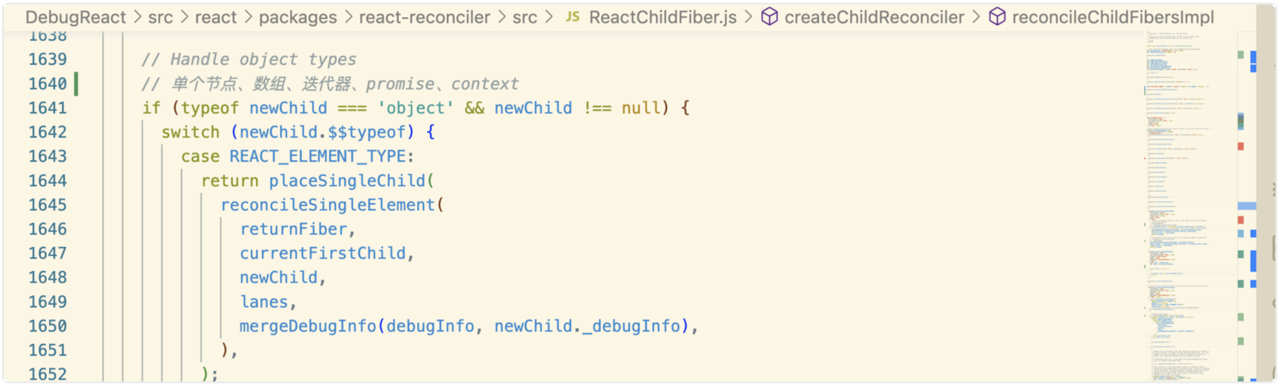
13-2 协调单个节点



DebugReact > src > react > packages > react-reconciler > src > JS ReactChildFiber.js > createChildReconciler > reconcileChildFibersImpl

```javascript
    // Handle object types
    // 单个节点、数组、迭代器、promise、context
    if (typeof newChild === 'object' && newChild !== null) {
      switch (newChild.$$typeof) {
        case REACT_ELEMENT_TYPE:
          return placeSingleChild(
            reconcileSingleElement(
              returnFiber,
              currentFirstChild,
              newChild,
              lanes,
              mergeDebugInfo(debugInfo, newChild._debugInfo),
            ),
          );
```

reconcileSingleElement

JavaScript

```javascript
function reconcileSingleElement(
  returnFiber: Fiber,
  currentFirstChild: Fiber | null,
  element: ReactElement,
  lanes: Lanes,
  debugInfo: ReactDebugInfo | null,
): Fiber {
  const key = element.key;
  let child = currentFirstChild;
```

```
while (child !== null) {
  if (child.key === key) {

    const elementType = element.type;
    if (elementType === REACT_FRAGMENT_TYPE) {
      if (child.tag === Fragment) {
        deleteRemainingChildren(returnFiber, child.sibling);
        const existing = useFiber(child, element.props.children);
        existing.return = returnFiber;
        if (__DEV__) {
          existing._debugOwner = element._owner;
          existing._debugInfo = debugInfo;
        }
        return existing;
      }
    } else {
      if (
        child.elementType === elementType
      ) {
        deleteRemainingChildren(returnFiber, child.sibling);
        const existing = useFiber(child, element.props);
        coerceRef(returnFiber, child, existing, element);
        existing.return = returnFiber;
        return existing;
      }
    }
    // Didn't match.
    deleteRemainingChildren(returnFiber, child);
    break;
  } else {
    deleteChild(returnFiber, child);
  }
  child = child.sibling;
}

if (element.type === REACT_FRAGMENT_TYPE) {
  const created = createFiberFromFragment(
    element.props.children,
    returnFiber.mode,
    lanes,
        element.key,
      );
      created.return = returnFiber;
      return created;
    } else {
      const created = createFiberFromElement(element, returnFiber.mode
      coerceRef(returnFiber, currentFirstChild, created, element);
      created.return = returnFiber;
      return created;
    }
  }
```

placeSingleChild

```javascript
function placeSingleChild(newFiber: Fiber): Fiber {
  if (shouldTrackSideEffects && newFiber.alternate === null) {
    newFiber.flags |= Placement | PlacementDEV;
  }
  return newFiber;
}
```