何时使用 {#when-to-use}

- 可提供成功、警告和错误等反馈信息。
- 顶部居中显示并自动消失，是一种不打断用户操作的轻量级提示方式。

代码演示

Hooks 调用（推荐）

```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;
```

其他提示类型

```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;
```

修改延时

```
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;
```

加载中

```
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 接口

```
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;
```

自定义样式

```
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;
```

更新消息内容

```
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;
```

静态方法（不推荐）

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

```tsx
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

```tsx
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" />;
```

组件 Token

Debug

```tsx
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

通用属性参考：通用属性

组件提供了一些静态方法，使用方式和参数如下：

- `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)`

参数	说明	类型	默认值
content	提示内容	ReactNode | config	-
duration	自动关闭的延时，单位秒。设为 0 时不自动关闭	number	3
onClose	关闭时触发的回调函数	function	-

组件同时提供 promise 接口。

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

其中 `message[level]` 是组件已经提供的静态方法。`then` 接口返回值是 Promise。

也可以对象的形式传递参数：

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

- `message.loading(config)`

`config` 对象属性如下：

参数	说明	类型	默认值
className	自定义 CSS class	string	-
content	提示内容	ReactNode	-
duration	自动关闭的延时，单位秒。设为 0 时不自动关闭	number	3
icon	自定义图标	ReactNode	-
key	当前提示的唯一标志	string | number	-
style	自定义内联样式	CSSProperties	-
onClick	点击 message 时触发的回调函数	function	-
onClose	关闭时触发的回调函数	function	-

全局方法

还提供了全局配置和全局销毁方法：

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

> *也可通过 `message.destroy(key)` 来关闭一条消息。*

message.config

> *当你使用 `ConfigProvider` 进行全局化配置时，系统会默认自动开启 RTL 模式。(4.3.0+)*
>
> *当你想单独使用，可通过如下设置开启 RTL 模式。*

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

参数	说明	类型	默认值	版本
duration	默认自动关闭延时，单位秒	number	3	
getContainer	配置渲染节点的输出位置，但依旧为全屏展示	() => HTMLElement	() => document.body	

maxCount	最大显示数，超过限制时，最早的消息会被自动关闭	number	-	
prefixCls	消息节点的 className 前缀	string	`ant-message`	4.5.0
rtl	是否开启 RTL 模式	boolean	false	
top	消息距离顶部的位置	string | number	8	

主题变量（Design Token）

FAQ

为什么 message 不能获取 context、redux 的内容和 ConfigProvider 的 `locale/prefixCls/theme` 等配置?

直接调用 message 方法，antd 会通过 `ReactDOM.render` 动态创建新的 React 实体。其 context 与当前代码所在 context 并不相同，因而无法获取 context 信息。

当你需要 context 信息（例如 ConfigProvider 配置的内容）时，可以通过 `message.useMessage` 方法会返回 `api` 实体以及 `contextHolder` 节点。将其插入到你需要获取 context 位置即可：

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

return (
  <Context1.Provider value="Ant">
    {/* contextHolder 在 Context1 内，它可以获得 Context1 的 context */}
    {contextHolder}
    <Context2.Provider value="Design">
      {/* contextHolder 在 Context2 外，因而不会获得 Context2 的 context */}
    </Context2.Provider>
  </Context1.Provider>
);
```

异同：通过 hooks 创建的 `contextHolder` 必须插入到子元素节点中才会生效，当你不需要上下文信息时请直接调用。

> *可通过 App 包裹组件 简化 `useMessage` 等方法需要手动植入 contextHolder 的问题。*

静态方法如何设置 prefixCls？

你可以通过 `ConfigProvider.config` 进行设置。