy
  show-checkbox>
</el-tree>

<script>
  export default {
    data() {
      return {
        props: {
          label: 'name',
          children: 'zones',
```

```
          isLeaf: 'leaf'
        },
      };
    },
    methods: {
      loadNode(node, resolve) {
        if (node.level === 0) {
          return resolve([{ name: 'region' }]);
        }
        if (node.level > 1) return resolve([]);

        setTimeout(() => {
          const data = [{
            name: 'leaf',
            leaf: true
          }, {
            name: 'zone'
          }];

          resolve(data);
        }, 500);
      }
    }
  };
</script>
```

:::

Checkbox désactivées

Les checkbox des noeuds peuvent être désactivées individuellement.

:::demo Dans cet exemple, la propriété `disabled` est ajoutée à `defaultProps`, et certains noeuds ont `disabled:true`. Les checkbox correspondantes sont donc désactivées.

```
<el-tree
  :data="data"
  :props="defaultProps"
  show-checkbox
  @check-change="handleCheckChange">
</el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          id: 1,
          label: 'Niveau un 1',
          children: [{
            id: 3,
            label: 'Niveau deux 2-1',
```

```
            children: [{
              id: 4,
              label: 'Niveau trois 3-1-1'
            }, {
              id: 5,
              label: 'Niveau trois 3-1-2',
              disabled: true
            }]
          }, {
            id: 2,
            label: 'Niveau deux 2-2',
            disabled: true,
            children: [{
              id: 6,
              label: 'Niveau trois 3-2-1'
            }, {
              id: 7,
              label: 'Niveau trois 3-2-2',
              disabled: true
            }]
          }]
        }],
        defaultProps: {
            children: 'children',
            label: 'label',
            disabled: 'disabled',
        },
      };
    }
  };
</script>
```

:::

Ouverture et sélection par défaut

Certains noeuds peuvent être ouverts et/ou sélectionnés par défaut.

:::demo Utilisez `default-expanded-keys` et `default-checked-keys` pour réglez respectivement les noeuds ouverts et les noeuds sélectionnés par défaut. Notez que `node-key` est requis dans ce cas. Sa valeurs est le nom d'une clé dans l'objets data, et sa valeur devrait être unique dans tout l'arbre.

```
<el-tree
  :data="data"
  show-checkbox
  node-key="id"
  :default-expanded-keys="[2, 3]"
  :default-checked-keys="[5]"
  :props="defaultProps">
</el-tree>

<script>
```

```
  export default {
    data() {
      return {
        data: [{
          id: 1,
          label: 'Niveau un 1',
          children: [{
            id: 4,
            label: 'Niveau deux 1-1',
            children: [{
              id: 9,
              label: 'Niveau trois 1-1-1'
            }, {
              id: 10,
              label: 'Niveau trois 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: 'Niveau un 2',
          children: [{
            id: 5,
            label: 'Niveau deux 2-1'
          }, {
            id: 6,
            label: 'Niveau deux 2-2'
          }]
        }, {
          id: 3,
          label: 'Niveau un 3',
          children: [{
            id: 7,
            label: 'Niveau deux 3-1'
          }, {
            id: 8,
            label: 'Niveau deux 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

