## 何时使用 {#when-to-use}

文件夹、组织架构、生物分类、国家地区等等，世间万物的大多数结构都是树形结构。使用 `树控件` 可以完整展现其中的层级关系，并具有展开收起选择等交互功能。

## 代码演示

### 基本

```
import React from 'react';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        disabled: true,
        children: [
          {
            title: 'leaf',
            key: '0-0-0-0',
            disableCheckbox: true,
          },
          {
            title: 'leaf',
            key: '0-0-0-1',
          },
        ],
      },
      {
        title: 'parent 1-1',
        key: '0-0-1',
        children: [{ title: <span style={{ color: '#1677ff' }}>sss</span>,
key: '0-0-1-0' }],
      },
    ],
  },
];

const App: React.FC = () => {
  const onSelect: TreeProps['onSelect'] = (selectedKeys, info) => {
```

```
    console.log('selected', selectedKeys, info);
  };

  const onCheck: TreeProps['onCheck'] = (checkedKeys, info) => {
    console.log('onCheck', checkedKeys, info);
  };

  return (
    <Tree
      checkable
      defaultExpandedKeys={['0-0-0', '0-0-1']}
      defaultSelectedKeys={['0-0-1']}
      defaultCheckedKeys={['0-0-0', '0-0-1']}
      onSelect={onSelect}
      onCheck={onCheck}
      treeData={treeData}
    />
  );
};

export default App;
```

**受控操作示例**

```
import React, { useState } from 'react';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: '0-0',
    key: '0-0',
    children: [
      {
        title: '0-0-0',
        key: '0-0-0',
        children: [
          { title: '0-0-0-0', key: '0-0-0-0' },
          { title: '0-0-0-1', key: '0-0-0-1' },
          { title: '0-0-0-2', key: '0-0-0-2' },
        ],
      },
      {
        title: '0-0-1',
        key: '0-0-1',
        children: [
```

```
          { title: '0-0-1-0', key: '0-0-1-0' },
          { title: '0-0-1-1', key: '0-0-1-1' },
          { title: '0-0-1-2', key: '0-0-1-2' },
        ],
      },
      {
        title: '0-0-2',
        key: '0-0-2',
      },
    ],
  },
  {
    title: '0-1',
    key: '0-1',
    children: [
      { title: '0-1-0-0', key: '0-1-0-0' },
      { title: '0-1-0-1', key: '0-1-0-1' },
      { title: '0-1-0-2', key: '0-1-0-2' },
    ],
  },
  {
    title: '0-2',
    key: '0-2',
  },
];

const App: React.FC = () => {
  const [expandedKeys, setExpandedKeys] = useState<React.Key[]>(['0-0-0',
'0-0-1']);
  const [checkedKeys, setCheckedKeys] = useState<React.Key[]>(['0-0-0']);
  const [selectedKeys, setSelectedKeys] = useState<React.Key[]>([]);
  const [autoExpandParent, setAutoExpandParent] = useState<boolean>(true);

  const onExpand: TreeProps['onExpand'] = (expandedKeysValue) => {
    console.log('onExpand', expandedKeysValue);
    // if not set autoExpandParent to false, if children expanded, parent
can not collapse.
    // or, you can remove all expanded children keys.
    setExpandedKeys(expandedKeysValue);
    setAutoExpandParent(false);
  };

  const onCheck: TreeProps['onCheck'] = (checkedKeysValue) => {
    console.log('onCheck', checkedKeysValue);
    setCheckedKeys(checkedKeysValue as React.Key[]);
  };
```

```
  const onSelect: TreeProps['onSelect'] = (selectedKeysValue, info) => {
    console.log('onSelect', info);
    setSelectedKeys(selectedKeysValue);
  };

  return (
    <Tree
      checkable
      onExpand={onExpand}
      expandedKeys={expandedKeys}
      autoExpandParent={autoExpandParent}
      onCheck={onCheck}
      checkedKeys={checkedKeys}
      onSelect={onSelect}
      selectedKeys={selectedKeys}
      treeData={treeData}
    />
  );
};

export default App;
```

