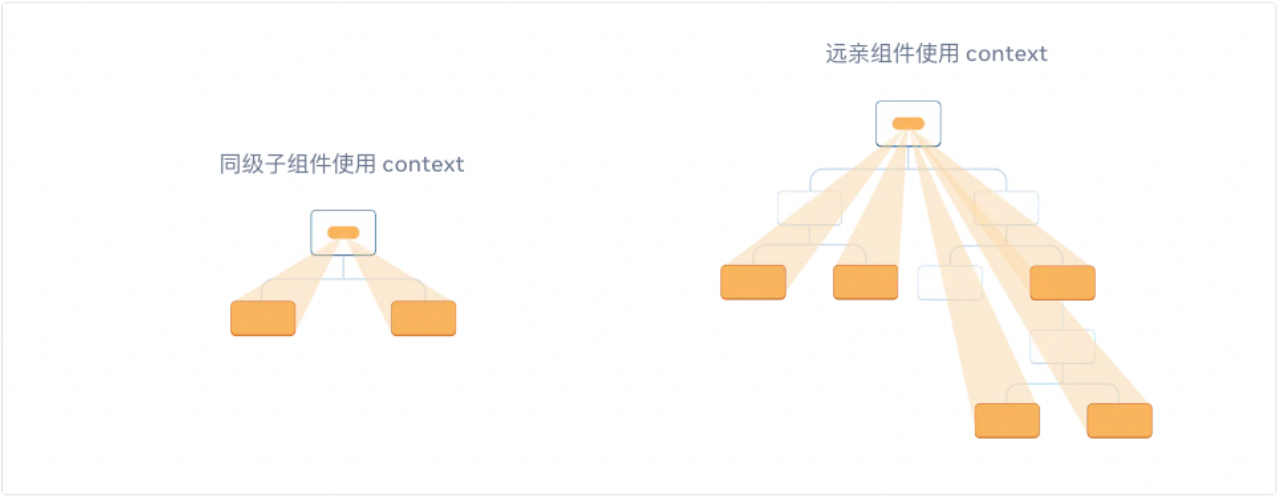
# 15–1 知识分析：Context 简介

## 资源

1. 使用 Context 深层传递参数




## 基础使用

Context 使用场景：当祖先组件想要和后代组件快速通信。

1. **创建 Context 对象**，可以设置默认 value。如果缺少匹配的 Provider，那么后代组件将会读取这里的默认值。
2. **Provider** 传递 value 给后代组件。
3. **后代组件消费** value：
    1). contextType:只能用在类组件且只能订阅单一的 Context 来源
    2). useContext: 只能用在函数组件或者自定义 Hook 中
    3). Consumer 组件，无限制。

TypeScript

```
// !1. 创建context对象
const CountContext = createContext(100); // 默认值
const ThemeContext = createContext("red"); // 默认值

// !2. 创建Provider组件，用于向后代组件传递value

function FunctionComponent() {
  const [count, setCount] = useReducer((x) => x + 1, 0);

  return (
    <div className="border">
      <h1>函数组件</h1>
      <button onClick={() => setCount()}>{count}</button>

      <ThemeContext.Provider value="green">
        <CountContext.Provider value={count}>
          <CountContext.Provider value={count + 1}>
            <Child />
          </CountContext.Provider>
        </CountContext.Provider>
      </ThemeContext.Provider>
    </div>
  );
}

function Child() {
  // !3. 后代组件消费value，寻找的最近的匹配的Provider组件的value
  const count = useContext(CountContext);
  const theme = useContext(ThemeContext);
```

```
  return (
    <div className={"border " + theme}>
      <h1>Child</h1>

      <p>第一种消费方式：useContext</p>
      <p>{count}</p>

      <p>第二种消费方式：Consumer</p>
      <ThemeContext.Consumer>
        {(theme) => (
          <div className={theme}>
            <CountContext.Consumer>
              {(value) => <p>{value}</p>}
            </CountContext.Consumer>
          </div>
        )}
      </ThemeContext.Consumer>

      <p>第三种消费方式：contextType，只能消费单一的context来源</p>
      <ClassComponent />
    </div>
  );
}

class ClassComponent extends Component {
  static contextType = CountContext;
  render() {
    console.log("ClassComponent render");
    return (
      <div className="border">
        <h1>类组件</h1>
        <p>{this.context as number}</p>
      </div>
    );
  }
}
```