02

OPEN ORIENTED

凹 凸 实 验 室

一起来造个轮子（一）

Virtual Dom

luckyadam

1 缘起

2 Virtual Dom

3 Virtual Dom diff

1

缘起

缘起

02

为啥又要造个轮子?

缘起

故事得从一次**摸索**说起

缘起

PC页面

缘起

传统

jQuery

优势

- 团队上手快
- 配套方案成熟
- 基础设施完善

劣势

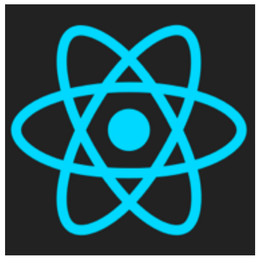
- 风格不一
- 后续维护困难
- DOM操作
- 技术过时

缘起

升级?

缘起

流行框架



缘起

体积大

兼容性

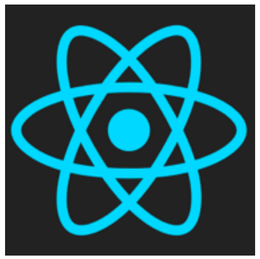
可扩展性

缘起

那么来考虑造个轮子吧！

缘起

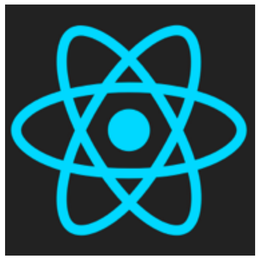
流行框架



virtual dom

缘起

流行框架



jsx

↓

virtual dom

template string

↓

virtual dom

缘起

virtual dom

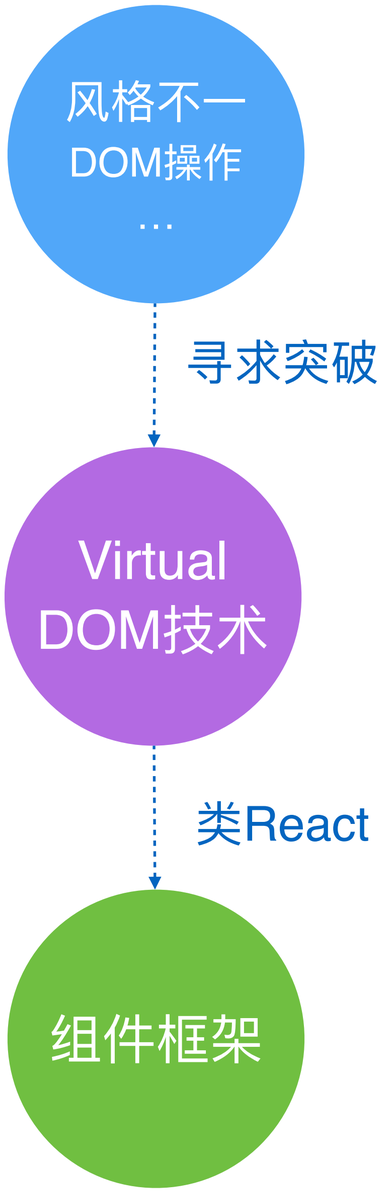
细粒度完成dom更新

自动找到最优的修改dom操作，避免繁重的手工dom更改

更加方便地进行组件化，有利于开发维护

缘起

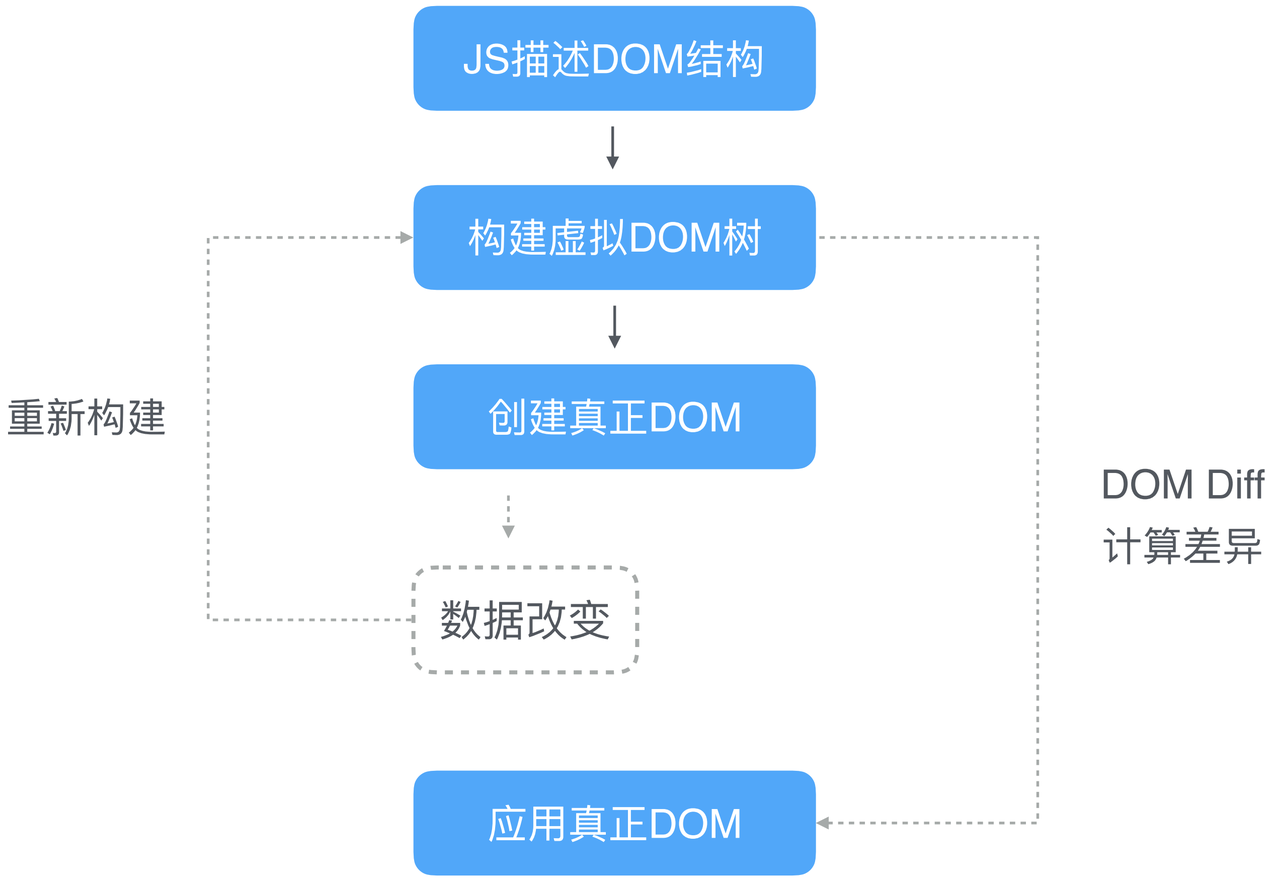
缘起



2

Virtual Dom

Virtual Dom


JS描述DOM结构
构建虚拟DOM树
创建真正DOM
数据改变
应用真正DOM
重新构建
DOM Diff
计算差异

Virtual Dom


```
<div class="container">
  <ul class="list">
    <li class="list_item">1</li>
    <li class="list_item">2</li>
    <li class="list_item">3</li>
    <li class="list_item">4</li>
    <li class="list_item">5</li>
  </ul>
</div>

```

dom结构

Virtual Dom


```
{
  "tagName": "div",
  "props": {
    "className": "container"
  },
  "children": [
    {
      "tagName": "ul",
      "props": {
        "className": "list"
      },
      "children": [
        { "tagName": "li", "props": { "className": "list_item" }, "children": [1] },
        { "tagName": "li", "props": { "className": "list_item" }, "children": [2] },
        { "tagName": "li", "props": { "className": "list_item" }, "children": [3] },
        { "tagName": "li", "props": { "className": "list_item" }, "children": [4] },
        { "tagName": "li", "props": { "className": "list_item" }, "children": [5] }
      ]
    }
  ]
}

```

json描述

Virtual Dom

Virtual Dom


```
function createElement (tagName, props, ...children) {
  return { tagName, props, children }
}
```

Virtual Dom


```
createElement('div', { className: 'container' },
  createElement('ul', { className: 'list' },
    createElement('li', { className: 'list_item' }, 1),
    createElement('li', { className: 'list_item' }, 2),
    createElement('li', { className: 'list_item' }, 3),
    createElement('li', { className: 'list_item' }, 4),
    createElement('li', { className: 'list_item' }, 5)
  )
)
```