**拖动示例**

```
import React, { useState } from 'react';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const x = 3;
const y = 2;
const z = 1;
const defaultData: TreeDataNode[] = [];

const generateData = (_level: number, _preKey?: React.Key, _tns?:
TreeDataNode[]) => {
  const preKey = _preKey || '0';
  const tns = _tns || defaultData;

  const children: React.Key[] = [];
  for (let i = 0; i < x; i++) {
    const key = `${preKey}-${i}`;
    tns.push({ title: key, key });
    if (i < y) {
      children.push(key);
    }
```

```
  }
  if (_level < 0) {
    return tns;
  }
  const level = _level - 1;
  children.forEach((key, index) => {
    tns[index].children = [];
    return generateData(level, key, tns[index].children);
  });
};
generateData(z);

const App: React.FC = () => {
  const [gData, setGData] = useState(defaultData);
  const [expandedKeys] = useState(['0-0', '0-0-0', '0-0-0-0']);

  const onDragEnter: TreeProps['onDragEnter'] = (info) => {
    console.log(info);
    // expandedKeys, set it when controlled is needed
    // setExpandedKeys(info.expandedKeys)
  };

  const onDrop: TreeProps['onDrop'] = (info) => {
    console.log(info);
    const dropKey = info.node.key;
    const dragKey = info.dragNode.key;
    const dropPos = info.node.pos.split('-');
    const dropPosition = info.dropPosition - Number(dropPos[dropPos.length
- 1]); // the drop position relative to the drop node, inside 0, top -1,
bottom 1

    const loop = (
      data: TreeDataNode[],
      key: React.Key,
      callback: (node: TreeDataNode, i: number, data: TreeDataNode[]) =>
void,
    ) => {
      for (let i = 0; i < data.length; i++) {
        if (data[i].key === key) {
          return callback(data[i], i, data);
        }
        if (data[i].children) {
          loop(data[i].children!, key, callback);
        }
      }
    };
```

```
    const data = [...gData];

    // Find dragObject
    let dragObj: TreeDataNode;
    loop(data, dragKey, (item, index, arr) => {
      arr.splice(index, 1);
      dragObj = item;
    });

    if (!info.dropToGap) {
      // Drop on the content
      loop(data, dropKey, (item) => {
        item.children = item.children || [];
        // where to insert. New item was inserted to the start of the array
in this example, but can be anywhere
        item.children.unshift(dragObj);
      });
    } else {
      let ar: TreeDataNode[] = [];
      let i: number;
      loop(data, dropKey, (_item, index, arr) => {
        ar = arr;
        i = index;
      });
      if (dropPosition === -1) {
        // Drop on the top of the drop node
        ar.splice(i!, 0, dragObj!);
      } else {
        // Drop on the bottom of the drop node
        ar.splice(i! + 1, 0, dragObj!);
      }
    }
    setGData(data);
  };

  return (
    <Tree
      className="draggable-tree"
      defaultExpandedKeys={expandedKeys}
      draggable
      blockNode
      onDragEnter={onDragEnter}
      onDrop={onDrop}
      treeData={gData}
    />
  );
```

```
};

export default App;
```

**异步数据加载**

```
import React, { useState } from 'react';
import { Tree } from 'antd';

interface DataNode {
  title: string;
  key: string;
  isLeaf?: boolean;
  children?: DataNode[];
}

const initTreeData: DataNode[] = [
  { title: 'Expand to load', key: '0' },
  { title: 'Expand to load', key: '1' },
  { title: 'Tree Node', key: '2', isLeaf: true },
];

// It's just a simple demo. You can use tree map to optimize update perf.
const updateTreeData = (list: DataNode[], key: React.Key, children:
DataNode[]): DataNode[] =>
  list.map((node) => {
    if (node.key === key) {
      return {
        ...node,
        children,
      };
    }
    if (node.children) {
      return {
        ...node,
        children: updateTreeData(node.children, key, children),
      };
    }
    return node;
  });

const App: React.FC = () => {
  const [treeData, setTreeData] = useState(initTreeData);

  const onLoadData = ({ key, children }: any) =>
    new Promise<void>((resolve) => {
```

```
      if (children) {
        resolve();
        return;
      }
      setTimeout(() => {
        setTreeData((origin) =>
          updateTreeData(origin, key, [
            { title: 'Child Node', key: `${key}-0` },
            { title: 'Child Node', key: `${key}-1` },
          ]),
        );

        resolve();
      }, 1000);
    });

  return <Tree loadData={onLoadData} treeData={treeData} />;
};

export default App;
```

