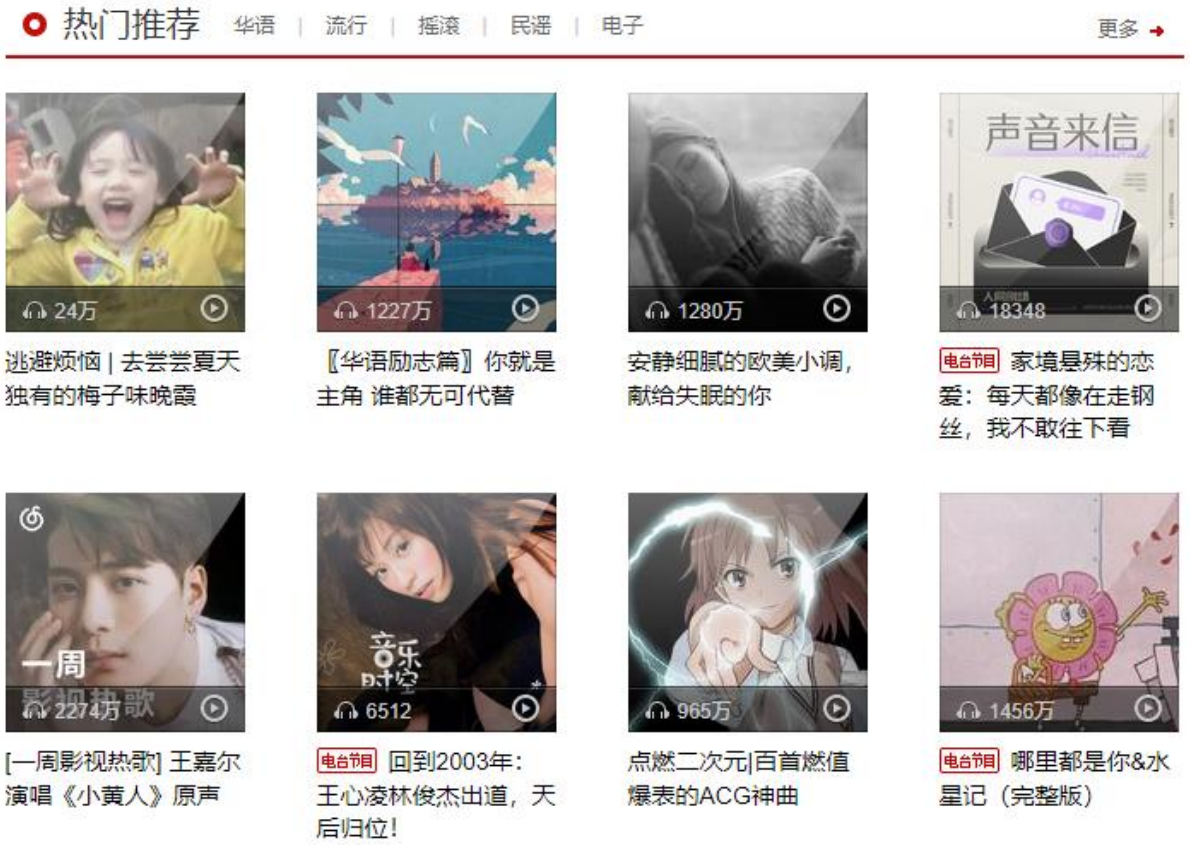
Vue基础 – 模板语法（二）

王红元 coderwhy

目录
content

列表渲染

- **在真实开发中，我们往往会从服务器拿到一组数据，并且需要对其进行渲染。**
  - 这个时候我们可以使用v-for来完成；
  - v-for类似于JavaScript的for循环，可以用于遍历一组数据；



