When To Use

When requiring users to interact with the application, but without jumping to a new page and interrupting the user's workflow, you can use `Modal` to create a new floating layer over the current page to get user feedback or display information.

Additionally, if you need to show a simple confirmation dialog, you can use `App.useApp` hooks.

Examples

Basic

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

const App: React.FC = () => {
  const [isModalOpen, setIsModalOpen] = useState(false);

  const showModal = () => {
    setIsModalOpen(true);
  };

  const handleOk = () => {
    setIsModalOpen(false);
  };

  const handleCancel = () => {
    setIsModalOpen(false);
  };

  return (
    <>
      <Button type="primary" onClick={showModal}>
        Open Modal
      </Button>
      <Modal title="Basic Modal" open={isModalOpen} onOk={handleOk}
onCancel={handleCancel}>
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
      </Modal>
    </>
  );
};

export default App;
```

Asynchronously close

```tsx
import React, { useState } from 'react';
import { Button, Modal } from 'antd';

const App: React.FC = () => {
  const [open, setOpen] = useState(false);
  const [confirmLoading, setConfirmLoading] = useState(false);
  const [modalText, setModalText] = useState('Content of the modal');

  const showModal = () => {
    setOpen(true);
  };

  const handleOk = () => {
    setModalText('The modal will be closed after two seconds');
    setConfirmLoading(true);
    setTimeout(() => {
      setOpen(false);
      setConfirmLoading(false);
    }, 2000);
  };

  const handleCancel = () => {
    console.log('Clicked cancel button');
    setOpen(false);
  };

  return (
    <>
      <Button type="primary" onClick={showModal}>
        Open Modal with async logic
      </Button>
      <Modal
        title="Title"
        open={open}
        onOk={handleOk}
        confirmLoading={confirmLoading}
        onCancel={handleCancel}
      >
        <p>{modalText}</p>
      </Modal>
    </>
  );
};
```

```
export default App;
```

Customized Footer

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

const App: React.FC = () => {
  const [loading, setLoading] = useState(false);
  const [open, setOpen] = useState(false);

  const showModal = () => {
    setOpen(true);
  };

  const handleOk = () => {
    setLoading(true);
    setTimeout(() => {
      setLoading(false);
      setOpen(false);
    }, 3000);
  };

  const handleCancel = () => {
    setOpen(false);
  };

  return (
    <>
      <Button type="primary" onClick={showModal}>
        Open Modal with customized footer
      </Button>
      <Modal
        open={open}
        title="Title"
        onOk={handleOk}
        onCancel={handleCancel}
        footer={[
          <Button key="back" onClick={handleCancel}>
            Return
          </Button>,
          <Button key="submit" type="primary" loading={loading} onClick=
{handleOk}>
            Submit
          </Button>,
```

```
          <Button
            key="link"
            href="https://google.com"
            target="_blank"
            type="primary"
            loading={loading}
            onClick={handleOk}
          >
            Search on Google
          </Button>,
        ]}
      >
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
      </Modal>
    </>
  );
};

export default App;
```

Loading

v5.18.0

```
import React from 'react';
import { Button, Modal } from 'antd';

const App: React.FC = () => {
  const [open, setOpen] = React.useState<boolean>(false);
  const [loading, setLoading] = React.useState<boolean>(true);

  const showLoading = () => {
    setOpen(true);
    setLoading(true);

    // Simple loading mock. You should add cleanup logic in real world.
    setTimeout(() => {
      setLoading(false);
    }, 2000);
  };

  return (
```

```
    <>
      <Button type="primary" onClick={showLoading}>
        Open Modal
      </Button>
      <Modal
        title={<p>Loading Modal</p>}
        footer={
          <Button type="primary" onClick={showLoading}>
            Reload
          </Button>
        }
        loading={loading}
        open={open}
        onCancel={() => setOpen(false)}
      >
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
      </Modal>
    </>
  );
};

export default App;
```

Customized Footer render function

v5.9.0

```
import React, { useState } from 'react';
import { Button, Modal, Space } from 'antd';

const App: React.FC = () => {
  const [open, setOpen] = useState(false);

  const showModal = () => {
    setOpen(true);
  };
  const handleOk = () => {
    setOpen(false);
  };

  const handleCancel = () => {
    setOpen(false);
  };
  return (
```

```
    <>
      <Space>
        <Button type="primary" onClick={showModal}>
          Open Modal
        </Button>
        <Button
          type="primary"
          onClick={() => {
            Modal.confirm({
              title: 'Confirm',
              content: 'Bla bla ...',
              footer: (_, { OkBtn, CancelBtn }) => (
                <>
                  <Button>Custom Button</Button>
                  <CancelBtn />
                  <OkBtn />
                </>
              ),
            });
          }}
        >
          Open Modal Confirm
        </Button>
      </Space>
      <Modal
        open={open}
        title="Title"
        onOk={handleOk}
        onCancel={handleCancel}
        footer={(_, { OkBtn, CancelBtn }) => (
          <>
            <Button>Custom Button</Button>
            <CancelBtn />
            <OkBtn />
          </>
        )}
      >
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
      </Modal>
    </>
  );
};
```

```
export default App;
```

Use hooks to get context

```
import React, { createContext } from 'react';
import { Button, Modal, Space } from 'antd';

const ReachableContext = createContext<string | null>(null);
const UnreachableContext = createContext<string | null>(null);

const config = {
  title: 'Use Hook!',
  content: (
    <>
      <ReachableContext.Consumer>{(name) => `Reachable: ${name}!`}
</ReachableContext.Consumer>
      <br />
      <UnreachableContext.Consumer>{(name) => `Unreachable: ${name}!`}
</UnreachableContext.Consumer>
    </>
  ),
};

const App: React.FC = () => {
  const [modal, contextHolder] = Modal.useModal();

  return (
    <ReachableContext.Provider value="Light">
      <Space>
        <Button
          onClick={async () => {
            const confirmed = await modal.confirm(config);
            console.log('Confirmed: ', confirmed);
          }}
        >
          Confirm
        </Button>
        <Button
          onClick={() => {
            modal.warning(config);
          }}
        >
          Warning
        </Button>
        <Button
```

```
          onClick={async () => {
            modal.info(config);
          }}
        >
          Info
        </Button>
        <Button
          onClick={async () => {
            modal.error(config);
          }}
        >
          Error
        </Button>
      </Space>
      {/* `contextHolder` should always be placed under the context you
want to access */}
      {contextHolder}

      {/* Can not access this context since `contextHolder` is not in it
*/}
      <UnreachableContext.Provider value="Bamboo" />
    </ReachableContext.Provider>
  );
};

export default App;
```

