Tree 树形控件

用清晰的层级结构展示信息，可展开或折叠。

基础用法

基础的树形结构展示。

:::demo

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

<script>
  export default {
    data() {
      return {
        data: [{
          label: '一级 1',
          children: [{
            label: '二级 1-1',
            children: [{
              label: '三级 1-1-1'
            }]
          }]
        }, {
          label: '一级 2',
          children: [{
            label: '二级 2-1',
            children: [{
              label: '三级 2-1-1'
            }]
          }, {
            label: '二级 2-2',
            children: [{
              label: '三级 2-2-1'
            }]
          }]
        }, {
          label: '一级 3',
          children: [{
            label: '二级 3-1',
            children: [{
              label: '三级 3-1-1'
            }]
          }, {
            label: '二级 3-2',
            children: [{
              label: '三级 3-2-1'
            }]
          }]
        }],
        defaultProps: {
```

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

:::

可选择

适用于需要选择层级时使用。

:::demo 本例还展示了动态加载节点数据的方法。

```
<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: 'region1' }, { name: 'region2' }]);
        }
        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>
```

:::

懒加载自定义叶子节点

:::demo 由于在点击节点时才进行该层数据的获取，默认情况下 Tree 无法预知某个节点是否为叶子节点，所以会为每个节点添加一个下拉按钮，如果节点没有下层数据，则点击后下拉按钮会消失。同时，你也可以提前告知 Tree 某个节点是否为叶子节点，从而避免在叶子节点前渲染下拉按钮。

```
<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>
```

:::

默认展开和默认选中

可将 Tree 的某些节点设置为默认展开或默认选中

:::demo 分别通过 `default-expanded-keys` 和 `default-checked-keys` 设置默认展开和默认选中的节点。需要注意的是，此时必须设置 `node-key`，其值为节点数据中的一个字段名，该字段在整棵树中是唯一的。

```
<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: '一级 1',
          children: [{
            id: 4,
```

```
            label: '二级 1-1',
            children: [{
              id: 9,
              label: '三级 1-1-1'
            }, {
              id: 10,
              label: '三级 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: '一级 2',
          children: [{
            id: 5,
            label: '二级 2-1'
          }, {
            id: 6,
            label: '二级 2-2'
          }]
        }, {
          id: 3,
          label: '一级 3',
          children: [{
            id: 7,
            label: '二级 3-1'
          }, {
            id: 8,
            label: '二级 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

禁用状态

可将 Tree 的某些节点设置为禁用状态

:::demo 通过 `disabled` 设置禁用状态。

```
<el-tree
  :data="data"
  show-checkbox
  node-key="id"
  :default-expanded-keys="[2, 3]"
```

```
  :default-checked-keys="[5]">
</el-tree>

<script>
  export default {
    data() {
      return {
        data: [{
          id: 1,
          label: '一级 2',
          children: [{
            id: 3,
            label: '二级 2-1',
            children: [{
              id: 4,
              label: '三级 3-1-1'
            }, {
              id: 5,
              label: '三级 3-1-2',
              disabled: true
            }]
          }, {
            id: 2,
            label: '二级 2-2',
            disabled: true,
            children: [{
              id: 6,
              label: '三级 3-2-1'
            }, {
              id: 7,
              label: '三级 3-2-2',
              disabled: true
            }]
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

树节点的选择

:::demo 本例展示如何获取和设置选中节点。获取和设置各有两种方式：通过 node 或通过 key。如果需要通过 key 来获取或设置，则必须设置 `node-key` 。

```
<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">通过 node 获取</el-button>
  <el-button @click="getCheckedKeys">通过 key 获取</el-button>
  <el-button @click="setCheckedNodes">通过 node 设置</el-button>
  <el-button @click="setCheckedKeys">通过 key 设置</el-button>
  <el-button @click="resetChecked">清空</el-button>
</div>

<script>
  export default {
    methods: {
      getCheckedNodes() {
        console.log(this.$refs.tree.getCheckedNodes());
      },
      getCheckedKeys() {
        console.log(this.$refs.tree.getCheckedKeys());
      },
      setCheckedNodes() {
        this.$refs.tree.setCheckedNodes([{
          id: 5,
          label: '二级 2-1'
        }, {
          id: 9,
          label: '三级 1-1-1'
        }]);
      },
      setCheckedKeys() {
        this.$refs.tree.setCheckedKeys([3]);
      },
      resetChecked() {
        this.$refs.tree.setCheckedKeys([]);
      }
    },

    data() {
      return {
        data: [{
          id: 1,
          label: '一级 1',
          children: [{
            id: 4,
```

