何时使用 {#when-to-use}

- 需要在多个可选项中进行多选时。
- 比起 Select 和 TreeSelect，穿梭框占据更大的空间，可以展示可选项的更多信息。

穿梭选择框用直观的方式在两栏中移动元素，完成选择行为。

选择一个或以上的选项后，点击对应的方向键，可以把选中的选项移动到另一栏。其中，左边一栏为 `source`，右边一栏为 `target`，API 的设计也反映了这两个概念。

> *注意：穿梭框组件只支持受控使用，不支持非受控模式。*

代码演示

基本用法

```
import React, { useState } from 'react';
import { Transfer } from 'antd';
import type { TransferProps } from 'antd';

interface RecordType {
  key: string;
  title: string;
  description: string;
}

const mockData = Array.from({ length: 20 }).map<RecordType>((_, i) => ({
  key: i.toString(),
  title: `content${i + 1}`,
  description: `description of content${i + 1}`,
}));

const initialTargetKeys = mockData.filter((item) => Number(item.key) >
10).map((item) => item.key);

const App: React.FC = () => {
  const [targetKeys, setTargetKeys] = useState<TransferProps['targetKeys']>
(initialTargetKeys);
  const [selectedKeys, setSelectedKeys] =
useState<TransferProps['targetKeys']>([]);

  const onChange: TransferProps['onChange'] = (nextTargetKeys, direction,
moveKeys) => {
    console.log('targetKeys:', nextTargetKeys);
    console.log('direction:', direction);
    console.log('moveKeys:', moveKeys);
    setTargetKeys(nextTargetKeys);
  };
  const onSelectChange: TransferProps['onSelectChange'] = (
    sourceSelectedKeys,
    targetSelectedKeys,
  ) => {
    console.log('sourceSelectedKeys:', sourceSelectedKeys);
    console.log('targetSelectedKeys:', targetSelectedKeys);
    setSelectedKeys([...sourceSelectedKeys, ...targetSelectedKeys]);
  };

  const onScroll: TransferProps['onScroll'] = (direction, e) => {
    console.log('direction:', direction);
    console.log('target:', e.target);
  };

  return (
    <Transfer
      dataSource={mockData}
      titles={['Source', 'Target']}
      targetKeys={targetKeys}
      selectedKeys={selectedKeys}
      onChange={onChange}
      onSelectChange={onSelectChange}
      onScroll={onScroll}
      render={(item) => item.title}
    />
  );
};

export default App;
```

单向样式

```
import React, { useState } from 'react';
import { Switch, Transfer } from 'antd';
import type { TransferProps } from 'antd';

interface RecordType {
  key: string;
  title: string;
  description: string;
  disabled: boolean;
}

const mockData = Array.from({ length: 20 }).map<RecordType>((_, i) => ({
  key: i.toString(),
  title: `content${i + 1}`,
  description: `description of content${i + 1}`,
  disabled: i % 3 < 1,
}));

const oriTargetKeys = mockData.filter((item) => Number(item.key) % 3 >
1).map((item) => item.key);

const App: React.FC = () => {
  const [targetKeys, setTargetKeys] = useState<React.Key[]>(oriTargetKeys);
  const [selectedKeys, setSelectedKeys] = useState<React.Key[]>([]);
  const [disabled, setDisabled] = useState(false);

  const handleChange: TransferProps['onChange'] = (newTargetKeys,
direction, moveKeys) => {
    setTargetKeys(newTargetKeys);

    console.log('targetKeys: ', newTargetKeys);
    console.log('direction: ', direction);
    console.log('moveKeys: ', moveKeys);
  };

  const handleSelectChange: TransferProps['onSelectChange'] = (
    sourceSelectedKeys,
    targetSelectedKeys,
  ) => {
    setSelectedKeys([...sourceSelectedKeys, ...targetSelectedKeys]);

    console.log('sourceSelectedKeys: ', sourceSelectedKeys);
    console.log('targetSelectedKeys: ', targetSelectedKeys);
  };

  const handleScroll: TransferProps['onScroll'] = (direction, e) => {
    console.log('direction:', direction);
    console.log('target:', e.target);
  };

  const handleDisable = (checked: boolean) => {
    setDisabled(checked);
  };

  return (
    <>
      <Transfer
        dataSource={mockData}
        titles={['Source', 'Target']}
        targetKeys={targetKeys}
        selectedKeys={selectedKeys}
        onChange={handleChange}
        onSelectChange={handleSelectChange}
        onScroll={handleScroll}
        render={(item) => item.title}
        disabled={disabled}
        oneWay
        style={{ marginBottom: 16 }}
      />
      <Switch
        unCheckedChildren="disabled"
        checkedChildren="disabled"
        checked={disabled}
        onChange={handleDisable}
      />
    </>
  );
};

export default App;
```