Internationalization

```
import React, { useState } from 'react';
import { ExclamationCircleOutlined } from '@ant-design/icons';
import { Button, Modal, Space } from 'antd';

const LocalizedModal = () => {
  const [open, setOpen] = useState(false);

  const showModal = () => {
    setOpen(true);
  };

  const hideModal = () => {
    setOpen(false);
  };

  return (
    <>
```

```
      <Button type="primary" onClick={showModal}>
        Modal
      </Button>
      <Modal
        title="Modal"
        open={open}
        onOk={hideModal}
        onCancel={hideModal}
        okText="确认"
        cancelText="取消"
      >
        <p>Bla bla ...</p>
        <p>Bla bla ...</p>
        <p>Bla bla ...</p>
      </Modal>
    </>
  );
};

const App: React.FC = () => {
  const [modal, contextHolder] = Modal.useModal();

  const confirm = () => {
    modal.confirm({
      title: 'Confirm',
      icon: <ExclamationCircleOutlined />,
      content: 'Bla bla ...',
      okText: '确认',
      cancelText: '取消',
    });
  };

  return (
    <>
      <Space>
        <LocalizedModal />
        <Button onClick={confirm}>Confirm</Button>
      </Space>
      {contextHolder}
    </>
  );
};

export default App;
```

Manual to update destroy

```tsx
import React from 'react';
import { Button, Modal } from 'antd';

const App: React.FC = () => {
  const [modal, contextHolder] = Modal.useModal();

  const countDown = () => {
    let secondsToGo = 5;

    const instance = modal.success({
      title: 'This is a notification message',
      content: `This modal will be destroyed after ${secondsToGo} second.`,
    });

    const timer = setInterval(() => {
      secondsToGo -= 1;
      instance.update({
        content: `This modal will be destroyed after ${secondsToGo}
second.`,
      });
    }, 1000);

    setTimeout(() => {
      clearInterval(timer);
      instance.destroy();
    }, secondsToGo * 1000);
  };

  return (
    <>
      <Button onClick={countDown}>Open modal to close in 5s</Button>
      {contextHolder}
    </>
  );
};

export default App;
```

To customize the position of modal

```tsx
import React, { useState } from 'react';
import { Button, Modal } from 'antd';

const App: React.FC = () => {
  const [modal1Open, setModal1Open] = useState(false);
```

```
  const [modal2Open, setModal2Open] = useState(false);

  return (
    <>
      <Button type="primary" onClick={() => setModal1Open(true)}>
        Display a modal dialog at 20px to Top
      </Button>
      <Modal
        title="20px to Top"
        style={{ top: 20 }}
        open={modal1Open}
        onOk={() => setModal1Open(false)}
        onCancel={() => setModal1Open(false)}
      >
        <p>some contents...</p>
        <p>some contents...</p>
        <p>some contents...</p>
      </Modal>
      <br />
      <br />
      <Button type="primary" onClick={() => setModal2Open(true)}>
        Vertically centered modal dialog
      </Button>
      <Modal
        title="Vertically centered modal dialog"
        centered
        open={modal2Open}
        onOk={() => setModal2Open(false)}
        onCancel={() => setModal2Open(false)}
      >
        <p>some contents...</p>
        <p>some contents...</p>
        <p>some contents...</p>
      </Modal>
    </>
  );
};

export default App;
```

Dark Bg

Debug

```
import React, { useState } from 'react';
import { ClockCircleOutlined, DownOutlined } from '@ant-design/icons';
```

```tsx
import {
  Anchor,
  Badge,
  Button,
  Calendar,
  Card,
  Collapse,
  DatePicker,
  Dropdown,
  Modal,
  Slider,
  Switch,
  Table,
  Tabs,
  Timeline,
  Transfer,
  Tree,
  Typography,
} from 'antd';
import type { TableProps, TransferProps } from 'antd';
import type { TransferKey } from 'antd/es/transfer/interface';
import dayjs from 'dayjs';
import customParseFormat from 'dayjs/plugin/customParseFormat';
import difference from 'lodash/difference';

dayjs.extend(customParseFormat);

const { Panel } = Collapse;
const { TreeNode } = Tree;
const { TabPane } = Tabs;
const { Meta } = Card;
const { Link } = Anchor;
const { Text } = Typography;

const text = `
  A dog is a type of domesticated animal.
  Known for its loyalty and faithfulness,
  it can be found as a welcome guest in many households across the world.
`;

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

```ts
interface RecordType {
  key: string;
  name: string;
  age: number;
  address: string;
}

interface DataTableType {
  key: string;
  name: string;
  borrow: number;
  repayment: number;
}

interface ExpandDataType {
  key: React.Key;
  date: string;
  name: string;
  upgradeNum: string;
}

interface NestDataType {
  key: React.Key;
  name: string;
  platform: string;
  version: string;
  upgradeNum: number;
  creator: string;
  createdAt: string;
}

interface FixedDataType {
  key: string;
  name: string;
  age: number;
  address: string;
}

const mockData = Array.from({ length: 20 }).map<DataType>((_, 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<TransferKey>((item) => item.key);

const dataSource: RecordType[] = [
  { key: '1', name: 'John Brown', age: 32, address: 'New York No. 1 Lake
Park' },
  { key: '2', name: 'Jim Green', age: 42, address: 'London No. 1 Lake Park'
},
  { key: '3', name: 'Joe Black', age: 32, address: 'Sydney No. 1 Lake Park'
},
  { key: '4', name: 'Jim Red', age: 32, address: 'London No. 2 Lake Park'
},
];

const columnsTable: TableProps<DataTableType>['columns'] = [
  { title: 'Name', dataIndex: 'name' },
  { title: 'Borrow', dataIndex: 'borrow' },
  { title: 'Repayment', dataIndex: 'repayment' },
];

const summaryDataSource: DataTableType[] = [
  { key: '1', name: 'John Brown', borrow: 10, repayment: 33 },
  { key: '2', name: 'Jim Green', borrow: 100, repayment: 0 },
  { key: '3', name: 'Joe Black', borrow: 10, repayment: 10 },
  { key: '4', name: 'Jim Red', borrow: 75, repayment: 45 },
];

const expandDataSource = Array.from({ length: 3 }).map<ExpandDataType>((_,
i) => ({
  key: i,
  date: '2014-12-24 23:12:00',
  name: 'This is production name',
  upgradeNum: 'Upgraded: 56',
}));

const expandColumns: TableProps<ExpandDataType>['columns'] = [
  { title: 'Date', dataIndex: 'date', key: 'date' },
  { title: 'Name', dataIndex: 'name', key: 'name' },
  {
    title: 'Status',
    key: 'state',
    render: () => (
      <span>
        <Badge status="success" />
        Finished
```