v-for基本使用

- **v-for的基本格式是 "item in 数组"：**
  - 数组通常是来自data或者prop，也可以是其他方式；
  - item是我们给每项元素起的一个别名，这个别名可以自定来定义；
- **我们知道，在遍历一个数组的时候会经常需要拿到数组的索引：**
  - 如果我们需要索引，可以使用格式： "(item, index) in 数组"；
  - 注意上面的顺序：数组元素项item是在前面的，索引项index是在后面的；

```
<template id="my-app">
  <h2>电影列表</h2>
  <ul>
    <li v-for="item in movies">{{item}}</li>
  </ul>
</template>
```

```
<template id="my-app">
  <h2>电影列表</h2>
  <ul>
    <li v-for="(item, index) in movies">{{index}}-{{item}}</li>
  </ul>
</template>
```

v-for支持的类型

- **v-for也支持遍历对象，并且支持有一二三个参数：**
  - 一个参数： "value in object";
  - 二个参数： "(value, key) in object";
  - 三个参数： "(value, key, index) in object";
- **v-for同时也支持数字的遍历：**
  - 每一个item都是一个数字；
- **v-for也可以遍历其他可迭代对象(Iterable)**

```
<template id="my-app">
  <h2>遍历对象</h2>
  <ul>
    <li v-for="(value, key, index) in info">
      {{index}} - {{key}} - {{value}}
    </li>
  </ul>
</template>
```

```
<template id="my-app">
  <ul>
    <li v-for="item in 10">{{item}}</li>
  </ul>
</template>
```

template元素

- **类似于v-if，你可以使用 template 元素来循环渲染一段包含多个元素的内容：**
  - 我们使用template来对多个元素进行包裹，而不是使用div来完成；

```
<template id="my-app">
  <ul>
    <template v-for="(value, key) in info">
      <li>{{key}}</li>
      <li>{{value}}</li>
      <hr>
    </template>
  </ul>
</template>
```

数组更新检测

- **Vue 将被侦听的数组的变更方法进行了包裹，所以它们也将会触发视图更新。**
- **这些被包裹过的方法包括：**
  - push()
  - pop()
  - shift()
  - unshift()
  - splice()
  - sort()
  - reverse()
- **替换数组的方法**
  - 上面的方法会直接修改原来的数组；
  - 但是某些方法不会替换原来的数组，而是会生成新的数组，比如 filter()、concat() 和 slice()；

v-for中的key是什么作用?

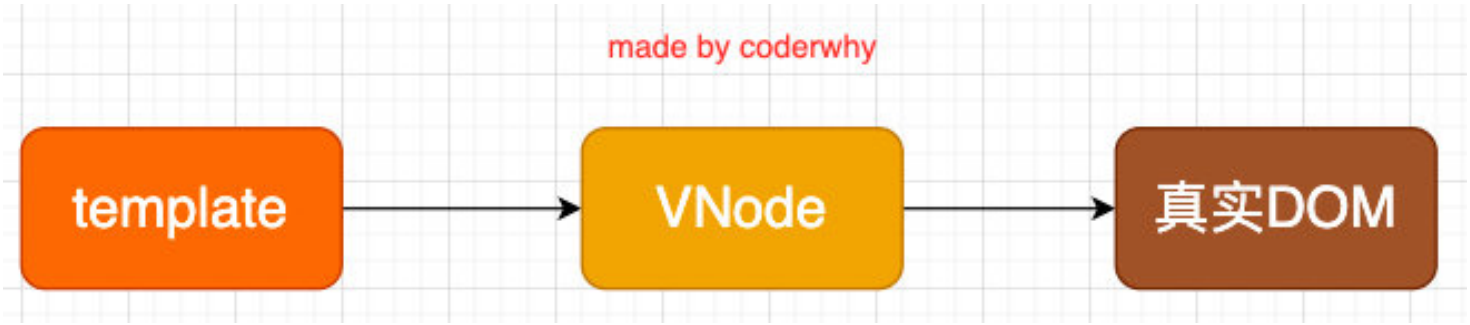
- **在使用v-for进行列表渲染时，我们通常会给元素或者组件绑定一个key属性。**
- **这个key属性有什么作用呢？我们先来看一下官方的解释：**
  - key属性主要用在Vue的虚拟DOM算法，在新旧nodes对比时辨识VNodes；
  - 如果不使用key，Vue会使用一种最大限度减少动态元素并且尽可能的尝试就地修改/复用相同类型元素的算法；
  - 而使用key时，它会基于key的变化重新排列元素顺序，并且会移除/销毁key不存在的元素；
- **官方的解释对于初学者来说并不好理解，比如下面的问题：**
  - 什么是新旧nodes，什么是VNode?
  - 没有key的时候，如何尝试修改和复用的?
  - 有key的时候，如何基于key重新排列的?

