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

Hooks usage (recommended)

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

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

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

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

export default App;
```

Other types of message

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

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

  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</Button>
        <Button onClick={error}>Error</Button>
        <Button onClick={warning}>Warning</Button>
      </Space>
    </>
  );
};

export default App;
```

Customize duration

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

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

  const success = () => {
    messageApi.open({
      type: 'success',
      content: 'This is a prompt message for success, and it will disappear
in 10 seconds',
      duration: 10,
    });
  };
```

```
  return (
    <>
      {contextHolder}
      <Button onClick={success}>Customized display duration</Button>
    </>
  );
};

export default App;
```

Message with loading indicator

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

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

  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 loading indicator</Button>
    </>
  );
};

export default App;
```

Promise interface

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

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

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

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

export default App;
```

Customized style

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

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

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

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

```
};

export default App;
```

Update Message Content

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

const App: React.FC = () => {
  const [messageApi, contextHolder] = message.useMessage();
  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={openMessage}>
        Open the message box
      </Button>
    </>
  );
};

export default App;
```

Static method (deprecated)

```
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}>
    Static Method
  </Button>
);

export default App;
```

_InternalPanelDoNotUseOrYouWillBeFired

Debug

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

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

export default () => <InternalPanel content="Hello World!" type="error" />;
```

Component Token

Debug

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

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

export default () => (
  <>
    <ConfigProvider
      theme={{
        components: {
          Message: {
            contentPadding: 40,
            contentBg: '#e6f4ff',
          },
        },
      }}
```

```
    >
      <InternalPanel content="Hello World!" type="error" />
    </ConfigProvider>
    <ConfigProvider
      theme={{
        components: {
          Message: {
            colorBgElevated: '#e6f4ff',
          },
        },
      }}
    >
      <InternalPanel content="Hello World!" type="error" />
    </ConfigProvider>
  </>
);
```

API

Common props ref：Common props

This component 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, but still display at fullScreen	() => 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	string | number	8	

Design Token

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`