React 基础

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

使用状态操作表单元素的值

使用状态操作表单元素的值

步骤，分为两步：

1. 准备一个状态

```
const [value, setValue] = useState('')
```

2. 绑定 value 值（状态值）和 onChange 属性（设置状态）

```
<input value={value} onChange={e => setValue(e.target.value)} />
```

受控组件：受到 React 状态控制的表单元素

注意：React 中文本框的 change 事件类似于原生的 input 事件（React 对 change 事件做过处理）

useRef 与 DOM 操作

useRef 与 DOM 操作

在 React 组件中操作 DOM，需要使用 useRef，分为两步：

1. 准备：创建 ref 对象，并与 JSX 绑定

```
const inputRef = useRef(null)
```

```
<input ref={inputRef} />
```

2. 根据业务进行 DOM 操作：通过 inputRef.current 拿到 DOM 对象

```
inputRef.current.value
```

```
inputRef.current.focus()
```

注意1：不要在组件渲染时使用 ref 进行 DOM 操作，因此，此时 ref 还没有值

注意2：操作文本框的值，推荐使用状态（受控组件）操作

案例：B站评论 —— 发表评论

B站评论案例 —— 发布评论

1. 获取评论内容
2. 发布评论

★ 发布评论

B站评论案例 —— 发布评论总结

1. 使用受控组件获取评论内容

```
const [value, setValue] = useState('')
<textarea
  className="reply-box-textarea"
  placeholder="发一条友善的评论"
  value={value}
  onChange={e => setValue(e.target.value)}
/>
```

2.1 发布评论（有内容，发布评论同时排序）

```
const comment = {
  rpid: Date.now(),
  user,
  content: value,
  ctime: dayjs().format('MM-DD HH:mm'),
  like: 0,
  action: 0,
}
const newList = [comment, ...list]
if (activeTab === 'time') {
  // 按照时间降序排序
  setList(orderBy(newList, 'ctime', 'desc'))
} else {
  // 按照喜欢数量降序排序
  setList(orderBy(newList, 'like', 'desc'))
}
```

2.2 发布评论（无内容，获得焦点提升体验）

```
const textRef = useRef(null)
<textarea
  className="reply-box-textarea"
  placeholder="发一条友善的评论"
  ref={textRef}
/>
if (value.trim() === '') {
  return textRef.current.focus()
}
```

组件 props

组件 props

作用：给组件传递数据，是 React 组件通讯的基础

使用步骤：

1. 传递 props：在组件标签上添加属性

```
<Avatar imgUrl="https://...zhoujielun.jpeg" size={80} />
```

2. 接收 props：通过参数拿到

```
const Avatar = props => {
  return <img src={props.imgUrl} width={props.size} alt="" />
}
```

props 是只读对象

推荐：使用解构，接收 props

```
const Avatar = ({ imgUrl, size = 100 }) => {
  return <img src={imgUrl} width={size} alt="" />
}
```