认识VNode

- **我们先来解释一下VNode的概念：**
  - 因为目前我们还没有比较完整的学习组件的概念，所以目前我们先理解HTML元素创建出来的VNode；
  - VNode的全称是Virtual Node，也就是虚拟节点；
  - 事实上，无论是组件还是元素，它们最终在Vue中表示出来的都是一个个VNode；
  - VNode的本质是一个JavaScript的对象；

```
<div class="title" style="font-size: 30px; color: red;">哈哈哈</div>
```

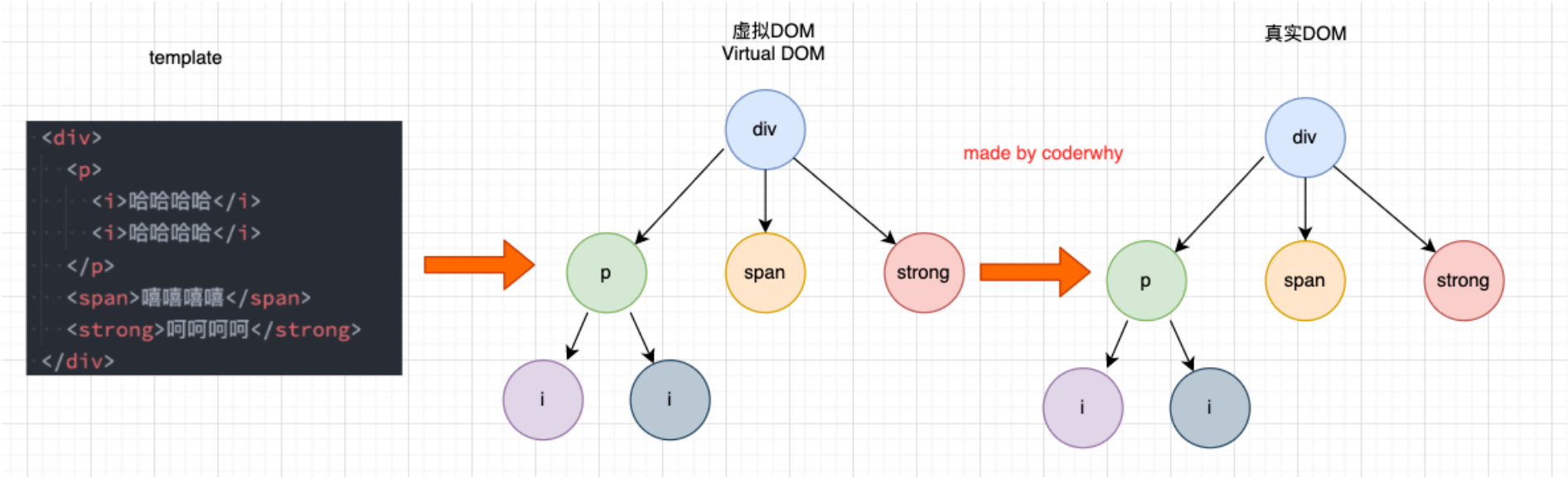
```
const vnode = {
  type: "div",
  props: {
    class: "title",
    style: {
      "font-size": "30px",
      color: "red",
    },
  },
  children: "哈哈哈",
};
```



虚拟DOM

- 如果我们不只是一个简单的div，而是有一大堆的元素，那么它们应该会形成一个VNode Tree：

```
<div>
  <p>
    <i>哈哈哈哈</i>
    <i>哈哈哈哈</i>
  </p>
  <span>嘻嘻嘻嘻</span>
  <strong>呵呵呵呵</strong>
</div>
```



