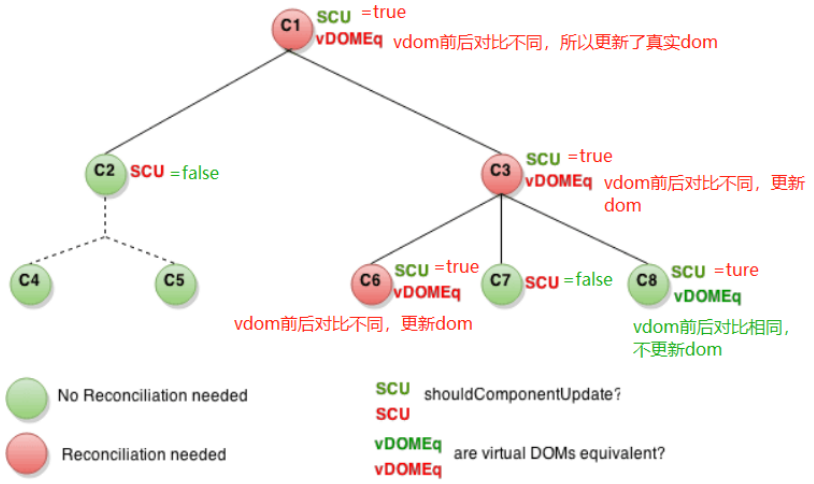
react渲染原理

从官方性能优化要点谈到虚拟dom： https://zh-hans.reactjs.org/docs/optimizing-performance.html * **压缩版本**

- **虚拟列表**
- **shouldComponentUpdate**（默认返回true），**返回false，不更新**
  ——>React.PureComponent(props 和 state 的**浅**比较：复杂对象变更不更新->也可通过component.forceUpdate()强制调用render)
  React.PureComponent 中的 shouldComponentUpdate() return false将跳过所有子组件树的 prop 更新。

shouldComponentUpdate 的作用

这是一个组件的子树。每个节点中，`SCU` 代表 `shouldComponentUpdate` 返回的值，而 `vDOMEq` 代表返回的 React 元素是否相同。最后，圆圈的颜色代表了该组件是否需要被调停。



> dom操作相对于js对象操作更慢，通过js对象表示虚拟dom进行diff对比后统一修改真实dom
> 组件props或state变更，react会将最新返回的元素vdom和之前渲染的元素vdom对比，决定是否更新真实dom。
> 即使只更新改变了的dom节点，重新渲染也要时间，所以还可以通过shouldComponentUpdate 来进行提速

- **不可变数据immutable**

复杂对象变更触发更新的方式：
通过array.concat([])、Object.assign({}, colormap, {right: 'blue'})
或者扩展运算符[...array,[]],{...colormap, right: 'blue'}

React.memo

```
const MyComponent = React.memo(function MyComponent(props) {
  /* 使用 props 渲染 */
});
```

在相同**props**的情况下渲染相同的结果,用React.memo包裹：通过记忆组件渲染结果的方式来提高组件的性能

默认情况下其只会对复杂对象做浅层对比，如果你想要控制对比过程，那么请将自定义的比较函数通过第二个参数传入来实现。

```
function MyComponent(props) {
  /* 使用 props 渲染 */
}
function areEqual(prevProps, nextProps) {
  /*
  如果把 nextProps 传入 render 方法的返回结果与
  将 prevProps 传入 render 方法的返回结果一致则返回 true,
  否则返回 false
  */
}
export default React.memo(MyComponent, areEqual);
```

areEqual返回false说明props前后不相等，重新渲染

Diff算法

https://zh-hans.reactjs.org/docs/reconciliation.html

Diff算法：对比props或state变化前后返回的两颗React元素树之间的差别，生成将一棵树转换成另一棵树的最小操作次数的算法叫Diff算法，通过Diff算法高效实现UI更新

react的Diff算法设计原则

1. 两个不同类型的元素会产生出不同的树；
2. 开发者可以通过设置 key 属性，来告知渲染哪些子元素在不同的渲染下可以保存不变；