# Transfer

Double column transfer choice box.

## When To Use

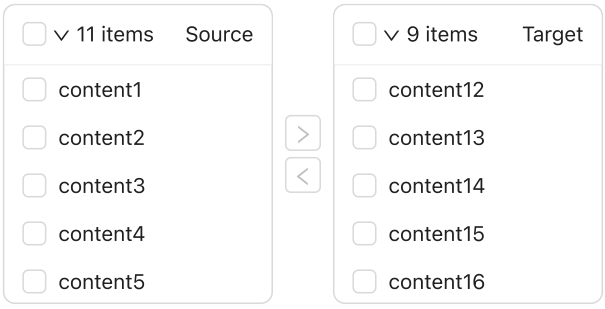
- It is a select control essentially which can be use for selecting multiple items.
- Transfer can display more information for items and take up more space.

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.

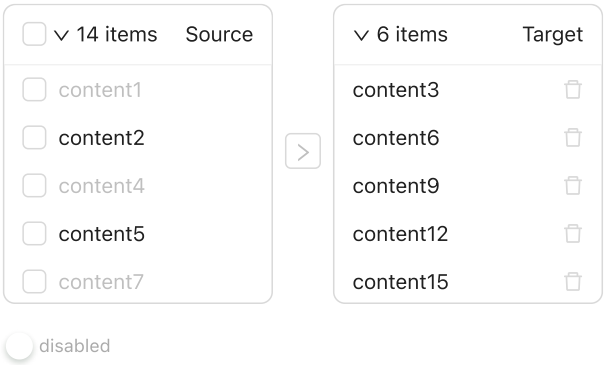
## Examples




## Basic

The most basic usage of `Transfer` involves providing the source data and target keys arrays, plus the rendering and some callback functions.

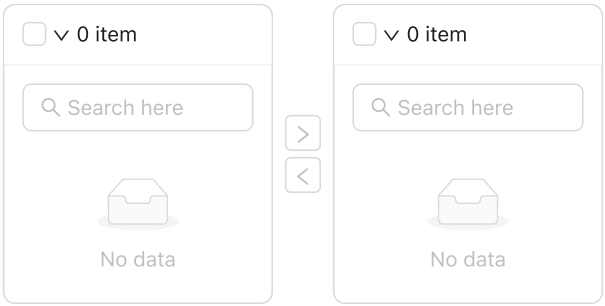
```
import React, { useState } from 'react';
import { Transfer } from 'antd';
import type { TransferDirection } from 'antd/es/transfer';

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

const mockData: RecordType[] = Array.from({ length: 20 }).map((_, 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.ke

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

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

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

  const onScroll = (direction: TransferDirection, e: React.SyntheticEvent<HTMLUListElement>) =>
    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}
    />
  );
};
```




## One Way

Use `oneWay` to makes Transfer to one way style.

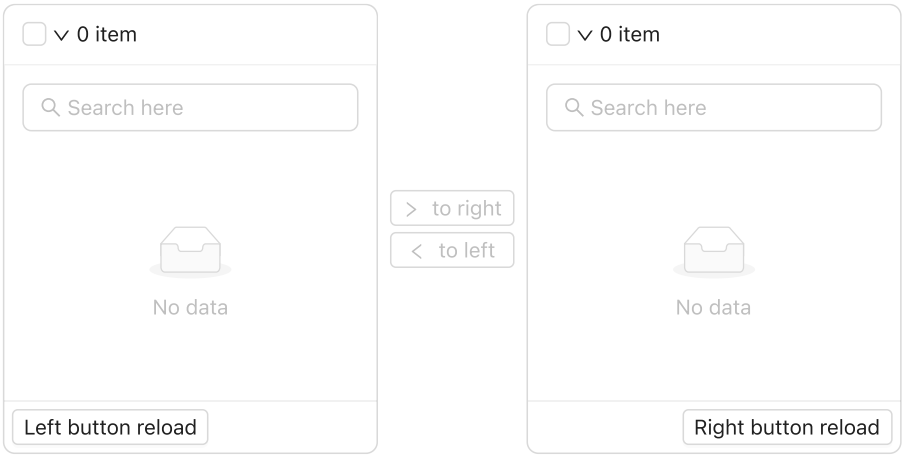
```
import React, { useState } from 'react';
import { Switch, Transfer } from 'antd';
import type { TransferDirection } from 'antd/es/transfer';

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

const mockData: RecordType[] = Array.from({ length: 20 }).map((_, 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<string[]>(oriTargetKeys);
  const [selectedKeys, setSelectedKeys] = useState<string[]>([]);
  const [disabled, setDisabled] = useState(false);

  const handleChange = (
    newTargetKeys: string[],
    direction: TransferDirection,
    moveKeys: string[],
  ) => {
    setTargetKeys(newTargetKeys);

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

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

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

  const handleScroll = (
    direction: TransferDirection,
    e: React.SyntheticEvent<HTMLUListElement, Event>,
  ) => {
    console.log('direction:', direction);
    console.log('target:', e.target);
  };

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




## Search

Transfer with a search box.