带搜索框

```
import React, { useEffect, useState } from 'react';
import { Transfer } from 'antd';
import type { TransferProps } from 'antd';

interface RecordType {
  key: string;
  title: string;
  description: string;
  chosen: boolean;
}

const App: React.FC = () => {
  const [mockData, setMockData] = useState<RecordType[]>([]);
  const [targetKeys, setTargetKeys] = useState<TransferProps['targetKeys']>
([]);

  const getMock = () => {
    const tempTargetKeys = [];
    const tempMockData = [];
    for (let i = 0; i < 20; i++) {
      const data = {
        key: i.toString(),
        title: `content${i + 1}`,
        description: `description of content${i + 1}`,
        chosen: i % 2 === 0,
      };
      if (data.chosen) {
        tempTargetKeys.push(data.key);
      }
      tempMockData.push(data);
    }
    setMockData(tempMockData);
    setTargetKeys(tempTargetKeys);
  };

  useEffect(() => {
    getMock();
  }, []);

  const filterOption = (inputValue: string, option: RecordType) =>
    option.description.indexOf(inputValue) > -1;

  const handleChange: TransferProps['onChange'] = (newTargetKeys) => {
    setTargetKeys(newTargetKeys);
  };

  const handleSearch: TransferProps['onSearch'] = (dir, value) => {
    console.log('search:', dir, value);
  };

  return (
    <Transfer
      dataSource={mockData}
      showSearch
      filterOption={filterOption}
      targetKeys={targetKeys}
      onChange={handleChange}
      onSearch={handleSearch}
      render={(item) => item.title}
    />
  );
};

export default App;
```

高级用法

```tsx
import React, { useEffect, useState } from 'react';
import { Button, Transfer } from 'antd';
import type { TransferProps } from 'antd';

interface RecordType {
  key: string;
  title: string;
  description: string;
  chosen: boolean;
}

const App: React.FC = () => {
  const [mockData, setMockData] = useState<RecordType[]>([]);
  const [targetKeys, setTargetKeys] = useState<TransferProps['targetKeys']>
([]);

  const getMock = () => {
    const tempTargetKeys = [];
    const tempMockData = [];
    for (let i = 0; i < 20; i++) {
      const data = {
        key: i.toString(),
        title: `content${i + 1}`,
        description: `description of content${i + 1}`,
        chosen: i % 2 === 0,
      };
      if (data.chosen) {
        tempTargetKeys.push(data.key);
      }
      tempMockData.push(data);
    }
    setMockData(tempMockData);
    setTargetKeys(tempTargetKeys);
  };

  useEffect(() => {
    getMock();
  }, []);

  const handleChange: TransferProps['onChange'] = (newTargetKeys) => {
    setTargetKeys(newTargetKeys);
  };

  const renderFooter: TransferProps['footer'] = (_, info) => {
    if (info?.direction === 'left') {
      return (
        <Button
          size="small"
          style={{ display: 'flex', margin: 8, marginInlineEnd: 'auto' }}
          onClick={getMock}
        >
          Left button reload
        </Button>
      );
    }
    return (
      <Button
        size="small"
        style={{ display: 'flex', margin: 8, marginInlineStart: 'auto' }}
        onClick={getMock}
      >
        Right button reload
      </Button>
    );
  };

  return (
    <Transfer
      dataSource={mockData}
      showSearch
      listStyle={{
        width: 250,
        height: 300,
      }}
      operations={['to right', 'to left']}
      targetKeys={targetKeys}
      onChange={handleChange}
      render={(item) => `${item.title}-${item.description}`}
      footer={renderFooter}
    />
  );
};

export default App;
```