```
      </span>
    ),
  },
  { title: 'Upgrade Status', dataIndex: 'upgradeNum', key: 'upgradeNum' },
  {
    title: 'Action',
    dataIndex: 'operation',
    key: 'operation',
    render: () => (
      <span className="table-operation">
        <a>Pause</a>
        <a>Stop</a>
        <Dropdown>
          <a>
            More <DownOutlined />
          </a>
        </Dropdown>
      </span>
    ),
  },
];

const expandedRowRender = () => (
  <Table<ExpandDataType> columns={expandColumns} dataSource=
{expandDataSource} pagination={false} />
);

const columnsNest: TableProps<NestDataType>['columns'] = [
  { title: 'Name', dataIndex: 'name', key: 'name' },
  { title: 'Platform', dataIndex: 'platform', key: 'platform' },
  { title: 'Version', dataIndex: 'version', key: 'version' },
  { title: 'Upgraded', dataIndex: 'upgradeNum', key: 'upgradeNum' },
  { title: 'Creator', dataIndex: 'creator', key: 'creator' },
  { title: 'Date', dataIndex: 'createdAt', key: 'createdAt' },
  { title: 'Action', key: 'operation', render: () => <a>Publish</a> },
];

const nestDataSource = Array.from({ length: 3 }).map<NestDataType>((_, i)
=> ({
  key: i,
  name: 'Screem',
  platform: 'iOS',
  version: '10.3.4.5654',
  upgradeNum: 500,
  creator: 'Jack',
  createdAt: '2014-12-24 23:12:00',
```

```
}));

const columnsFixed: TableProps<FixedDataType>['columns'] = [
  { title: 'Full Name', width: 100, dataIndex: 'name', key: 'name', fixed:
'left' },
  { title: 'Age', width: 100, dataIndex: 'age', key: 'age', fixed: 'left'
},
  { title: 'Column 1', dataIndex: 'address', key: '1' },
  { title: 'Column 2', dataIndex: 'address', key: '2' },
  { title: 'Column 3', dataIndex: 'address', key: '3' },
  { title: 'Column 4', dataIndex: 'address', key: '4' },
  { title: 'Column 5', dataIndex: 'address', key: '5' },
  { title: 'Column 6', dataIndex: 'address', key: '6' },
  { title: 'Column 7', dataIndex: 'address', key: '7' },
  { title: 'Column 8', dataIndex: 'address', key: '8' },
  { title: 'Action', key: 'operation', fixed: 'right', width: 100, render:
() => <a>action</a> },
];

const fixedDataSource: FixedDataType[] = [
  { key: '1', name: 'John Brown', age: 32, address: 'New York Park' },
  { key: '2', name: 'Jim Green', age: 40, address: 'London Park' },
];

const TableTransfer: React.FC<
  Readonly<Partial<Record<'leftColumns' | 'rightColumns',
TableProps<DataType>['columns']>>> &
    TransferProps<DataType>
> = (props) => {
  const { leftColumns, rightColumns, ...restProps } = props;
  return (
    <Transfer<DataType> {...restProps} showSelectAll={false}>
      {(transferProps) => {
        const {
          direction,
          filteredItems,
          onItemSelectAll,
          onItemSelect,
          selectedKeys: listSelectedKeys,
          disabled: listDisabled,
        } = transferProps;

        const columns = (direction === 'left' ? leftColumns : rightColumns)
?? [];

        const rowSelection: TableProps<DataType>['rowSelection'] = {
```

```
          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, selected);
          },
          onSelect({ key }, selected) {
            onItemSelect(key, selected);
          },
          selectedRowKeys: listSelectedKeys,
        };

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

const columns: TableProps<RecordType>['columns'] = [
  {
    title: 'Name',
    dataIndex: 'name',
    key: 'name',
    filters: [
```

```
      { text: 'Joe', value: 'Joe' },
      { text: 'Jim', value: 'Jim' },
    ],
    filteredValue: null,
    onFilter: (value, record) => record.name.includes(value as string),
    sorter: (a, b) => a.name.length - b.name.length,
    sortOrder: 'ascend',
    ellipsis: true,
  },
  {
    title: 'Age',
    dataIndex: 'age',
    key: 'age',
    sorter: false,
    sortOrder: 'ascend',
    ellipsis: true,
  },
  {
    title: 'Address',
    dataIndex: 'address',
    key: 'address',
    filters: [
      { text: 'London', value: 'London' },
      { text: 'New York', value: 'New York' },
    ],
    filteredValue: null,
    onFilter: (value, record) => record.address.includes(value as string),
    sorter: false,
    sortOrder: 'ascend',
    ellipsis: true,
  },
];

const tableTransferColumns: TableProps<DataType>['columns'] = [
  { dataIndex: 'title', title: 'Name' },
  { dataIndex: 'description', title: 'Description' },
];

const Demo: React.FC = () => {
  const [open, setOpen] = useState(false);
  const [targetKeys, setTargetKeys] = useState<TransferKey[]>
(oriTargetKeys);
  const [selectedKeys, setSelectedKeys] = useState<TransferKey[]>([]);
  const [disabled, setDisabled] = useState(false);
  const [showSearch, setShowSearch] = useState(false);
```

```tsx
  const handleDisable = (isDisabled: boolean) => {
    setDisabled(isDisabled);
  };

  const handleTableTransferChange = (nextTargetKeys: TransferKey[]) => {
    setTargetKeys(nextTargetKeys);
  };

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

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

  const handleTransferChange = (keys: TransferKey[]) => {
    setTargetKeys(keys);
  };

  const handleTransferSelectChange = (
    sourceSelectedKeys: TransferKey[],
    targetSelectedKeys: TransferKey[],
  ) => {
    setSelectedKeys([...sourceSelectedKeys, ...targetSelectedKeys]);
  };

  const showModal = () => {
    setOpen(true);
  };

  const handleOk = (e: React.MouseEvent<HTMLButtonElement, MouseEvent>) =>
{
    console.log(e);
    setOpen(false);
  };

  const handleCancel = (e: React.MouseEvent<HTMLButtonElement, MouseEvent>)
=> {
    console.log(e);
    setOpen(false);
  };

  return (
    <>
      <Button type="primary" onClick={showModal}>
```

```
        Open Modal
      </Button>
      <Modal title="Basic Modal" open={open} onOk={handleOk} onCancel=
{handleCancel}>
        <Switch
          unCheckedChildren="disabled"
          checkedChildren="disabled"
          checked={disabled}
          onChange={handleDisable}
          style={{ marginBottom: 16 }}
        />
        <Card title="Card Title">
          <Card.Grid>Content</Card.Grid>
          <Card.Grid hoverable={false}>Content</Card.Grid>
          <Card.Grid>Content</Card.Grid>
          <Card.Grid>Content</Card.Grid>
          <Card.Grid>Content</Card.Grid>
          <Card.Grid>Content</Card.Grid>
          <Card.Grid>Content</Card.Grid>
        </Card>
        <Collapse>
          <Panel header="This is panel header 1" key="1">
            <Collapse defaultActiveKey="1">
              <Panel header="This is panel nest panel" key="1">
                <p>{text}</p>
              </Panel>
            </Collapse>
          </Panel>
          <Panel header="This is panel header 2" key="2">
            <p>{text}</p>
          </Panel>
          <Panel header="This is panel header 3" key="3">
            <p>{text}</p>
          </Panel>
        </Collapse>
        <Transfer<DataType>
          dataSource={mockData}
          titles={['Source', 'Target']}
          targetKeys={targetKeys}
          selectedKeys={selectedKeys}
          onChange={handleTransferChange}
          onSelectChange={handleTransferSelectChange}
          render={(item) => item.title}
          disabled={disabled}
        />
        <TableTransfer
```

