Tree

Muestra un conjunto de datos jerárquicos.

Uso básico

Estructura básica de árbol.

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

<script>
  export default {
    data() {
      return {
        data: [{
          label: 'Level one 1',
          children: [{
            label: 'Level two 1-1',
            children: [{
              label: 'Level three 1-1-1'
            }]
          }]
        }, {
          label: 'Level one 2',
          children: [{
            label: 'Level two 2-1',
            children: [{
              label: 'Level three 2-1-1'
            }]
          }, {
            label: 'Level two 2-2',
            children: [{
              label: 'Level three 2-2-1'
            }]
          }]
        }, {
          label: 'Level one 3',
          children: [{
            label: 'Level two 3-1',
            children: [{
              label: 'Level three 3-1-1'
            }]
          }, {
            label: 'Level two 3-2',
            children: [{
              label: 'Level three 3-2-1'
            }]
```

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

Seleccionable

Usado para la selección de nodos.

:::demo Este ejemplo también muestra como cargar los datos de forma asíncrona.

```
<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>
```

:::

Nodos hoja en modo perezoso (lazy load)

:::demo Los datos de un nodo no son cargados hasta que no es pinchado, así que el árbol no puede predecir si es una hoja. Por eso a cada nodo se le añade el botón de desplegar, y si el nodo es una hoja el botón desaparecerá al pinchar en él. También puede decirle al árbol que el nodo es una hoja de antemano, y así evita que muestre el botón de desplegar.

```
<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 desactivados

El checkbox de un nodo se puede poner como desactivado.

:::demo En el ejemplo, la propiedad `disabled` se declara en `defaultProps`, y algunos nodos se ponen como `disabled:true`. Los checkboxes correspondientes son desactivados y no se pueden pinchar.

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

<script>
  export default {
    data() {
      return {
        data: [{
          id: 1,
          label: 'Level one 1',
          children: [{
            id: 3,
            label: 'Level two 2-1',
            children: [{
              id: 4,
              label: 'Level three 3-1-1'
            }, {
              id: 5,
              label: 'Level three 3-1-2',
              disabled: true
            }]
          }, {
            id: 2,
            label: 'Level two 2-2',
            disabled: true,
            children: [{
              id: 6,
              label: 'Level three 3-2-1'
            }, {
              id: 7,
              label: 'Level three 3-2-2',
              disabled: true
            }]
          }]
        }],
        defaultProps: {
            children: 'children',
            label: 'label',
            disabled: 'disabled',
        },
      };
    }
```

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

:::

Desplegado o seleccionado por defecto

Los nodos pueden estar desplegados o seleccionados por defecto.

:::demo Utilice `default-expanded-keys` y `default-checked-keys` para establecer los nodos desplegados y seleccionados respectivamente. Tenga en cuenta que para que funcione es necesario que tengan `node-key`. Su valor es el nombre de una clave en el objeto data, y el valor de la clave debe ser único en todo el árbol.

```
<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: 'Level one 1',
          children: [{
            id: 4,
            label: 'Level two 1-1',
            children: [{
              id: 9,
              label: 'Level three 1-1-1'
            }, {
              id: 10,
              label: 'Level three 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: 'Level one 2',
          children: [{
            id: 5,
            label: 'Level two 2-1'
```

```
          }, {
            id: 6,
            label: 'Level two 2-2'
          }]
        }, {
          id: 3,
          label: 'Level one 3',
          children: [{
            id: 7,
            label: 'Level two 3-1'
          }, {
            id: 8,
            label: 'Level two 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