Virtual Dom


```
React.createElement('div', { className: 'container' },
  React.createElement('ul', { className: 'list' },
    React.createElement('li', { className: 'list_item' }, 1),
    React.createElement('li', { className: 'list_item' }, 2),
    React.createElement('li', { className: 'list_item' }, 3),
    React.createElement('li', { className: 'list_item' }, 4),
    React.createElement('li', { className: 'list_item' }, 5)
  )
)
```

https://facebook.github.io/react/docs/jsx-in-depth.html

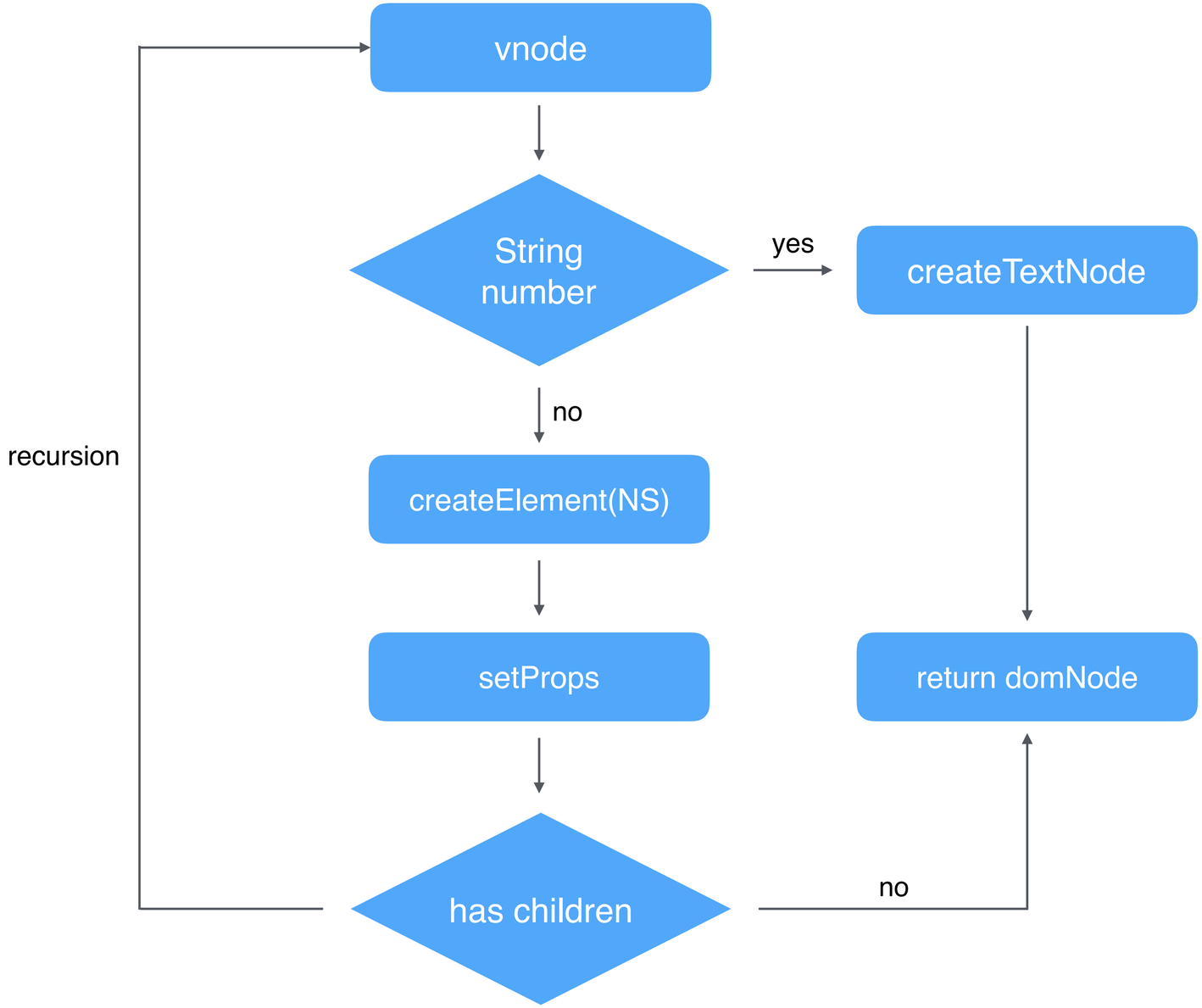
Virtual Dom


```jsx
const node = (
  <div className='test'>
    <ul className='list'>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
    </ul>
  </div>
)
```

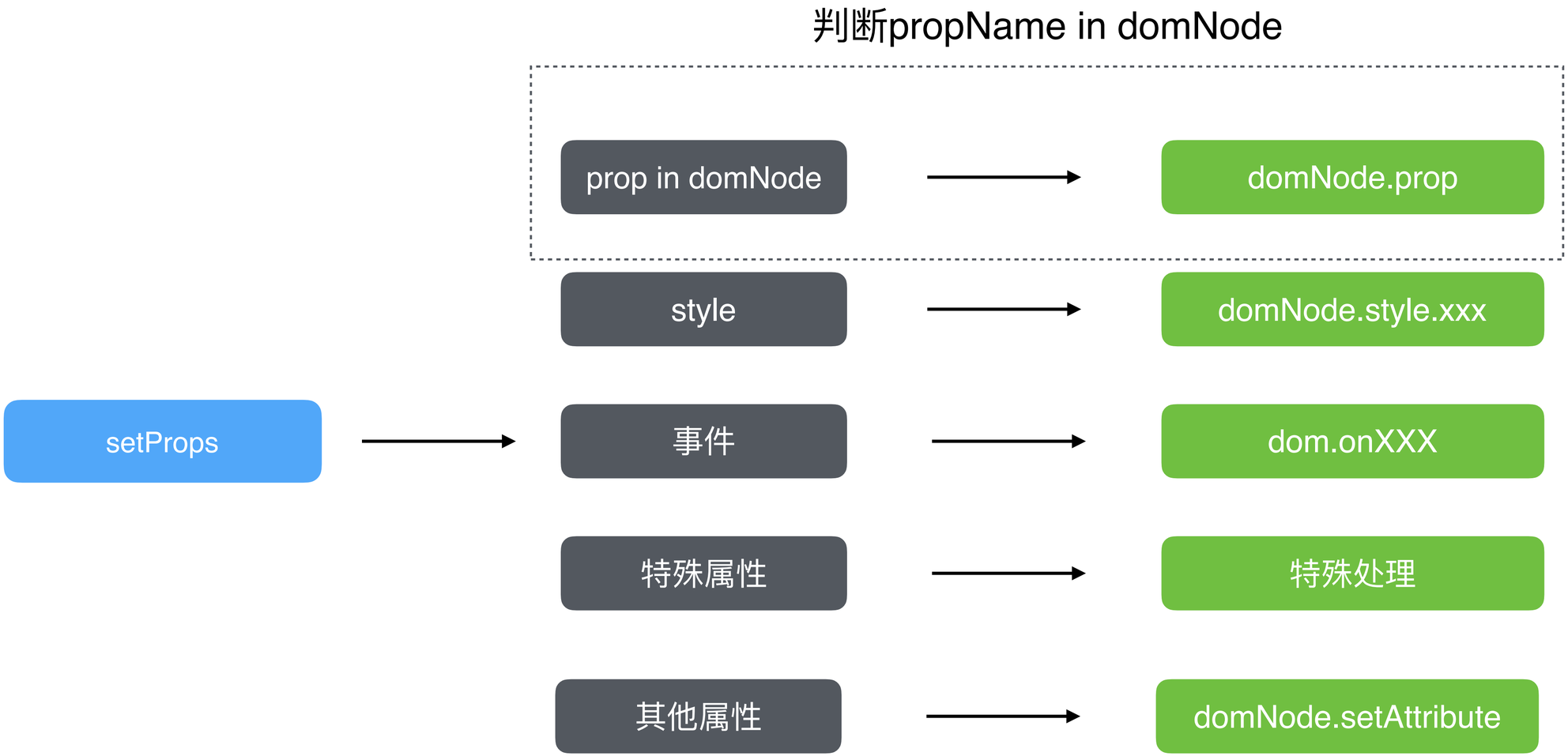
babel-plugin-transform-react-jsx

Virtual Dom

Virtual Dom


vnode
String
number
yes
createTextNode
no
recursion
createElement(NS)
setProps
return domNode
has children
no