```
          dataSource={mockData}
          targetKeys={targetKeys}
          disabled={disabled}
          showSearch={showSearch}
          leftColumns={tableTransferColumns}
          rightColumns={tableTransferColumns}
          onChange={handleTableTransferChange}
          filterOption={(inputValue: string, item: any) =>
            item.title?.includes(inputValue) ||
item.tag?.includes(inputValue)
          }
        />
        <Switch
          unCheckedChildren="disabled"
          checkedChildren="disabled"
          checked={disabled}
          onChange={triggerDisable}
          style={{ marginTop: 16 }}
        />
        <Switch
          unCheckedChildren="showSearch"
          checkedChildren="showSearch"
          checked={showSearch}
          onChange={triggerShowSearch}
          style={{ marginTop: 16 }}
        />
        <Anchor>
          <Link href="#anchor-demo-basic" title="Basic demo" />
          <Link href="#anchor-demo-static" title="Static demo" />
          <Link href="#anchor-demo-basic" title="Basic demo with Target"
target="_blank" />
          <Link href="#API" title="API">
            <Link href="#Anchor-Props" title="Anchor Props" />
            <Link href="#Link-Props" title="Link Props" />
          </Link>
        </Anchor>
        <Tabs type="card">
          <TabPane tab="Tab 1" key="1">
            Content of Tab Pane 1
          </TabPane>
          <TabPane tab="Tab 2" key="2">
            Content of Tab Pane 2
          </TabPane>
          <TabPane tab="Tab 3" key="3">
            Content of Tab Pane 3
          </TabPane>
```

```
        </Tabs>
        <Timeline>
          <Timeline.Item>Create a services site 2015-09-01</Timeline.Item>
          <Timeline.Item>Solve initial network problems 2015-09-
01</Timeline.Item>
          <Timeline.Item dot={<ClockCircleOutlined style={{ fontSize:
'16px' }} />} color="red">
            Technical testing 2015-09-01
          </Timeline.Item>
          <Timeline.Item>Network problems being solved 2015-09-
01</Timeline.Item>
        </Timeline>
        <Calendar />
        <Tree showLine switcherIcon={<DownOutlined />} defaultExpandedKeys=
{['0-0-0']}>
          <TreeNode title="parent 1" key="0-0">
            <TreeNode title="parent 1-0" key="0-0-0">
              <TreeNode title="leaf" key="0-0-0-0" />
              <TreeNode title="leaf" key="0-0-0-1" />
              <TreeNode title="leaf" key="0-0-0-2" />
            </TreeNode>
            <TreeNode title="parent 1-1" key="0-0-1">
              <TreeNode title="leaf" key="0-0-1-0" />
            </TreeNode>
            <TreeNode title="parent 1-2" key="0-0-2">
              <TreeNode title="leaf" key="0-0-2-0" />
              <TreeNode title="leaf" key="0-0-2-1" />
            </TreeNode>
          </TreeNode>
        </Tree>
        <Table<RecordType> columns={columns} dataSource={dataSource}
footer={() => 'Footer'} />
        <Table<DataTableType>
          columns={columnsTable}
          dataSource={summaryDataSource}
          pagination={false}
          id="table-demo-summary"
          bordered
          summary={(pageData) => {
            let totalBorrow = 0;
            let totalRepayment = 0;
            pageData.forEach(({ borrow, repayment }) => {
              totalBorrow += borrow;
              totalRepayment += repayment;
            });
            return (
```

```
              <>
                <tr>
                  <th>Total</th>
                  <td>
                    <Text type="danger">{totalBorrow}</Text>
                  </td>
                  <td>
                    <Text>{totalRepayment}</Text>
                  </td>
                </tr>
                <tr>
                  <th>Balance</th>
                  <td colSpan={2}>
                    <Text type="danger">{totalBorrow - totalRepayment}
</Text>
                  </td>
                </tr>
              </>
            );
          }}
        />
        <br />
        <Table<NestDataType>
          columns={columnsNest}
          expandable={{ expandedRowRender }}
          dataSource={nestDataSource}
        />
        <Table<FixedDataType>
          columns={columnsFixed}
          dataSource={fixedDataSource}
          scroll={{ x: 1300, y: 100 }}
        />
        <Card
          hoverable
          style={{ width: 240 }}
          cover={
            <img
              draggable={false}
              alt="example"

src="https://os.alipayobjects.com/rmsportal/QBnOOoLaAfKPirc.png"
              />
          }
        >
          <Meta title="Europe Street beat" description="www.instagram.com"
/>
```

```
        </Card>
        <Slider defaultValue={30} />
        <DatePicker defaultValue={dayjs('2015/01/01', 'YYYY/MM/DD')}
format="YYYY/MM/DD" />
        <Badge count={5}>
          <a href="#" className="head-example" />
        </Badge>
      </Modal>
    </>
  );
};

export default Demo;
```

Customize footer buttons props

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

const App: React.FC = () => {
  const [open, setOpen] = useState(false);

  const showModal = () => {
    setOpen(true);
  };

  const handleOk = (e: React.MouseEvent<HTMLElement>) => {
    console.log(e);
    setOpen(false);
  };

  const handleCancel = (e: React.MouseEvent<HTMLElement>) => {
    console.log(e);
    setOpen(false);
  };

  return (
    <>
      <Button type="primary" onClick={showModal}>
        Open Modal with customized button props
      </Button>
      <Modal
        title="Basic Modal"
        open={open}
        onOk={handleOk}
        onCancel={handleCancel}
```

```
        okButtonProps={{ disabled: true }}
        cancelButtonProps={{ disabled: true }}
      >
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
      </Modal>
    </>
  );
};

export default App;
```