Seleccionando nodos

:::demo Este ejemplo muestra como establecer y leer nodos seleccionados. Esto se puede hacer por nodos o por claves. Si lo hace por claves el atributo `node-key` es necesario.

```
<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">get by node</el-button>
  <el-button @click="getCheckedKeys">get by key</el-button>
  <el-button @click="setCheckedNodes">set by node</el-button>
  <el-button @click="setCheckedKeys">set by key</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.$refs.tree.setCheckedNodes([{
          id: 5,
          label: 'Level two 2-1'
        }, {
          id: 9,
          label: 'Level three 1-1-1'
        }]);
      },
      setCheckedKeys() {
        this.$refs.tree.setCheckedKeys([3]);
      },
      resetChecked() {
        this.$refs.tree.setCheckedKeys([]);
      }
    },

    data() {
      return {
        data: [{
          id: 1,
          label: 'Level one 1',
          children: [{
            id: 4,
            label: 'Level two 1-1',
            children: [{
              id: 9,
              label: 'Level three 1-1-1'
            }, {
              id: 10,
              label: 'Level three 1-1-2'
            }]
          }]
        }, {
```

```
          id: 2,
          label: 'Level one 2',
          children: [{
            id: 5,
            label: 'Level two 2-1'
          }, {
            id: 6,
            label: 'Level two 2-2'
          }]
        }, {
          id: 3,
          label: 'Level one 3',
          children: [{
            id: 7,
            label: 'Level two 3-1'
          }, {
            id: 8,
            label: 'Level two 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

Contenido personalizado en los nodos

El contenido de los nodos puede ser personalizado, así que puede añadir iconos y botones a su gusto.

:::demo Hay dos maneras de personalizar la plantilla para los nodos de árbol: `render-content` y scoped slot. Utilice `render-content` para asignar una función de renderizado que devuelve el contenido del árbol de nodos. Mire la documentación de node para una introducción detallada a las funciones de renderizado. Si prefiere scoped slot, tendrá acceso a los `nodos` y `datos` en el ámbito de aplicación, representando el objeto TreeNode y los datos del nodo actual respectivamente. Tenga en cuenta que este ejemplo no puede ejecutarse en codepen.io ya que no soporta la sintaxis JSX. En un proyecto real `render-content` funcionará si las dependencias relevantes están configuradas correctamente.

```
<div class="custom-tree-container">
```

```
  <div class="block">
    <p>Using 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>Using scoped 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)">
            Append
          </el-button>
          <el-button
            type="text"
            size="mini"
            @click="() => remove(node, data)">
            Delete
          </el-button>
        </span>
      </span>
    </el-tree>
  </div>
</div>

<script>
  let id = 1000;

  export default {
    data() {
      const data = [{
```

```
    id: 1,
    label: 'Level one 1',
    children: [{
      id: 4,
      label: 'Level two 1-1',
      children: [{
        id: 9,
        label: 'Level three 1-1-1'
      }, {
        id: 10,
        label: 'Level three 1-1-2'
      }]
    }]
  }, {
    id: 2,
    label: 'Level one 2',
    children: [{
      id: 5,
      label: 'Level two 2-1'
    }, {
      id: 6,
      label: 'Level two 2-2'
    }]
  }, {
    id: 3,
    label: 'Level one 3',
    children: [{
      id: 7,
      label: 'Level two 3-1'
    }, {
      id: 8,
      label: 'Level two 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) }>Appenc
              <el-button size="mini" type="text" on-click={ () => this.remove(node, data) }>
            </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>
```

:::

Filtrado de nodos

Los nodos del árbol se pueden filtrar.

:::demo Invoque el método `filter` de la instancia de Tree para filtrar los nodos. Su parámetro es la palabra de filtrado. Tenga en cuenta que para que funcione es necesario `filter-node-method`, y su valor el método de filtrado.

```
<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: 'Level one 1',
          children: [{
            id: 4,
            label: 'Level two 1-1',
            children: [{
              id: 9,
              label: 'Level three 1-1-1'
            }, {
              id: 10,
              label: 'Level three 1-1-2'
            }]
          }]
        }, {
          id: 2,
```

```
          label: 'Level one 2',
          children: [{
            id: 5,
            label: 'Level two 2-1'
          }, {
            id: 6,
            label: 'Level two 2-2'
          }]
        }, {
          id: 3,
          label: 'Level one 3',
          children: [{
            id: 7,
            label: 'Level two 3-1'
          }, {
            id: 8,
            label: 'Level two 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

Acordeón

Solo puede ser expandido un nodo del mismo nivel a la vez.