**可搜索**

```
import React, { useMemo, useState } from 'react';
import { Input, Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const { Search } = Input;

const x = 3;
const y = 2;
const z = 1;
const defaultData: TreeDataNode[] = [];

const generateData = (_level: number, _preKey?: React.Key, _tns?:
TreeDataNode[]) => {
  const preKey = _preKey || '0';
  const tns = _tns || defaultData;

  const children: React.Key[] = [];
  for (let i = 0; i < x; i++) {
    const key = `${preKey}-${i}`;
    tns.push({ title: key, key });
    if (i < y) {
      children.push(key);
    }
```

```
  }
  if (_level < 0) {
    return tns;
  }
  const level = _level - 1;
  children.forEach((key, index) => {
    tns[index].children = [];
    return generateData(level, key, tns[index].children);
  });
};
generateData(z);

const dataList: { key: React.Key; title: string }[] = [];
const generateList = (data: TreeDataNode[]) => {
  for (let i = 0; i < data.length; i++) {
    const node = data[i];
    const { key } = node;
    dataList.push({ key, title: key as string });
    if (node.children) {
      generateList(node.children);
    }
  }
};
generateList(defaultData);

const getParentKey = (key: React.Key, tree: TreeDataNode[]): React.Key => {
  let parentKey: React.Key;
  for (let i = 0; i < tree.length; i++) {
    const node = tree[i];
    if (node.children) {
      if (node.children.some((item) => item.key === key)) {
        parentKey = node.key;
      } else if (getParentKey(key, node.children)) {
        parentKey = getParentKey(key, node.children);
      }
    }
  }
  return parentKey!;
};

const App: React.FC = () => {
  const [expandedKeys, setExpandedKeys] = useState<React.Key[]>([]);
  const [searchValue, setSearchValue] = useState('');
  const [autoExpandParent, setAutoExpandParent] = useState(true);

  const onExpand = (newExpandedKeys: React.Key[]) => {
```

```
    setExpandedKeys(newExpandedKeys);
    setAutoExpandParent(false);
  };

  const onChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const { value } = e.target;
    const newExpandedKeys = dataList
      .map((item) => {
        if (item.title.indexOf(value) > -1) {
          return getParentKey(item.key, defaultData);
        }
        return null;
      })
      .filter((item, i, self): item is React.Key => !!(item &&
self.indexOf(item) === i));
    setExpandedKeys(newExpandedKeys);
    setSearchValue(value);
    setAutoExpandParent(true);
  };

  const treeData = useMemo(() => {
    const loop = (data: TreeDataNode[]): TreeDataNode[] =>
      data.map((item) => {
        const strTitle = item.title as string;
        const index = strTitle.indexOf(searchValue);
        const beforeStr = strTitle.substring(0, index);
        const afterStr = strTitle.slice(index + searchValue.length);
        const title =
          index > -1 ? (
            <span key={item.key}>
              {beforeStr}
              <span className="site-tree-search-value">{searchValue}</span>
              {afterStr}
            </span>
          ) : (
            <span key={item.key}>{strTitle}</span>
          );
        if (item.children) {
          return { title, key: item.key, children: loop(item.children) };
        }

        return {
          title,
          key: item.key,
        };
      });
```

```
    return loop(defaultData);
  }, [searchValue]);

  return (
    <div>
      <Search style={{ marginBottom: 8 }} placeholder="Search" onChange=
{onChange} />
      <Tree
        onExpand={onExpand}
        expandedKeys={expandedKeys}
        autoExpandParent={autoExpandParent}
        treeData={treeData}
      />
    </div>
  );
};

export default App;
```

**连接线**

```
import React, { useState } from 'react';
import { CarryOutOutlined, CheckOutlined, FormOutlined } from '@ant-
design/icons';
import { Select, Switch, Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    icon: <CarryOutOutlined />,
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        icon: <CarryOutOutlined />,
        children: [
          { title: 'leaf', key: '0-0-0-0', icon: <CarryOutOutlined /> },
          {
            title: (
              <>
                <div>multiple line title</div>
                <div>multiple line title</div>
              </>
```

```
            ),
            key: '0-0-0-1',
            icon: <CarryOutOutlined />,
          },
          { title: 'leaf', key: '0-0-0-2', icon: <CarryOutOutlined /> },
        ],
      },
      {
        title: 'parent 1-1',
        key: '0-0-1',
        icon: <CarryOutOutlined />,
        children: [{ title: 'leaf', key: '0-0-1-0', icon: <CarryOutOutlined
/> }],
      },
      {
        title: 'parent 1-2',
        key: '0-0-2',
        icon: <CarryOutOutlined />,
        children: [
          { title: 'leaf', key: '0-0-2-0', icon: <CarryOutOutlined /> },
          {
            title: 'leaf',
            key: '0-0-2-1',
            icon: <CarryOutOutlined />,
            switcherIcon: <FormOutlined />,
          },
        ],
      },
    ],
  },
  {
    title: 'parent 2',
    key: '0-1',
    icon: <CarryOutOutlined />,
    children: [
      {
        title: 'parent 2-0',
        key: '0-1-0',
        icon: <CarryOutOutlined />,
        children: [
          { title: 'leaf', key: '0-1-0-0', icon: <CarryOutOutlined /> },
          { title: 'leaf', key: '0-1-0-1', icon: <CarryOutOutlined /> },
        ],
      },
    ],
  },
```

```
];

const App: React.FC = () => {
  const [showLine, setShowLine] = useState<boolean>(true);
  const [showIcon, setShowIcon] = useState<boolean>(false);
  const [showLeafIcon, setShowLeafIcon] = useState<React.ReactNode>(true);

  const onSelect = (selectedKeys: React.Key[], info: any) => {
    console.log('selected', selectedKeys, info);
  };

  const handleLeafIconChange = (value: 'true' | 'false' | 'custom') => {
    if (value === 'custom') {
      return setShowLeafIcon(<CheckOutlined />);
    }

    if (value === 'true') {
      return setShowLeafIcon(true);
    }

    return setShowLeafIcon(false);
  };

  return (
    <div>
      <div style={{ marginBottom: 16 }}>
        showLine: <Switch checked={!!showLine} onChange={setShowLine} />
        <br />
        <br />
        showIcon: <Switch checked={showIcon} onChange={setShowIcon} />
        <br />
        <br />
        showLeafIcon:{' '}
        <Select defaultValue="true" onChange={handleLeafIconChange}>
          <Select.Option value="true">True</Select.Option>
          <Select.Option value="false">False</Select.Option>
          <Select.Option value="custom">Custom icon</Select.Option>
        </Select>
      </div>
      <Tree
        showLine={showLine ? { showLeafIcon } : false}
        showIcon={showIcon}
        defaultExpandedKeys={['0-0-0']}
        onSelect={onSelect}
        treeData={treeData}
      />
```

```
    </div>
  );
};

export default App;
```