Virtual Dom

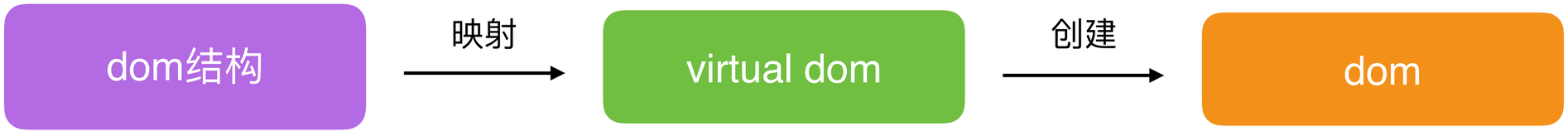

判断propName in domNode
prop in domNode
domNode.prop
style
domNode.style.xxx
setProps
事件
dom.onXXX
特殊属性
特殊处理
其他属性
domNode.setAttribute

Virtual Dom

事件处理

```
if (/^on/.test(propName)) {
  domNode[propName.toLowerCase()] = propValue
}
```

Virtual Dom



Virtual Dom

02

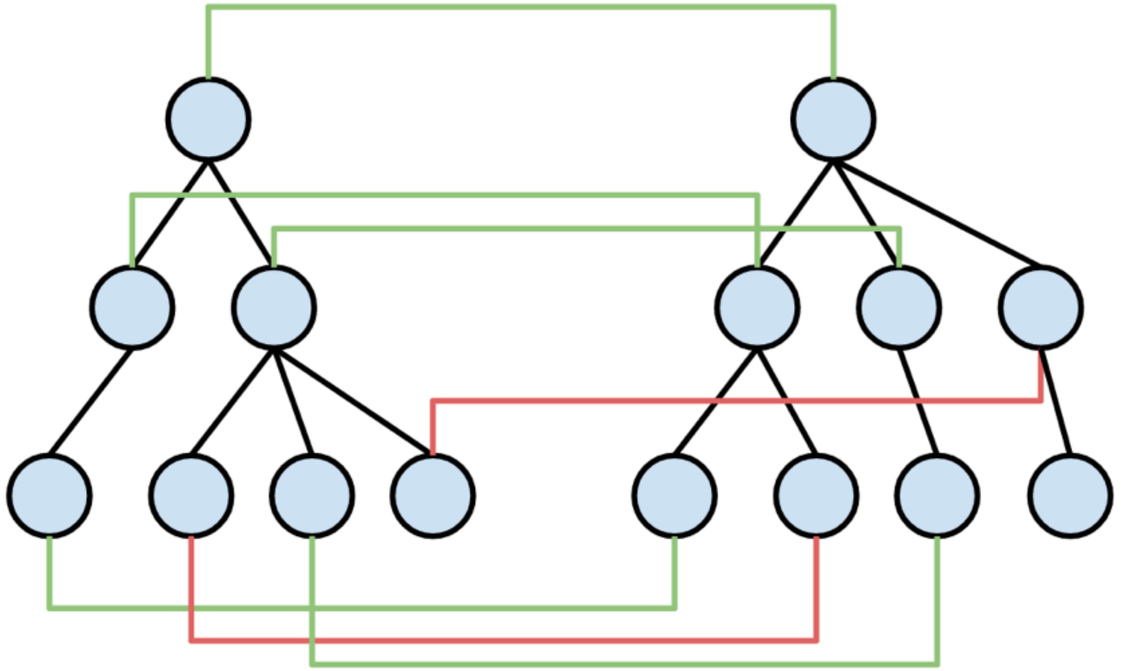
一个小栗子

3

Virtual Dom
diff

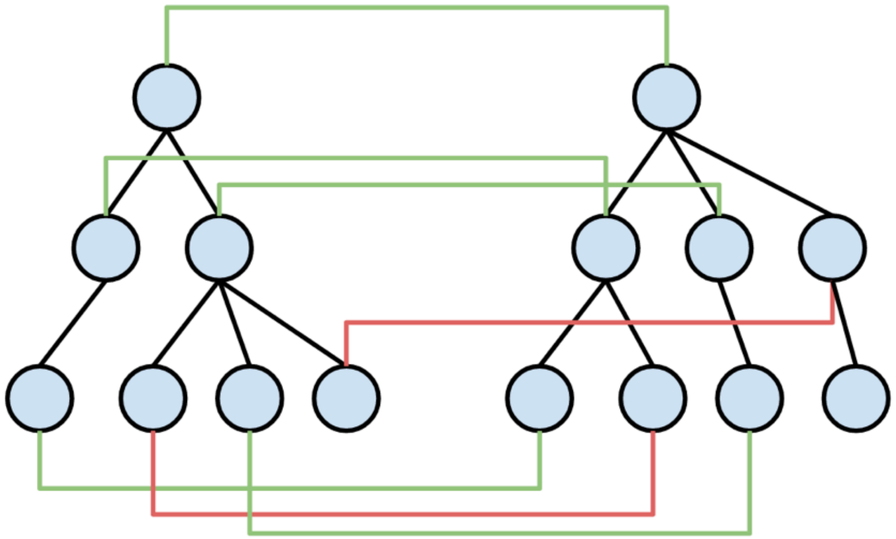
diff

Virtual Dom diff



tree diff

Virtual Dom diff



tree diff

时间复杂度o(n^3)