```
import React, { useEffect, useState } from 'react';
import { Transfer } from 'antd';
import type { TransferDirection } from 'antd/es/transfer';

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

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

  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 = (newTargetKeys: string[]) => {
    setTargetKeys(newTargetKeys);
  };

  const handleSearch = (dir: TransferDirection, value: string) => {
    console.log('search:', dir, value);
  };

  return (
    <Transfer
      dataSource={mockData}
      showSearch
      filterOption={filterOption}
      targetKeys={targetKeys}
      onChange={handleChange}
```

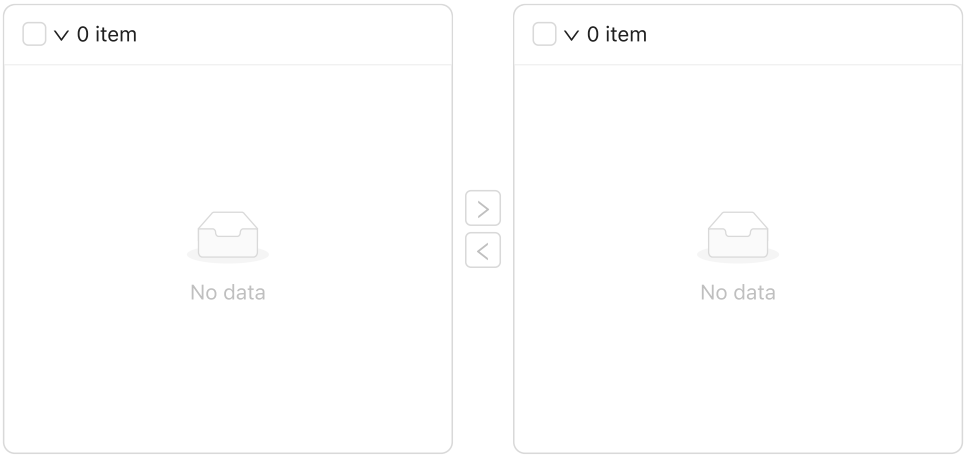



## Advanced

Advanced Usage of Transfer.

You can customize the labels of the transfer buttons, the width and height of the columns, and what should be displayed in the footer.

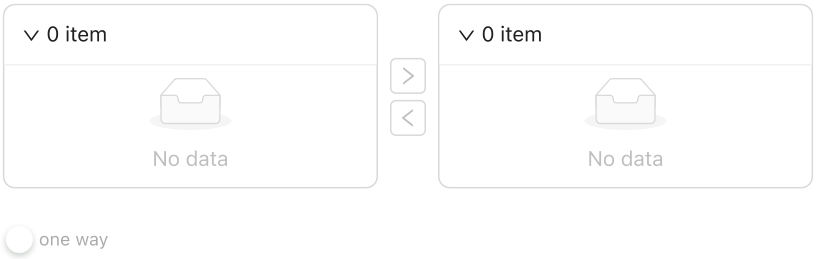
```
import React, { useEffect, useState } from 'react';
import { Button, Transfer } from 'antd';
import type { TransferDirection, TransferListProps } from 'antd/es/transfer';

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

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

  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 = (newTargetKeys: string[]) => {
    setTargetKeys(newTargetKeys);
  };

  const renderFooter = (
    _: TransferListProps<any>,
    { direction }: {
      direction: TransferDirection;
    },
```




## Custom datasource

Custom each Transfer Item, and in this way you can render a complex datasource.

```
import React, { useEffect, useState } from 'react';
import { Transfer } from 'antd';
import type { TransferDirection } from 'antd/es/transfer';

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

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

  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 = (
    newTargetKeys: string[],
    direction: TransferDirection,
    moveKeys: string[],
  ) => {
    console.log(newTargetKeys, direction, moveKeys);
    setTargetKeys(newTargetKeys);
  };

  const renderItem = (item: RecordType) => {
    const customLabel = (
      <span className="custom-item">
```




## Pagination

large count of items with pagination.