**自定义图标**

```
import React from 'react';
import {
  DownOutlined,
  FrownFilled,
  FrownOutlined,
  MehOutlined,
  SmileOutlined,
} from '@ant-design/icons';
import { Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    icon: <SmileOutlined />,
    children: [
      {
        title: 'leaf',
        key: '0-0-0',
        icon: <MehOutlined />,
      },
      {
        title: 'leaf',
        key: '0-0-1',
        icon: ({ selected }) => (selected ? <FrownFilled /> :
<FrownOutlined />),
      },
    ],
  },
];

const App: React.FC = () => (
  <Tree
    showIcon
    defaultExpandAll
    defaultSelectedKeys={['0-0-0']}
    switcherIcon={<DownOutlined />}
```

```
    treeData={treeData}
  />
);

export default App;
```

**目录**

```
import React from 'react';
import { Tree } from 'antd';
import type { GetProps, TreeDataNode } from 'antd';

type DirectoryTreeProps = GetProps<typeof Tree.DirectoryTree>;

const { DirectoryTree } = Tree;

const treeData: TreeDataNode[] = [
  {
    title: 'parent 0',
    key: '0-0',
    children: [
      { title: 'leaf 0-0', key: '0-0-0', isLeaf: true },
      { title: 'leaf 0-1', key: '0-0-1', isLeaf: true },
    ],
  },
  {
    title: 'parent 1',
    key: '0-1',
    children: [
      { title: 'leaf 1-0', key: '0-1-0', isLeaf: true },
      { title: 'leaf 1-1', key: '0-1-1', isLeaf: true },
    ],
  },
];

const App: React.FC = () => {
  const onSelect: DirectoryTreeProps['onSelect'] = (keys, info) => {
    console.log('Trigger Select', keys, info);
  };

  const onExpand: DirectoryTreeProps['onExpand'] = (keys, info) => {
    console.log('Trigger Expand', keys, info);
  };

  return (
    <DirectoryTree
```

```
      multiple
      draggable
      defaultExpandAll
      onSelect={onSelect}
      onExpand={onExpand}
      treeData={treeData}
    />
  );
};

export default App;
```

**目录 Debug**

Debug

```
import React from 'react';
import { Flex, Tree } from 'antd';
import type { GetProps, TreeDataNode } from 'antd';

const { DirectoryTree } = Tree;

const treeData: TreeDataNode[] = [
  {
    title: 'parent 0',
    key: '0-0',
    children: [
      { title: 'leaf 0-0', key: '0-0-0', isLeaf: true, disabled: true },
      { title: 'leaf 0-1', key: '0-0-1', isLeaf: true, disableCheckbox:
true },
    ],
  },
  {
    title: 'parent 1',
    key: '0-1',
    children: [
      { title: 'leaf 1-0', key: '0-1-0', isLeaf: true },
      { title: 'leaf 1-1', key: '0-1-1', isLeaf: true },
    ],
  },
];

const sharedProps: GetProps<typeof DirectoryTree> = {
  treeData,
  defaultExpandAll: true,
  onSelect: (keys, info) => {
```

```
    console.log('Trigger Select', keys, info);
  },
  onExpand: (keys, info) => {
    console.log('Trigger Expand', keys, info);
  },
};

const DemoOne = () => <DirectoryTree draggable defaultSelectedKeys={['0-0-
0']} />;

const DemoTwo = () => <DirectoryTree {...sharedProps} checkable
defaultSelectedKeys={['0-1-0']} />;

const DemoThree = () => (
  <DirectoryTree {...sharedProps} draggable checkable defaultSelectedKeys=
{['0-1']} />
);

const BasicDemo = () => <DirectoryTree {...sharedProps} multiple treeData=
{treeData} />;

const NormalDemo = () => <Tree {...sharedProps} defaultSelectedKeys={['0-
1']} />;

const NormalCheckDemo = () => (
  <Tree {...sharedProps} checkable defaultSelectedKeys={['0-1', '0-0-0',
'0-0-1', '0-1-1']} />
);

const NormalDragDemo = () => <Tree {...sharedProps} draggable
defaultSelectedKeys={['0-1-0']} />;

const App = () => (
  <Flex wrap gap="large">
    <DemoOne />
    <DemoTwo />
    <DemoThree />
    <BasicDemo />
    <NormalDemo />
    <NormalCheckDemo />
    <NormalDragDemo />
  </Flex>
);

export default App;
```