不适合web

Virtual Dom diff

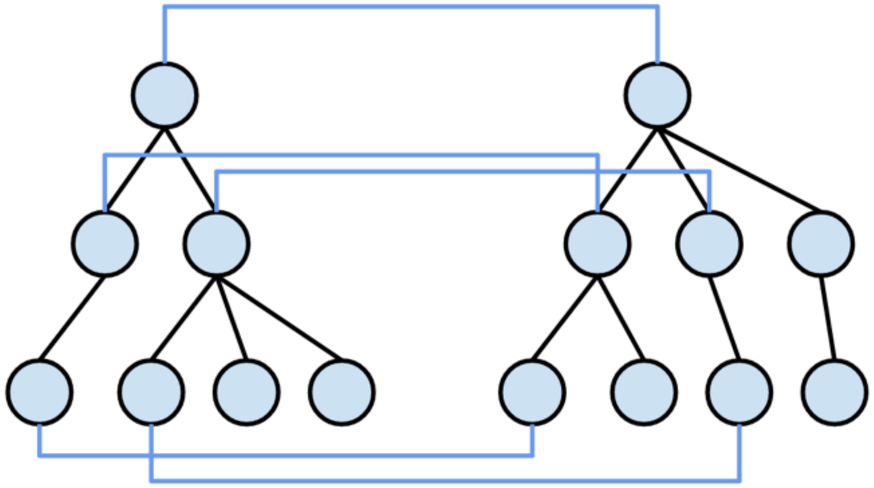
一点微小的工作

两个相同组件产生类似的DOM结构，不同的组件产生不同的DOM结构

对于同一层次的一组子节点，它们可以通过唯一的id进行区分

对树进行逐层比较

Virtual Dom diff

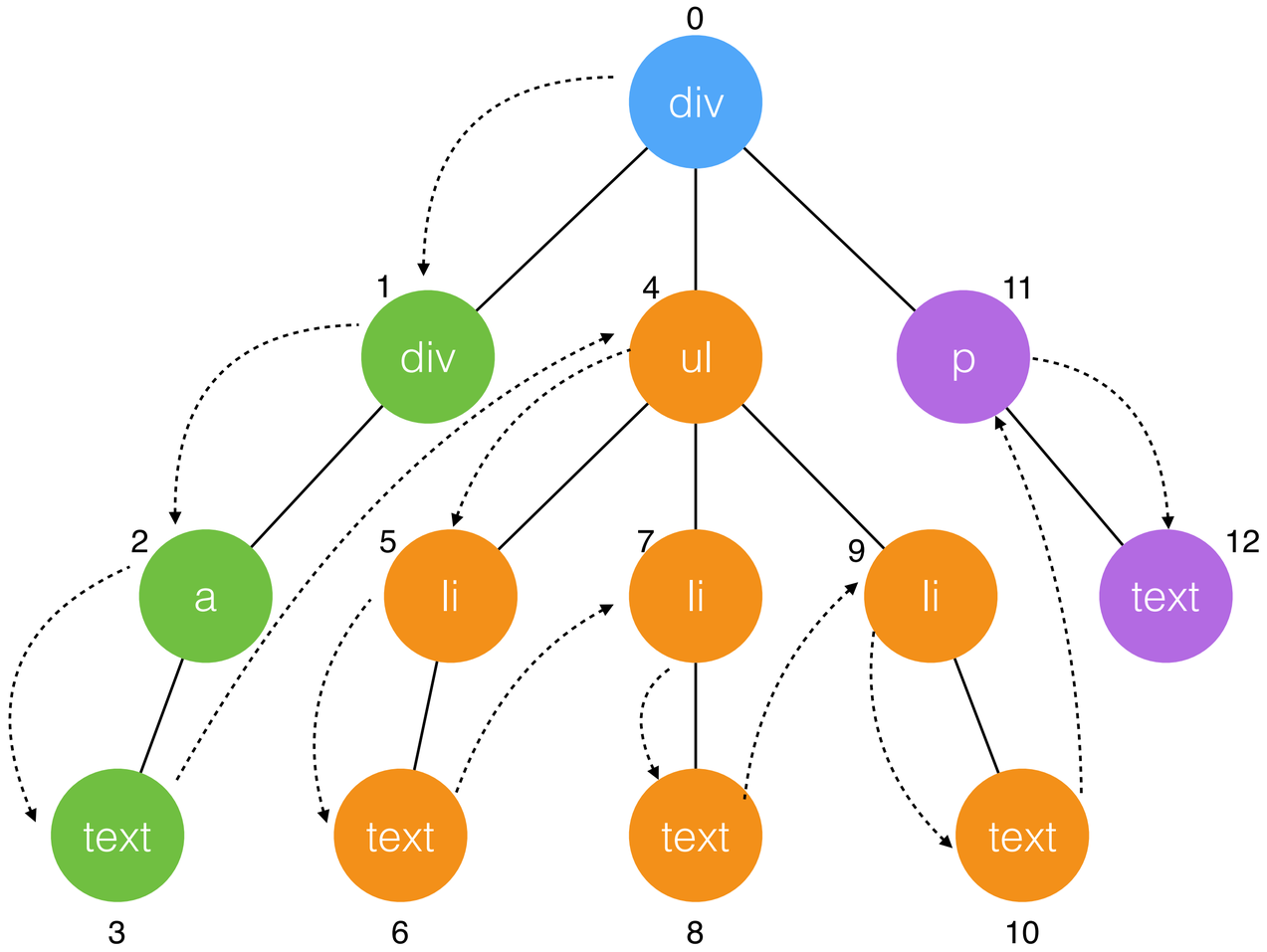


Virtual DOM tree diff

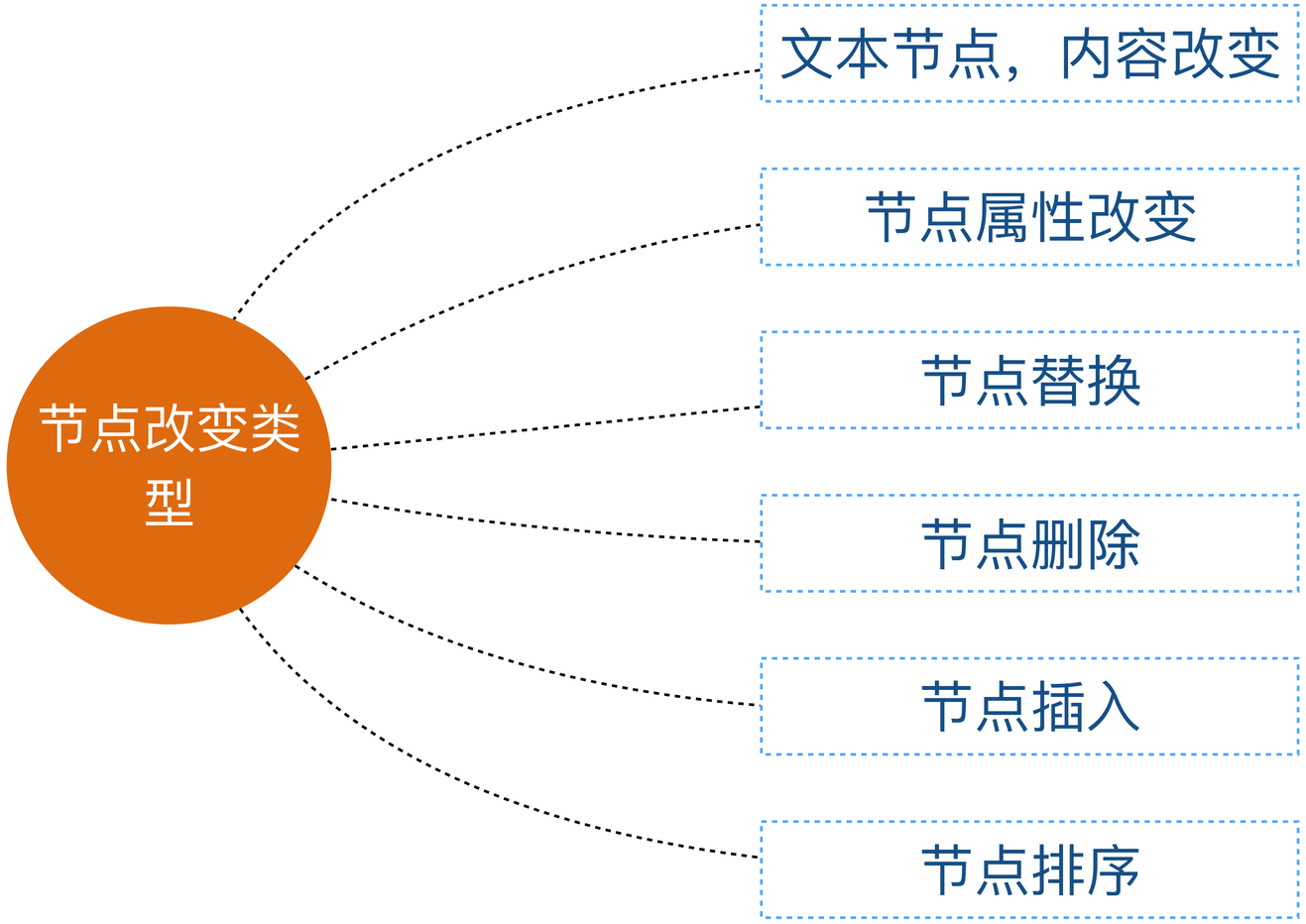
深度优先，逐层遍历

时间复杂度o(n)

Virtual Dom diff



Virtual Dom diff



Virtual Dom diff


```
1   function diff(a, b) {
2     let patches = { old: a }
3     walk(a, b, patches, 0)
4     return patches
5   }
6
7   function walk(a, b, patches, index) {
8
9   }
```

