学习反应 >

快速开始

欢迎使用 React 文档！本页将向您介绍您每天会用到的 80% 的 React 概念。

你将学习

- 如何创建和嵌套组件
- 如何添加标记和样式
- 如何显示数据
- 如何呈现条件和列表
- 如何响应事件和更新屏幕
- 如何在组件之间共享数据

创建和嵌套组件

*React 应用程序由组件*组成。组件是 UI（用户界面）的一部分，具有自己的逻辑和外观。组件可以小到一个按钮，也可以大到整个页面。

React 组件是返回标记的 JavaScript 函数：

```
function MyButton() {
  return (
    <button>I'm a button</button>
  );
}
```

现在你已经声明了 `MyButton`，你可以将它嵌套到另一个组件中：

```
export default function MyApp() {
  return (
    <div>
      <h1>Welcome to my app</h1>
      <MyButton />
    </div>
  );
}
```

请注意，它 `<MyButton />` 以大写字母开头。这就是您知道它是 React 组件的方式。React 组件名称必须始终以大写字母开头，而 HTML 标签必须是小写字母。

看看结果：

应用程序.js 下载 重置

```
1  function MyButton() {
2    return (
3      <button>
4        I'm a button
5      </button>
6    );
7  }
8
9  export default function MyApp() {
10   return (
11     <div>
12       <h1>Welcome to my app</h1>
```

展示更多

关键字 `export default` 指定文件中的主要组件。如果您不熟悉某些 JavaScript 语法，MDN和javascript.info有很好的参考资料。

使用 JSX 编写标记

您在上面看到的标记语法称为*JSX*。它是可选的，但大多数 React 项目为了方便起见都使用 JSX。我们为本地开发推荐的所有工具都支持开箱即用的 JSX。

JSX 比 HTML 更严格。你必须关闭像 `<br />` . 您的组件也不能返回多个 JSX 标签。您必须将它们包装到一个共享的父级中，例如一个 `<div>...</div>` 或一个空 `<>...</>` 包装器：

```
function AboutPage() {
  return (
    <>
      <h1>About</h1>
      <p>Hello there.<br />How do you do?</p>
    </>
  );
}
```

如果你有很多 HTML 要移植到 JSX，你可以使用在线转换器。

添加样式

在 React 中，你指定一个 CSS 类 `className` 。它的工作方式与 HTML 属性相同 `class` ：

```
<img className="avatar" />
```

然后在单独的 CSS 文件中为其编写 CSS 规则：

```
/* In your CSS */
.avatar {
  border-radius: 50%;
}
```

React 没有规定你如何添加 CSS 文件。在最简单的情况下，您将向 `<link>` HTML 添加一个标记。如果您使用构建工具或框架，请查阅其文档以了解如何将 CSS 文件添加到您的项目中。

显示数据

JSX 允许您将标记放入 JavaScript。花括号让你“逃回”到 JavaScript 中，这样你就可以从代码中嵌入一些变量并将其显示给用户。例如，这将显示 `user.name`：

```
return (
  <h1>
    {user.name}
  </h1>
);
```

你也可以从 JSX 属性“转义到 JavaScript”，但你必须使用花括号*而不是*引号。例如，将字符串作为 CSS 类 `className="avatar"` 传递，但读取 JavaScript变量值，然后将该值作为属性传递： `"avatar" src={user.imageUrl} user.imageUrl src`

```
return (
  <img
    className="avatar"
    src={user.imageUrl}
  />
);
```

您也可以在 JSX 大括号内放置更复杂的表达式，例如，字符串连接：

应用程序.js 下载 重置

```
8   return (
9     <>
10      <h1>{user.name}</h1>
11      <img
12        className="avatar"
13        src={user.imageUrl}
14        alt={'Photo of ' + user.name}
15        style={{
16          width: user.imageSize,
17          height: user.imageSize
18        }}
19      />
```

展示更多

在上面的示例中， `style={{}}` 不是特殊语法，而是JSX 大括号 `{}` 内的常规对象 `style={}` 。 `style` 当您的样式依赖于 JavaScript 变量时，您可以使用该属性。