Custom modal content render

```
import React, { useRef, useState } from 'react';
import { Button, Modal } from 'antd';
import type { DraggableData, DraggableEvent } from 'react-draggable';
import Draggable from 'react-draggable';

const App: React.FC = () => {
  const [open, setOpen] = useState(false);
  const [disabled, setDisabled] = useState(true);
  const [bounds, setBounds] = useState({ left: 0, top: 0, bottom: 0, right:
0 });
  const draggleRef = useRef<HTMLDivElement>(null!);

  const showModal = () => {
    setOpen(true);
  };

  const handleOk = (e: React.MouseEvent<HTMLElement>) => {
    console.log(e);
    setOpen(false);
  };

  const handleCancel = (e: React.MouseEvent<HTMLElement>) => {
    console.log(e);
    setOpen(false);
  };

  const onStart = (_event: DraggableEvent, uiData: DraggableData) => {
    const { clientWidth, clientHeight } = window.document.documentElement;
    const targetRect = draggleRef.current?.getBoundingClientRect();
    if (!targetRect) {
      return;
```

```
    }
    setBounds({
      left: -targetRect.left + uiData.x,
      right: clientWidth - (targetRect.right - uiData.x),
      top: -targetRect.top + uiData.y,
      bottom: clientHeight - (targetRect.bottom - uiData.y),
    });
  };

  return (
    <>
      <Button onClick={showModal}>Open Draggable Modal</Button>
      <Modal
        title={
          <div
            style={{ width: '100%', cursor: 'move' }}
            onMouseOver={() => {
              if (disabled) {
                setDisabled(false);
              }
            }}
            onMouseOut={() => {
              setDisabled(true);
            }}
            // fix eslintjsx-a11y/mouse-events-have-key-events
            // https://github.com/jsx-eslint/eslint-plugin-jsx-
a11y/blob/master/docs/rules/mouse-events-have-key-events.md
            onFocus={() => {}}
            onBlur={() => {}}
            // end
          >
            Draggable Modal
          </div>
        }
        open={open}
        onOk={handleOk}
        onCancel={handleCancel}
        modalRender={(modal) => (
          <Draggable
            disabled={disabled}
            bounds={bounds}
            nodeRef={draggleRef}
            onStart={(event, uiData) => onStart(event, uiData)}
          >
            <div ref={draggleRef}>{modal}</div>
          </Draggable>
```

```
        )}
      >
        <p>
          Just don&apos;t learn physics at school and your life will be
full of magic and miracles.
        </p>
        <br />
        <p>Day before yesterday I saw a rabbit, and yesterday a deer, and
today, you.</p>
      </Modal>
    </>
  );
};

export default App;
```

To customize the width of modal

```
import React, { useState } from 'react';
import { Button, Flex, Modal } from 'antd';

const App: React.FC = () => {
  const [open, setOpen] = useState(false);
  const [openResponsive, setOpenResponsive] = useState(false);

  return (
    <Flex vertical gap="middle" align="flex-start">
      {/* Basic */}
      <Button type="primary" onClick={() => setOpen(true)}>
        Open Modal of 1000px width
      </Button>
      <Modal
        title="Modal 1000px width"
        centered
        open={open}
        onOk={() => setOpen(false)}
        onCancel={() => setOpen(false)}
        width={1000}
      >
        <p>some contents...</p>
        <p>some contents...</p>
        <p>some contents...</p>
      </Modal>

      {/* Responsive */}
      <Button type="primary" onClick={() => setOpenResponsive(true)}>
```

```
        Open Modal of responsive width
      </Button>
      <Modal
        title="Modal responsive width"
        centered
        open={openResponsive}
        onOk={() => setOpenResponsive(false)}
        onCancel={() => setOpenResponsive(false)}
        width={{
          xs: '90%',
          sm: '80%',
          md: '70%',
          lg: '60%',
          xl: '50%',
          xxl: '40%',
        }}
      >
        <p>some contents...</p>
        <p>some contents...</p>
        <p>some contents...</p>
      </Modal>
    </Flex>
  );
};

export default App;
```

Static Method

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

const info = () => {
  Modal.info({
    title: 'This is a notification message',
    content: (
      <div>
        <p>some messages...some messages...</p>
        <p>some messages...some messages...</p>
      </div>
    ),
    onOk() {},
  });
};

const success = () => {
```

```
  Modal.success({
    content: 'some messages...some messages...',
  });
};

const error = () => {
  Modal.error({
    title: 'This is an error message',
    content: 'some messages...some messages...',
  });
};

const warning = () => {
  Modal.warning({
    title: 'This is a warning message',
    content: 'some messages...some messages...',
  });
};

const App: React.FC = () => (
  <Space wrap>
    <Button onClick={info}>Info</Button>
    <Button onClick={success}>Success</Button>
    <Button onClick={error}>Error</Button>
    <Button onClick={warning}>Warning</Button>
  </Space>
);

export default App;
```

Static confirmation

```
import React from 'react';
import { ExclamationCircleFilled } from '@ant-design/icons';
import { Button, Modal, Space } from 'antd';

const { confirm } = Modal;

const showConfirm = () => {
  confirm({
    title: 'Do you want to delete these items?',
    icon: <ExclamationCircleFilled />,
    content: 'Some descriptions',
    onOk() {
      console.log('OK');
    },
```

```
    onCancel() {
      console.log('Cancel');
    },
  });
};

const showPromiseConfirm = () => {
  confirm({
    title: 'Do you want to delete these items?',
    icon: <ExclamationCircleFilled />,
    content: 'When clicked the OK button, this dialog will be closed after
1 second',
    onOk() {
      return new Promise((resolve, reject) => {
        setTimeout(Math.random() > 0.5 ? resolve : reject, 1000);
      }).catch(() => console.log('Oops errors!'));
    },
    onCancel() {},
  });
};

const showDeleteConfirm = () => {
  confirm({
    title: 'Are you sure delete this task?',
    icon: <ExclamationCircleFilled />,
    content: 'Some descriptions',
    okText: 'Yes',
    okType: 'danger',
    cancelText: 'No',
    onOk() {
      console.log('OK');
    },
    onCancel() {
      console.log('Cancel');
    },
  });
};

const showPropsConfirm = () => {
  confirm({
    title: 'Are you sure delete this task?',
    icon: <ExclamationCircleFilled />,
    content: 'Some descriptions',
    okText: 'Yes',
    okType: 'danger',
    okButtonProps: {
```

```
      disabled: true,
    },
    cancelText: 'No',
    onOk() {
      console.log('OK');
    },
    onCancel() {
      console.log('Cancel');
    },
  });
};

const App: React.FC = () => (
  <Space wrap>
    <Button onClick={showConfirm}>Confirm</Button>
    <Button onClick={showPromiseConfirm}>With promise</Button>
    <Button onClick={showDeleteConfirm} type="dashed">
      Delete
    </Button>
    <Button onClick={showPropsConfirm} type="dashed">
      With extra props
    </Button>
  </Space>
);

export default App;
```

Customize className for build-in module

```
import React, { useState } from 'react';
import { Button, ConfigProvider, Modal, Space } from 'antd';
import { createStyles, useTheme } from 'antd-style';

const useStyle = createStyles(({ token }) => ({
  'my-modal-body': {
    background: token.blue1,
    padding: token.paddingSM,
  },
  'my-modal-mask': {
    boxShadow: `inset 0 0 15px #fff`,
  },
  'my-modal-header': {
    borderBottom: `1px dotted ${token.colorPrimary}`,
  },
  'my-modal-footer': {
    color: token.colorPrimary,
```