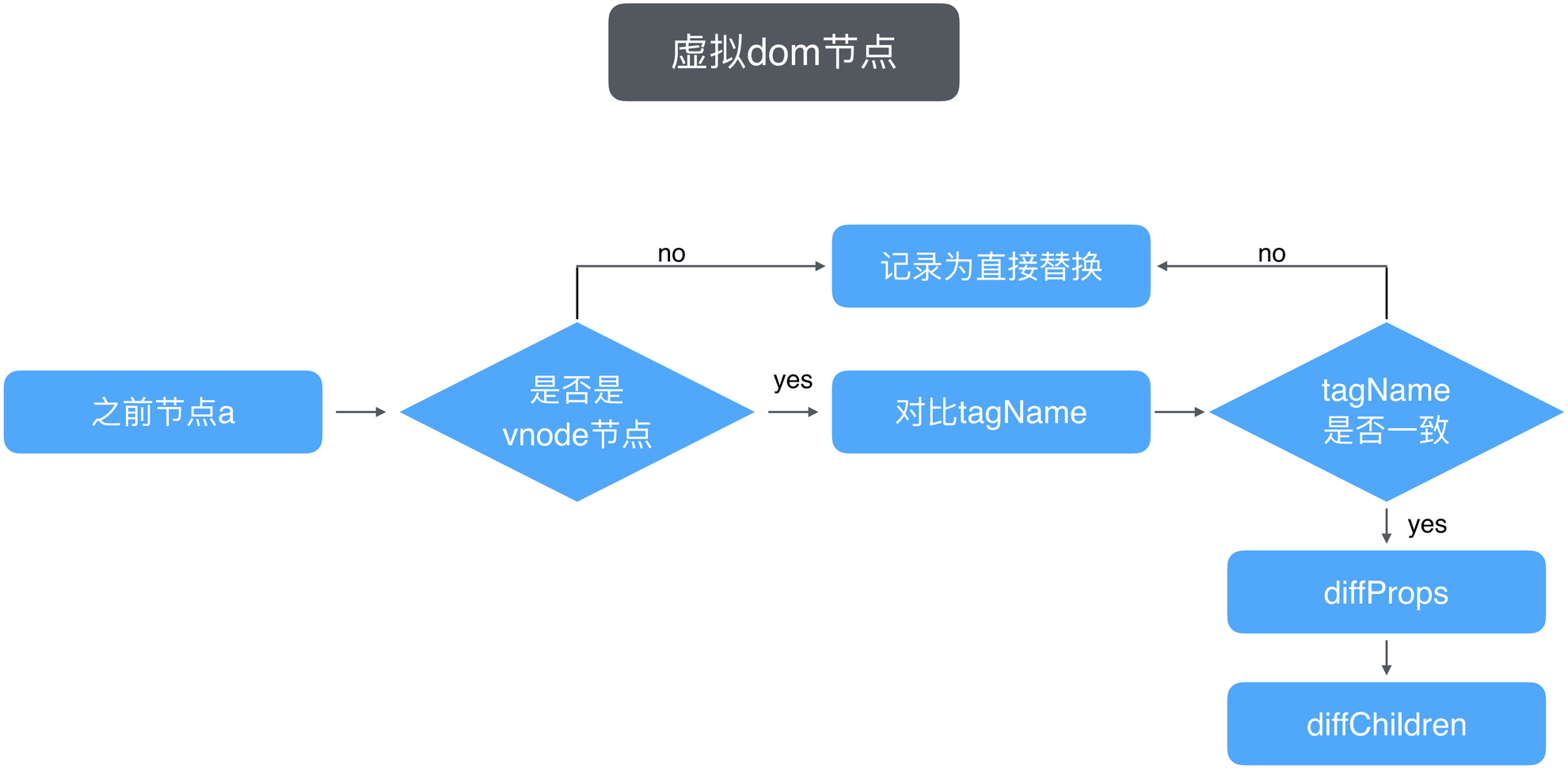
diff.js

Virtual Dom diff

文本节点

```
if (isText(b)) {
  if (!isText(a)) {
    apply = appendPatch(apply, { type: 'text', patch: b, old: a })
  } else if (a !== b) {
    apply = appendPatch(apply, { type: 'text', patch: b, old: a })
  }
}
```

Virtual Dom diff



Virtual Dom diff

虚拟dom节点

```
if (!isVNode(a)) {
  apply = appendPatch(apply, { type: 'vnode', patch: b, old: a })
} else if (a.tagName === b.tagName) {
  const propsPatch = diffProps(a.props, b.props)
  if (propsPatch) {
    apply = appendPatch(apply, { type: 'props', patch: propsPatch, old: a })
  }
  apply = diffChildren(a, b, apply, patches, index)
} else {
  apply = appendPatch(apply, { type: 'vnode', patch: b, old: a })
}
```

Virtual Dom diff

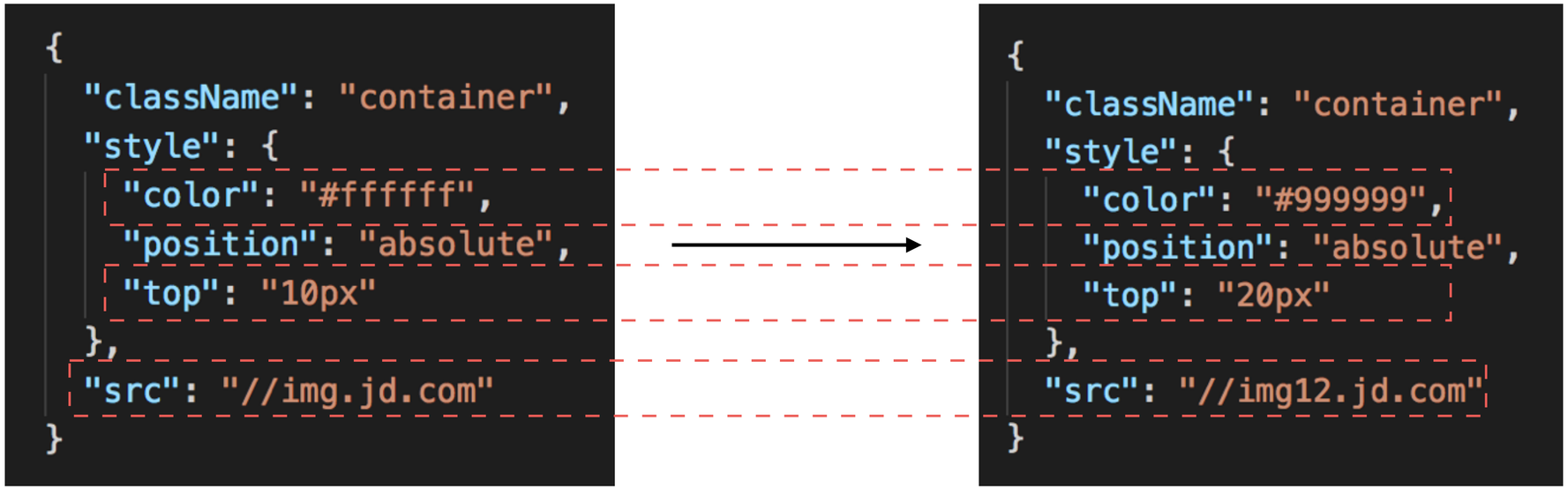
深度遍历对比对象

递归遍历

列表排序

Virtual Dom diff

diffProps



Virtual Dom diff

diffProps

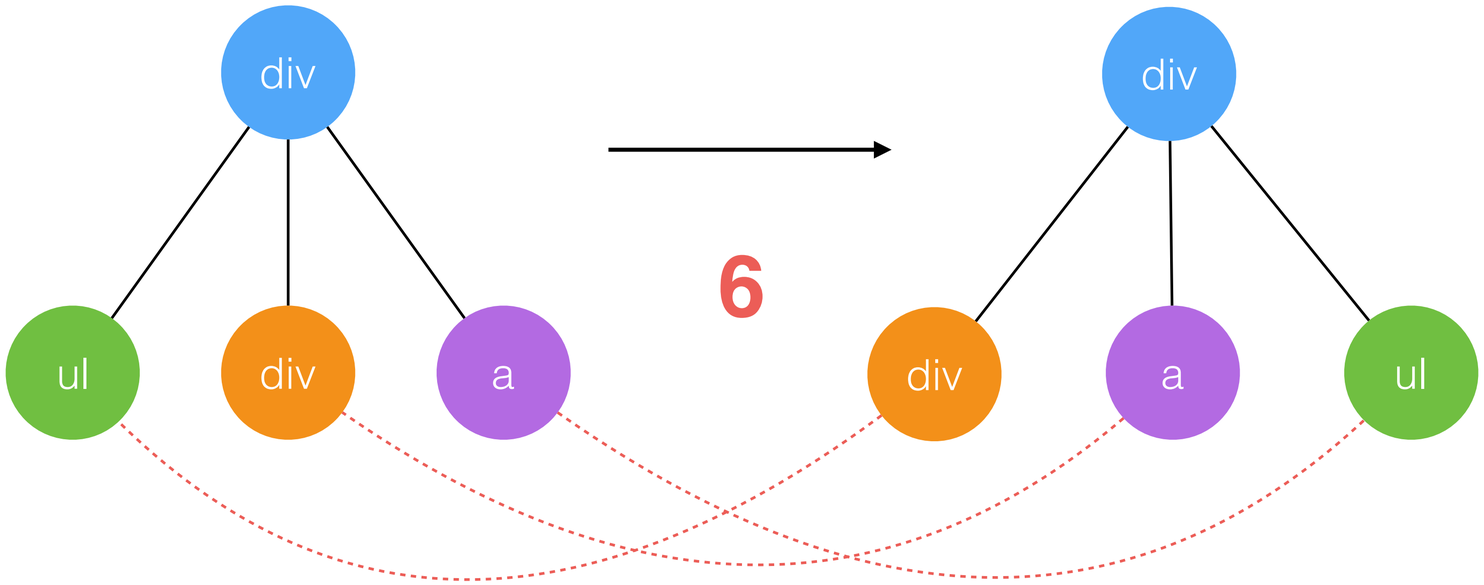
```
{
  "style": {
    "color": "#999999",
    "top": "20px"
  },
  "src": "//img12.jd.com"
}
```