props 默认值

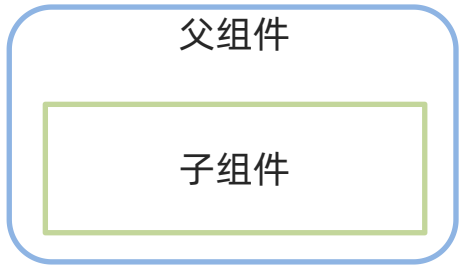
组件通讯

组件通讯

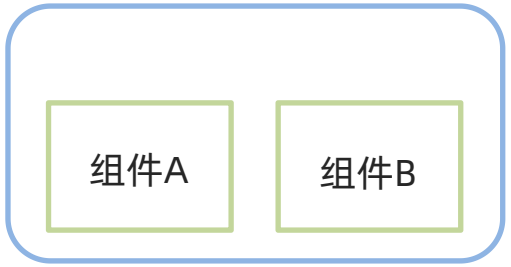
使用场景：一个组件需要使用另一个组件的数据

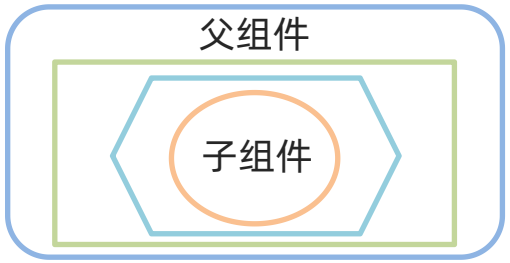
根据组件之间的层级关系，常见的 React 组件通讯分为 3 种：

1. 父子组件通讯

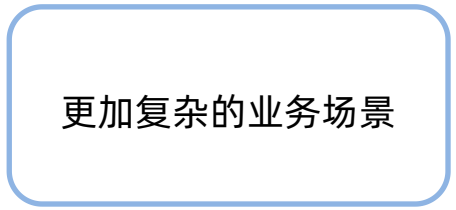


2. 非父子组件通讯





3. 状态管理工具（Redux 等）



父子组件通讯

原则：谁的数据谁负责

1. 父 → 子 （传递 props）

① 父组件提供数据，通过 props 传递给子组件使用

```
<Todo id={id} text={text} done={done} />
<Todo {...item} />
```

2. 子 → 父 （回调函数）

① 父组件准备修改数据的函数，传递给子组件

```
const onToggle = id => {}
<Todo onToggle={onToggle} />
```

② 子组件调用函数，将数据作为参数回传给父组件

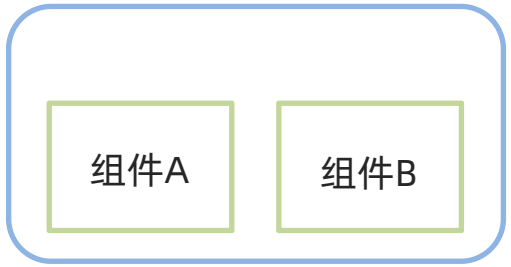
```
const Todo = ({ id, onToggle }) => {
  return <li onClick={() => onToggle(id)}></li>
}
```

兄弟组件通讯

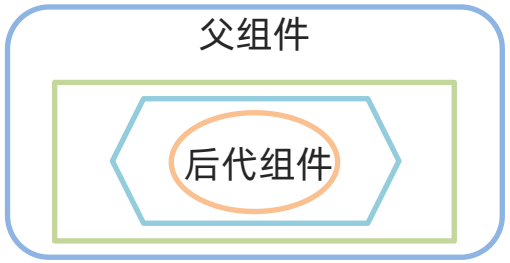
非父子组件通讯

根据组件之间的层级关系，常见的 2 种情况：

1. 兄弟关系



2. 后代关系



非父子组件通讯 —— 兄弟关系

父组件　　数据：选中的好友

状态提升：

如果两个兄弟组件要通讯，就把共享数据提升到公共父组件中

1. 找到父组件，提供共享数据

```
// 父组件
const App = () => {
  // 共享数据：选中的好友
  const [chatFriend, setChatFriend] = useState(friends[0])
  // 修改状态的函数
  const onSelectFriend = friend => {
    setChatFriend(friend)
  }
```

2. 使用数据，展示好友名称（父到子通讯）

```
{/* 聊天窗口 */}
<Chat chatFriend={chatFriend} />
const Chat = ({ chatFriend }) => {
<div className="header">{chatFriend.name}</div>
```

