Tree

Display a set of data with hierarchies.

Basic usage

Basic tree structure.

:::demo

```
<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>
```

:::

Selectable

Used for node selection.

:::demo This example also shows how to load node data asynchronously.

```
<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>
```

:::

Custom leaf node in lazy mode

:::demo A node's data is not fetched until it is clicked, so the Tree cannot predict whether a node is a leaf node. That's why a drop-down button is added to each node, and if it is a leaf node, the drop-down button will disappear when clicked. That being said, you can also tell the Tree in advance whether the node is a leaf node, avoiding the render of the drop-down button before a leaf node.

```
<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>
```

:::

Disabled checkbox

The checkbox of a node can be set as disabled.

:::demo In the example, 'disabled' property is declared in defaultProps, and some nodes are set as 'disabled:true'. The corresponding checkboxes are disabled and can't be clicked.

```
<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>
```

:::

Default expanded and default checked

Tree nodes can be initially expanded or checked

:::demo Use `default-expanded-keys` and `default-checked-keys` to set initially expanded and initially checked nodes respectively. Note that for them to work, `node-key` is required. Its value is the name of a key in the data object, and the value of that key should be unique across the whole tree.

```
<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>
```

:::

Checking tree nodes

:::demo This example shows how to get and set checked nodes. They both can be done in two approaches: node and key. If you are taking the key approach, `node-key` is required.

```
<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>
```

:::

Custom node content

The content of tree nodes can be customized, so you can add icons or buttons as you will

:::demo There are two ways to customize template for tree nodes: `render-content` and scoped slot. Use `render-content` to assign a render function that returns the content of tree nodes. See Vue's documentation for a detailed introduction of render functions. If you prefer scoped slot, you'll have access to `node` and `data` in the scope, standing for the TreeNode object and node data of the current node respectively. Note that the `render-`

`content` demo can't run in jsfiddle because it doesn't support JSX syntax. In a real project, `render-content` will work if relevant dependencies are correctly configured.

```
<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) }>Append</el-button>
              <el-button size="mini" type="text" on-click={ () => this.remove(node, data) }>Delete</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>
```

:::

Tree node filtering

Tree nodes can be filtered

:::demo Invoke the `filter` method of the Tree instance to filter tree nodes. Its parameter is the filtering keyword. Note that for it to work, `filter-node-method` is required, and its value is the filtering method.

```
<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>
```

:::

Accordion

Only one node among the same level can be expanded at one time.

:::demo

```
<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

You can drag and drop Tree nodes by adding a `draggable` attribute.

:::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: '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>
```

:::

Attributes

Attribute	Description	Type	Accepted Values	Default
data	tree data	array	—	—
empty-text	text displayed when data is void	string	—	—
node-key	unique identity key name for nodes, its value should be unique across the whole tree	string	—	—
props	configuration options, see the following table	object	—	—
render-after-expand	whether to render child nodes only after a parent node is expanded for the first time	boolean	—	true
load	method for loading subtree data, only works when `lazy` is true	function(node, resolve)	—	—
render-content	render function for tree node	Function(h, { node, data, store }	—	—
highlight-	whether current node is highlighted	boolean	—	false

current				
default-expand-all	whether to expand all nodes by default	boolean	—	false
expand-on-click-node	whether to expand or collapse node when clicking on the node, if false, then expand or collapse node only when clicking on the arrow icon.	boolean	—	true
check-on-click-node	whether to check or uncheck node when clicking on the node, if false, the node can only be checked or unchecked by clicking on the checkbox.	boolean	—	false
auto-expand-parent	whether to expand father node when a child node is expanded	boolean	—	true
default-expanded-keys	array of keys of initially expanded nodes	array	—	—
show-checkbox	whether node is selectable	boolean	—	false
check-strictly	whether checked state of a node not affects its father and child nodes when `show-checkbox` is `true`	boolean	—	false
default-checked-keys	array of keys of initially checked nodes	array	—	—
current-node-key	key of initially selected node	string, number	—	—
filter-node-method	this function will be executed on each node when use filter method. if return `false`, tree node will be hidden.	Function(value, data, node)	—	—
accordion	whether only one node among the same level can be expanded at one time	boolean	—	false
indent	horizontal indentation of nodes in adjacent levels in pixels	number	—	16
icon-class	custome tree node icon	string	-	-
lazy	whether to lazy load leaf node, used with `load` attribute	boolean	—	false
draggable	whether enable tree nodes drag and drop	boolean	—	false
allow-drag	this function will be executed before	Function(node)	—	—

	dragging a node. If `false` is returned, the node can not be dragged			
allow-drop	this function will be executed before the dragging node is dropped. If `false` is returned, the dragging node can not be dropped at the target node. `type` has three possible values: 'prev' (inserting the dragging node before the target node), 'inner' (inserting the dragging node to the target node) and 'next' (inserting the dragging node after the target node)	Function(draggingNode, dropNode, type)	—	—

props

Attribute	Description	Type	Accepted Values	Default
label	specify which key of node object is used as the node's label	string, function(data, node)	—	—
children	specify which node object is used as the node's subtree	string	—	—
disabled	specify which key of node object represents if node's checkbox is disabled	boolean, function(data, node)	—	—
isLeaf	specify whether the node is a leaf node, only works when lazy load is enabled	boolean, function(data, node)	—	—

Method

`Tree` has the following method, which returns the currently selected array of nodes.