条件渲染

在 React 中，没有用于编写条件的特殊语法。相反，您将使用与编写常规 JavaScript 代码时相同的技术。例如，您可以使用 `if` 语句有条件地包含 JSX：

```
let content;
if (isLoggedIn) {
  content = <AdminPanel />;
} else {
  content = <LoginForm />;
}
return (
  <div>
    {content}
  </div>
);
```

如果您喜欢更紧凑的代码，可以使用条件 `?` 运算符。与 不同 `if` ，它在 JSX 内部工作：

```
<div>
  {isLoggedIn ? (
    <AdminPanel />
  ) : (
    <LoginForm />
  )}
</div>
```

当你不需要 `else` 分支时，你也可以使用更短的逻辑 `&&` 语法：

```
<div>
  {isLoggedIn && <AdminPanel />}
</div>
```

所有这些方法也适用于有条件地指定属性。如果您不熟悉这种 JavaScript 语法中的某些语法，您可以从始终使用 `if...else` .

渲染列表

您将依赖 `for` 循环和数组 `map()` 函数等JavaScript 功能来呈现组件列表。

例如，假设您有一系列产品：

```
const products = [
  { title: 'Cabbage', id: 1 },
  { title: 'Garlic', id: 2 },
  { title: 'Apple', id: 3 },
];
```

在您的组件内，使用该 `map()` 函数将产品数组转换为 `<li>` 项目数组：

```
const listItems = products.map(product =>
  <li key={product.id}>
    {product.title}
  </li>
);

return (
  <ul>{listItems}</ul>
);
```

注意如何 `<li>` 具有 `key` 属性。对于列表中的每个项目，您应该传递一个字符串或数字，以在其兄弟项中唯一标识该项目。通常，键应该来自您的数据，例如数据库 ID。如果您稍后插入、删除或重新排序项目，React 使用您的键来了解发生了什么。

应用程序.js　下载　重置

```
1  const products = [
2    { title: 'Cabbage', isFruit: false, id: 1 },
3    { title: 'Garlic', isFruit: false, id: 2 },
4    { title: 'Apple', isFruit: true, id: 3 },
5  ];
6
```

```
 7  export default function ShoppingList() {
 8    const listItems = products.map(product =>
 9      <li
10        key={product.id}
11        style={{
```

展示更多

响应事件

您可以通过在组件内*声明事件处理*函数来响应事件：

```
function MyButton() {
  function handleClick() {
    alert('You clicked me!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

注意 `onClick={handleClick}` 末尾没有括号！不要*调用*事件处理函数：你只需要*传递下去*。当用户单击按钮时，React 将调用您的事件处理程序。

更新画面

通常，您会希望您的组件“记住”一些信息并显示它。例如，您可能想要计算单击按钮的次数。为此，请将*状态*添加到您的组件。

`useState` 首先，从 React导入：

```
import { useState } from 'react';
```

现在你可以在你的组件中声明一个*状态变量*：

```
function MyButton() {
  const [count, setCount] = useState(0);
  // ...
```

您将从 获得两件事 `useState`： 当前状态（ `count`）和允许您更新它的函数（ `setCount`）。你可以给它们起任何名字，但惯例是写 `[something, setSomething]`.

第一次显示该按钮， `count` 是 `0` 因为您传递 `0` 给了 `useState()`.当你想改变状态时，调用 `setCount()` 并将新值传递给它。单击此按钮将增加计数器：

```
function MyButton() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <button onClick={handleClick}>
      Clicked {count} times
    </button>
```

```
  );
}
```

React 会再次调用你的组件函数。这一次， `count` 将是 `1` 。然后它会 `2` 。等等。

如果多次渲染同一个组件，每个组件都会获得自己的状态。分别点击每个按钮：

应用程序.js 下载 重置

