Modal

Modal dialogs.

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 show a simple confirmation dialog, you can use `App.useApp` hooks.

Examples

Open Modal

Basic

Basic modal.

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

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

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

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

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

  return (
    <>
      <Button type="primary" onClick={showModa
        Open Modal
      </Button>
      <Modal title="Basic Modal" open={isModal
        <p>Some contents...</p>
        <p>Some contents...</p>
        <p>Some contents...</p>
      </Modal>
    </>
  );
};

export default App;
```

Open Modal with async logic

Asynchronously close

Asynchronously close a modal dialog when the OK button is pressed. For example, you can use this pattern when you submit a form.

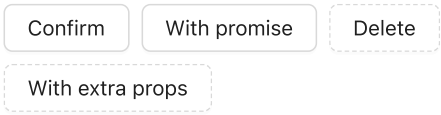
```
import React, { useState } from 'react';
import { Button, Modal } from 'antd';

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

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

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

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

  return (
    <>
      <Button type="primary" onClick={showModa
        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

A more complex example which define a customized footer button bar. The dialog will change to loading state after clicking the submit button, and when the loading is done, the modal dialog will be closed.

You could set `footer` to `null` if you don't need default footer buttons.

```
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={showModa
        Open Modal with customized footer
      </Button>
      <Modal
        open={open}
        title="Title"
        onOk={handleOk}
        onCancel={handleCancel}
        footer={[
          <Button key="back" onClick={handleCa
            Return
          </Button>,
          <Button key="submit" type="primary"
            Submit
          </Button>,
          <Button
            key="link"
            href="https://google.com"
            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>
```



Confirmation modal dialog

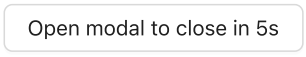
Use `confirm()` to show a confirmation modal dialog. Let onCancel/onOk function return a promise object to delay closing the dialog.

```
import React from 'react';
import { ExclamationCircleFilled } from '@ant-
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
    onOk() {
      return new Promise((resolve, reject) =>
        setTimeout(Math.random() > 0.5 ? resol
      }).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: {
```

Internationalization

To customize the text of the buttons, you need to set `okText` and `cancelText` props.

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

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

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

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

  return (
    <>
      <Button type="primary" onClick={showModa
        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.useModa

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

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

export default App;
```



Manual to update destroy

Manually updating and destroying a modal through instance.

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

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

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

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

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

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

  return (
    <>
      <Button onClick={countDown}>Open modal t
      {contextHolder}
    </>
  );
};

export default App;
```

Display a modal dialog at 20px to Top

Vertically centered modal dialog

To customize the position of modal

You can use `centered`, `style.top` or other styles to set position of modal dialog.

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

const App: React.FC = () => {
  const [modal1Open, setModal1Open] = useState
  const [modal2Open, setModal2Open] = useState

  return (
    <>
      <Button type="primary" onClick={() => se
        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={() => se
        Vertically centered modal dialog
      </Button>
      <Modal
        title="Vertically centered modal dialo
        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;
```

Open Modal with customized button props

Customize footer buttons props

Passing `okButtonProps` and `cancelButtonProps` will customize the OK button and cancel button 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<HTMLE
    console.log(e);
    setOpen(false);
  };

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

  return (
    <>
      <Button type="primary" onClick={showModa
        Open Modal with customized button prop
      </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;
```

Confirm | Warning | Info | Error

Use hooks to get context

Use `Modal.useModal` to get `contextHolder` with context accessible issue.

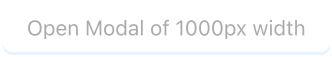
```
import React, { createContext } from 'react';
import { Button, Modal, Space } from 'antd';

const ReachableContext = createContext<string
const UnreachableContext = createContext<strir

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

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

  return (
    <ReachableContext.Provider value="Light">
      <Space>
        <Button
          onClick={() => {
            modal.confirm(config);
          }}
        >
          Confirm
        </Button>
        <Button
          onClick={() => {
            modal.warning(config);
          }}
        >
          Warning
        </Button>
        <Button
          onClick={() => {
            modal.info(config);
          }}
        >
          Info
        </Button>
        <Button
          onClick={() => {
            modal.error(config);
          }}
        >
          Error
        </Button>
      </Space>
      {/* `contextHolder` should always be pla
      {contextHolder}

      {/* Can not access this context since `c
      <UnreachableContext.Provider value="Bamb
    </ReachableContext.Provider>
  );
};

export default App;
```

Open Draggable Modal

Custom modal content render

Custom modal content render. use `react-draggable` implements draggable.