Method	Description	Parameters
filter	filter all tree nodes, filtered nodes will be hidden	Accept a parameter which will be used as first parameter for filter-node-method
updateKeyChildren	set new data to node, only works when `node-key` is assigned	(key, data) Accept two parameters: 1. key of node 2. new data
getCheckedNodes	If the node can be selected (`show-checkbox` is `true`), it returns the currently	(leafOnly, includeHalfChecked) Accept two boolean type parameters: 1. default value is `false`. If the parameter is `true`, it only returns the currently selected array of sub-

	selected array of nodes	nodes. 2. default value is `false`. If the parameter is `true`, the return value contains halfchecked nodes
setCheckedNodes	set certain nodes to be checked, only works when `node-key` is assigned	an array of nodes to be checked
getCheckedKeys	If the node can be selected (`show-checkbox` is `true`), it returns the currently selected array of node's keys	(leafOnly) Accept a boolean type parameter whose default value is `false`. If the parameter is `true`, it only returns the currently selected array of sub-nodes.
setCheckedKeys	set certain nodes to be checked, only works when `node-key` is assigned	(keys, leafOnly) Accept two parameters: 1. an array of node's keys to be checked 2. a boolean type parameter whose default value is `false`. If the parameter is `true`, it only returns the currently selected array of sub-nodes.
setChecked	set node to be checked or not, only works when `node-key` is assigned	(key/data, checked, deep) Accept three parameters: 1. node's key or data to be checked 2. a boolean typed parameter indicating checked or not. 3. a boolean typed parameter indicating deep or not.
getHalfCheckedNodes	If the node can be selected (`show-checkbox` is `true`), it returns the currently half selected array of nodes	-
getHalfCheckedKeys	If the node can be selected (`show-checkbox` is `true`), it returns the currently half selected array of node's keys	-
getCurrentKey	return the highlight node's key (null if no node is highlighted)	—
getCurrentNode	return the highlight node's data (null if no node is highlighted)	—
setCurrentKey	set highlighted node by key, only works when `node-key` is assigned	(key) the node's key to be highlighted. If `null`, cancel the currently highlighted node
setCurrentNode	set highlighted	(node) the node to be highlighted

	node, only works when `node-key` is assigned	
getNode	get node by data or key	(data) the node's data or key
remove	remove a node, only works when node-key is assigned	(data) the node's data or node to be deleted
append	append a child node to a given node in the tree	(data, parentNode) 1. child node's data to be appended 2. parent node's data, key or node
insertBefore	insert a node before a given node in the tree	(data, refNode) 1. node's data to be inserted 2. reference node's data, key or node
insertAfter	insert a node after a given node in the tree	(data, refNode) 1. node's data to be inserted 2. reference node's data, key or node

Events

Event Name	Description	Parameters
node-click	triggers when a node is clicked	three parameters: node object corresponding to the node clicked, `node` property of TreeNode, TreeNode itself
node-contextmenu	triggers when a node is clicked by right button	four parameters: event, node object corresponding to the node clicked, `node` property of TreeNode, TreeNode itself
check-change	triggers when the selected state of the node changes	three parameters: node object corresponding to the node whose selected state is changed, whether the node is selected, whether node's subtree has selected nodes
check	triggers after clicking the checkbox of a node	two parameters: node object corresponding to the node that is checked / unchecked, tree checked status object which has four props: checkedNodes, checkedKeys, halfCheckedNodes, halfCheckedKeys
current-change	triggers when current node changes	two parameters: node object corresponding to the current node, `node` property of TreeNode
node-expand	triggers when current node open	three parameters: node object corresponding to the node opened, `node` property of TreeNode, TreeNode itself
node-collapse	triggers when current node close	three parameters: node object corresponding to the node closed, `node` property of TreeNode, TreeNode itself
node-drag-start	triggers when dragging starts	two parameters: node object corresponding to the dragging node, event.

node-drag-enter	triggers when the dragging node enters another node	three parameters: node object corresponding to the dragging node, node object corresponding to the entering node, event.
node-drag-leave	triggers when the dragging node leaves a node	three parameters: node object corresponding to the dragging node, node object corresponding to the leaving node, event.
node-drag-over	triggers when dragging over a node (like mouseover event)	three parameters: node object corresponding to the dragging node, node object corresponding to the dragging over node, event.
node-drag-end	triggers when dragging ends	four parameters: node object corresponding to the dragging node, node object corresponding to the dragging end node (may be `undefined`), node drop type (before / after / inner), event.
node-drop	triggers after the dragging node is dropped	four parameters: node object corresponding to the dragging node, node object corresponding to the dropped node, node drop type (before / after / inner), event.

Scoped Slot

Name	Description
—	Custom content for tree nodes. The scope parameter is { node, data }