Message

Display global messages as feedback in response to user operations.

When To Use

- To provide feedback such as success, warning, error etc.
- A message is displayed at top and center and will be dismissed automatically, as a non-interrupting light-weighted prompt.

Examples

Display normal message

Hooks usage (recommended)

Use `message.useMessage` to get `contextHolder` with context accessible issue. Please note that, we recommend to use top level registration instead of `message` static method, because static method cannot consume context, and ConfigProvider data will not work.

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

const App: React.FC = () => {
  const [messageApi, contextHolder] = message.

  const info = () => {
    messageApi.info('Hello, Ant Design!');
  };

  return (
    <>
      {contextHolder}
      <Button type="primary" onClick={info}>
        Display normal message
      </Button>
    </>
  );
};

export default App;
```

Success Error Warning

Other types of message

Messages of success, error and warning types.

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

const App: React.FC = () => {
  const [messageApi, contextHolder] = message.

  const success = () => {
    messageApi.open({
      type: 'success',
      content: 'This is a success message',
    });
  };

  const error = () => {
    messageApi.open({
      type: 'error',
      content: 'This is an error message',
    });
  };

  const warning = () => {
    messageApi.open({
      type: 'warning',
      content: 'This is a warning message',
    });
  };

  return (
    <>
      {contextHolder}
      <Space>
        <Button onClick={success}>Success</Bu
        <Button onClick={error}>Error</Button>
        <Button onClick={warning}>Warning</Bu
      </Space>
    </>
  );
};

export default App;
```

Customized display duration

Customize duration

Customize message display duration from default `3s` to `10s`.

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

const App: React.FC = () => {
  const [messageApi, contextHolder] = message.

  const success = () => {
    messageApi.open({
      type: 'success',
      content: 'This is a prompt message for s
      duration: 10,
    });
  };

  return (
    <>
      {contextHolder}
      <Button onClick={success}>Customized dis
    </>
  );
};

export default App;
```

Display a loading indicator

Message with loading indicator

Display a global loading indicator, which is dismissed by itself asynchronously.

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

const App: React.FC = () => {
  const [messageApi, contextHolder] = message.

  const success = () => {
    messageApi.open({
      type: 'loading',
      content: 'Action in progress..',
      duration: 0,
    });
    // Dismiss manually and asynchronously
    setTimeout(messageApi.destroy, 2500);
  };
  return (
    <>
      {contextHolder}
      <Button onClick={success}>Display a loa
    </>
  );
};

export default App;
```

Display sequential messages

Promise interface

`message` provides a promise interface for `onClose`. The above example will display a new message when the old message is about to close.

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

const App: React.FC = () => {
  const [messageApi, contextHolder] = message.

  const success = () => {
    messageApi
      .open({
        type: 'loading',
        content: 'Action in progress..',
        duration: 2.5,
      })
      .then(() => message.success('Loading fir
      .then(() => message.info('Loading finist
  };

  return (
    <>
      {contextHolder}
      <Button onClick={success}>Display sequer
    </>
  );
};

export default App;
```

Customized style

Customized style

The `style` and `className` are available to customize Message.

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

const App: React.FC = () => {
  const [messageApi, contextHolder] = message.

  const success = () => {
    messageApi.open({
      type: 'success',
      content: 'This is a prompt message with
      className: 'custom-class',
      style: {
        marginTop: '20vh',
      },
    });
  };

  return (
    <>
      {contextHolder}
      <Button onClick={success}>Customized sty
    </>
  );
};

export default App;
```

Display normal message

Normal prompt

Normal message for information.