```
import { useState } from 'react';

export default function MyApp() {
  return (
    <div>
      <h1>Counters that update separately</h1>
      <MyButton />
      <MyButton />
    </div>
  );
}

```

展示更多

注意每个按钮如何“记住”它自己的 `count` 状态并且不影响其他按钮。

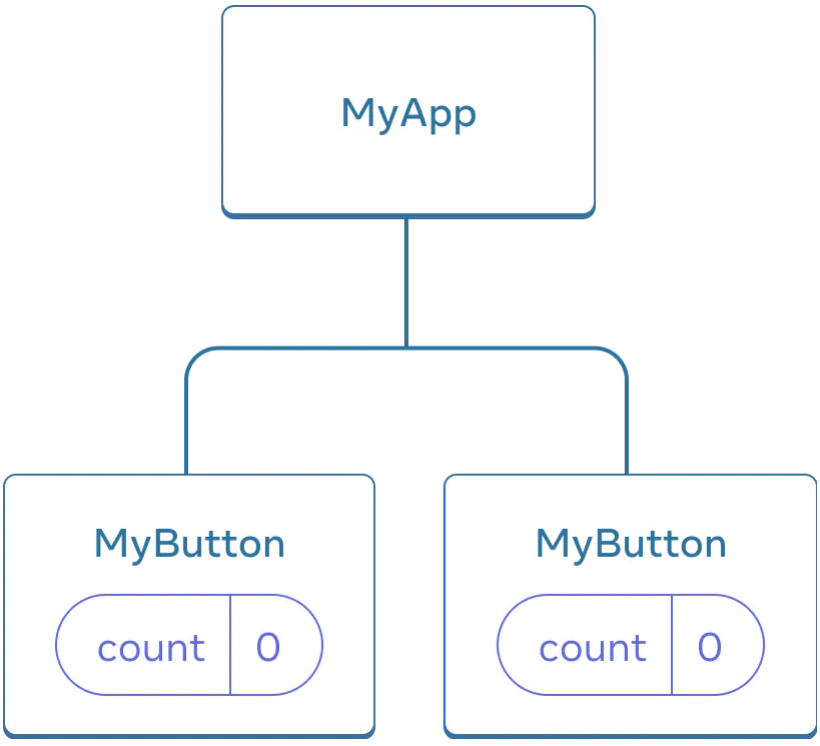
使用钩子

以 开头的函数 `use` 称为*Hooks*。 `useState` 是 React 提供的内置 Hook。您可以在 API 参考中找到其他内置 Hook 。您还可以通过组合现有的 Hooks 来编写自己的 Hooks。

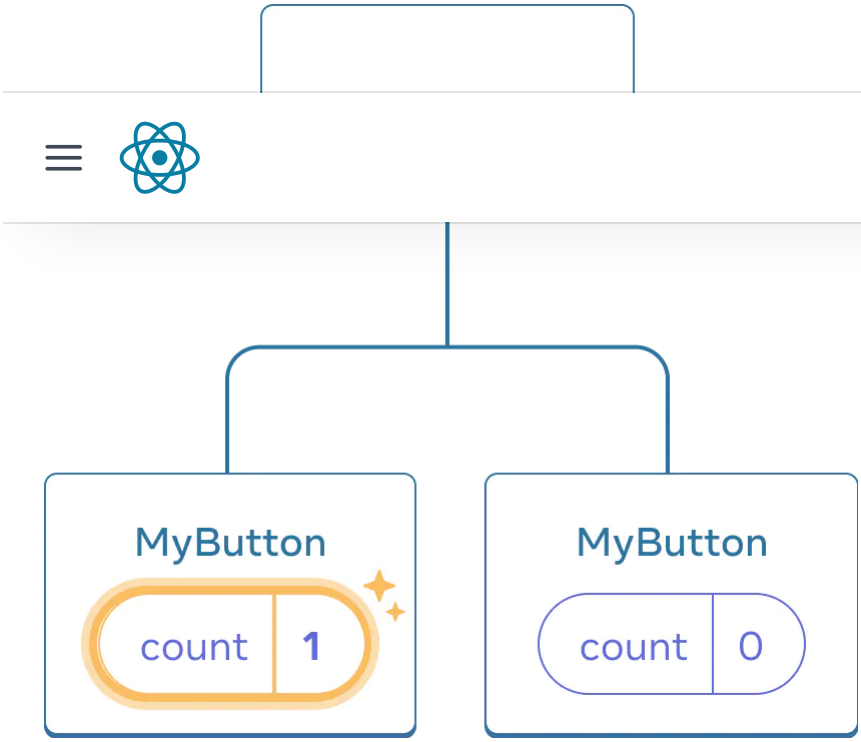
钩子比其他功能更具限制性。您只能在组件（或其他 Hooks）的*顶部调用 Hooks*。如果您想 `useState` 在条件或循环中使用，请提取一个新组件并将其放在那里。

在组件之间共享数据