**自定义展开/折叠图标**

```
import React from 'react';
import { DownOutlined } from '@ant-design/icons';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        children: [
          {
            title: 'leaf',
            key: '0-0-0-0',
          },
          {
            title: 'leaf',
            key: '0-0-0-1',
          },
          {
            title: 'leaf',
            key: '0-0-0-2',
          },
        ],
      },
      {
        title: 'parent 1-1',
        key: '0-0-1',
        children: [
          {
            title: 'leaf',
            key: '0-0-1-0',
          },
        ],
      },
      {
        title: 'parent 1-2',
        key: '0-0-2',
        children: [
          {
            title: 'leaf',
```

```
            key: '0-0-2-0',
          },
          {
            title: 'leaf',
            key: '0-0-2-1',
          },
        ],
      },
    ],
  },
];

const App: React.FC = () => {
  const onSelect: TreeProps['onSelect'] = (selectedKeys, info) => {
    console.log('selected', selectedKeys, info);
  };

  return (
    <Tree
      showLine
      switcherIcon={<DownOutlined />}
      defaultExpandedKeys={['0-0-0']}
      onSelect={onSelect}
      treeData={treeData}
    />
  );
};

export default App;
```

**虚拟滚动**

```
import React from 'react';
import { Tooltip, Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const dig = (path = '0', level = 3) => {
  const list = [];
  for (let i = 0; i < 10; i += 1) {
    const key = `${path}-${i}`;
    const treeNode: TreeDataNode = {
      title: key,
      key,
    };

    if (level > 0) {
```

```
      treeNode.children = dig(key, level - 1);
    }

    list.push(treeNode);
  }
  return list;
};

const treeData = dig();

const MemoTooltip = Tooltip || React.memo(Tooltip);

const App: React.FC = () => (
  <Tree
    treeData={treeData}
    height={233}
    defaultExpandAll
    titleRender={(item) => {
      const title = item.title as React.ReactNode;
      return <MemoTooltip title={title}>{title}</MemoTooltip>;
    }}
  />
);

export default App;
```

**Drag Debug**

Debug

```
import React from 'react';
import { CarryOutOutlined } from '@ant-design/icons';
import type { TreeDataNode, TreeProps } from 'antd';
import { Switch, Tree } from 'antd';

const x = 3;
const y = 2;
const z = 1;
const data: TreeDataNode[] = [];

const generateData = (_level: number, preKey = '0', tns = data):
TreeDataNode[] | undefined => {
  const children: string[] = [];
  for (let i = 0; i < x; i++) {
    const key = `${preKey}-${i}`;
    tns.push({ title: key, key, icon: <CarryOutOutlined /> });
```

```
    if (i < y) {
      children.push(key);
    }
  }
  if (_level < 0) {
    return tns;
  }
  const level = _level - 1;
  children.forEach((key, index) => {
    tns[index].children = [];
    return generateData(level, key, tns[index].children);
  });
};

generateData(z);

const App: React.FC = () => {
  const [gData, setGData] = React.useState<TreeDataNode[]>(data);
  const [showLine, setShowLine] = React.useState<any>(true);
  const [showIcon, setShowIcon] = React.useState<boolean>(true);
  const [showLeafIcon, setShowLeafIcon] = React.useState<boolean>(true);
  const [expandedKeys, setExpandedKeys] = React.useState<React.Key[]>(['0-
0', '0-0-0', '0-0-0-0']);

  const onDragEnter: TreeProps['onDragEnter'] = (info) => {
    console.log(info);
    // expandedKeys, set it when controlled is needed
    setExpandedKeys(info.expandedKeys);
  };

  const onDrop: TreeProps['onDrop'] = (info) => {
    console.log(info);
    const dropKey = info.node.key as number;
    const dragKey = info.dragNode.key as number;
    const dropPos = info.node.pos.split('-');
    const dropPosition = info.dropPosition - Number(dropPos[dropPos.length
- 1]);

    const loop = (
      data: TreeDataNode[],
      key: number,
      callback: (item: TreeDataNode, index: number, err: TreeDataNode[]) =>
void,
    ): void => {
      for (let i = 0; i < data.length; i++) {
        if (data[i].key === key) {
```

```
          return callback(data[i], i, data);
        }
        if (data[i].children) {
          loop(data[i].children!, key, callback);
        }
      }
    };

    const data = [...gData];

    // Find dragObject
    let dragObj: TreeDataNode;
    loop(data, dragKey, (item, index, arr) => {
      arr.splice(index, 1);
      dragObj = item;
    });

    if (!info.dropToGap) {
      // Drop on the content
      loop(data, dropKey, (item) => {
        item.children = item.children || [];
        // where to insert. New item was inserted to the end of the array
in this example, but can be anywhere
        item.children.push(dragObj);
      });
    } else if (
      ((info.node as any).props.children || []).length > 0 && // Has
children
      (info.node as any).props.expanded && // Is expanded
      dropPosition === 1 // On the bottom gap
    ) {
      loop(data, dropKey, (item) => {
        item.children = item.children || [];
        // where to insert. New item was inserted to the start of the array
in this example, but can be anywhere
        item.children.unshift(dragObj);
      });
    } else {
      let ar: TreeDataNode[];
      let i: number;
      loop(data, dropKey, (_, index, arr) => {
        ar = arr;
        i = index;
      });
      if (dropPosition === -1) {
        ar!.splice(i!, 0, dragObj!);
```

```
      } else {
        ar!.splice(i! + 1, 0, dragObj!);
      }
    }
    setGData(data);
  };

  const innerSetShowLine = (showLine: boolean) => {
    if (showLine) {
      if (showLeafIcon) {
        setShowLine({ showLeafIcon: true });
      } else {
        setShowLine(true);
      }
    } else {
      setShowLine(false);
    }
  };

  const innerSetShowLeafIcon = (showLeafIcon: boolean) => {
    setShowLeafIcon(showLeafIcon);
    setShowLine({ showLeafIcon });
  };

  return (
    <>
      <div style={{ marginBottom: 16 }}>
        showLine: <Switch checked={showLine} onChange={innerSetShowLine} />
        <br />
        <br />
        showIcon: <Switch checked={showIcon} onChange={() =>
setShowIcon(showIcon)} />
        <br />
        <br />
        showLeafIcon: <Switch checked={showLeafIcon} onChange=
{innerSetShowLeafIcon} />
      </div>
      <Tree
        showLine={showLine}
        showIcon={showIcon}
        className="draggable-tree"
        defaultExpandedKeys={expandedKeys}
        draggable
        blockNode
        onDragEnter={onDragEnter}
        onDrop={onDrop}
```

```
        treeData={gData}
      />
    </>
  );
};

export default App;
```