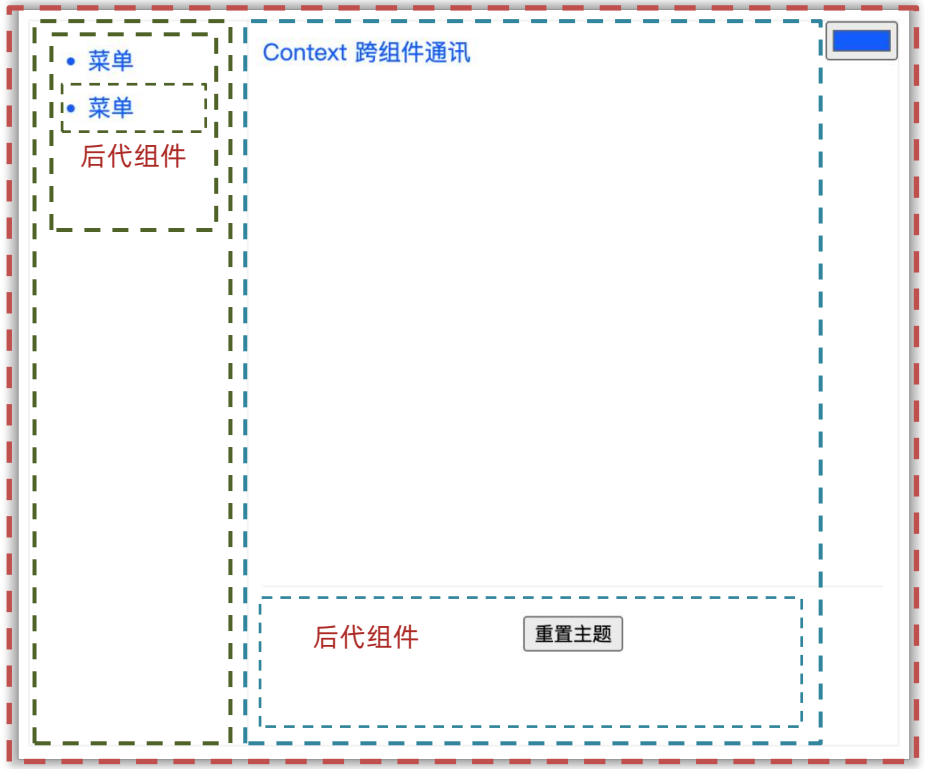
3. 修改数据，切换选中好友（子到父通讯）

```
{/* 好友列表 */}
<Friends onSelectFriend={onSelectFriend} />
const Friends = ({ onSelectFriend }) => {
<div onClick={() => onSelectFriend(item)}>...</div>
```

跨组件通讯

非父子组件通讯 —— 后代关系

父组件



Context（上下文）：范围，无视组件层级关系，跨组件通讯

1. 创建 Context 对象

```
const ThemeContext = createContext()
```

2. 划定范围，提供共享数据

```
<ThemeContext.Provider value={共享数据}>
  父组件
</ThemeContext.Provider>
```

3. 范围内的组件，获取共享数据

```
const 共享数据 = useContext(ThemeContext)
```

useEffect 的使用

useEffect 的使用

useEffect 的作用：在组件生命周期的三个阶段（挂载、更新、卸载），执行网络请求、浏览器 API 等操作

这些操作，也叫：副作用（side effects）

语法：`useEffect(Effect函数, 依赖项数组)`

副作用代码　　控制 Effect函数 的执行时机，可选

比如，在线聊天室：

挂载时：

```
useEffect(() => {
  // 发送请求，连接聊天室
}, [])
```

更新时：

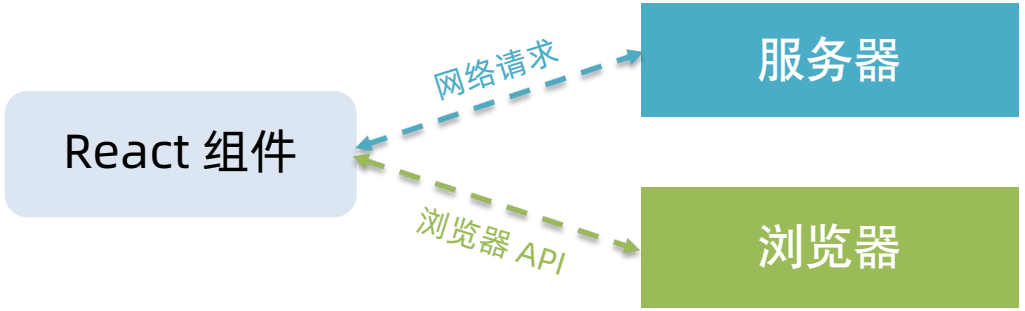
```
useEffect(() => {
  // 重新发送请求，连接聊天室
}, [roomId])
```