自定义渲染行数据

```
import React, { useEffect, useState } from 'react';
import { Transfer } from 'antd';
import type { TransferProps } from 'antd';

interface RecordType {
  key: string;
  title: string;
  description: string;
  chosen: boolean;
}

const App: React.FC = () => {
  const [mockData, setMockData] = useState<RecordType[]>([]);
  const [targetKeys, setTargetKeys] = useState<React.Key[]>([]);

  const getMock = () => {
    const tempTargetKeys = [];
    const tempMockData = [];
    for (let i = 0; i < 20; i++) {
      const data = {
        key: i.toString(),
        title: `content${i + 1}`,
        description: `description of content${i + 1}`,
        chosen: i % 2 === 0,
      };
      if (data.chosen) {
        tempTargetKeys.push(data.key);
      }
      tempMockData.push(data);
    }
    setMockData(tempMockData);
    setTargetKeys(tempTargetKeys);
  };

  useEffect(() => {
    getMock();
  }, []);

  const handleChange: TransferProps['onChange'] = (newTargetKeys,
direction, moveKeys) => {
    console.log(newTargetKeys, direction, moveKeys);
    setTargetKeys(newTargetKeys);
  };

  const renderItem = (item: RecordType) => {
    const customLabel = (
      <span className="custom-item">
        {item.title} - {item.description}
      </span>
    );
    return {
      label: customLabel, // for displayed item
      value: item.title, // for title and filter matching
    };
  };

  return (
    <Transfer
      dataSource={mockData}
      listStyle={{
        width: 300,
        height: 300,
      }}
      targetKeys={targetKeys}
      onChange={handleChange}
      render={renderItem}
    />
  );
};

export default App;
```

分页

```
import React, { useEffect, useState } from 'react';
import { Switch, Transfer } from 'antd';
import type { TransferProps } from 'antd';

interface RecordType {
  key: string;
  title: string;
  description: string;
  chosen: boolean;
}

const App: React.FC = () => {
  const [oneWay, setOneWay] = useState(false);
  const [mockData, setMockData] = useState<RecordType[]>([]);
  const [targetKeys, setTargetKeys] = useState<React.Key[]>([]);

  useEffect(() => {
    const newTargetKeys = [];
    const newMockData = [];
    for (let i = 0; i < 2000; i++) {
      const data = {
        key: i.toString(),
        title: `content${i + 1}`,
        description: `description of content${i + 1}`,
        chosen: i % 2 === 0,
      };
      if (data.chosen) {
        newTargetKeys.push(data.key);
      }
      newMockData.push(data);
    }

    setTargetKeys(newTargetKeys);
    setMockData(newMockData);
  }, []);

  const onChange: TransferProps['onChange'] = (newTargetKeys, direction,
moveKeys) => {
    console.log(newTargetKeys, direction, moveKeys);
    setTargetKeys(newTargetKeys);
  };

  return (
    <>
      <Transfer
        dataSource={mockData}
        targetKeys={targetKeys}
        onChange={onChange}
        render={(item) => item.title}
        oneWay={oneWay}
        pagination
      />
      <br />
      <Switch
        unCheckedChildren="one way"
        checkedChildren="one way"
        checked={oneWay}
        onChange={setOneWay}
      />
    </>
  );
};

export default App;
```