props patch

Virtual Dom diff

diffChildren

遍历递归子节点，调用walk方法进行diff

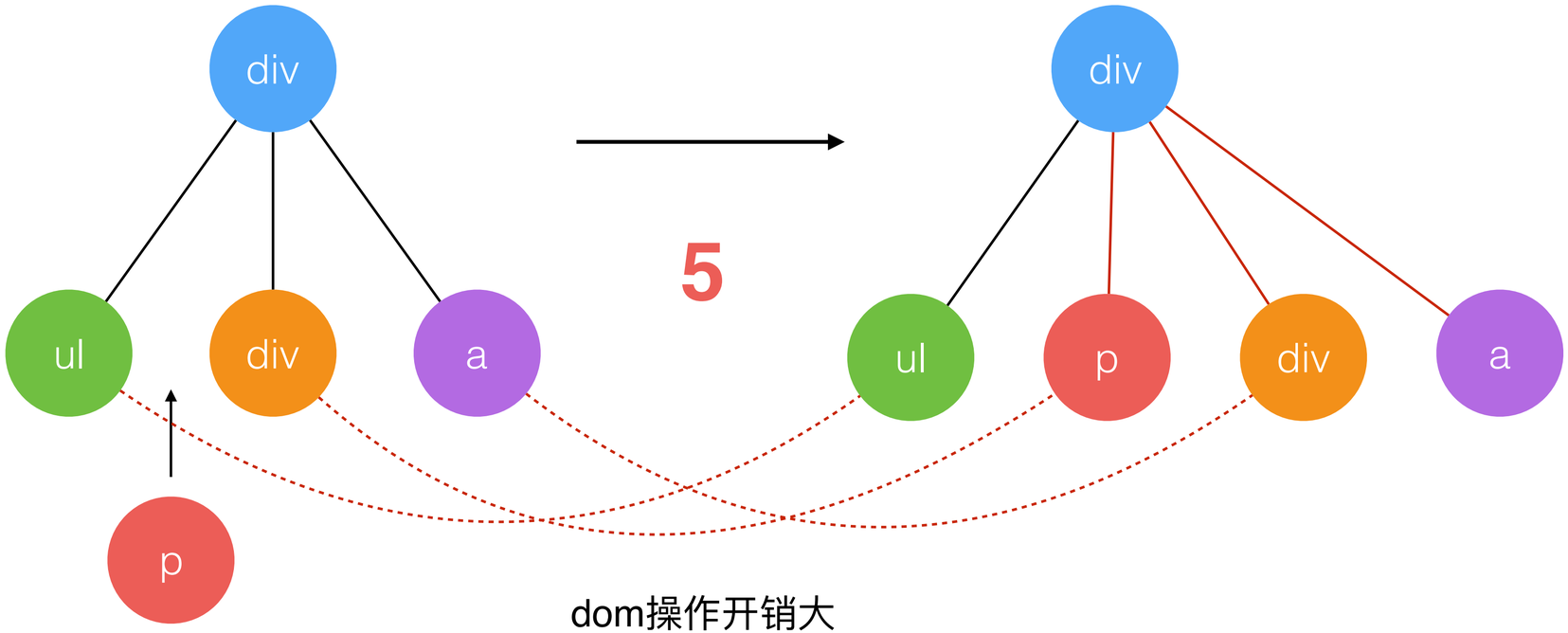


节点未经排序，一一对比

Virtual Dom diff

diffChildren

节点未经排序，一一对比



dom操作开销大

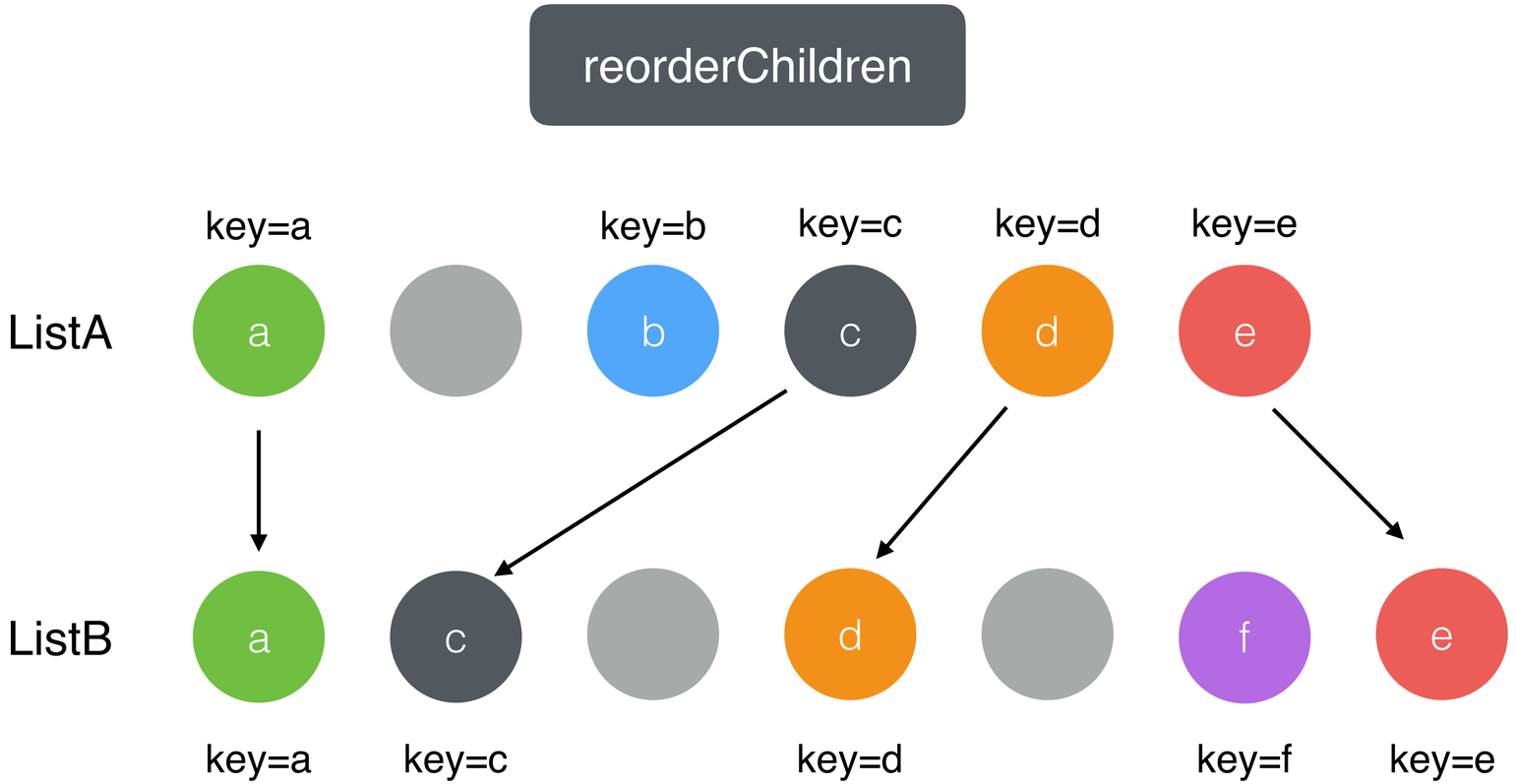
Virtual Dom diff

02

reorderChildren

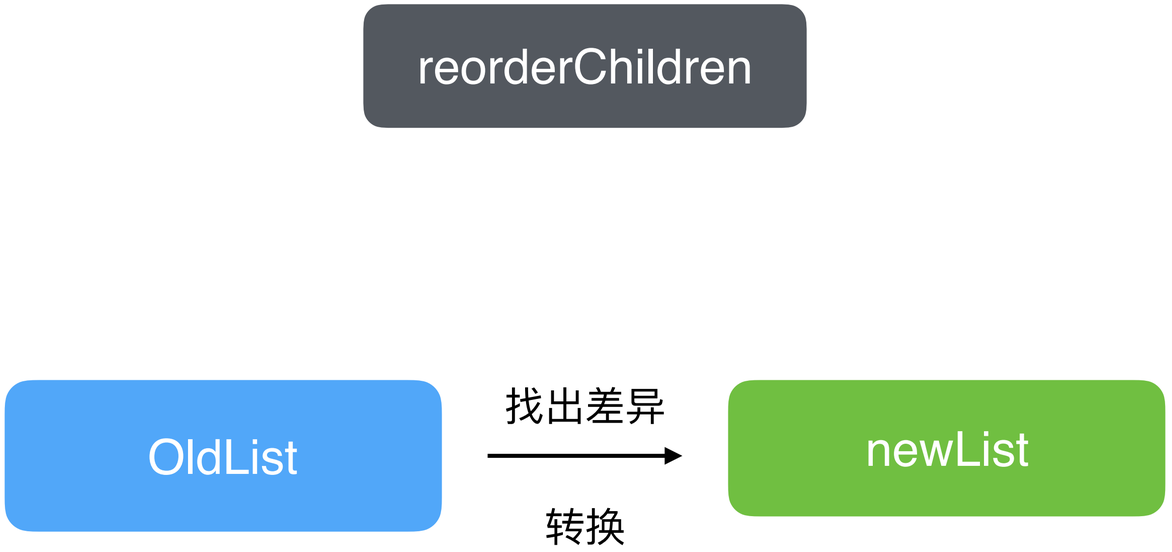
给元素增加key属性，基于key属性进行重新排序

Virtual Dom diff



基于key重新排序

Virtual Dom diff

