When To Use

- It is a select control essentially which can be use for selecting multiple items.
- Transfer can display more information for items and take up more space.

Transfer the elements between two columns in an intuitive and efficient way.

One or more elements can be selected from either column, one click on the proper `direction` button, and the transfer is done. The left column is considered the `source` and the right column is considered the `target`. As you can see in the API description, these names are reflected in.

> *notice: Transfer is a controlled component, uncontrolled mode is not supported.*

Examples

Basic

```
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;
```

One Way

```
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;
```

Search

```
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;
```

Advanced

```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;
```

Custom datasource

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

interface RecordType {
```

```tsx
  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;
```

Pagination

```
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;
```

Table Transfer

```
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;
```

Tree Transfer

```
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;
```

Status

```
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;
```

Custom Select All Labels

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;
```

Component 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

Common props ref: Common props

Property	Description	Type	Default
dataSource	Used for setting the source data. The elements that are part of this array will be present the left column. Except the elements whose keys are included in	RecordType extends TransferItem = TransferItem[]	[]

	`targetKeys` prop		
disabled	Whether disabled transfer	boolean	false
selectionsIcon	custom dropdown icon	React.ReactNode	
filterOption	A function to determine whether an item should show in search result list, only works when searching, (add `direction` support since 5.9.0+)	(inputValue, option, direction: `left` | `right`): boolean	-
footer	A function used for rendering the footer	(props, { direction }) => ReactNode	-
listStyle	A custom CSS style used for rendering the transfer columns	object | ({direction: `left` | `right`}) => object	-
locale	The i18n text including filter, empty text, item unit, etc	{ itemUnit: string; itemsUnit: string; searchPlaceholder: string; notFoundContent: ReactNode | ReactNode[]; }	{ itemUnit: `item`, itemsUnit: `items`, notFoundContent: `The list is empty`, searchPlaceholder: `Search here` }

oneWay	Display as single direction style	boolean	false
operations	A set of operations that are sorted from top to bottom	string[]	[>, <]
operationStyle	A custom CSS style used for rendering the operations column	object	-
pagination	Use pagination. Not work in render props	boolean | { pageSize: number, simple: boolean, showSizeChanger?: boolean, showLessItems?: boolean }	false
render	The function to generate the item shown on a column. Based on an record (element of the dataSource array), this function should return a React element which is generated from that record.	(record) => ReactNode	-

	Also, it can return a plain object with `value` and `label`, `label` is a React element and `value` is for title		
selectAllLabels	A set of customized labels for select all checkboxes on the header	(ReactNode | (info: { selectedCount: number, totalCount: number }) => ReactNode)[]	-
selectedKeys	A set of keys of selected items	string[] | number[]	[]
showSearch	If included, a search box is shown on each column	boolean | { placeholder:string,defaultValue:string }	false
showSelectAll	Show select all checkbox on the header	boolean	true
status	Set validation status	'error' | 'warning'	-
targetKeys	A set of keys of elements that are listed on the right column	string[] | number[]	[]
titles	A set of titles that	ReactNode[]	-

	are sorted from left to right		
onChange	A callback function that is executed when the transfer between columns is complete	(targetKeys, direction, moveKeys): void	-
onScroll	A callback function which is executed when scroll options list	(direction, event): void	-
onSearch	A callback function which is executed when search field are changed	(direction: `left` | `right`, value: string): void	-
onSelectChange	A callback function which is executed when selected items are changed	(sourceSelectedKeys, targetSelectedKeys): void	-

Render Props

Transfer accept `children` to customize render list, using follow props:

Property	Description	Type	Version
direction	List render direction	`left` | `right`	
disabled	Disable list or not	boolean	
filteredItems	Filtered items	RecordType[]	

selectedKeys	Selected items	string[] | number[]	
onItemSelect	Select item	(key: string | number, selected: boolean)	
onItemSelectAll	Select a group of items	(keys: string[] | number[], selected: boolean)	

example

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

Warning

According the standard of React, the key should always be supplied directly to the elements in the array. In Transfer, the keys should be set on the elements included in `dataSource` array. By default, `key` property is used as an unique identifier.

If there's no `key` in your data, you should use `rowKey` to specify the key that will be used for uniquely identify each element.

```
// eg. your primary key is `uid`
return <Transfer rowKey={(record) => record.uid} />;
```

Design Token

FAQ

How to support fetch and present data from a remote server in Transfer column.

In order to keep the page number synchronized, you can disable columns you checked without removing the option: https://codesandbox.io/s/objective-wing-6iqbx