表格穿梭框

```
import React, { useState } from 'react';
import { Flex, Switch, Table, Tag, Transfer } from 'antd';
import type { GetProp, TableColumnsType, TableProps, TransferProps } from
'antd';

type TransferItem = GetProp<TransferProps, 'dataSource'>[number];
type TableRowSelection<T extends object> = TableProps<T>['rowSelection'];

interface DataType {
  key: string;
  title: string;
  description: string;
  tag: string;
}

interface TableTransferProps extends TransferProps<TransferItem> {
  dataSource: DataType[];
  leftColumns: TableColumnsType<DataType>;
  rightColumns: TableColumnsType<DataType>;
}

// Customize Table Transfer
const TableTransfer: React.FC<TableTransferProps> = (props) => {
  const { leftColumns, rightColumns, ...restProps } = props;
  return (
    <Transfer style={{ width: '100%' }} {...restProps}>
      {({
        direction,
        filteredItems,
        onItemSelect,
        onItemSelectAll,
        selectedKeys: listSelectedKeys,
        disabled: listDisabled,
      }) => {
        const columns = direction === 'left' ? leftColumns : rightColumns;
        const rowSelection: TableRowSelection<TransferItem> = {
          getCheckboxProps: () => ({ disabled: listDisabled }),
          onChange(selectedRowKeys) {
            onItemSelectAll(selectedRowKeys, 'replace');
          },
          selectedRowKeys: listSelectedKeys,
          selections: [Table.SELECTION_ALL, Table.SELECTION_INVERT,
Table.SELECTION_NONE],
        };

        return (
          <Table
            rowSelection={rowSelection}
            columns={columns}
            dataSource={filteredItems}
            size="small"
            style={{ pointerEvents: listDisabled ? 'none' : undefined }}
            onRow={({ key, disabled: itemDisabled }) => ({
              onClick: () => {
                if (itemDisabled || listDisabled) {
                  return;
                }
                onItemSelect(key, !listSelectedKeys.includes(key));
              },
            })}
          />
        );
      }}
    </Transfer>
  );
};

const mockTags = ['cat', 'dog', 'bird'];

const mockData = Array.from({ length: 20 }).map<DataType>((_, i) => ({
  key: i.toString(),
  title: `content${i + 1}`,
  description: `description of content${i + 1}`,
  tag: mockTags[i % 3],
}));

const columns: TableColumnsType<DataType> = [
  {
    dataIndex: 'title',
    title: 'Name',
  },
  {
    dataIndex: 'tag',
    title: 'Tag',
    render: (tag: string) => (
      <Tag style={{ marginInlineEnd: 0 }} color="cyan">
        {tag.toUpperCase()}
      </Tag>
    ),
  },
  {
    dataIndex: 'description',
    title: 'Description',
  },
];

const filterOption = (input: string, item: DataType) =>
  item.title?.includes(input) || item.tag?.includes(input);

const App: React.FC = () => {
  const [targetKeys, setTargetKeys] = useState<TransferProps['targetKeys']>
([]);
  const [disabled, setDisabled] = useState(false);

  const onChange: TableTransferProps['onChange'] = (nextTargetKeys) => {
    setTargetKeys(nextTargetKeys);
  };

  const toggleDisabled = (checked: boolean) => {
    setDisabled(checked);
  };

  return (
    <Flex align="start" gap="middle" vertical>
      <TableTransfer
        dataSource={mockData}
        targetKeys={targetKeys}
        disabled={disabled}
        showSearch
        showSelectAll={false}
        onChange={onChange}
        filterOption={filterOption}
        leftColumns={columns}
        rightColumns={columns}
      />
      <Switch
        unCheckedChildren="disabled"
        checkedChildren="disabled"
        checked={disabled}
        onChange={toggleDisabled}
      />
    </Flex>
  );
};

export default App;
```

树穿梭框

```tsx
import React, { useState } from 'react';
import { theme, Transfer, Tree } from 'antd';
import type { GetProp, TransferProps, TreeDataNode } from 'antd';

type TransferItem = GetProp<TransferProps, 'dataSource'>[number];

interface TreeTransferProps {
  dataSource: TreeDataNode[];
  targetKeys: TransferProps['targetKeys'];
  onChange: TransferProps['onChange'];
}

// Customize Table Transfer
const isChecked = (selectedKeys: React.Key[], eventKey: React.Key) =>
  selectedKeys.includes(eventKey);

const generateTree = (
  treeNodes: TreeDataNode[] = [],
  checkedKeys: TreeTransferProps['targetKeys'] = [],
): TreeDataNode[] =>
  treeNodes.map(({ children, ...props }) => ({
    ...props,
    disabled: checkedKeys.includes(props.key as string),
    children: generateTree(children, checkedKeys),
  }));

const TreeTransfer: React.FC<TreeTransferProps> = ({
  dataSource,
  targetKeys = [],
  ...restProps
}) => {
  const { token } = theme.useToken();

  const transferDataSource: TransferItem[] = [];
  function flatten(list: TreeDataNode[] = []) {
    list.forEach((item) => {
      transferDataSource.push(item as TransferItem);
      flatten(item.children);
    });
  }
  flatten(dataSource);

  return (
    <Transfer
      {...restProps}
      targetKeys={targetKeys}
      dataSource={transferDataSource}
      className="tree-transfer"
      render={(item) => item.title!}
      showSelectAll={false}
    >
      {({ direction, onItemSelect, selectedKeys }) => {
        if (direction === 'left') {
          const checkedKeys = [...selectedKeys, ...targetKeys];
          return (
            <div style={{ padding: token.paddingXS }}>
              <Tree
                blockNode
                checkable
                checkStrictly
                defaultExpandAll
                checkedKeys={checkedKeys}
                treeData={generateTree(dataSource, targetKeys)}
                onCheck={(_, { node: { key } }) => {
                  onItemSelect(key as string, !isChecked(checkedKeys,
key));
                }}
                onSelect={(_, { node: { key } }) => {
                  onItemSelect(key as string, !isChecked(checkedKeys,
key));
                }}
              />
            </div>
          );
        }
      }}
    </Transfer>
  );
};

const treeData: TreeDataNode[] = [
  { key: '0-0', title: '0-0' },
  {
    key: '0-1',
    title: '0-1',
    children: [
      { key: '0-1-0', title: '0-1-0' },
      { key: '0-1-1', title: '0-1-1' },
    ],
  },
  { key: '0-2', title: '0-2' },
  { key: '0-3', title: '0-3' },
  { key: '0-4', title: '0-4' },
];

const App: React.FC = () => {
  const [targetKeys, setTargetKeys] =
useState<TreeTransferProps['targetKeys']>([]);
  const onChange: TreeTransferProps['onChange'] = (keys) => {
    setTargetKeys(keys);
  };
  return <TreeTransfer dataSource={treeData} targetKeys={targetKeys}
onChange={onChange} />;
};

export default App;
```