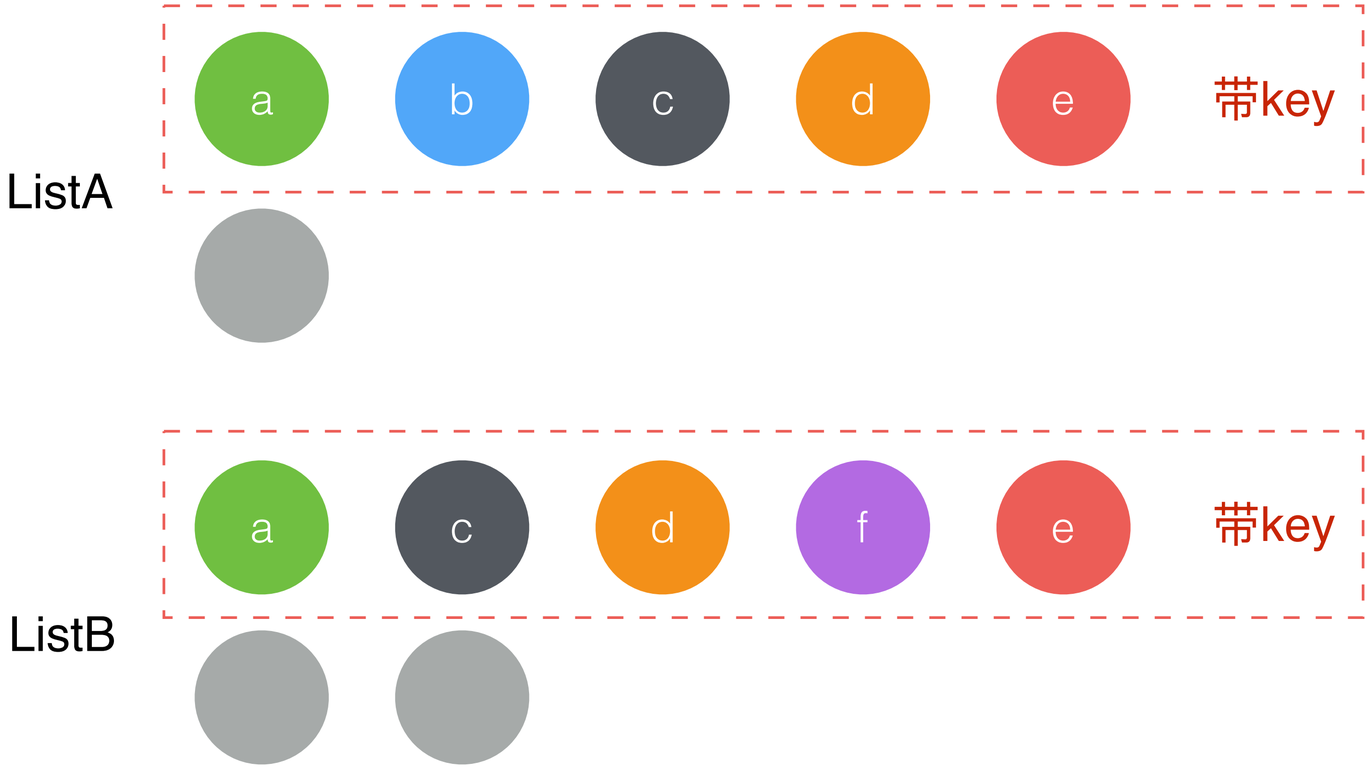


将newList转换为OldList的模样

Virtual Dom diff

reorderChildren

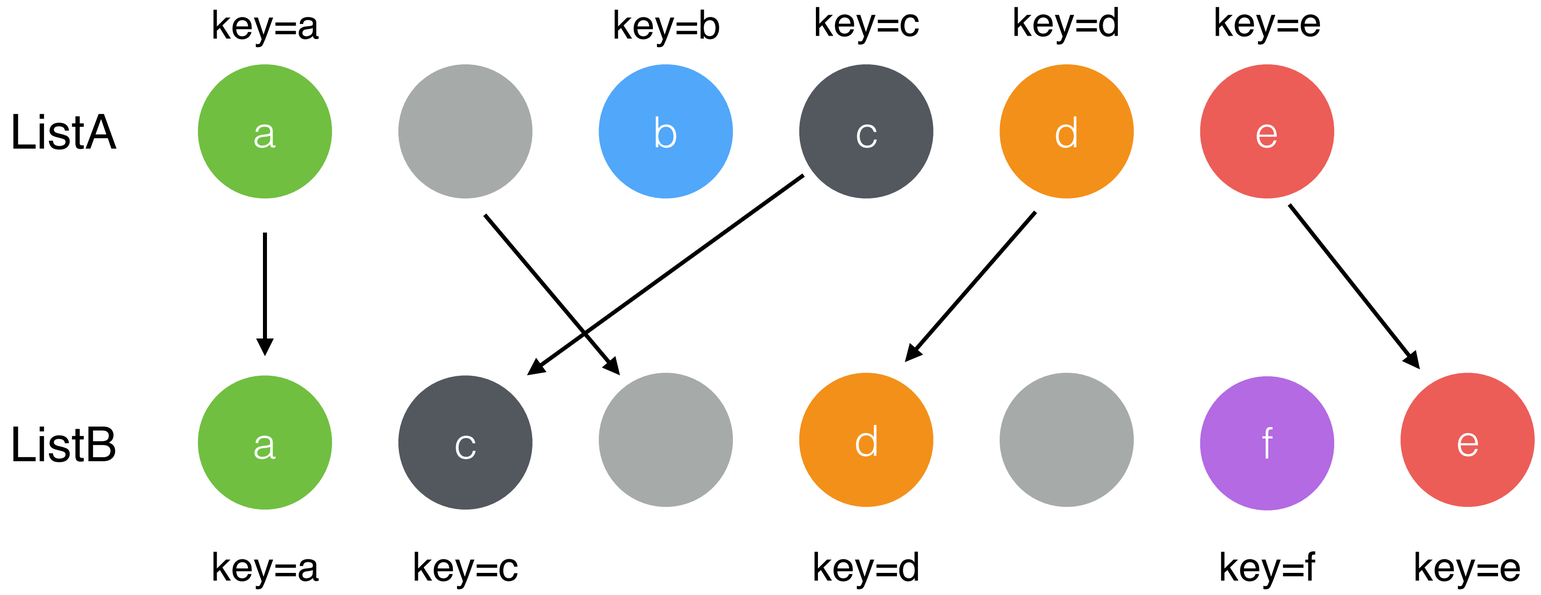
1、整理list，找出带有key属性的元素和不带key属性的元素



Virtual Dom diff

reorderChildren

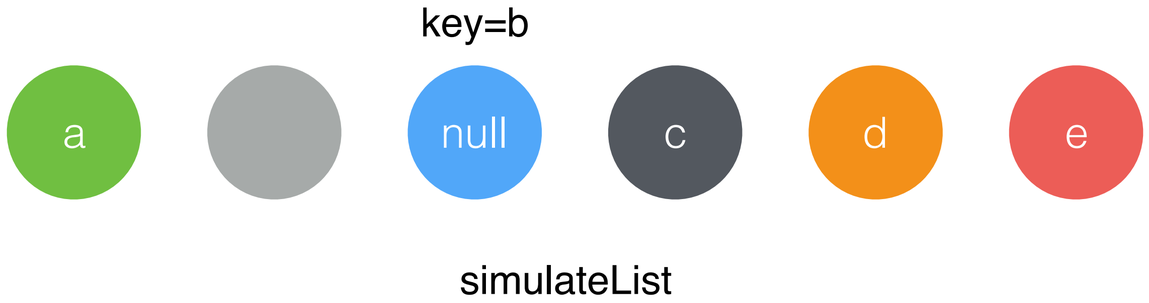
2、根据ListA的带key节点顺序，对ListB进行还原，不带key元素根据位置进行对应，ListB中已经不存在带key元素标记为删除，ListB中新增的带key元素添加到后面，得到一个新list，命名为simulateList