Sélectionner des noeuds

:::demo Cet exemple montre comment récupérer et sélectionner des noeuds. Vous pouvez utiliser deux approches: les noeuds ou les clés. Dans le cas des clés, `node-key` est requis.

```
<el-tree
  :data="data"
  show-checkbox
  default-expand-all
  node-key="id"
  ref="tree"
  highlight-current
  :props="defaultProps">
</el-tree>

<div class="buttons">
  <el-button @click="getCheckedNodes">Récupération par noeud</el-button>
  <el-button @click="getCheckedKeys">Récupération par clé</el-button>
  <el-button @click="setCheckedNodes">Sélection par noeud</el-button>
  <el-button @click="setCheckedKeys">Sélection par clé</el-button>
  <el-button @click="resetChecked">Reset</el-button>
</div>

<script>
  export default {
    methods: {
      getCheckedNodes() {
        console.log(this.$refs.tree.getCheckedNodes());
      },
      getCheckedKeys() {
        console.log(this.$refs.tree.getCheckedKeys());
      },
      setCheckedNodes() {
        this.$refs.tree.setCheckedNodes([{
          id: 5,
          label: 'Niveau deux 2-1'
        }, {
          id: 9,
          label: 'Niveau trois 1-1-1'
        }]);
      },
      setCheckedKeys() {
        this.$refs.tree.setCheckedKeys([3]);
      },
      resetChecked() {
        this.$refs.tree.setCheckedKeys([]);
      }
    },

    data() {
      return {
        data: [{
          id: 1,
```

```
          label: 'Niveau un 1',
          children: [{
            id: 4,
            label: 'Niveau deux 1-1',
            children: [{
              id: 9,
              label: 'Niveau trois 1-1-1'
            }, {
              id: 10,
              label: 'Niveau trois 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: 'Niveau un 2',
          children: [{
            id: 5,
            label: 'Niveau deux 2-1'
          }, {
            id: 6,
            label: 'Niveau deux 2-2'
          }]
        }, {
          id: 3,
          label: 'Niveau un 3',
          children: [{
            id: 7,
            label: 'Niveau deux 3-1'
          }, {
            id: 8,
            label: 'Niveau deux 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

Contenu personnalisé

Le contenu des noeuds peut être personnalisé, afin de pouvoir ajouter des icônes ou des boutons par exemple.

:::demo Il existe deux méthodes de personnalisation: `render-content` et les slots avec portée. Utilisez `render-content` pour assigner une fonction de rendu qui va générer le contenu des noeuds. Voire la documentation de Vue pour plus d'informations. Si vous préférez les slots, vous aurez accès à `node` et `data` dans le scope, corerspondant à l'objet TreeNode et aux données du noeud courant. Notez que la démo de `render-content` ne

marche pas dans jsfiddle car il ne supporte pas JSX. Dans un vrai projet, `render-content` marchera si les dépendances sont satisfaites.

```
<div class="custom-tree-container">
  <div class="block">
    <p>Avec render-content</p>
    <el-tree
      :data="data"
      show-checkbox
      node-key="id"
      default-expand-all
      :expand-on-click-node="false"
      :render-content="renderContent">
    </el-tree>
  </div>
  <div class="block">
    <p>Avec un slot</p>
    <el-tree
      :data="data"
      show-checkbox
      node-key="id"
      default-expand-all
      :expand-on-click-node="false">
      <span class="custom-tree-node" slot-scope="{ node, data }">
        <span>{{ node.label }}</span>
        <span>
          <el-button
            type="text"
            size="mini"
            @click="() => append(data)">
            Ajouter
          </el-button>
          <el-button
            type="text"
            size="mini"
            @click="() => remove(node, data)">
            Supprimer
          </el-button>
        </span>
      </span>
    </el-tree>
  </div>
</div>

<script>
  let id = 1000;

  export default {
    data() {
      const data = [{
        id: 1,
```

```
      label: 'Niveau un 1',
      children: [{
        id: 4,
        label: 'Niveau deux 1-1',
        children: [{
          id: 9,
          label: 'Niveau trois 1-1-1'
        }, {
          id: 10,
          label: 'Niveau trois 1-1-2'
        }]
      }]
    }, {
      id: 2,
      label: 'Niveau un 2',
      children: [{
        id: 5,
        label: 'Niveau deux 2-1'
      }, {
        id: 6,
        label: 'Niveau deux 2-2'
      }]
    }, {
      id: 3,
      label: 'Niveau un 3',
      children: [{
        id: 7,
        label: 'Niveau deux 3-1'
      }, {
        id: 8,
        label: 'Niveau deux 3-2'
      }]
    }];
    return {
      data: JSON.parse(JSON.stringify(data)),
      data: JSON.parse(JSON.stringify(data))
    }
  },

  methods: {
    append(data) {
      const newChild = { id: id++, label: 'testtest', children: [] };
      if (!data.children) {
        this.$set(data, 'children', []);
      }
      data.children.push(newChild);
    },

    remove(node, data) {
      const parent = node.parent;
      const children = parent.data.children || parent.data;
      const index = children.findIndex(d => d.id === data.id);
```