自定义状态

```
import React from 'react';
import { Flex, Transfer } from 'antd';

const App: React.FC = () => (
  <Flex gap="middle" vertical>
    <Transfer status="error" />
    <Transfer status="warning" showSearch />
  </Flex>
);

export default App;
```

自定义全选文字

Debug

```
import React, { useState } from 'react';
import { Transfer } from 'antd';
import type { TransferProps } from 'antd';

interface RecordType {
  key: string;
  title: string;
  description: string;
}

const mockData = Array.from({ length: 10 }).map<RecordType>((_, i) => ({
  key: i.toString(),
  title: `content${i + 1}`,
  description: `description of content${i + 1}`,
}));

const oriTargetKeys = mockData.filter((item) => Number(item.key) % 3 >
1).map((item) => item.key);

const selectAllLabels: TransferProps['selectAllLabels'] = [
  'Select All',
  ({ selectedCount, totalCount }) => `${selectedCount}/${totalCount}`,
];

const App: React.FC = () => {
  const [targetKeys, setTargetKeys] = useState<React.Key[]>(oriTargetKeys);
  return (
    <Transfer
      dataSource={mockData}
      targetKeys={targetKeys}
      onChange={setTargetKeys}
      render={(item) => item.title}
      selectAllLabels={selectAllLabels}
    />
  );
};

export default App;
```