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

const App: React.FC = () => {
  const [open, setOpen] = useState(false);
  const [disabled, setDisabled] = useState(tru
  const [bounds, setBounds] = useState({ left:
  const draggleRef = useRef<HTMLDivElement>(nu

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

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

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

  const onStart = (_event: DraggableEvent, uiD
    const { clientWidth, clientHeight } = winc
    const targetRect = draggleRef.current?.get
    if (!targetRect) {
      return;
    }
    setBounds({
      left: -targetRect.left + uiData.x,
      right: clientWidth - (targetRect.right -
      top: -targetRect.top + uiData.y,
      bottom: clientHeight - (targetRect.bottc
    });
  };

  return (
    <>
      <Button onClick={showModal}>Open Draggab
      <Modal
        title={
          <div
            style={{
              width: '100%',
              cursor: 'move',
            }}
            onMouseOver={() => {
              if (disabled) {
                setDisabled(false);
              }
            }}
            onMouseOut={() => {
              setDisabled(true);
            }}
            // fix eslintjsx-a11y/mouse-events
            // https://github.com/jsx-eslint/e
            onFocus={() => {}}
            onBlur={() => {}}
            // end
          >
```



To customize the width of modal

Use `width` to set the width of the modal dialog.

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

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

  return (
    <>
      <Button type="primary" onClick={() => se
        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>
    </>
  );
};

export default App;
```

Static Method

In most case, you do not need static method. It can not consume context like dynamic theme. Please use hooks version or `App` provided instance first.

```
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;
```

Confirm

destroy confirmation modal dialog

`Modal.destroyAll()` will destroy all confirmation modal dialogs. Usually, you can use it in router change event to destroy confirm modal dialog automatically.

```
import React from 'react';
import { ExclamationCircleOutlined } from '@ar
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}>
        onOk() {
          console.log('OK');
        },
        onCancel() {
          console.log('Cancel');
        },
      });
    }, i * 500);
  }
};

const App: React.FC = () => <Button onClick={s

export default App;
```

API

Property	Description	Type	Default
afterClose	Specify a function that will be called when modal is closed completely	function	-
bodyStyle	Body style for modal body element. Such as height, padding etc	CSSProperties	
cancelButtonProps	The cancel button props	ButtonProps	-
cancelText	Text of the Cancel button	ReactNode	`Cancel`
centered	Centered Modal	boolean	false

Property	Description	Type	Default	V
closable	Whether a close (x) button is visible on top right of the modal dialog or not	boolean	true	
closeIcon	Custom close icon	ReactNode	<CloseOutlined />	
confirmLoading	Whether to apply loading visual effect for OK button or not	boolean	false	
destroyOnClose	Whether to unmount child components on onClose	boolean	false	
focusTriggerAfterClose	Whether need to focus trigger element after dialog is closed	boolean	true	4
footer	Footer content, set as `footer={null}` when you don't need default buttons	ReactNode	(OK and Cancel buttons)	
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 modal) is clicked	boolean	true	
maskStyle	Style for modal's mask element	CSSProperties		
modalRender	Custom modal content render	(node: ReactNode) => ReactNode	-	4
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	CSSProperties	-	

Property	Description	Type	Default	V
	used at least for adjusting the position			
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	520	
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	-	

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
afterClose	Specify a function that will be called when modal is closed completely	function	-
autoFocusButton	Specify which button to autofocus	null | `ok` | `cancel`	`ok`
bodyStyle	Body style for modal body element. Such as height, padding etc	CSSProperties	
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
closeIcon	Custom close icon	ReactNode	undefined
content	Content	ReactNode	-
footer	Footer content, set as `footer: null` when you don't need default buttons	ReactNode	-
getContainer	Return the mount node for Modal	HTMLElement | () => HTMLElement | Selectors | false	document.body
icon	Custom icon	ReactNode	<ExclamationCircleFille />
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
maskStyle	Style for modal's mask element	object	{}

Property	Description	Type	Default
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	-
zIndex	The `z-index` of the Modal	number	1000
onCancel	Specify a function that will be called when the user clicks the Cancel button. The parameter of this function is a function whose execution should include closing the dialog. If the function does not take any parameter (`!onCancel.length`) then modal dialog will be closed unless returned value is `true` (`!!onCancel()`). You can also just return a promise and when the promise is resolved, the modal dialog will also be closed	function(close)	-
onOk	Specify a function that will be called when the user clicks the OK button. The parameter of this function is a function whose execution should include closing the dialog. If the function does not	function(close)	-