**大数据**

Debug

```
import React from 'react';
import { Tree } from 'antd';
import type { TreeDataNode } from 'antd';

const treeData: TreeDataNode[] = [];

for (let i = 0; i < 100; i += 1) {
  const children: TreeDataNode[] = [];

  for (let j = 0; j < 100; j += 1) {
    children.push({
      title: `child ${i}-${j}`,
      key: `l-${i}-${j}`,
    });
  }

  treeData.push({
    title: `parent ${i}`,
    key: `l-${i}`,
    children,
  });
}

const App: React.FC = () => <Tree defaultExpandAll height={400} treeData=
{treeData} />;

export default App;
```

**占据整行**

```
import React from 'react';
import { Tree } from 'antd';
import type { TreeDataNode } from 'antd';
```

```
const treeData: TreeDataNode[] = [
  {
    title: 'parent',
    key: '0',
    children: [
      {
        title: 'child 1',
        key: '0-0',
        disabled: true,
      },
      {
        title: 'child 2',
        key: '0-1',
        disableCheckbox: true,
      },
    ],
  },
];

const App: React.FC = () => (
  <Tree checkable defaultSelectedKeys={['0-1']} defaultExpandAll treeData=
{treeData} blockNode />
);

export default App;
```

**组件 Token**

Debug

```
import React from 'react';
import { ConfigProvider, Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        disabled: true,
        children: [
          {
            title: 'leaf',
```

```
            key: '0-0-0-0',
            disableCheckbox: true,
          },
          {
            title: 'leaf',
            key: '0-0-0-1',
          },
        ],
      },
      {
        title: 'parent 1-1',
        key: '0-0-1',
        children: [{ title: <span style={{ color: '#1677ff' }}>sss</span>,
key: '0-0-1-0' }],
      },
    ],
  },
];

const App: React.FC = () => {
  const onSelect: TreeProps['onSelect'] = (selectedKeys, info) => {
    console.log('selected', selectedKeys, info);
  };

  const onCheck: TreeProps['onCheck'] = (checkedKeys, info) => {
    console.log('onCheck', checkedKeys, info);
  };

  return (
    <ConfigProvider
      theme={{
        components: {
          Tree: {
            nodeHoverBg: '#fff2f0',
            nodeHoverColor: '#1677ff',
            nodeSelectedBg: '#ffa39e',
            nodeSelectedColor: '#fff',
            indentSize: 80,
          },
        },
      }}
    >
      <Tree
        checkable
        defaultExpandedKeys={['0-0-0', '0-0-1']}
        defaultSelectedKeys={['0-0-1']}
```

```
        defaultCheckedKeys={['0-0-0', '0-0-1']}
        onSelect={onSelect}
        onCheck={onCheck}
        treeData={treeData}
      />
    </ConfigProvider>
  );
};

export default App;
```