```
  },
  'my-modal-content': {
    border: '1px solid #333',
  },
}));

const App: React.FC = () => {
  const [isModalOpen, setIsModalOpen] = useState([false, false]);
  const { styles } = useStyle();
  const token = useTheme();

  const toggleModal = (idx: number, target: boolean) => {
    setIsModalOpen((p) => {
      p[idx] = target;
      return [...p];
    });
  };

  const classNames = {
    body: styles['my-modal-body'],
    mask: styles['my-modal-mask'],
    header: styles['my-modal-header'],
    footer: styles['my-modal-footer'],
    content: styles['my-modal-content'],
  };

  const modalStyles = {
    header: {
      borderLeft: `5px solid ${token.colorPrimary}`,
      borderRadius: 0,
      paddingInlineStart: 5,
    },
    body: {
      boxShadow: 'inset 0 0 5px #999',
      borderRadius: 5,
    },
    mask: {
      backdropFilter: 'blur(10px)',
    },
    footer: {
      borderTop: '1px solid #333',
    },
    content: {
      boxShadow: '0 0 30px #999',
    },
  };
```

```
  return (
    <>
      <Space>
        <Button type="primary" onClick={() => toggleModal(0, true)}>
          Open Modal
        </Button>
        <Button type="primary" onClick={() => toggleModal(1, true)}>
          ConfigProvider
        </Button>
      </Space>
      <Modal
        title="Basic Modal"
        open={isModalOpen[0]}
        onOk={() => toggleModal(0, false)}
        onCancel={() => toggleModal(0, false)}
        footer="Footer"
        classNames={classNames}
        styles={modalStyles}
      >
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
      </Modal>
      <ConfigProvider
        modal={{
          classNames,
          styles: modalStyles,
        }}
      >
        <Modal
          title="Basic Modal"
          open={isModalOpen[1]}
          onOk={() => toggleModal(1, false)}
          onCancel={() => toggleModal(1, false)}
          footer="Footer"
        >
          <p>Some contents...</p>
          <p>Some contents...</p>
          <p>Some contents...</p>
        </Modal>
      </ConfigProvider>
    </>
  );
};
```

```
export default App;
```

destroy confirmation modal dialog

```
import React from 'react';
import { ExclamationCircleOutlined } from '@ant-design/icons';
import { Button, Modal } from 'antd';

const { confirm } = Modal;

const destroyAll = () => {
  Modal.destroyAll();
};

const showConfirm = () => {
  for (let i = 0; i < 3; i += 1) {
    setTimeout(() => {
      confirm({
        icon: <ExclamationCircleOutlined />,
        content: <Button onClick={destroyAll}>Click to destroy
all</Button>,
        onOk() {
          console.log('OK');
        },
        onCancel() {
          console.log('Cancel');
        },
      });
    }, i * 500);
  }
};

const App: React.FC = () => <Button onClick={showConfirm}>Confirm</Button>;

export default App;
```

Nested Modal

Debug

```
import React, { useState } from 'react';
import { Button, message, Modal, notification, Select, Space, Switch } from
'antd';

const options = [
```

```
  {
    label: 'Option 1',
    value: '1',
  },
  {
    label: 'Option 2',
    value: '2',
  },
];

const Demo: React.FC = () => {
  const [messageInstance, messageHolder] = message.useMessage();
  const [notificationInstance, notificationHolder] =
notification.useNotification();

  const [isModalOpen, setIsModalOpen] = useState<boolean>(false);

  const onShowStatic = () => {
    Modal.confirm({
      content: <Select open value="1" options={options} />,
    });
  };

  return (
    <Space>
      <Switch
        style={{ position: 'relative', zIndex: isModalOpen ? 4000 : 0 }}
        checkedChildren="Open"
        unCheckedChildren="Close"
        onChange={(open) => setIsModalOpen(open)}
      />
      <Button onClick={onShowStatic}>Static</Button>
      <Modal
        title="Basic Modal"
        open={isModalOpen}
        footer={null}
        destroyOnClose
        onCancel={() => setIsModalOpen(false)}
        maskClosable={false}
        closable={false}
        styles={{
          content: {
            marginBlockStart: 100,
          },
        }}
      >
```

```
<Select open value="1" options={options} />
<Modal
  title="Nested Modal"
  open={isModalOpen}
  footer={null}
  destroyOnClose
  mask={false}
  onCancel={() => setIsModalOpen(false)}
  maskClosable={false}
  closable={false}
  styles={{
    content: {
      marginBlockStart: 250,
    },
    body: {
      display: 'flex',
      justifyContent: 'center',
    },
  }}
>
  <Select open value="1" options={options} />

  <Modal
    title="Nested Modal"
    open={isModalOpen}
    footer={null}
    destroyOnClose
    mask={false}
    maskClosable={false}
    onCancel={() => setIsModalOpen(false)}
    closable={false}
    styles={{
      content: {
        marginBlockStart: 400,
      },
      body: {
        display: 'flex',
        justifyContent: 'flex-end',
      },
    }}
  >
    <Space wrap>
      <Button
        onClick={() => {
          Modal.confirm({
            title: 'Are you OK?',
```

```
                content: 'I am OK',
              });
            }}
          >
            Static Confirm
          </Button>

          <Button
            onClick={() => {
              message.success('Hello World');
              notification.success({
                message: 'Hello World',
              });
            }}
          >
            Static Message, Notification
          </Button>

          <Button
            onClick={() => {
              messageInstance.success('Hello World');
              notificationInstance.success({
                message: 'Hello World',
              });
            }}
          >
            Hook Message, Notification
          </Button>

          <Select open value="1" options={options} />
        </Space>
      </Modal>
    </Modal>
  </Modal>

  {messageHolder}
  {notificationHolder}
</Space>
);
};

export default Demo;
```

_InternalPanelDoNotUseOrYouWillBeFired

Debug

```tsx
import React from 'react';
import { Button, Modal, Space, Typography } from 'antd';
import type { ModalFuncProps } from 'antd';

/** Test usage. Do not use in your production. */
const { _InternalPanelDoNotUseOrYouWillBeFired: InternalPanel } = Modal;

const customFooterFn: ModalFuncProps['footer'] = (originNode, { OkBtn,
CancelBtn }) => (
  <Space direction="vertical">
    <Space>{originNode}</Space>
    <Space>
      <CancelBtn />
      <Button danger type="primary">
        Custom
      </Button>
      <OkBtn />
    </Space>
  </Space>
);

export default () => (
  <div style={{ display: 'flex', flexDirection: 'column', rowGap: 16 }}>
    <InternalPanel title="Hello World!" style={{ width: '100%', height: 200
}}>
      Hello World?!
    </InternalPanel>
    <InternalPanel type="success" style={{ width: 200, height: 150 }}>
      A good news!
    </InternalPanel>
    <InternalPanel title="Confirm This?" type="confirm" style={{ width:
300, height: 200 }}>
      Some descriptions.
    </InternalPanel>