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

const info = () => {
  message.info('This is a normal message');
};

const App: React.FC = () => (
  <Button type="primary" onClick={info}>
    Display normal message
  </Button>
);

export default App;
```

Open the message box

Update Message Content

Update message content with unique `key`.

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

const App: React.FC = () => {
  const [messageApi, contextHolder] = message.
  const key = 'updatable';

  const openMessage = () => {
    messageApi.open({
      key,
      type: 'loading',
      content: 'Loading...',
    });
    setTimeout(() => {
      messageApi.open({
        key,
        type: 'success',
        content: 'Loaded!',
        duration: 2,
      });
    }, 1000);
  };

  return (
    <>
      {contextHolder}
      <Button type="primary" onClick={openMess
        Open the message box
      </Button>
    </>
  );
};

export default App;
```

API

This components provides some static methods, with usage and arguments as following:

- `message.success(content, [duration], onClose)`
- `message.error(content, [duration], onClose)`
- `message.info(content, [duration], onClose)`
- `message.warning(content, [duration], onClose)`
- `message.loading(content, [duration], onClose)`

Argument	Description	Type	Default
content	The content of the message	ReactNode | config	-
duration	Time(seconds) before auto-dismiss, don't dismiss if set to 0	number	1.5
onClose	Specify a function that will be called when the message is closed	function	-

`afterClose` can be called in thenable interface:

- `message[level](content, [duration]).then(afterClose)`
- `message[level](content, [duration], onClose).then(afterClose)`

where `level` refers one static methods of `message`. The result of `then` method will be a Promise.

Supports passing parameters wrapped in an object:

- `message.open(config)`
- `message.success(config)`
- `message.error(config)`
- `message.info(config)`
- `message.warning(config)`
- `message.loading(config)`

The properties of config are as follows:

Property	Description	Type	Default
className	Customized CSS class	string	-
content	The content of the message	ReactNode	-
duration	Time(seconds) before auto-dismiss, don't dismiss if set to 0	number	3
icon	Customized Icon	ReactNode	-
key	The unique identifier of the Message	string | number	-
style	Customized inline style	[CSSProperties]()	-
onClick	Specify a function that will be called when the message is clicked	function	-
onClose	Specify a function that will be called when the message is closed	function	-

Global static methods

Methods for global configuration and destruction are also provided:

- `message.config(options)`
- `message.destroy()`

> use `message.destroy(key)` to remove a message。

message.config

> When you use `ConfigProvider` for global configuration, the system will automatically start RTL mode by default.(4.3.0+)
>
> When you want to use it alone, you can start the RTL mode through the following settings.

```
message.config({
  top: 100,
  duration: 2,
  maxCount: 3,
  rtl: true,
  prefixCls: 'my-message',
});
```

Argument	Description	Type	Default	Version
duration	Time before auto-dismiss, in seconds	number	3	
getContainer	Return the mount node for Message	() => HTMLElement	() => document.body	
maxCount	Max message show, drop oldest if exceed limit	number	-	
prefixCls	The prefix className of message node	string	`ant-message`	4.5.0
rtl	Whether to enable RTL mode	boolean	false	
top	Distance from top	number	8	

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
colorError	Used to represent the visual elements of the operation failure, such as the error Button, error Result component, etc.	`string`	#ff4d4f
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
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)
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
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
fontSizeLG	Large font size	`number`	16
lineHeight	Line height of text.	`number`	1.5714285714285714
marginXS	Control the margin of an element, with a small size.	`number`	8
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)
paddingSM	Control the small padding of the element.	`number`	12
paddingXS	Control the extra small padding of the element.	`number`	8
zIndexPopupBase	Base zIndex of component like FloatButton, Affix which can be cover by large popup	`number`	1000

FAQ

Why I can not access context, redux, ConfigProvider `locale/prefixCls/theme` in message?

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

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

```
const [api, contextHolder] = message.useMessage();

return (
  <Context1.Provider value="Ant">
    {/* contextHolder is inside Context1 which means api will get value of Context1 */}
    {contextHolder}
```

```
      <Context2.Provider value="Design">
        {/* contextHolder is outside Context2 which means api will **not** get value 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 `useMessage` and other methods that need to manually implant contextHolder.

How to set static methods prefixCls ?

You can config with `ConfigProvider.config`