卸载时：

```
useEffect(() => {
  return () => { /* 断开连接 */ }
}, [])
```

useEffect 官方解释：

连接外部系统（不受 React 控制的系统）



useEffect 的扩展

useEffect 的扩展

1. 推荐：一个 useEffect 负责一个完整功能

```
useEffect(() => {
  console.log(`建立连接：${roomId} 房间`)
  return () => {
    console.log(`断开连接：${roomId} 房间`)
  }
}, [roomId])
```

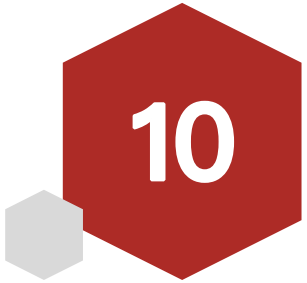
执行过程： 挂载1 更新2 1 卸载2

2. 依赖项

```
useEffect(() => { /* 可变值 */ }, [可变值1, 可变值2,])
```

需要作为依赖项的值：Effect 中用到的可变值，比如，props/state/组件内创建的变量等

不需要作为依赖项的值：组件外创建的或导入的变量、函数等



useEffect 应用 —— 发送请求

useEffect 应用 —— 发送请求

场景：组件初次渲染时，发送请求获取数据

```
useEffect(() => {
  const loadData = async () => {
    const res = await axios.get('http://xxx')
    setTodos(res.data)
  }
  loadData()
}, [])
```

1. 调用 useEffect，依赖项为空数组
2. 创建新函数，在该函数上使用 async，并调用
3. 发送请求，通过 await 获取到数据，然后使用

注意1：不要直接在 Effect函数 前面添加 asnyc 关键字，因为它是同步的

```
useEffect(async () => {
```

注意2：

- useEffect：只处理跟组件挂载、更新、卸载相关的请求代码
- 事件处理程序：处理点击等用户操作时的请求代码

React Hooks 解释和使用规则

React Hooks 解释和使用规则

React Hooks 是以 use 开头的函数，比如，useState/useEffect/useContext 等

Hooks（钩子）：为组件提供不同的 React 特性，比如，useState Hook 为组件提供状态

版本：React v16.8+

使用规则：只能在组件的顶层调用，不能嵌套在 if、for、其他函数中

原理：React 组件依赖 Hooks 的调用顺序，必须保证所有 Hooks，在每次渲染时，调用顺序一致

正确：

```
const App = () => {
  const [count, setCount] = useState(0)

  useEffect(() => {
    // ...
  }, [])
}
```

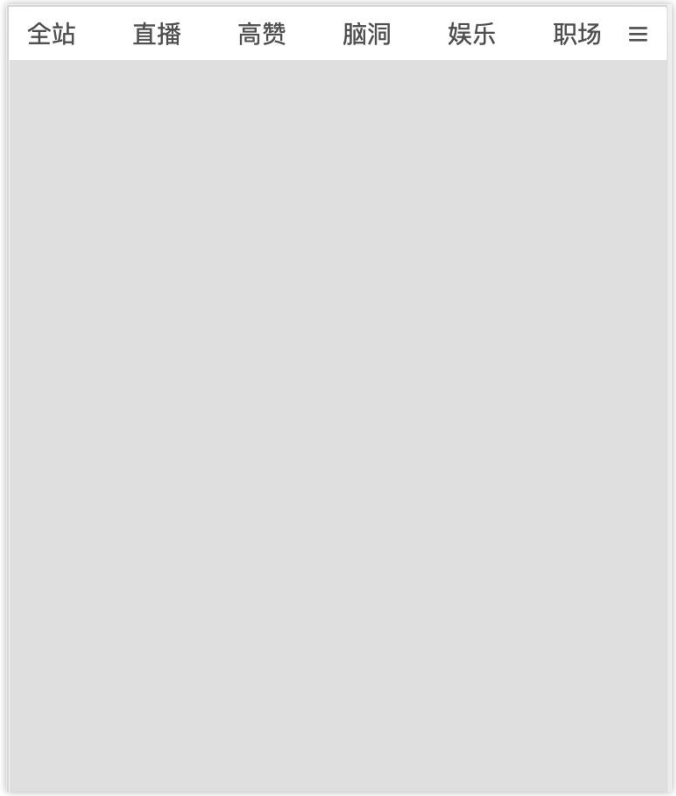
错误：

```
const App = () => {
  if (Math.random() > 0.5) {
    const [count, setCount] = useState(0)
  }
  useEffect(() => {
    // ...
  }, [])
}
```