```
            label: '二级 1-1',
            children: [{
              id: 9,
              label: '三级 1-1-1'
            }, {
              id: 10,
              label: '三级 1-1-2'
            }]
          }]
        }, {
          id: 2,
          label: '一级 2',
          children: [{
            id: 5,
            label: '二级 2-1'
          }, {
            id: 6,
            label: '二级 2-2'
          }]
        }, {
          id: 3,
          label: '一级 3',
          children: [{
            id: 7,
            label: '二级 3-1'
          }, {
            id: 8,
            label: '二级 3-2'
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

自定义节点内容

节点的内容支持自定义，可以在节点区添加按钮或图标等内容

:::demo 可以通过两种方法进行树节点内容的自定义：`render-content` 和 scoped slot。使用 `render-content` 指定渲染函数，该函数返回需要的节点区内容即可。渲染函数的用法请参考 Vue 文档。使用 scoped slot 会传入两个参数 `node` 和 `data`，分别表示当前节点的 Node 对象和当前节点的数据。注意：由于 jsfiddle 不支持 JSX 语法，所以 `render-content` 示例在 jsfiddle 中无法运行。但是在实际的项目中，只要正确地配置了相关依赖，就可以正常运行。

```
<div class="custom-tree-container">
  <div class="block">
    <p>使用 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>使用 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: '一级 1',
        children: [{
          id: 4,
```

```
        label: '二级 1-1',
        children: [{
          id: 9,
          label: '三级 1-1-1'
        }, {
          id: 10,
          label: '三级 1-1-2'
        }]
      }]
    }, {
      id: 2,
      label: '一级 2',
      children: [{
        id: 5,
        label: '二级 2-1'
      }, {
        id: 6,
        label: '二级 2-2'
      }]
    }, {
      id: 3,
      label: '一级 3',
      children: [{
        id: 7,
        label: '二级 3-1'
      }, {
        id: 8,
        label: '二级 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>
```

:::

节点过滤

通过关键字过滤树节点

:::demo 在需要对节点进行过滤时，调用 Tree 实例的 `filter` 方法，参数为关键字。需要注意的是，此时需要设置 `filter-node-method`，值为过滤函数。

```
<el-input
  placeholder="输入关键字进行过滤"
  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: '一级 1',
      children: [{
        id: 4,
        label: '二级 1-1',
        children: [{
          id: 9,
          label: '三级 1-1-1'
        }, {
          id: 10,
          label: '三级 1-1-2'
        }]
      }]
    }, {
      id: 2,
      label: '一级 2',
      children: [{
        id: 5,
        label: '二级 2-1'
      }, {
        id: 6,
        label: '二级 2-2'
      }]
    }, {
      id: 3,
      label: '一级 3',
      children: [{
        id: 7,
        label: '二级 3-1'
      }, {
        id: 8,
        label: '二级 3-2'
      }]
    }],
    defaultProps: {
```

```
          children: 'children',
          label: 'label'
        }
      };
    }
  };
</script>
```

:::

手风琴模式

对于同一级的节点，每次只能展开一个

:::demo

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

<script>
  export default {
    data() {
      return {
        data: [{
          label: '一级 1',
          children: [{
            label: '二级 1-1',
            children: [{
              label: '三级 1-1-1'
            }]
          }]
        }, {
          label: '一级 2',
          children: [{
            label: '二级 2-1',
            children: [{
              label: '三级 2-1-1'
            }]
          }, {
            label: '二级 2-2',
            children: [{
              label: '三级 2-2-1'
            }]
          }]
        }, {
          label: '一级 3',
          children: [{
            label: '二级 3-1',
```

```
            children: [{
              label: '三级 3-1-1'
            }]
          }, {
            label: '二级 3-2',
            children: [{
              label: '三级 3-2-1'
            }]
          }]
        }],
        defaultProps: {
          children: 'children',
          label: 'label'
        }
      };
    },
    methods: {
      handleNodeClick(data) {
        console.log(data);
      }
    }
  };
</script>
```

:::

可拖拽节点

通过 draggable 属性可让节点变为可拖拽。

:::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: [{
```

```
        id: 1,
        label: '一级 1',
        children: [{
          id: 4,
          label: '二级 1-1',
          children: [{
            id: 9,
            label: '三级 1-1-1'
          }, {
            id: 10,
            label: '三级 1-1-2'
          }]
        }]
      }, {
        id: 2,
        label: '一级 2',
        children: [{
          id: 5,
          label: '二级 2-1'
        }, {
          id: 6,
          label: '二级 2-2'
        }]
      }, {
        id: 3,
        label: '一级 3',
        children: [{
          id: 7,
          label: '二级 3-1'
        }, {
          id: 8,
          label: '二级 3-2',
          children: [{
           id: 11,
            label: '三级 3-2-1'
          }, {
            id: 12,
            label: '三级 3-2-2'
          }, {
            id: 13,
            label: '三级 3-2-3'
          }]
        }]
      }],
      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 === '二级 3-1') {
          return type !== 'inner';
        } else {
          return true;
        }
      },
      allowDrag(draggingNode) {
        return draggingNode.data.label.indexOf('三级 3-2-2') === -1;
      }
    }
  };
</script>
```