在前面的例子中，每个 `MyButton` 都有自己独立的 `count` ，当每个按钮被点击时，只有 `count` 被点击的按钮的 发生了变化：



最初，每个 `MyButton` 的 `count` 状态是 `0`

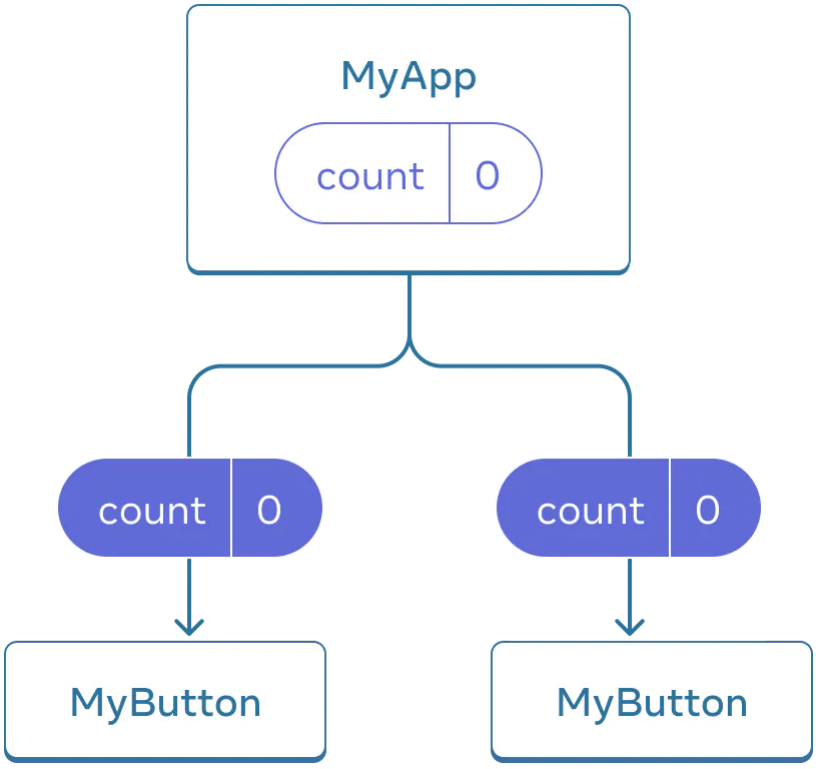


第一个 MyButton 更新 count 为 1

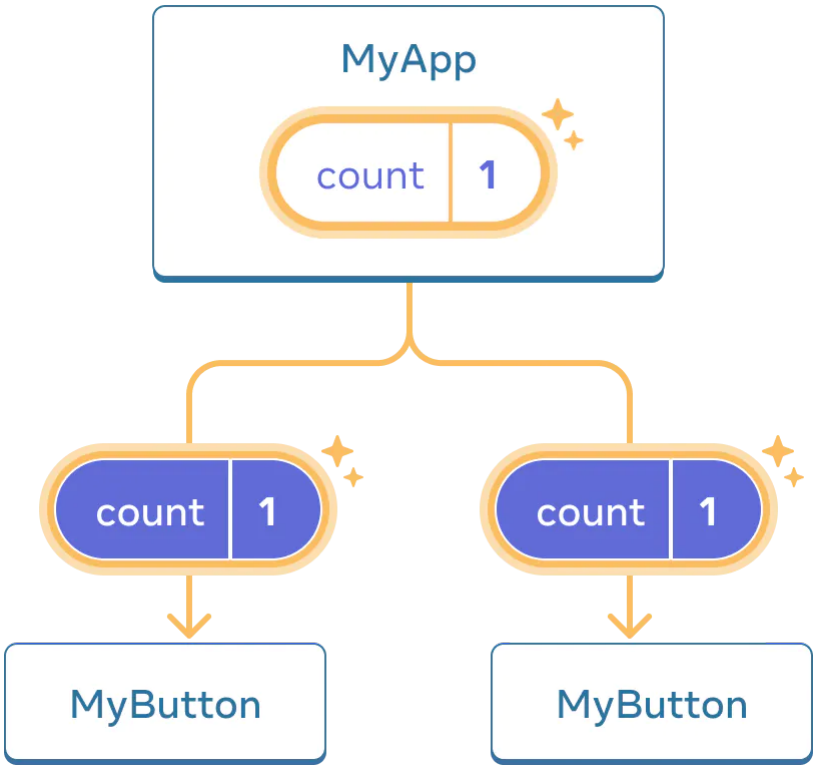
但是，通常您会需要组件来*共享数据并始终一起更新*。

要使两个 MyButton 组件显示相同 count 并一起更新，您需要将状态从单个按钮“向上”移动到包含所有按钮的最近组件。

在这个例子中，它是 MyApp：



最初， `MyApp` 的 `count` 状态是 `0` 并且被传递给两个孩子



单击时， `MyApp` 将其 `count` 状态更新为 `1` 并将其传递给两个孩子

现在，当您单击任一按钮时， `count` in `MyApp` 将发生变化，这将更改 .in 中的两个计数 `MyButton` 。以下是您如何在代码中表达这一点。

首先，*将状态*从 `MyButton` into向上移动 `MyApp` ：