案例：知乎频道管理

知乎频道管理案例

首页



频道弹窗

展示和隐藏频道弹窗

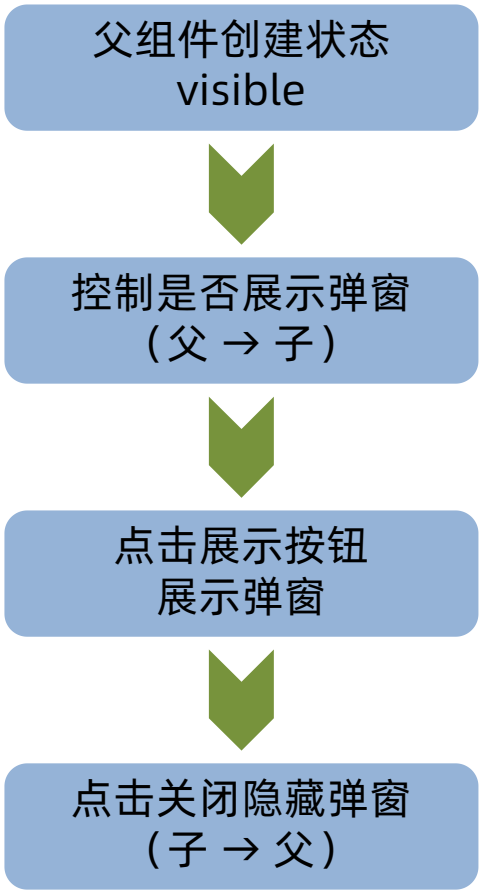
渲染频道数据

选择或移除频道

展示或隐藏频道弹窗

首页

频道弹窗



渲染频道数据

useEffect
获取频道数据

```
useEffect(() => {
  const loadData = async () => {
    const res = await axios.get('http://localhost:8000/channels')
    setChannels(res.data)
  }
  loadData()
}, [])
```

创建 Context 对象

ChannelContext.js

```
export const ChannelContext = createContext()
```

Provider
提供频道数据

```
const myChannels = channels.filter(item => item.selected)
const moreChannels = channels.filter(item => !item.selected)
<ChannelContext.Provider value={{ myChannels, moreChannels }}>
  <div className="app">
    <Home />
  </div>
</ChannelContext.Provider>
```

useContext
获取频道数据并渲染

```
const { myChannels } = useContext(ChannelContext)
```

移除或添加频道

切换编辑/完成状态

处理不可编辑频道

更新频道选中状态

```jsx
const [isEdit, setIsEdit] = useState(false)
<span onClick={() => setIsEdit(!isEdit)}>
  {isEdit ? '完成' : '编辑'}
</span>
<div className={classNames('list', isEdit && 'edit')}>
```

```jsx
<ChannelItem canEdit={item.canEdit} />
const ChannelItem = ({ name, className, canEdit }) => {
  return (
    <div className={classNames('channel-item', className)}>
      {name}
      {canEdit && <span className="icon">+</span>}
    </div>
```

```jsx
<ChannelContext.Provider value={{ onUpdateChannel }}>
<ChannelItem
  onClick={() => {
    if (isEdit && item.canEdit) {
      onUpdateChannel(item.id, !item.selected)
    }
  }}
/>
```

传智教育旗下高端IT教育品牌