```
import React, { useEffect, useState } from 'react';
import { Switch, Transfer } from 'antd';
import type { TransferDirection } from 'antd/es/transfer';

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<string[]>([]);

  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 = (newTargetKeys: string[], direction: TransferDirection, moveKeys: string[]) =
    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}
```

14 items

| Name | Tag | Description |
| --- | --- | --- |
| content1 | cat | description of content1 |
| content2 | dog | description of content2 |
| content4 | cat | description of content4 |
| content5 | dog | description of content5 |
| content7 | cat | description of content7 |
| content8 | dog | description of content8 |
| content10 | cat | description of content10 |
| content11 | dog | description of content11 |
| content13 | cat | description of content13 |
| content14 | dog | description of content14 |

1 2

6 items

| Name |
| --- |
| content3 |
| content6 |
| content9 |
| content12 |
| content15 |
| content18 |

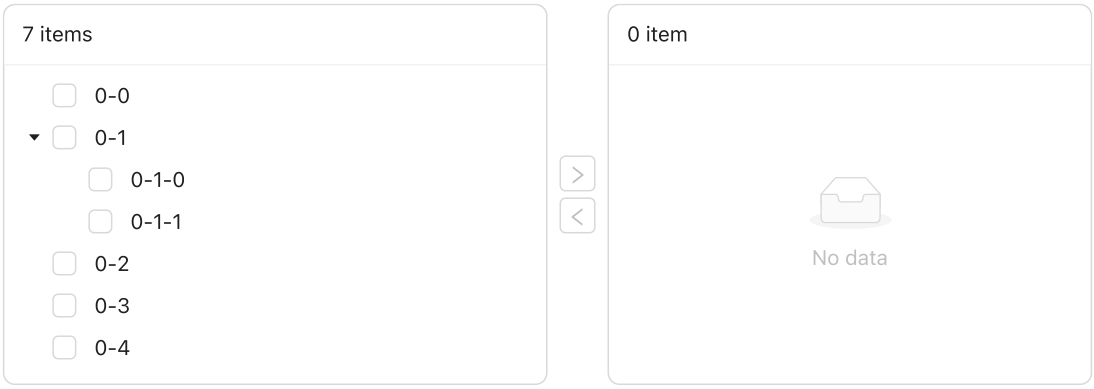
1

disabled showSearch

## Table Transfer

Customize render list with Table component.

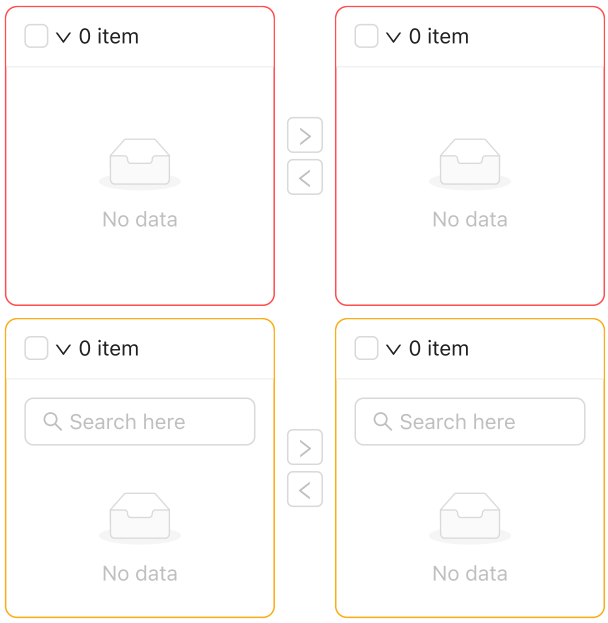
```
import React, { useState } from 'react';
import { Space, Switch, Table, Tag, Transfer } from 'antd';
import type { ColumnsType, TableRowSelection } from 'antd/es/table/interface';
import type { TransferItem, TransferProps } from 'antd/es/transfer';
import difference from 'lodash/difference';

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: ColumnsType<DataType>;
  rightColumns: ColumnsType<DataType>;
```




## Tree Transfer

Customize render list with Tree component.