Virtual Dom diff

reorderChildren

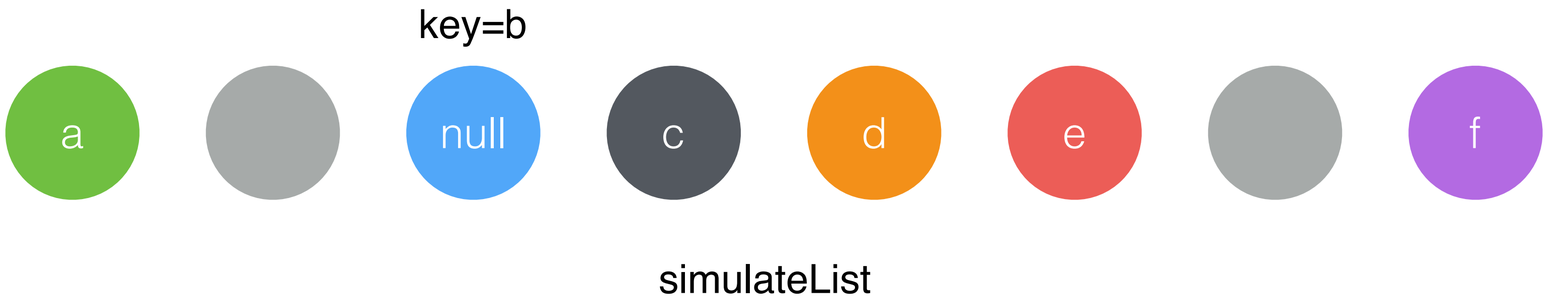
2、根据ListA的带key节点顺序，对ListB进行还原，不带key元素根据位置进行对应，ListB中已经不存在带key元素标记为删除，ListB中新增的带key元素添加到后面，得到一个新list，命名为simulateList



Virtual Dom diff

reorderChildren

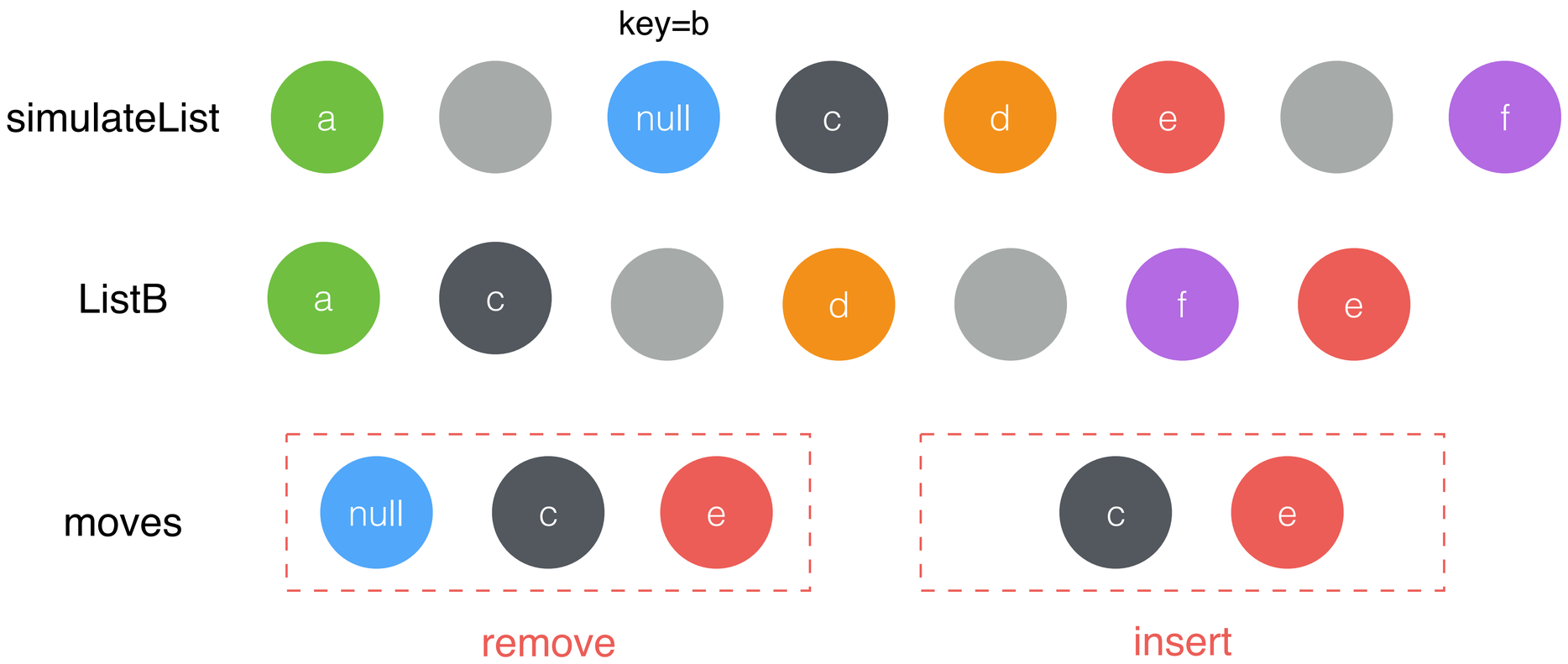
3、在上一步的基础上，将ListB中新增带key元素和不带key元素添加到simulateList后面



Virtual Dom diff

reorderChildren

4、找到将simulateList转换成ListB的操作



Virtual Dom diff

02

问题

何时应给元素增加key属性?

patch

Virtual Dom diff

patch

diff结果

```
{
  "old": VNode,
  "0": [
    {"type": "props", patch: { src: "//" }, old: VNode},
    {"type": "insert", patch: VNode, old: VNode}
  ],
  "1": [
    {"type": "insert", patch: VNode, old: VNode}
  ],
  "2": [
    {"type": "props", patch: { style: {color: '#fff'} }, old: VNode}
  ],
  "5": [
    {"type": "replace", patch: VNode, old: VNode}
  ],
  "6": [
    {"type": "reorder", patch: {remove: {}, insert: {}}, old: VNode}
  ]
}
```

Virtual Dom diff

02

patch

根据diff结果索引到对应dom

使用二分查找法对dom树进行检索

```
{
  0: dom0,
  1: dom1,
  2: dom2,
  5: dom5,
  6: dom6,
}
```

Virtual Dom diff

patch

根据diff结果索引到对应dom

```
function patchSingle (domNode, vpatch) {
  let type = vpatch.type
  let oldVNode = vpatch.old
  let patchObj = vpatch.patch

  switch (type) {
    case 'text':
      return patchVText(domNode, patchObj)
    case 'replace':
      return patchVNode(domNode, patchObj)
    case 'insert':
      return patchInsert(domNode, patchObj)
    case 'props':
      return patchProperties(domNode, patchObj, oldVNode.props)
    case 'reorder':
      return patchOrder(domNode, patchObj)
    case 'remove':
      return patchRemove(domNode, oldVNode)
    default:
      return domNode
  }
}
```

patch.js

Virtual Dom diff

调用方法

```
let patches = diff(lastVNode, newVNode)

let domNode = patch(lastDom, patches)
```

Virtual Dom diff

02

一个小栗子

课后作业

实现一个简单virtual dom

02

OPEN ORIENTED

凹 凸 实 验 室