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

<script>
  export default {
    data() {
      return {
        data: [{
          label: 'Level one 1',
```

```
          children: [{
            label: 'Level two 1-1',
            children: [{
              label: 'Level three 1-1-1'
            }]
          }]
        }, {
          label: 'Level one 2',
          children: [{
            label: 'Level two 2-1',
            children: [{
              label: 'Level three 2-1-1'
            }]
          }, {
            label: 'Level two 2-2',
            children: [{
              label: 'Level three 2-2-1'
            }]
          }]
        }, {
          label: 'Level one 3',
          children: [{
            label: 'Level two 3-1',
            children: [{
              label: 'Level three 3-1-1'
            }]
          }, {
            label: 'Level two 3-2',
            children: [{
              label: 'Level three 3-2-1'
            }]
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    },
    methods: {
      handleNodeClick(data) {
        console.log(data);
      }
    }
  };
</script>
```

Draggable

Puede arrastrar y soltar nodos de Tree añadiendo un atributo `draggable` .

```
<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: 'Level one 1',
          children: [{
            label: 'Level two 1-1',
            children: [{
              label: 'Level three 1-1-1'
            }]
          }]
        }, {
          label: 'Level one 2',
          children: [{
            label: 'Level two 2-1',
            children: [{
              label: 'Level three 2-1-1'
            }]
          }, {
            label: 'Level two 2-2',
            children: [{
              label: 'Level three 2-2-1'
            }]
          }]
        }, {
          label: 'Level one 3',
```