组件 Token

Debug

```
import React, { useState } from 'react';
import { ConfigProvider, Space, Switch, Table, Tag, Transfer } from 'antd';
import type { GetProp, TableColumnsType, TableProps, TransferProps } from
'antd';
import difference from 'lodash/difference';

type TableRowSelection<T> = TableProps<T>['rowSelection'];

type TransferItem = GetProp<TransferProps, 'dataSource'>[number];

interface RecordType {
  key: string;
  title: string;
  description: string;
  disabled: boolean;
  tag: string;
```

```
}

interface DataType {
  key: string;
  title: string;
  description: string;
  disabled: boolean;
  tag: string;
}

interface TableTransferProps extends TransferProps<TransferItem> {
  dataSource: DataType[];
  leftColumns: TableColumnsType<DataType>;
  rightColumns: TableColumnsType<DataType>;
}

// Customize Table Transfer
const TableTransfer = ({ leftColumns, rightColumns, ...restProps }:
TableTransferProps) => (
  <Transfer {...restProps}>
    {({
      direction,
      filteredItems,
      onItemSelectAll,
      onItemSelect,
      selectedKeys: listSelectedKeys,
      disabled: listDisabled,
    }) => {
      const columns = direction === 'left' ? leftColumns : rightColumns;

      const rowSelection: TableRowSelection<TransferItem> = {
        getCheckboxProps: (item) => ({ disabled: listDisabled ||
item.disabled }),
        onSelectAll(selected, selectedRows) {
          const treeSelectedKeys = selectedRows
            .filter((item) => !item.disabled)
            .map(({ key }) => key);
          const diffKeys = selected
            ? difference(treeSelectedKeys, listSelectedKeys)
            : difference(listSelectedKeys, treeSelectedKeys);
          onItemSelectAll(diffKeys as string[], selected);
        },
        onSelect({ key }, selected) {
          onItemSelect(key as string, selected);
        },
        selectedRowKeys: listSelectedKeys,
      };

      return (
        <Table
          rowSelection={rowSelection}
          columns={columns}
          dataSource={filteredItems}
          size="small"
          style={{ pointerEvents: listDisabled ? 'none' : undefined }}
          onRow={({ key, disabled: itemDisabled }) => ({
            onClick: () => {
              if (itemDisabled || listDisabled) {
                return;
              }
              onItemSelect(key as string, !listSelectedKeys.includes(key as
string));
            },
          })}
        />
      );
    }}
  </Transfer>
);

const mockTags = ['cat', 'dog', 'bird'];

const mockData = Array.from({ length: 20 }).map<RecordType>((_, i) => ({
  key: i.toString(),
  title: `content${i + 1}`,
  description: `description of content${i + 1}`,
  disabled: i % 4 === 0,
  tag: mockTags[i % 3],
}));

const leftTableColumns: TableColumnsType<DataType> = [
  {
    dataIndex: 'title',
    title: 'Name',
  },
  {
    dataIndex: 'tag',
    title: 'Tag',
    render: (tag) => <Tag>{tag}</Tag>,
  },
  {
    dataIndex: 'description',
    title: 'Description',
  },
];

const rightTableColumns: TableColumnsType<DataType> = [
  {
    dataIndex: 'title',
    title: 'Name',
  },
];

const initialTargetKeys = mockData.filter((item) => Number(item.key) >
10).map((item) => item.key);

const App: React.FC = () => {
  const [targetKeys, setTargetKeys] = useState<React.Key[]>
(initialTargetKeys);
  const [selectedKeys, setSelectedKeys] = useState<React.Key[]>([]);

  const onChange: TransferProps['onChange'] = (nextTargetKeys, direction,
moveKeys) => {
    console.log('targetKeys:', nextTargetKeys);
    console.log('direction:', direction);
    console.log('moveKeys:', moveKeys);
    setTargetKeys(nextTargetKeys);
  };

  const onSelectChange: TransferProps['onSelectChange'] = (
    sourceSelectedKeys,
    targetSelectedKeys,
  ) => {
    console.log('sourceSelectedKeys:', sourceSelectedKeys);
    console.log('targetSelectedKeys:', targetSelectedKeys);
    setSelectedKeys([...sourceSelectedKeys, ...targetSelectedKeys]);
  };

  const onScroll: TransferProps['onScroll'] = (direction, e) => {
    console.log('direction:', direction);
    console.log('target:', e.target);
  };

  const [disabled, setDisabled] = useState(false);
  const [showSearch, setShowSearch] = useState(false);

  const secondOnChange: TransferProps['onChange'] = (nextTargetKeys) => {
    setTargetKeys(nextTargetKeys);
  };

  const triggerDisable = (checked: boolean) => {
    setDisabled(checked);
  };

  const triggerShowSearch = (checked: boolean) => {
    setShowSearch(checked);
  };

  return (
    <ConfigProvider
      theme={{
        components: {
          Transfer: {
            listWidth: 40,
            listWidthLG: 50,
            listHeight: 30,
            itemHeight: 20,
            itemPaddingBlock: 10,
            headerHeight: 18,
          },
        },
      }}
    >
      <Transfer
        dataSource={mockData}
        titles={['Source', 'Target']}
        targetKeys={targetKeys}
        selectedKeys={selectedKeys}
        onChange={onChange}
        onSelectChange={onSelectChange}
        onScroll={onScroll}
        render={(item) => item.title}
      />
      <Transfer status="error" />
      <Transfer status="warning" showSearch />
      <TableTransfer
        dataSource={mockData}
        targetKeys={targetKeys}
        disabled={disabled}
        showSearch={showSearch}
        onChange={secondOnChange}
        filterOption={(inputValue, item) =>
          item.title!.indexOf(inputValue) !== -1 ||
item.tag.indexOf(inputValue) !== -1
```

```
        }
        leftColumns={leftTableColumns}
        rightColumns={rightTableColumns}
      />
      <Space style={{ marginTop: 16 }}>
        <Switch
          unCheckedChildren="disabled"
          checkedChildren="disabled"
          checked={disabled}
          onChange={triggerDisable}
        />
        <Switch
          unCheckedChildren="showSearch"
          checkedChildren="showSearch"
          checked={showSearch}
          onChange={triggerShowSearch}
        />
      </Space>
    </ConfigProvider>
  );
};

export default App;
```