```
import React, { useState } from 'react';
import { Transfer, Tree, theme } from 'antd';
import type { TransferDirection, TransferItem } from 'antd/es/transfer';
import type { DataNode } from 'antd/es/tree';

interface TreeTransferProps {
  dataSource: DataNode[];
  targetKeys: string[];
  onChange: (targetKeys: string[], direction: TransferDirection, moveKeys: string[]) => void;
}

// Customize Table Transfer
const isChecked = (selectedKeys: (string | number)[], eventKey: string | number) =>
  selectedKeys.includes(eventKey);

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

const TreeTransfer = ({ dataSource, targetKeys, ...restProps }: TreeTransferProps) => {
  const { token } = theme.useToken();

  const transferDataSource: TransferItem[] = [];
  function flatten(list: DataNode[] = []) {
    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
```




## Status

Add status to Transfer with `status`, which could be `error` or `warning`.

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

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

export default App;
```

```
        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>
    </>
  );
};

export default App;
```

## API

| 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 `targetKeys` prop | RecordType extends TransferItem = TransferItem[] | [] |
| disabled | Whether disabled transfer | boolean | false |
| filterOption | A function to determine whether an item should show in search result list, only works when searching | (inputValue, option): boolean | - |
| footer | A function used for rendering the footer | (props, { direction }) => ReactNode | - |

| Property | Description | Type | Default |
| --- | --- | --- | --- |
| 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. Also, it can return a plain object with `value` and `label`, `label` is a React element and `value` is for title | (record) => ReactNode | - |
| 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[] | [] |
| showSearch | If included, a search box is shown on each column | boolean | false |

| Property | Description | Type | Default |
| --- | --- | --- | --- |
| 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[] | [] |
| titles | A set of titles that are sorted from left to right | ReactNode[] | - |
| 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[] | |
| onItemSelect | Select item | (key: string, selected: boolean) | |
| onItemSelectAll | Select a group of items | (keys: string[], 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

### ▾ Global Token

| Token Name | Description | Type | Default Value |
|---|---|---|---|
| colorBgContainer | Container background color, e.g: default button, input box, etc. Be sure not to confuse this with `colorBgElevated`. | `string` | `#ffffff` |
| colorBgContainerDisabled | Control the background color of container in disabled state. | `string` | `rgba(0, 0, 0, 0.04)` |
| colorBorder | Default border color, used to separate different elements, such as: form separator, card separator, etc. | `string` | `#d9d9d9` |
| colorError | Used to represent the visual elements of the operation failure, such as the error Button, error Result component, etc. | `string` | `#ff4d4f` |
| colorLinkHover | Control the color of hyperlink when hovering. | `string` | `#69b1ff` |
| colorSplit | Used as the color of separator, this color is the same as colorBorderSecondary but with transparency. | `string` | `rgba(5, 5, 5, 0.06)` |
| colorText | Default text color which comply with W3C standards, and this color is also the darkest neutral color. | `string` | `rgba(0, 0, 0, 0.88)` |
| colorTextDisabled | Control the color of text in disabled state. | `string` | `rgba(0, 0, 0, 0.25)` |
| colorWarning | Used to represent the warning map token, such as Notification, Alert, etc. Alert or Control component(like Input) will use these map tokens. | `string` | `#faad14` |
| borderRadiusLG | LG size border radius, used in some large border radius components, such as Card, Modal and other components. | `number` | 8 |

| Token Name | Description | Type | Default Value |
| --- | --- | --- | --- |
| controlHeight | The height of the basic controls such as buttons and input boxes in Ant Design | `number` | 32 |
| controlHeightLG | LG component height | `number` | 40 |
| controlItemBgActive | Control the background color of control component item when active. | `string` | `#e6f4ff` |
| controlItemBgActiveHover | Control the background color of control component item when hovering and active. | `string` | `#bae0ff` |
| controlItemBgHover | Control the background color of control component item when hovering. | `string` | `rgba(0, 0, 0, 0.04)` |
| fontFamily | The font family of Ant Design prioritizes the default interface font of the system, and provides a set of alternative font libraries that are suitable for screen display to maintain the readability and readability of the font under different platforms and browsers, reflecting the friendly, stable and professional characteristics. | `string` | -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, 'Helvetica Neue', Arial, 'Noto Sans', sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol', 'Noto Color Emoji' |
| fontSize | The most widely used font size in the design system, from which the text gradient will be derived. | `number` | 14 |
| fontSizeIcon | Control the font size of operation icon in Select, Cascader, etc. Normally same as fontSizeSM. | `number` | 12 |
| lineHeight | Line height of text. | `number` | 1.5714285714285714 |
| lineType | Border style of base components | `string` | solid |
| lineWidth | Border width of base components | `number` | 1 |
| margin | Control the margin of an element, with a medium size. | `number` | 16 |
| marginXS | Control the margin of an element, with a small size. | `number` | 8 |
| marginXXS | Control the margin of an element, with the smallest size. | `number` | 4 |
| motionDurationSlow | Motion speed, slow speed. Used for large element animation interaction. | `string` | 0.3s |
| paddingSM | Control the small padding of the element. | `number` | 12 |
| paddingXS | Control the extra small padding of the element. | `number` | 8 |

## 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