Property	Description	Type	Default
	take any parameter (`!onOk.length`) then modal dialog will be closed unless returned value is `true` (`!!onOk()`). You can also just return a promise and when the promise is resolved, the modal dialog will also be closed		

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>;
```

Design Token

▼ Global Token

Token Name	Description	Type	Default Value
colorBgElevated	Container background color of the popup layer, in dark mode the color value of this token will be a little brighter than `colorBgContainer`. E.g: modal, pop-up, menu, etc.	`string`	`#ffffff`
colorBgMask	The background color of the mask, used to cover the content below the mask, Modal, Drawer and other components use this token	`string`	`rgba(0, 0, 0, 0.45)`
colorError	Used to represent the visual elements of the operation failure, such as the error Button, error Result component, etc.	`string`	`#ff4d4f`
colorIconHover	Weak action hover color. Such as `allowClear` or Alert close button	`string`	`rgba(0, 0, 0, 0.88)`
colorInfo	Used to represent the operation information of the Token sequence, such as Alert, Tag, Progress, and other components use these map tokens.	`string`	`#1677ff`
colorPrimaryBorder	The stroke color under the main color gradient, used on the stroke of components such as Slider.	`string`	`#91caff`
colorSplit	Used as the color of separator, this color is the same as colorBorderSecondary but with transparency.	`string`	`rgba(5, 5, 5, 0.06)`
colorSuccess	Used to represent the token sequence of operation success, such as Result, Progress and other components will use these map tokens.	`string`	`#52c41a`
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)`
colorTextDescription	Control the font color of text description.	`string`	`rgba(0, 0, 0, 0.45)`
colorTextHeading	Control the font color of heading.	`string`	`rgba(0, 0, 0, 0.88)`
colorWarning	Used to represent the warning map token, such as Notification, Alert, etc.	`string`	`#faad14`

Token Name	Description	Type	Default Value
	Alert or Control component(like Input) will use these map tokens.		
borderRadiusLG	LG size border radius, used in some large border radius components, such as Card, Modal and other components.	`number`	8
borderRadiusSM	SM size border radius, used in small size components, such as Button, Input, Select and other input components in small size	`number`	4
boxShadow	Control the box shadow style of an element.	`string`	0 6px 16px 0 rgba(0, 0, 0, 0.08), 0 3px 6px -4px rgba(0, 0, 0, 0.12), 0 9px 28px 8px rgba(0, 0, 0, 0.05)
controlHeightLG	LG component height	`number`	40
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
fontSizeHeading5	Font size of h5 tag.	`number`	16
fontSizeLG	Large font size	`number`	16
fontWeightStrong	Control the font weight of heading components (such as h1, h2, h3) or selected item.	`number`	600
lineHeight	Line height of text.	`number`	1.5714285714285714
lineHeightHeading5	Line height of h5 tag.	`number`	1.5
lineType	Border style of base components	`string`	solid
lineWidth	Border width of base components	`number`	1
lineWidthFocus	Control the width of the line when the component is in focus state.	`number`	4
margin	Control the margin of an element, with a medium size.	`number`	16
marginLG	Control the margin of an element, with a large size.	`number`	24
marginSM	Control the margin of an element, with a medium-small size.	`number`	12

Token Name	Description	Type	Default Value
marginXS	Control the margin of an element, with a small size.	`number`	8
motionDurationMid	Motion speed, medium speed. Used for medium element animation interaction.	`string`	0.2s
motionDurationSlow	Motion speed, slow speed. Used for large element animation interaction.	`string`	0.3s
motionEaseInOutCirc	Preset motion curve.	`string`	cubic-bezier(0.78, 0.14, 0.15, 0.86)
motionEaseOutCirc	Preset motion curve.	`string`	cubic-bezier(0.08, 0.82, 0.17, 1)
padding	Control the padding of the element.	`number`	16
paddingContentHorizontalLG	Control the horizontal padding of content element, suitable for large screen devices.	`number`	24
paddingLG	Control the large padding of the element.	`number`	24
paddingMD	Control the medium padding of the element.	`number`	20
paddingXS	Control the extra small padding of the element.	`number`	8
screenSMMax	Control the maximum width of small screens.	`number`	767
wireframe	Used to change the visual effect of the component to wireframe, if you need to use the V4 effect, you need to enable the configuration item	`boolean`	false
zIndexPopupBase	Base zIndex of component like FloatButton, Affix which can be cover by large popup	`number`	1000

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 disable motion?

You can config `transitionName=""` and `maskTransitionName=""` to remove motion class. But you should note that these prop is internal usage which we don't promise exist in next major version.

How to set static methods prefixCls ?

You can config with `ConfigProvider.config`