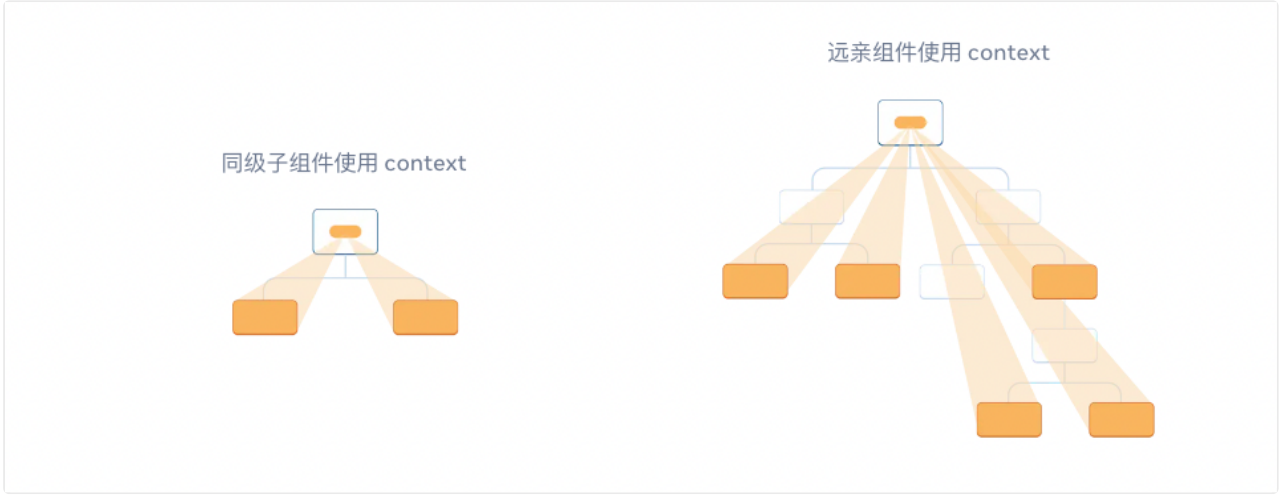
16–1 分析 Context 的底层结构与源码实现

资源

1. 使用 Context 深层传递参数

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: createContext

react/packages/react/src/ReactContext.js

JavaScript

```
import {
  REACT_PROVIDER_TYPE,
  REACT_CONSUMER_TYPE,
  REACT_CONTEXT_TYPE,
} from 'shared/ReactSymbols';

import type {ReactContext} from 'shared/ReactTypes';

export function createContext<T>(defaultValue: T): ReactContext<T> {

  const context: ReactContext<T> = {
    $$typeof: REACT_CONTEXT_TYPE,

    _currentValue: defaultValue,

    Provider: (null: any),
    Consumer: (null: any),
  };

  (context: any).Provider = {
    $$typeof: REACT_PROVIDER_TYPE,
    _context: context,
  };

  (context: any).Consumer = context;

  return context;
}
```

2. Provider 传递 value 给后代组件

beginWork

TypeScript

```
function updateContextProvider(
  current: Fiber | null,
```

```
  workInProgress: Fiber,
  renderLanes: Lanes,
) {
  let context: ReactContext<any> = workInProgress.type._context;

  const newProps = workInProgress.pendingProps;
  const oldProps = workInProgress.memoizedProps;

  const newValue = newProps.value;

  pushProvider(workInProgress, context, newValue);

  if (oldProps !== null) {
    const oldValue = oldProps.value;
    if (is(oldValue, newValue)) {
      // value没有发生变化，如果children也没有发生变化，不需要再进行后面的rende
      if (
        oldProps.children === newProps.children &&
        !hasLegacyContextChanged()
      ) {
        return bailoutOnAlreadyFinishedWork(
          current,
          workInProgress,
          renderLanes,
        );
      }
    } else {
      // context value 已更改。搜索匹配的消费者并调度它们进行更新。
      propagateContextChange(workInProgress, context, renderLanes);
    }
  }

  const newChildren = newProps.children;
  reconcileChildren(current, workInProgress, newChildren, renderLanes)
  return workInProgress.child;
}
```

3. 后代组件消费

3.1 useContext

只能用在函数组件或者自定义 Hook 中。

3.2 contextType

contextType:只能用在类组件且只能订阅单一的 Context 来源。

3.3 Consumer

Consumer 组件，无限制。