    <InternalPanel
      title="Custom Footer Render"
      style={{ width: 380, height: 200 }}
      footer={customFooterFn}
    >
      <Typography.Paragraph>
        <Typography.Link href="https://github.com/ant-design/ant-
design/pull/44318">
          Feature #44318
        </Typography.Link>
      </Typography.Paragraph>
```

```
    </InternalPanel>
  </div>
);
```

Control modal's animation origin position

Debug

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

const App: React.FC = () => {
  const [isModalOpen, setIsModalOpen] = useState(false);

  const showModal = () => {
    setIsModalOpen(true);
  };

  const handleOk = () => {
    setIsModalOpen(false);
  };

  const handleCancel = () => {
    setIsModalOpen(false);
  };

  return (
    <>
      <Button type="primary" onClick={showModal}>
        Open Modal
      </Button>
      <Modal
        title="Basic Modal"
        open={isModalOpen}
        onOk={handleOk}
        onCancel={handleCancel}
        mousePosition={{ x: 300, y: 300 }}
      >
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
      </Modal>
    </>
  );
};
```

```
export default App;
```

Wireframe

Debug

```
import React from 'react';
import { ConfigProvider, Modal } from 'antd';

/** Test usage. Do not use in your production. */
const { _InternalPanelDoNotUseOrYouWillBeFired: InternalPanel } = Modal;

export default () => (
  <ConfigProvider theme={{ token: { wireframe: true } }}>
    <div style={{ display: 'flex', flexDirection: 'column', rowGap: 16 }}>
      <InternalPanel title="Hello World!" style={{ width: '100%' }}>
        Hello World?!
      </InternalPanel>
      <InternalPanel type="success" style={{ width: 200 }}>
        A good news!
      </InternalPanel>
      <InternalPanel title="Confirm This?" type="confirm" style={{ width:
300 }}>
        Some descriptions.
      </InternalPanel>
    </div>
  </ConfigProvider>
);
```

Component Token

Debug

```
import React from 'react';
import { ConfigProvider, Modal } from 'antd';

/** Test usage. Do not use in your production. */
const { _InternalPanelDoNotUseOrYouWillBeFired: InternalPanel } = Modal;

export default () => (
  <ConfigProvider
    theme={{
      components: {
        Modal: {
          footerBg: '#f6ffed',
```

```
          contentBg: '#e6fffb',
          headerBg: '#f9f0ff',
          titleLineHeight: 3,
          titleFontSize: 12,
          titleColor: '#1d39c4',
        },
      },
    }}
  >
    <div style={{ display: 'flex', flexDirection: 'column', rowGap: 16 }}>
      <InternalPanel title="Hello World!" style={{ width: '100%' }}>
        Hello World?!
      </InternalPanel>
      <ConfigProvider theme={{ token: { wireframe: true } }}>
        <InternalPanel title="Hello World!" style={{ width: '100%' }}>
          Hello World?!
        </InternalPanel>
      </ConfigProvider>
      <InternalPanel type="success" style={{ width: 200 }}>
        A good news!
      </InternalPanel>
      <InternalPanel title="Confirm This?" type="confirm" style={{ width:
300 }}>
        Some descriptions.
      </InternalPanel>
    </div>
  </ConfigProvider>
);
```

API

Common props ref：Common props

Property	Description	Type	Default	Ver
afterClose	Specify a function that will be called when modal is closed completely	function	-	
classNames	Config Modal build-in	Record<SemanticDOM, string>	-	

	module's className			
styles	Config Modal build-in module's style	Record<SemanticDOM, CSSProperties>	-	5.10.0
cancelButtonProps	The cancel button props	ButtonProps	-	
cancelText	Text of the Cancel button	ReactNode	`Cancel`	
centered	Centered Modal	boolean	false	
closable	Whether a close (x) button is visible on top right or not	boolean | { closeIcon?: React.ReactNode; disabled?: boolean; }	true	
closeIcon	Custom close icon. 5.7.0: close button will be hidden when setting to `null` or `false`	ReactNode	<CloseOutlined />	
confirmLoading	Whether to apply loading visual effect for OK button or not	boolean	false	
destroyOnClose	Whether to unmount child	boolean	false	

	components on onClose			
focusTriggerAfterClose	Whether need to focus trigger element after dialog is closed	boolean	true	4.9.0
footer	Footer content, set as `footer={null}` when you don't need default buttons	ReactNode | (originNode: ReactNode, extra: { OkBtn: React.FC, CancelBtn: React.FC }) => ReactNode	(OK and Cancel buttons)	renderF 5.9.0
forceRender	Force render Modal	boolean	false	
getContainer	The mounted node for Modal but still display at fullscreen	HTMLElement | () => HTMLElement | Selectors | false	document.body	
keyboard	Whether support press esc to close	boolean	true	
mask	Whether show mask or not	boolean	true	
maskClosable	Whether to close the modal dialog when the mask (area outside the	boolean	true	

	modal) is clicked			
modalRender	Custom modal content render	(node: ReactNode) => ReactNode	-	4.7.0
okButtonProps	The ok button props	ButtonProps	-	
okText	Text of the OK button	ReactNode	OK	
okType	Button type of the OK button	string	`primary`	
style	Style of floating layer, typically used at least for adjusting the position	CSSProperties	-	
loading	Show the skeleton	boolean		5.18.0
title	The modal dialog's title	ReactNode	-	
open	Whether the modal dialog is visible or not	boolean	false	
width	Width of the modal dialog	string | number | Breakpoint	520	Breakp 5.23.0
wrapClassName	The class name of the container of the modal dialog	string	-	

zIndex	The z-index of the Modal	number	1000	
onCancel	Specify a function that will be called when a user clicks mask, close button on top right or Cancel button	function(e)	-	
onOk	Specify a function that will be called when a user clicks the OK button	function(e)	-	
afterOpenChange	Callback when the animation ends when Modal is turned on and off	(open: boolean) => void	-	5.4.0

Note

- The state of Modal will be preserved at it's component lifecycle by default, if you wish to open it with a brand new state every time, set `destroyOnClose` on it.
- There is a situation that using `<Modal />` with Form, which won't clear fields value when closing Modal even you have set `destroyOnClose`. You need `<Form preserve={false} />` in this case.
- `Modal.method()` RTL mode only supports hooks.