:::

Attributes

参数	说明	类型	可选值	默认值
data	展示数据	array	—	—
empty-text	内容为空的时候展示的文本	String	—	—
node-key	每个树节点用来作为唯一标识的属性，整棵树应该是唯一的	String	—	—
props	配置选项，具体看下表	object	—	—
render-after-expand	是否在第一次展开某个树节点后才渲染其子节点	boolean	—	true
load	加载子树数据的方法，仅当 lazy 属性为true 时生效	function(node, resolve)	—	—

render-content	树节点的内容区的渲染 Function	Function(h, { node, data, store }	—	—
highlight-current	是否高亮当前选中节点，默认值是 false。	boolean	—	false
default-expand-all	是否默认展开所有节点	boolean	—	false
expand-on-click-node	是否在点击节点的时候展开或者收缩节点， 默认值为 true，如果为 false，则只有点箭头图标的时候才会展开或者收缩节点。	boolean	—	true
check-on-click-node	是否在点击节点的时候选中节点，默认值为 false，即只有在点击复选框时才会选中节点。	boolean	—	false
auto-expand-parent	展开子节点的时候是否自动展开父节点	boolean	—	true
default-expanded-keys	默认展开的节点的 key 的数组	array	—	—
show-checkbox	节点是否可被选择	boolean	—	false
check-strictly	在显示复选框的情况下，是否严格的遵循父子不互相关联的做法，默认为 false	boolean	—	false
default-checked-keys	默认勾选的节点的 key 的数组	array	—	—
current-node-key	当前选中的节点	string, number	—	—
filter-node-method	对树节点进行筛选时执行的方法，返回 true 表示这个节点可以显示，返回 false 则表示这个节点会被隐藏	Function(value, data, node)	—	—
accordion	是否每次只打开一个同级树节点展开	boolean	—	false
indent	相邻级节点间的水平缩进，单位为像素	number	—	16
icon-class	自定义树节点的图标	string	-	-
lazy	是否懒加载子节点，需与 load 方法结合使用	boolean	—	false
draggable	是否开启拖拽节点功能	boolean	—	false
allow-drag	判断节点能否被拖拽	Function(node)	—	—
allow-drop	拖拽时判定目标节点能否被放置。`type` 参数有三种情况：'prev'、'inner' 和 'next'，分别表示放置在目标节点前、插入至目标节点和放置在目标节点后	Function(draggingNode, dropNode, type)	—	—

props

参数	说明	类型	可选值	默认值
label	指定节点标签为节点对象的某个属性值	string, function(data, node)	—	—
children	指定子树为节点对象的某个属性值	string	—	—
disabled	指定节点选择框是否禁用为节点对象的某个属性值	boolean, function(data, node)	—	—
isLeaf	指定节点是否为叶子节点，仅在指定了 lazy 属性的情况下生效	boolean, function(data, node)	—	—

方法

`Tree` 内部使用了 Node 类型的对象来包装用户传入的数据，用来保存目前节点的状态。 `Tree` 拥有如下方法：

方法名	说明	参数
filter	对树节点进行筛选操作	接收一个任意类型的参数，该参数会在 filter-node-method 中作为第一个参数
updateKeyChildren	通过 keys 设置节点子元素，使用此方法必须设置 node-key 属性	(key, data) 接收两个参数，1. 节点 key 2. 节点数据的数组
getCheckedNodes	若节点可被选择（即 `show-checkbox` 为 `true`），则返回目前被选中的节点所组成的数组	(leafOnly, includeHalfChecked) 接收两个 boolean 类型的参数，1. 是否只是叶子节点，默认值为 `false` 2. 是否包含半选节点，默认值为 `false`
setCheckedNodes	设置目前勾选的节点，使用此方法必须设置 node-key 属性	(nodes) 接收勾选节点数据的数组
getCheckedKeys	若节点可被选择（即 `show-checkbox` 为 `true`），则返回目前被选中的节点的 key 所组成的数组	(leafOnly) 接收一个 boolean 类型的参数，若为 `true` 则仅返回被选中的叶子节点的 keys，默认值为 `false`
setCheckedKeys	通过 keys 设置目前勾选的节点，使用此方法必须设置 node-key 属性	(keys, leafOnly) 接收两个参数，1. 勾选节点的 key 的数组 2. boolean 类型的参数，若为 `true` 则仅设置叶子节点的选中状态，默认值为 `false`
setChecked	通过 key / data 设置某个节点的勾选状态，使用此方法必须设置 node-key 属性	(key/data, checked, deep) 接收三个参数，1. 勾选节点的 key 或者 data 2. boolean 类型，节点是否选中 3. boolean 类型，是否设置子节点 ，默认为 false
getHalfCheckedNodes	若节点可被选择（即 `show-checkbox` 为 `true`），则返回目前半选中的节点所组成的数组	-