```
        children.splice(index, 1);
      },

      renderContent(h, { node, data, store }) {
        return (
          <span class="custom-tree-node">
            <span>{node.label}</span>
            <span>
              <el-button size="mini" type="text" on-click={ () => this.append(data)
}>Ajouter</el-button>
              <el-button size="mini" type="text" on-click={ () => this.remove(node,
data) }>Supprimer</el-button>
            </span>
          </span>);
      }
    }
  };
</script>

<style>
  .custom-tree-node {
    flex: 1;
    display: flex;
    align-items: center;
    justify-content: space-between;
    font-size: 14px;
    padding-right: 8px;
  }
</style>
```

:::

Filtrage

Les noeuds peuvent être filtrés par mot-clé.

:::demo Utilisez la méthode `filter` de l'instance de Tree pour pouvoir filtrer les noeuds, son paramètre étant le mot-clé. Notez que pour que cela fonctionne, `filter-node-method` est requis, sa valeur étant la méthode de filtrage.

```
<el-input
  placeholder="Filter keyword"
  v-model="filterText">
</el-input>

<el-tree
  class="filter-tree"
  :data="data"
  :props="defaultProps"
  default-expand-all
  :filter-node-method="filterNode"
  ref="tree">
```

```
</el-tree>

<script>
  export default {
    watch: {
      filterText(val) {
        this.$refs.tree.filter(val);
      }
    },

    methods: {
      filterNode(value, data) {
        if (!value) return true;
        return data.label.indexOf(value) !== -1;
      }
    },

    data() {
      return {
        filterText: '',
        data: [{
          id: 1,
          label: 'Niveau un 1',
          children: [{
            id: 4,
            label: 'Niveau deux 1-1',
            children: [{
              id: 9,
              label: 'Niveau trois 1-1-1'
            }, {
              id: 10,
              label: 'Niveau trois 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: 'Niveau un 2',
          children: [{
            id: 5,
            label: 'Niveau deux 2-1'
          }, {
            id: 6,
            label: 'Niveau deux 2-2'
          }]
        }, {
          id: 3,
          label: 'Niveau un 3',
          children: [{
            id: 7,
            label: 'Niveau deux 3-1'
          }, {
            id: 8,
```

```
            label: 'Niveau deux 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

Accordéon

Vous pouvez utiliser un mode accordéon afin que seul un noeud par niveau soit ouvert.

:::demo

```
<el-tree
  :data="data"
  :props="defaultProps"
  accordion
  @node-click="handleNodeClick">
</el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          label: 'Niveau un 1',
          children: [{
            label: 'Niveau deux 1-1',
            children: [{
              label: 'Niveau trois 1-1-1'
            }]
          }]
        }, {
          label: 'Niveau un 2',
          children: [{
            label: 'Niveau deux 2-1',
            children: [{
              label: 'Niveau trois 2-1-1'
            }]
          }, {
            label: 'Niveau deux 2-2',
            children: [{
              label: 'Niveau trois 2-2-1'
            }]
          }]
```

```
        }, {
          label: 'Niveau un 3',
          children: [{
            label: 'Niveau deux 3-1',
            children: [{
              label: 'Niveau trois 3-1-1'
            }]
          }, {
            label: 'Niveau deux 3-2',
            children: [{
              label: 'Niveau trois 3-2-1'
            }]
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    },
    methods: {
      handleNodeClick(data) {
        console.log(data);
      }
    }
  };
</script>
```

:::

Noeuds déplaçables

Vous pouvez déplacer les noeuds par drag'n drop en ajoutant l'attribut `draggable`.