### 多行

Debug

```
import React from 'react';
import { Tree } from 'antd';
import type { TreeDataNode, TreeProps } from 'antd';

const treeData: TreeDataNode[] = [
  {
    title: 'parent 1',
    key: '0-0',
    children: [
      {
        title: 'parent 1-0',
        key: '0-0-0',
        disabled: true,
        children: [
          {
            title: 'This is a very very very very long text',
            key: '0-0-0-0',
            disableCheckbox: true,
          },
          {
            title: 'This is also a very very very very very very long text',
            key: '0-0-0-1',
          },
        ],
      },
      {
        title: 'parent 1-1',
        key: '0-0-1',
        children: [{ title: <span style={{ color: '#1677ff' }}>sss</span>,
key: '0-0-1-0' }],
      },
```

```
    ],
  },
];

const App: React.FC = () => {
  const onSelect: TreeProps['onSelect'] = (selectedKeys, info) => {
    console.log('selected', selectedKeys, info);
  };

  const onCheck: TreeProps['onCheck'] = (checkedKeys, info) => {
    console.log('onCheck', checkedKeys, info);
  };

  return (
    <Tree
      checkable
      defaultExpandedKeys={['0-0-0', '0-0-1']}
      defaultSelectedKeys={['0-0-1']}
      defaultCheckedKeys={['0-0-0', '0-0-1']}
      onSelect={onSelect}
      onCheck={onCheck}
      treeData={treeData}
      style={{ width: 200 }}
    />
  );
};

export default App;
```

## API

通用属性参考：通用属性

### Tree props

| 参数 | 说明 | 类型 | 默认值 | 版本 |
| --- | --- | --- | --- | --- |
| allowDrop | 是否允许拖拽时放置在该节点 | ({ dropNode, dropPosition }) => boolean | - | |
| autoExpandParent | 是否自动展开父节点 | boolean | false | |
| blockNode | 是否节点占据一行 | boolean | false | |
| checkable | 节点前添加 Checkbox 复选框 | boolean | false | |

| | | | | |
|---|---|---|---|---|
| checkedKeys | （受控）选中复选框的树节点（注意：父子节点有关联，如果传入父节点 key，则子节点自动选中；相应当子节点 key 都传入，父节点也自动选中。当设置 `checkable` 和 `checkStrictly`，它是一个有 `checked`和 `halfChecked`属性的对象，并且父子节点的选中与否不再关联 | string[] \| {checked: string[], halfChecked: string[]} | [] | |
| checkStrictly | checkable 状态下节点选择完全受控（父子节点选中状态不再关联） | boolean | false | |
| defaultCheckedKeys | 默认选中复选框的树节点 | string[] | [] | |
| defaultExpandAll | 默认展开所有树节点 | boolean | false | |
| defaultExpandedKeys | 默认展开指定的树节点 | string[] | [] | |
| defaultExpandParent | 默认展开父节点 | boolean | true | |
| defaultSelectedKeys | 默认选中的树节点 | string[] | [] | |
| disabled | 将树禁用 | boolean | false | |
| draggable | 设置节点可拖拽，可以通过 `icon: false` 关闭拖拽提示图标 | boolean \| ((node: DataNode) => boolean) \| { icon?: React.ReactNode \| false, nodeDraggable?: (node: DataNode) => boolean } | false | `config` 4.17.0 |
| expandedKeys | （受控）展开指定的树节点 | string[] | [] | |

| | | | | |
|---|---|---|---|---|
| fieldNames | 自定义节点 title、key、children 的字段 | object | { title: `title`, key: key, children: `children` } | 4.17.0 |
| filterTreeNode | 按需筛选树节点（高亮），返回 true | function(node) | - | |
| height | 设置虚拟滚动容器高度，设置后内部节点不再支持横向滚动 | number | - | |
| icon | 在标题之前插入自定义图标。需要设置 `showIcon` 为 true | ReactNode \| (props) => ReactNode | - | |
| loadData | 异步加载数据 | function(node) | - | |
| loadedKeys | （受控）已经加载的节点，需要配合 `loadData` 使用 | string[] | [] | |
| multiple | 支持点选多个节点（节点本身） | boolean | false | |
| rootStyle | 添加在 Tree 最外层的 style | CSSProperties | - | 4.20.0 |
| selectable | 是否可选中 | boolean | true | |
| selectedKeys | （受控）设置选中的树节点，多选需设置 `multiple` 为 true | string[] | - | |
| showIcon | 控制是否展示 `icon` 节点，没有默认样式 | boolean | false | |
| showLine | 是否展示连接线 | boolean \| { showLeafIcon: ReactNode \| ((props: AntTreeNodeProps) => ReactNode) } | false | |

