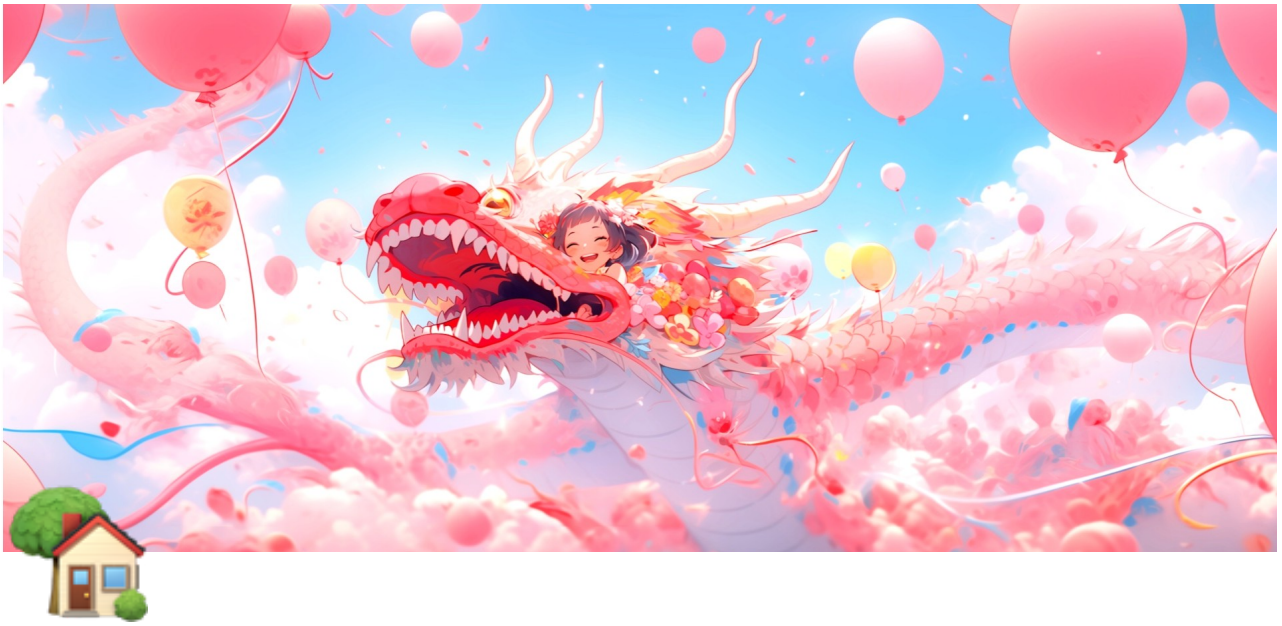

11–1 Hook 简介

资源

1. React 内置 Hook
2. 自定义 Hook

Hook 是 **React 16.8** 的新增特性。

Hook 分类

Hook —— 在组件中使用不同的 React 特性。

State Hook

状态帮助组件“记住”用户输入的信息。例如，一个表单组件可以使用状态存储输入值，而一个图像库组件可以使用状态存储所选的图像索引。

使用以下 Hook 以向组件添加状态：

- 使用 `useState` 声明可以直接更新的状态变量。

- 使用 `useReducer` 在 reducer 函数 中声明带有更新逻辑的 state 变量。

Context Hook

上下文帮助组件 从祖先组件接收信息，而无需将其作为 props 传递。例如，应用程序的顶层组件可以借助上下文将 UI 主题传递给所有下方的组件，无论这些组件层级有多深。

- 使用 `useContext` 读取订阅上下文。

Ref Hook

ref 允许组件 保存一些不用于渲染的信息，比如 DOM 节点或 timeout ID。与状态不同，更新 ref 不会重新渲染组件。ref 是从 React 范例中的“脱围机制”。当需要与非 React 系统如浏览器内置 API 一同工作时，ref 将会非常有用。

- 使用 `useRef` 声明 ref。你可以在其中保存任何值，但最常用于保存 DOM 节点。
- 使用 `useImperativeHandle` 自定义从组件中暴露的 ref，但是很少使用。

Effect Hook

Effect 允许组件 连接到外部系统并与之同步。这包括处理网络、浏览器、DOM、动画、使用不同 UI 库编写的小部件以及其他非 React 代码。

- 使用 `useEffect` 将组件连接到外部系统。

Effect 是从 React 范式中的“脱围机制”。避免使用 Effect 协调应用程序的数据流。如果不需要与外部系统交互，那么 可能不需要 Effect。

`useEffect` 有两个很少使用的变换形式，它们在执行时机有所不同：

- `useLayoutEffect` 在浏览器重新绘制屏幕前执行，可以在此处测量布局。
- `useInsertionEffect` 在 React 对 DOM 进行更改之前触发，库可以在此处插入动态 CSS。

性能 Hook

优化重新渲染性能的一种常见方法是跳过不必要的工作。例如，可以告诉 React 重用缓存的计算结果，或者如果数据自上次渲染以来没有更改，则跳过重新渲染。

可以使用以下 Hook 跳过计算和不必要的重新渲染：

- 使用 `useMemo` 缓存计算代价昂贵的计算结果。
- 使用 `useCallback` 将函数传递给优化组件之前缓存函数定义。

有时由于屏幕确实需要更新，无法跳过重新渲染。在这种情况下，可以通过将必须同步的阻塞更新（比如使用输入法输入内容）与不需要阻塞用户界面的非阻塞更新（比如更新图表）分离以提高性能。

使用以下 Hook 处理渲染优先级：

- `useTransition` 允许将状态转换标记为非阻塞，并允许其他更新中断它。
- `useDeferredValue` 允许延迟更新 UI 的非关键部分，以让其他部分先更新。

资源 Hook

资源可以被组件访问，而无需将它们作为状态的一部分。例如，组件可以从 Promise 中读取消息，或从上下文中读取样式信息。

使用以下 Hook 以从资源中读取值：

- `use` 允许读取像 Promise 或 上下文 这样的资源的值。

其他 Hook

这些 Hook 主要适用于库作者，不常在应用程序代码中使用。

- 使用 `useDebugValue` 自定义 React 开发者工具为自定义 Hook 添加的标签。
- 使用 `useId` 将唯一的 ID 与组件相关联，其通常与可访问性 API 一起使用。
- 使用 `useSyncExternalStore` 订阅外部 store。

自定义 Hook

开发者可以 自定义 Hook 作为 JavaScript 函数。

如 React-Redux 的 useDispatch、useSelector、AntD4/5 Form 的 useForm、React-Router 的 useMatch、useLocation 等

自定义 hooks 库：ahooks、useHooks