:::demo

```
<el-tree
  :data="data"
  node-key="id"
  default-expand-all
  @node-drag-start="handleDragStart"
  @node-drag-enter="handleDragEnter"
  @node-drag-leave="handleDragLeave"
  @node-drag-over="handleDragOver"
  @node-drag-end="handleDragEnd"
  @node-drop="handleDrop"
  draggable
  :allow-drop="allowDrop"
  :allow-drag="allowDrag">
</el-tree>

<script>
```

```
export default {
  data() {
    return {
      data: [{
        label: 'Niveau un 1',
        children: [{
          label: 'Niveau deux 1-1',
          children: [{
            label: 'Niveau trois 1-1-1'
          }]
        }]
      }, {
        label: 'Niveau un 2',
        children: [{
          label: 'Niveau deux 2-1',
          children: [{
            label: 'Niveau trois 2-1-1'
          }]
        }, {
          label: 'Niveau deux 2-2',
          children: [{
            label: 'Niveau trois 2-2-1'
          }]
        }]
      }, {
        label: 'Niveau un 3',
        children: [{
          label: 'Niveau deux 3-1',
          children: [{
            label: 'Niveau trois 3-1-1'
          }]
        }, {
          label: 'Niveau deux 3-2',
          children: [{
            label: 'Niveau trois 3-2-1'
          }]
        }]
      }],
      defaultProps: {
        children: 'children',
        label: 'label'
      }
    };
  },
  methods: {
    handleDragStart(node, ev) {
      console.log('drag start', node);
    },
    handleDragEnter(draggingNode, dropNode, ev) {
      console.log('tree drag enter: ', dropNode.label);
    },
    handleDragLeave(draggingNode, dropNode, ev) {
```

```
        console.log('tree drag leave: ', dropNode.label);
      },
      handleDragOver(draggingNode, dropNode, ev) {
        console.log('tree drag over: ', dropNode.label);
      },
      handleDragEnd(draggingNode, dropNode, dropType, ev) {
        console.log('tree drag end: ', dropNode && dropNode.label, dropType);
      },
      handleDrop(draggingNode, dropNode, dropType, ev) {
        console.log('tree drop: ', dropNode.label, dropType);
      },
      allowDrop(draggingNode, dropNode, type) {
        if (dropNode.data.label === 'Niveau deux 3-1') {
          return type !== 'inner';
        } else {
          return true;
        }
      },
      allowDrag(draggingNode) {
        return draggingNode.data.label.indexOf('Niveau trois 3-1-1') === -1;
      }
    }
  };
</script>
```

:::

Attributs

Attribut	Description	Type	Valeurs acceptées	Défaut
data	Données de l'arbre	array	—	—
empty-text	Texte à afficher quand il n'y a pas de données.	string	—	—
node-key	Identifiant unique pour chaque noeud, doit être unique dans tout l'arbre.	string	—	—
props	Options de configuration, voir table suivante.	object	—	—
render-after-expand	Si les noeuds enfants doivent être générés seulement après la première ouverture du parent.	boolean	—	true
load	Méthode pour charger les noeuds enfants, uniquement en mode `lazy`.	function(node, resolve)	—	—
render-content	Fonction de rendu pour les noeuds.	Function(h, { node, data, store }	—	—
highlight-	Si le noeud courant est mis en valeur.	boolean	—	false