| | | | | |
|---|---|---|---|---|
| switcherIcon | 自定义树节点的展开/折叠图标（带有默认 rotate 角度样式） | ReactNode \| ((props: AntTreeNodeProps) => ReactNode) | - | renderI 4.20.0 |
| switcherLoadingIcon | 自定义树节点的加载图标 | ReactNode | - | 5.20.0 |
| titleRender | 自定义渲染节点 | (nodeData) => ReactNode | - | 4.5.0 |
| treeData | treeNodes 数据，如果设置则不需要手动构造 TreeNode 节点（key 在整个树范围内唯一） | array<{key, title, children, [disabled, selectable]}> | - | |
| virtual | 设置 false 时关闭虚拟滚动 | boolean | true | 4.1.0 |
| onCheck | 点击复选框触发 | function(checkedKeys, e:{checked: boolean, checkedNodes, node, event, halfCheckedKeys}) | - | |
| onDragEnd | dragend 触发时调用 | function({event, node}) | - | |
| onDragEnter | dragenter 触发时调用 | function({event, node, expandedKeys}) | - | |
| onDragLeave | dragleave 触发时调用 | function({event, node}) | - | |
| onDragOver | dragover 触发时调用 | function({event, node}) | - | |
| onDragStart | 开始拖拽时调用 | function({event, node}) | - | |
| onDrop | drop 触发时调用 | function({event, node, dragNode, dragNodesKeys}) | - | |
| onExpand | 展开/收起节点时触发 | function(expandedKeys, {expanded: boolean, node}) | - | |

| | | | | |
|---|---|---|---|---|
| onLoad | 节点加载完毕时触发 | function(loadedKeys, {event, node}) | - | |
| onRightClick | 响应右键点击 | function({event, node}) | - | |
| onSelect | 点击树节点触发 | function(selectedKeys, e:{selected: boolean, selectedNodes, node, event}) | - | |

**TreeNode props**

| 参数 | 说明 | 类型 | 默认值 | |
|---|---|---|---|---|
| checkable | 当树为 checkable 时，设置独立节点是否展示 Checkbox | boolean | - | |
| disableCheckbox | 禁掉 checkbox | boolean | false | |
| disabled | 禁掉响应 | boolean | false | |
| icon | 自定义图标。可接收组件，props 为当前节点 props | ReactNode \| (props) => ReactNode | - | |
| isLeaf | 设置为叶子节点 (设置了 `loadData` 时有效)。为 `false` 时会强制将其作为父节点 | boolean | - | |
| key | 被树的 (default)ExpandedKeys / (default)CheckedKeys / (default)SelectedKeys 属性所用。注意：整个树范围内的所有节点的 key 值不能重复！ | string | (内部计算出的节点位置) | |
| selectable | 设置节点是否可被选中 | boolean | true | |
| title | 标题 | ReactNode | --- | |

**DirectoryTree props**

| 参数 | 说明 | 类型 | 默认值 |
|---|---|---|---|
| expandAction | 目录展开逻辑，可选：false \| `click` \| `doubleClick` | string \| boolean | `click` |

## 注意

在 `3.4.0` 之前：树节点可以有很多，但在设置 `checkable` 时，将会花费更多的计算时间，因此我们缓存了一些计算结果（ `this.treeNodesStates` ）来复用，避免多次重复计算，以此提高性能。但这也带来了一些限制，当你异步加载树节点时，你需要这样渲染树：

```
{
  this.state.treeData.length ? (
    <Tree>
      {this.state.treeData.map((data) => (
        <TreeNode />
      ))}
    </Tree>
  ) : (
    'loading tree'
  );
}
```

**Tree 方法**

| 名称 | 说明 |
| --- | --- |
| scrollTo({ key: string \| number; align?: 'top' \| 'bottom' \| 'auto'; offset?: number }) | 虚拟滚动下，滚动到指定 key 条目 |

## 主题变量（Design Token）

## FAQ

### defaultExpandAll 在异步加载数据时为何不生效?

`default` 前缀属性只有在初始化时生效，因而异步加载数据时 `defaultExpandAll` 已经执行完成。你可以通过受控 `expandedKeys` 或者在数据加载完成后渲染 Tree 来实现全部展开。

### 虚拟滚动的限制

虚拟滚动通过在仅渲染可视区域的元素来提升渲染性能。但是同时由于不会渲染所有节点，所以无法自动拓转横向宽度（比如超长 `title` 的横向滚动条）。

### `disabled` 节点在树中的关系是什么?

Tree 通过传导方式进行数据变更。无论是展开还是勾选，它都会从变更的节点开始向上、向下传导变化，直到遍历的当前节点是 `disabled` 时停止。因而如果控制的节点本身为 `disabled` 时，那么它只会修改本身而不会影响其他节点。举例来说，一个父节点包含 3 个子节点，其中一个为 `disabled` 状态。那么勾选父节点，只会影响其余两个子节点变成勾选状态。勾选两个子节点后，无论 `disabled` 节点什么状态，父节点都会变成勾选状态。

这种传导终止的方式是为了防止通过勾选子节点使得 `disabled` 父节点变成勾选状态，而用户无法直接勾选 `disabled` 父节点更改其状态导致的交互矛盾。如果你有着自己的传导需求，可以通过 `checkStrictly` 自定义勾选逻辑。