getHalfCheckedKeys	若节点可被选择（即 `show-checkbox` 为 `true`），则返回目前半选中的节点的 key 所组成的数组	-
getCurrentKey	获取当前被选中节点的 key，使用此方法必须设置 node-key 属性，若没有节点被选中则返回 null	—
getCurrentNode	获取当前被选中节点的 data，若没有节点被选中则返回 null	—
setCurrentKey	通过 key 设置某个节点的当前选中状态，使用此方法必须设置 node-key 属性	(key) 待被选节点的 key，若为 null 则取消当前高亮的节点
setCurrentNode	通过 node 设置某个节点的当前选中状态，使用此方法必须设置 node-key 属性	(node) 待被选节点的 node
getNode	根据 data 或者 key 拿到 Tree 组件中的 node	(data) 要获得 node 的 key 或者 data
remove	删除 Tree 中的一个节点，使用此方法必须设置 node-key 属性	(data) 要删除的节点的 data 或者 node
append	为 Tree 中的一个节点追加一个子节点	(data, parentNode) 接收两个参数，1. 要追加的子节点的 data 2. 子节点的 parent 的 data、key 或者 node
insertBefore	为 Tree 的一个节点的前面增加一个节点	(data, refNode) 接收两个参数，1. 要增加的节点的 data 2. 要增加的节点的后一个节点的 data、key 或者 node
insertAfter	为 Tree 的一个节点的后面增加一个节点	(data, refNode) 接收两个参数，1. 要增加的节点的 data 2. 要增加的节点的前一个节点的 data、key 或者 node

Events

事件名称	说明	回调参数
node-click	节点被点击时的回调	共三个参数，依次为：传递给 `data` 属性的数组中该节点所对应的对象、节点对应的 Node、节点组件本身。
node-contextmenu	当某一节点被鼠标右键点击时会触发该事件	共四个参数，依次为：event、传递给 `data` 属性的数组中该节点所对应的对象、节点对应的 Node、节点组件本身。
check-change	节点选中状态发生变化时的回调	共三个参数，依次为：传递给 `data` 属性的数组中该节点所对应的对象、节点本身是否被选中、节点的子树中是否有被选中的节点
check	当复选框被点击的时	共两个参数，依次为：传递给 `data` 属性的数组中该节点所对应的对

	候触发	象、树目前的选中状态对象，包含 checkedNodes、checkedKeys、halfCheckedNodes、halfCheckedKeys 四个属性
current-change	当前选中节点变化时触发的事件	共两个参数，依次为：当前节点的数据，当前节点的 Node 对象
node-expand	节点被展开时触发的事件	共三个参数，依次为：传递给 `data` 属性的数组中该节点所对应的对象、节点对应的 Node、节点组件本身
node-collapse	节点被关闭时触发的事件	共三个参数，依次为：传递给 `data` 属性的数组中该节点所对应的对象、节点对应的 Node、节点组件本身
node-drag-start	节点开始拖拽时触发的事件	共两个参数，依次为：被拖拽节点对应的 Node、event
node-drag-enter	拖拽进入其他节点时触发的事件	共三个参数，依次为：被拖拽节点对应的 Node、所进入节点对应的 Node、event
node-drag-leave	拖拽离开某个节点时触发的事件	共三个参数，依次为：被拖拽节点对应的 Node、所离开节点对应的 Node、event
node-drag-over	在拖拽节点时触发的事件（类似浏览器的 mouseover 事件）	共三个参数，依次为：被拖拽节点对应的 Node、当前进入节点对应的 Node、event
node-drag-end	拖拽结束时（可能未成功）触发的事件	共四个参数，依次为：被拖拽节点对应的 Node、结束拖拽时最后进入的节点（可能为空）、被拖拽节点的放置位置（before、after、inner）、event
node-drop	拖拽成功完成时触发的事件	共四个参数，依次为：被拖拽节点对应的 Node、结束拖拽时最后进入的节点、被拖拽节点的放置位置（before、after、inner）、event

Scoped Slot

name	说明
—	自定义树节点的内容，参数为 { node, data }