current				
default-expand-all	Si tous les noeuds sont ouverts par défaut.	boolean	—	false
expand-on-click-node	Si l'ouverture se fait aussi en cliquant sur le noeud. Si `false`, l'ouverture ne se fera qu'en cliquant sur l'icône.	boolean	—	true
check-on-click-node	Si la sélection se fait aussi en cliquant sur le noeud. Si `false`, la sélection ne se fera qu'en cliquant sur la checkbox.	boolean	—	false
auto-expand-parent	Si un noeud parent est automatiquement ouvert quand un noeud enfant s'ouvre.	boolean	—	true
default-expanded-keys	Tableau des clés des noeuds initialement ouverts.	array	—	—
show-checkbox	Si un noeud est sélectionnable.	boolean	—	false
check-strictly	Si la sélection d'un noeud affecte celle de son parent quand `show-checkbox` est `true`.	boolean	—	false
default-checked-keys	Tableau des clés des noeuds initialement sélectionnés.	array	—	—
current-node-key	Clé du noeud initialement sélectionné.	string, number	—	—
filter-node-method	Fonction exécutée sur chaque noeud pour le filtrage. Si elle retourne `false`, les noeuds seront cachés.	Function(value, data, node)	—	—
accordion	Si les noeuds fonctionnent en mode accordéon.	boolean	—	false
indent	Indentation horizontale des noeuds en px.	number	—	16
icon-class	Icône pour chaque noeud.	string	-	-
lazy	Si les noeuds sont chargés en mode lazy, utilisé avec l'attribut `load`.	boolean	—	false
draggable	Si les noeuds sont déplaçables par drag'n drop.	boolean	—	false
allow-drag	Fonction exécutée avant le déplacement d'un noeud. Si `false` est retourné, le noeud ne sera pas déplaçable.	Function(node)	—	—

allow-drop	Fonction exécutée avant le placement d'un noeud. Si `false` est retourné, le noeud ne pourra être placé sur la zone en question. `type` a trois valeurs possibles: 'prev' (insertion avant le noeud cible), 'inner' (insertion dans le noeud cible) and 'next' (insertion après le noeud cible).	Function(draggingNode, dropNode, type)	—	—

props

Attribut	Description	Type	Valeurs acceptées	Défaut
label	Détermine quelle clé de l'objet noeud représente le label.	string, function(data, node)	—	—
children	Détermine quelle clé de l'objet noeud représente les noeuds enfants.	string	—	—
disabled	Détermine quelle clé de l'objet noeud représente la désactivation du noeud.	boolean, function(data, node)	—	—
isLeaf	Détermine si le noeud est une feuille, ne marche qu'avec le mode lazy loading.	boolean, function(data, node)	—	—

Méthodes

`Tree` possède les méthodes suivantes, qui retourne la sélection de noeuds actuelle.

Méthode	Description	Paramètres
filter	Filtre les noeuds.	Accepte un paramètre qui sera le premier paramètre de `filter-node-method`.
updateKeyChildren	Ajoute de nouvelles données au noeud, ne marche que lorsque `node-key` est assigné.	(key, data) Accepte deux paramètres: 1. clé du noeud 2. nouvelles données.
getCheckedNodes	Si le noeud peut-être sélectionné (`show-checkbox` est `true`), il retourne un tableau des noeuds sélectionnés.	(leafOnly, includeHalfChecked) Accepte deux booléen: 1. Défaut à `false`. Si `true`, retourne seulement un tableau des sous-noeuds sélectionnés. 2. Défaut à `false`. Si `true`, retourne la moitié des noeuds sélectionnés.
setCheckedNodes	Détermine quels noeuds sont sélectionnés, ne marche que si `node_key` est assigné.	Un tableau de noeuds qui doivent être sélectionnés.