```
        children: [{
          label: 'Level two 3-1',
          children: [{
            label: 'Level three 3-1-1'
          }]
        }, {
          label: 'Level two 3-2',
          children: [{
            label: 'Level three 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 === 'Level two 3-1') {
        return type !== 'inner';
      } else {
        return true;
      }
    },
    allowDrag(draggingNode) {
      return draggingNode.data.label.indexOf('Level three 3-1-1') === -1;
```

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

Atributos

Atributo	Descripción	Tipo	Valores aceptados	Por defecto
data	Datos del árbol	array	—	—
empty-text	Texto a mostrar cuando data es void	string	—	—
node-key	Identificador único en todo el árbol para los nodos	string	—	—
props	Opciones de configuración	object	—	—
render-after-expand	si se mostrarán los nodos hijo sólo después de que se desglose por primera vez un nodo padre	boolean	—	true
load	Método para cargar los datos de subárboles	function(node, resolve)	—	—
render-content	Función de renderizado para los nodos	Function(h, { node, data, store }	—	—
highlight-current	Si el nodo actual está resaltado	boolean	—	false
default-expand-all	Expandir todos los nodos por defecto	boolean	—	false
expand-on-click-node	Si expandir o contraer un nodo al pincharlo, si es false solo se hará al pinchar en la flecha	boolean	—	true
check-on-click-node	si marcar o desmarcar el nodo al hacer clic en el. Si es `false`, el nodo sólo se puede marcar o desmarcar haciendo clic en la casilla de verificación.	boolean	—	false

Atributo	Descripción	Tipo	Valores aceptados	Por defecto
auto-expand-parent	Expandir un nodo padre si el hijo está seleccionado	boolean	—	true
default-expanded-keys	Array de claves de los nodos expandidos inicialmente	array	—	—
show-checkbox	Si un nodo es seleccionable	boolean	—	false
check-strictly	El estado de selección de un nodo no afecta a sus padres o hijos, cuando `show-checkbox` es `true`	boolean	—	false
default-checked-keys	Array con claves de los nodos seleccionados inicialmente	array	—	—
current-node-key	la clave del nodo inicialmente seleccionado	string, number	—	—
filter-node-method	Esta función se ejecutará en cada nodo cuando se use el método filtrar, si devuelve `false` el nodo se oculta	Function(value, data, node)	—	—
accordion	Si solo un nodo de cada nivel puede expandirse a la vez	boolean	—	false
indent	Indentación horizontal de los nodos en niveles adyacentes, en pixeles	number	—	16
icon-class	Icono del nodo del árbol de cliente	string	-	-
lazy	si se trata de un nodo de hoja lazy load, utilizado con el atributo `load`	boolean	—	false
draggable	si se habilita la función de drag & drop en los nodos	boolean	—	false

Atributo	Descripción	Tipo	Valores aceptados	Por defecto
allow-drag	esta función se ejecutará antes de arrastrar un nodo. si devuelve `false`, el nodo no puede ser arrastrado.	Function(nodo)	—	—
allow-drop	esta función se ejecutará al arrastrar y soltar un nodo. si devuelve false, el nodo arrastrando no se puede soltar en el nodo destino. `type` tiene tres valores posibles: 'prev' (insertar el nodo de arrastre antes del nodo de destino), 'inner' (insertar el nodo de arrastre en el nodo de destino) y 'next' (insertar el nodo de arrastre después del nodo de destino)	Function(Nodoquesearrastra, Nododestino, type)		—

props

Atributo	Descripción	Tipo	Valores aceptados	Por defecto
label	Especifica que clave del objecto nodo se utilizará como label	string, function(data, node)	—	—
children	Especifica que objeto del nodo se utiliza como subárbol	string	—	—
isLeaf	Especifica si el nodo es una hoja, sólo funciona cuando lazy load está activado	boolean, function(datos, nodo)	—	—

Métodos

`Tree` tiene los siguientes métodos, que devuelven el array de nodos seleccionados.

Método	Descripción	Parámetros
filter	Filtra los nodos del árbol, los nodos filtrados estarán ocultos	Acepta un parámetro que será usado como primer parámetro para filter-node-method
updateKeyChildren	Asocia un nuevo dato al nodo, solo funciona si `node-key` está asignado	(key, data)Acepta dos parámetros: 1. clave del nodo 2. nuevo dato
getCheckedNodes	si el nodo puede ser seleccionado (`show-checkbox` es `true`), devuelve el array de nodos actualmente seleccionada.	(leafOnly, includeHalfChecked) Acepta dos parámetros de tipo booleano: 1. El valor por defecto es `false`. Si el parámetro es `true`, sólo devuelve el array de subnodos actualmente seleccionado. 2. El valor por defecto es `false`. Si el parámetro es `true`, el valor de retorno contiene nodos halfchecked.
setCheckedNodes	Establece algunos nodos como seleccionados, solo funciona cuando `node-key` está asignado	Un array de nodos a seleccionar
getCheckedKeys	Si los nodos pueden ser seleccionados (`show-checkbox` es `true`), devuelve un array con las claves de los nodos seleccionados	(leafOnly) Acepta un booleano que por defecto es `false`.