Modal.method()

There are five ways to display the information based on the content's nature:

- `Modal.info`
- `Modal.success`
- `Modal.error`
- `Modal.warning`

- `Modal.confirm`

The items listed above are all functions, expecting a settings object as parameter. The properties of the object are follows:

Property	Description	Type	Default	Vers
afterClose	Specify a function that will be called when modal is closed completely	function	-	4.9.0
autoFocusButton	Specify which button to autofocus	null | ok | `cancel`	`ok`	
cancelButtonProps	The cancel button props	ButtonProps	-	
cancelText	Text of the Cancel button with Modal.confirm	string	`Cancel`	
centered	Centered Modal	boolean	false	
className	The className of container	string	-	
closable	Whether a close (x) button is visible on top right of the confirm dialog or not	boolean	false	4.9.0
closeIcon	Custom close icon	ReactNode	undefined	4.9.0
content	Content	ReactNode	-	
footer	Footer content, set as `footer: null` when	ReactNode | (originNode: ReactNode, extra: { OkBtn: React.FC,	-	renderFu 5.9.0

	you don't need default buttons	CancelBtn: React.FC }) => ReactNode		
getContainer	Return the mount node for Modal	HTMLElement | () => HTMLElement | Selectors | false	document.body	
icon	Custom icon	ReactNode	<ExclamationCircleFilled />	
keyboard	Whether support press esc to close	boolean	true	
mask	Whether show mask or not.	boolean	true	
maskClosable	Whether to close the modal dialog when the mask (area outside the modal) is clicked	boolean	false	
okButtonProps	The ok button props	ButtonProps	-	
okText	Text of the OK button	string	OK	
okType	Button `type` of the OK button	string	`primary`	
style	Style of floating layer, typically used at least for adjusting the position	CSSProperties	-	
title	Title	ReactNode	-	
width	Width of the modal dialog	string | number	416	

wrapClassName	The class name of the container of the modal dialog	string	-	4.18.0
zIndex	The `z-index` of the Modal	number	1000	
onCancel	Click to onCancel the callback, the parameter is the closing function, if it returns a promise, resolve means normal closing, reject means not closing	function(close)	-	
onOk	Click to onOk the callback, the parameter is the closing function, if it returns a promise, resolve means normal closing, reject means not closing	function(close)	-	

All the `Modal.method`s will return a reference, and then we can update and close the modal dialog by the reference.

```
const modal = Modal.info();

modal.update({
  title: 'Updated title',
  content: 'Updated content',
});

// on 4.8.0 or above, you can pass a function to update modal
```

```
modal.update((prevConfig) => ({
  ...prevConfig,
  title: `${prevConfig.title} (New)`,
}));

modal.destroy();
```

- `Modal.destroyAll`

`Modal.destroyAll()` could destroy all confirmation modal dialogs(`Modal.confirm|success|info|error|warning`). Usually, you can use it in router change event to destroy confirm modal dialog automatically without use modal reference to close(it's too complex to use for all modal dialogs)

```
import { browserHistory } from 'react-router';

// router change
browserHistory.listen(() => {
  Modal.destroyAll();
});
```

Modal.useModal()

When you need using Context, you can use `contextHolder` which created by `Modal.useModal` to insert into children. Modal created by hooks will get all the context where `contextHolder` are. Created `modal` has the same creating function with `Modal.method` .

```
const [modal, contextHolder] = Modal.useModal();

React.useEffect(() => {
  modal.confirm({
    // ...
  });
}, []);

return <div>{contextHolder}</div>;
```

`modal.confirm` return method:

- `destroy` : Destroy current modal
- `update` : Update current modal
- `then` : (Hooks only) Promise chain call, support `await` operation

```
// Return `true` when click `onOk` and `false` when click `onCancel`
const confirmed = await modal.confirm({ ... });
```

Semantic DOM

演示

```tsx
import React from 'react';
import type { ModalProps } from 'antd';
import { Modal } from 'antd';

import SemanticPreview from '../../../.dumi/components/SemanticPreview';
import useLocale from '../../../.dumi/hooks/useLocale';

const locales = {
  cn: {
    mask: '遮罩层元素',
    wrapper: '包裹层元素，一般用于动画容器',
    content: 'Modal 容器元素',
    header: '头部元素',
    body: '内容元素',
    footer: '底部元素',
  },
  en: {
    mask: 'Mask element',
    wrapper: 'Wrapper element. Used for motion container',
    content: 'Modal container element',
    header: 'Header element',
    body: 'Body element',
    footer: 'Footer element',
  },
};

const BlockModal = (props: ModalProps) => {
  const divRef = React.useRef<HTMLDivElement>(null);

  return (
    <div ref={divRef} style={{ position: 'absolute', inset: 0 }}>
      <Modal
        getContainer={() => divRef.current!}
        {...props}
        styles={{
          mask: {
            position: 'absolute',
            zIndex: 1,
          },
          wrapper: {
            position: 'absolute',
            zIndex: 1,
```

```
          },
        }}
        style={{
          top: '50%',
          transform: 'translateY(-50%)',
          marginBottom: 0,
          paddingBottom: 0,
        }}
      />
    </div>
  );
};

const App: React.FC = () => {
  const [locale] = useLocale(locales);
  return (
    <SemanticPreview
      componentName="Modal"
      semantics={[
        { name: 'mask', desc: locale.mask, version: '5.13.0' },
        { name: 'content', desc: locale.content, version: '5.13.0' },
        { name: 'wrapper', desc: locale.wrapper, version: '5.13.0' },
        { name: 'header', desc: locale.header, version: '5.13.0' },
        { name: 'body', desc: locale.body, version: '5.13.0' },
        { name: 'footer', desc: locale.footer, version: '5.13.0' },
      ]}
    >
      <BlockModal title="Title" closable={false} open getContainer={false}
width={400}>
        <p>Some contents...</p>
      </BlockModal>
    </SemanticPreview>
  );
};

export default App;
```

Design Token

FAQ

Why content not update when Modal closed?

Modal will use memo to avoid content jumping when closed. Also, if you use Form in Modal, you can reset `initialValues` by calling `resetFields` in effect.

Why I can not access context, redux, ConfigProvider `locale/prefixCls` in Modal.xxx?

antd will dynamic create React instance by `ReactDOM.render` when call Modal methods. Whose context is different with origin code located context.

When you need context info (like ConfigProvider context), you can use `Modal.useModal` to get `modal` instance and `contextHolder` node. And put it in your children:

```
const [modal, contextHolder] = Modal.useModal();

// then call modal.confirm instead of Modal.confirm

return (
  <Context1.Provider value="Ant">
    {/* contextHolder is in Context1, which means modal will get context of
Context1 */}
    {contextHolder}
    <Context2.Provider value="Design">
      {/* contextHolder is out of Context2, which means modal will not get
context of Context2 */}
    </Context2.Provider>
  </Context1.Provider>
);
```

Note: You must insert `contextHolder` into your children with hooks. You can use origin method if you do not need context connection.

> *App Package Component can be used to simplify the problem of `useModal` and other methods that need to manually implant contextHolder.*

How to set static methods prefixCls ?

You can config with `ConfigProvider.config`