getCheckedKeys	Si le noeud peut être sélectionné (`show-checkbox` est `true`), retourne un tableau de clés des noeuds sélectionnés.	(leafOnly) Booléen, défaut à `false`. Si à `true`, Il retourne seulement un tableau des sous-noeuds sélectionnés.
setCheckedKeys	Détermine quels noeuds sont sélectionnés, ne marche que si `node_key` est assigné.	(keys, leafOnly) Accepte deux paramètres: 1. un tableau de clés de noeuds à sélectionner 2. un booléen avec pour défaut `false`. Si à `true`, Il retourne seulement un tableau des sous-noeuds sélectionnés.
setChecked	Détermine si un noeud est sélectionnable, ne marche que si `node_key` est assigné.	(key/data, checked, deep) Accepte trois paramètres: 1. La clé ou les données du noeud 2. un booléen indiquant si sélectionné. 3. un booléen indiquant si profond.
getHalfCheckedNodes	Si le noeud peut être sélectionné (`show-checkbox` est `true`), retourne la moitié des noeuds sélectionnés.	-
getHalfCheckedKeys	Si le noeud peut être sélectionné (`show-checkbox` est `true`), retourne les clés de la moitié des noeuds sélectionnés.	-
getCurrentKey	retourne la clé du noeud actuellement en valeur (`null` si aucun noeud n'est en valeur).	—
getCurrentNode	retourne les données du noeud actuellement en valeur (`null` si aucun noeud n'est en valeur).	—
setCurrentKey	Met un noeud en valeur par sa clé, ne marche que si `node_key` est assigné.	(key) la clé du noeud. Si `null`, annule la sélection actuelle.
setCurrentNode	Met un noeud en valeur, ne marche que si `node_key` est assigné.	(node) le noeud.
getNode	Retourne le noeud grâce à sa clé ou ses données.	(data) la clé ou les données du noeud.
remove	Supprime un noeud, ne marche que si node-key est assigné.	(data) le noeud ou ses données à supprimer.

append	Ajoute un noeud à un autre noeud.	(data, parentNode) 1. les données du noeud à ajouter 2. les données du parent, clé ou données.
insertBefore	Insère un noeud avant un autre noeud.	(data, refNode) 1. Les données du noeud à insérer 2. Clé ou noeud de référence.
insertAfter	Insère un noeud après un autre noeud.	(data, refNode) 1. Les données du noeud à insérer 2. Clé ou noeud de référence.

Évènements

Nom	Description	Paramètres
node-click	Se déclenche quand est cliqué.	Le noeud cliqué, la propriété `node` de TreeNode, TreeNode lui-même.
node-contextmenu	Se déclenche quand un noeud reçoit un clic droit.	L'évènement, le noeud cliqué, la propriété `node` de TreeNode, TreeNode lui-même.
check-change	Se déclenche quand quand la sélection d'un noeud change.	Le noeud modifié, si le noeud est sélectionné, si des enfants sont sélectionnés.
check	Se déclenche après un clic sur le checkbox.	Le noeud modifié, l'objet statut de l'arbre avec quatre propriétés: checkedNodes, checkedKeys, halfCheckedNodes, halfCheckedKeys.
current-change	Se déclenche quand le noeud courant changes.	Le noeud courant, la propriété `node` de TreeNode
node-expand	Se déclenche quand le noeud courant s'ouvre.	Le noeud ouvert, la propriété `node` de TreeNode, TreeNode lui-même.
node-collapse	Se déclenche quand le noeud courant se ferme.	Le noeud fermé, la propriété `node` de TreeNode, TreeNode lui-même.
node-drag-start	Se déclenche quand le déplacement commence.	Le noeud déplacé, l'évènement.
node-drag-enter	Se déclenche quand le noeud déplacé entre dans un autre noeud.	Le noeud déplacé, l'autre noeud, l'évènement.
node-drag-leave	Se déclenche quand le noeud déplacé quitte un autre noeud.	Le noeud déplacé, l'autre noeud, l'évènement.
node-drag-over	Se déclenche quand le noeud passe au-dessus d'un autre noeud (comme l'évènement mouseover).	Le noeud déplacé, l'autre noeud, l'évènement.
node-drag-end	Se déclenche quand le déplacement se termine.	Le noeud déplacé, le noeud d'arrivée (peut-être `undefined`), le type de placement (before / after / inner), l'évènement.
node-drop	Se déclenche après que le noeud déplacé soit placé.	Le noeud déplacé, le noeud d'arrivée, le type de placement (before / after / inner), l'évènement.

Slot avec portée

Nom	Description
—	Le contenu personnalisé des noeuds. les paramètres sont { node, data }.