setCheckedKeys	Establece algunos nodos como seleccionados, solo si `node-key` está asignado	(keys, leafOnly) Acepta dos parametros: 1. un array de claves 2. un booleano cuyo valor por defecto es `false`. Si el parámetro es `true`, solo devuelve los nodos seleccionados

Método	Descripción	Parámetros
setChecked	Establece si un nodo está seleccionado, solo funciona si `node-key` esta asignado	(key/data, checked, deep) Acepta tres parámetros: 1. la clave o dato del nodo a ser seleccionado 2. un booleano que indica si un nodo el nodo estará seleccionado 3. un booleanoque indica si se hará en profundidad
getHalfCheckedNodes	Si el nodo puede ser seleccionado (`show-checkbox` es `true`), devuelve la mitad del array de nodos actualmente seleccionada.	-
getHalfCheckedKeys	Si el nodo puede ser seleccionado (`show-checkbox` es `true`), devuelve la mitad del array de claves del nodo actualmente seleccionado.	-
getCurrentKey	devuelve la clave del nodo resaltado actualmente (null si no hay ninguno)	—
getCurrentNode	devuelve los datos del nodo de resaltado (nulo si no hay ningún nodo resaltado)	—
setCurrentKey	establece el nodo resaltado por la clave, solo funciona si `node-key` está asignado	(key) la clave del nodo a ser resaltado. Si es `null`, cancela los nodos actualmente resaltados
setCurrentNode	establece el nodo resaltado, solo funciona si `node-key` está asignado	(node) nodo a ser resaltado
getNode	devuelve el nodo por el dato o la clave	(data) los datos o clave del nodo
remove	elimina un nodo, solo funciona si `node-key` está asignado	(data) los datos del nodo o nodo a borrar
append	añadir un nodo hijo a un nodo determinado del árbol	(data, parentNode) 1. los datos del nodo hijo que se añadirán 2. los datos del nodo padre, clave o nodo
insertBefore	insertar un nodo antes de un nodo dado en el árbol	(data, refNode) 1. Datos del nodo que se insertarán 2. Datos del nodo de referencia, clave o nodo

Método	Descripción	Parámetros
insertAfter	insertar un nodo después de un nodo dado en el árbol	(data, refNode) 1. Datos del nodo que se insertarán 2. Datos del nodo de referencia, clave o nodo

Eventos

Nombre del evento	Descripción	Parámetros
node-click	se lanza cuando un nodo es pinchado	tres parámetros: el objeto del nodo seleccionado, propiedad `node` de TreeNode y el TreeNode en si
node-contextmenu	se lanza cuando en un nodo se hace clic con el botón derecho	cuatro parámetros: evento, el objeto nodo sobre el que se hizo clic, la propiedad `node` del TreeNode, el TreeNode en si mismo
check-change	se lanza cuando el estado de selección del nodo cambia	tres parámetros: objeto nodo que se corresponde con el que ha cambiado, booleano que dice si esta seleccionado, booleano que dice si el nodo tiene hijos seleccionados
check	se activa al hacer clic en la casilla de selección de un nodo	dos parámetros: objeto de nodo correspondiente al nodo que se marca/desmarca, objeto de estatus de árbol verificado que tiene cuatro puntales: checkedNodes, checkedKeys, halfCheckedNodes, halfCheckedKeys
current-change	cambia cuando el nodo actual cambia	dos parámetros: objeto nodo que se corresponde al nodo actual y propiedad `node` del TreeNode
node-expand	se lanza cuando el nodo actual se abre	tres parámetros: el objeto del nodo abierto, propiedad `node` de TreeNode y el TreeNode en si

Nombre del evento	Descripción	Parámetros
node-collapse	se lanza cuando el nodo actual se cierra	tres parámetros: el objeto del nodo cerrado, propiedad `node` de TreeNode y el TreeNode en si
node-drag-start	se activa cuando se inicia el arrastre	dos parámetros: el objeto del nodo que se arrastrara, evento.
node-drag-enter	se desencadena cuando el nodo de arrastre entra en otro nodo	tres parámetros: objeto del nodo que se arrastra, objeto del nodo en el que entra, evento.
node-drag-leave	se desencadena cuando el nodo de arrastre sale de un nodo	tres parámetros: objeto del nodo que se arrastra, objeto del nodo del cual se sale, evento.
node-drag-over	se activa cuando se arrastra sobre un nodo (como el evento mouseover)	tres parámetros: objeto del nodo que se arrastra, objeto del nodo sobre el que esta el arrastre, evento.
node-drag-end	se activa cuando se termina de arrastrar	cuatro parámetros: objeto del nodo que se arrastra, objeto del nodo que corresponde al final del arrastre (puede ser `undefined`), tipo de integración (antes (before), después (after), dentro (inner)), evento.
node-drop	después de soltar el nodo de arrastre	cuatro parámetros: objeto del nodo que se esta arrastrando, objeto del nodo sobre el que se esta soltando, tipo de integración (antes (before), después (after), dentro (inner)), evento.

Scoped Slot

Nombre	Descripción
—	Contenido personalizado para nodos de tree. El parámetro del scope es { node, data }.