```
export default function MyApp() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <div>
      <h1>Counters that update separately</h1>
      <MyButton />
      <MyButton />
```

```
    </div>
  );
}

function MyButton() {
  // ... we're moving code from here ...
}
```

然后，*将状态*从向下传递 `MyApp` 到每个 `MyButton` ，连同共享的点击处理程序。您可以 `MyButton` 使用 JSX 花括号传递信息，就像您之前使用内置标签所做的那样 `<img>` ：

```
export default function MyApp() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <div>
      <h1>Counters that update together</h1>
      <MyButton count={count} onClick={handleClick} />
      <MyButton count={count} onClick={handleClick} />
    </div>
  );
}
```

你像这样传递下来的信息叫做*props*。现在 `MyApp` 组件包含 `count` 状态和 `handleClick` 事件处理程序，并将*它们作为 props 传递*给每个按钮。

最后，更改 `MyButton` 为*读取*您从其父组件传递的道具：

```
function MyButton({ count, onClick }) {
  return (
    <button onClick={onClick}>
      Clicked {count} times
    </button>
  );
```

```
}
```

单击按钮时， `onClick` 处理程序将触发。每个按钮的 `onClick` prop 都设置为 `handleClick` 里面的函数 `MyApp` ，所以它里面的代码运行。该代码调用 `setCount(count + 1)` ，递增 `count` 状态变量。新 `count` 值作为 prop 传递给每个按钮，因此它们都显示新值。这被称为“提升状态”。通过向上移动状态，您已经在组件之间共享它。

应用程序.js 下载 重置

```
1  import { useState } from 'react';
2
3  export default function MyApp() {
4    const [count, setCount] = useState(0);
5
6    function handleClick() {
7      setCount(count + 1);
8    }
9
10   return (
11     <div>
12       <h1>Counters that update together</h1>
```

展示更多

Counters that update together

Clicked 0 times

Clicked 0 times

下一步

到目前为止，您已经了解了如何编写 React 代码的基础知识！

查看教程，将它们付诸实践，并使用 React 构建您的第一个迷你应用程序。

下一个
教程：井字游戏 ›

- 认识团队
- 文档贡献者
- 致谢
- 反应本机
- 隐私
- 条款