API

通用属性参考：通用属性

Transfer

参数	说明	类型	默认值
dataSource	数据源，其中的数据将会被渲染到左边一栏中，`targetKeys` 中指定的除外	RecordType extends TransferItem = TransferItem[]	[]
disabled	是否禁用	boolean	false
selectionsIcon	自定义下拉菜单图标	React.ReactNode	
filterOption	根据搜索内容进行筛选，接收 `inputValue` `option` `direction` 三	(inputValue, option, direction: `left` | `right`): boolean	-

	个参数，(`direction` 自 5.9.0+支持)，当 `option` 符合筛选条件时，应返回 true，反之则返回 false		
footer	底部渲染函数	(props, { direction }) => ReactNode	-
listStyle	两个穿梭框的自定义样式	object|({direction: `left` | `right`}) => object	-
locale	各种语言	{ itemUnit: string; itemsUnit: string; searchPlaceholder: string; notFoundContent: ReactNode | ReactNode[]; }	{ itemUnit: 项, itemsUnit: 项, searchPlacehol 请输入搜索内容
oneWay	展示为单向样式	boolean	false
operations	操作文案集合，顺序从上至下	string[]	[>, <]
operationStyle	操作栏的自定义样式	CSSProperties	-
pagination	使用分页样式，自定义渲染列表下无效	boolean | { pageSize: number, simple: boolean, showSizeChanger?: boolean, showLessItems?: boolean }	false
render	每行数据渲染函数，该函数的入参为 `dataSource` 中的项，返回值为 ReactElement。或者返回一个普通对象，其中 `label` 字段为 ReactElement，`value` 字段为 title	(record) => ReactNode	-
selectAllLabels	自定义顶部多选框标题的集合	(ReactNode | (info: { selectedCount: number, totalCount: number }) => ReactNode)[]	-

selectedKeys	设置哪些项应该被选中	string[] | number[]	[]
showSearch	是否显示搜索框，或可对两侧搜索框进行配置	boolean | { placeholder:string,defaultValue:string }	false
showSelectAll	是否展示全选勾选框	boolean	true
status	设置校验状态	'error' | 'warning'	-
targetKeys	显示在右侧框数据的 key 集合	string[] | number[]	[]
titles	标题集合，顺序从左至右	ReactNode[]	-
onChange	选项在两栏之间转移时的回调函数	(targetKeys, direction, moveKeys): void	-
onScroll	选项列表滚动时的回调函数	(direction, event): void	-
onSearch	搜索框内容时改变时的回调函数	(direction: `left` | `right`, value: string): void	-
onSelectChange	选中项发生改变时的回调函数	(sourceSelectedKeys, targetSelectedKeys): void	-

Render Props

Transfer 支持接收 `children` 自定义渲染列表，并返回以下参数：

参数	说明	类型	版本
direction	渲染列表的方向	`left` | `right`	
disabled	是否禁用列表	boolean	
filteredItems	过滤后的数据	RecordType[]	
selectedKeys	选中的条目	string[] | number[]	
onItemSelect	勾选条目	(key: string | number, selected: boolean)	
onItemSelectAll	勾选一组条目	(keys: string[] | number[], selected: boolean)	

参考示例

```
<Transfer {...props}>{(listProps) => <YourComponent {...listProps} />}
</Transfer>
```

注意

按照 React 的规范，所有的组件数组必须绑定 key。在 Transfer 中， `dataSource` 里的数据值需要指定 `key` 值。对于 `dataSource` 默认将每列数据的 `key` 属性作为唯一的标识。

如果你的数据没有这个属性，务必使用 `rowKey` 来指定数据列的主键。

```
// 比如你的数据主键是 uid
return <Transfer rowKey={(record) => record.uid} />;
```

主题变量（Design Token）

FAQ

怎样让 Transfer 穿梭框列表支持异步数据加载

为了保持页码同步，在勾选时可以不移除选项而以禁用代替：https://codesandbox.io/s/objective-wing-6iqbx