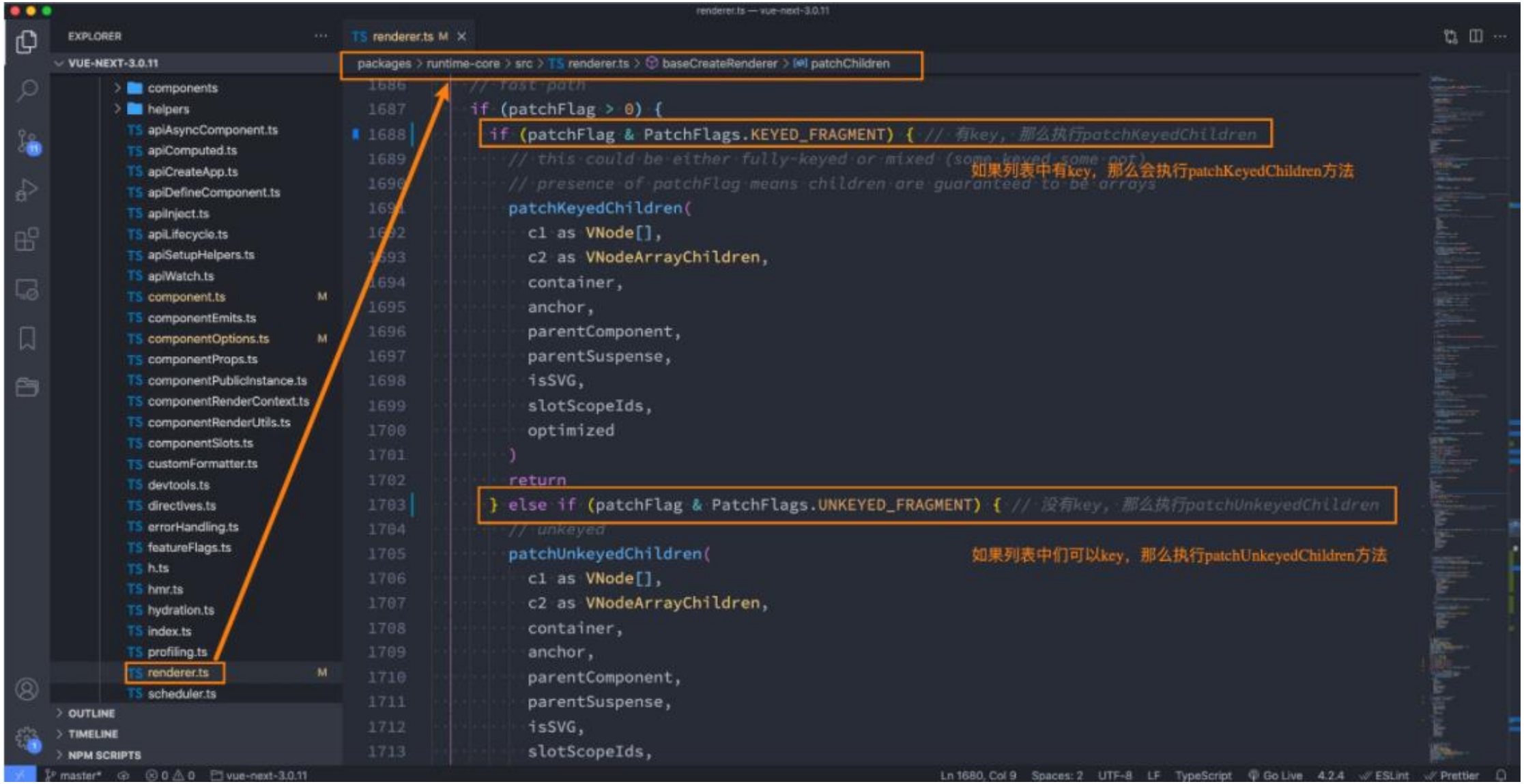
插入F的案例

- 我们先来看一个案例：这个案例是当我点击按钮时会在中间插入一个f；

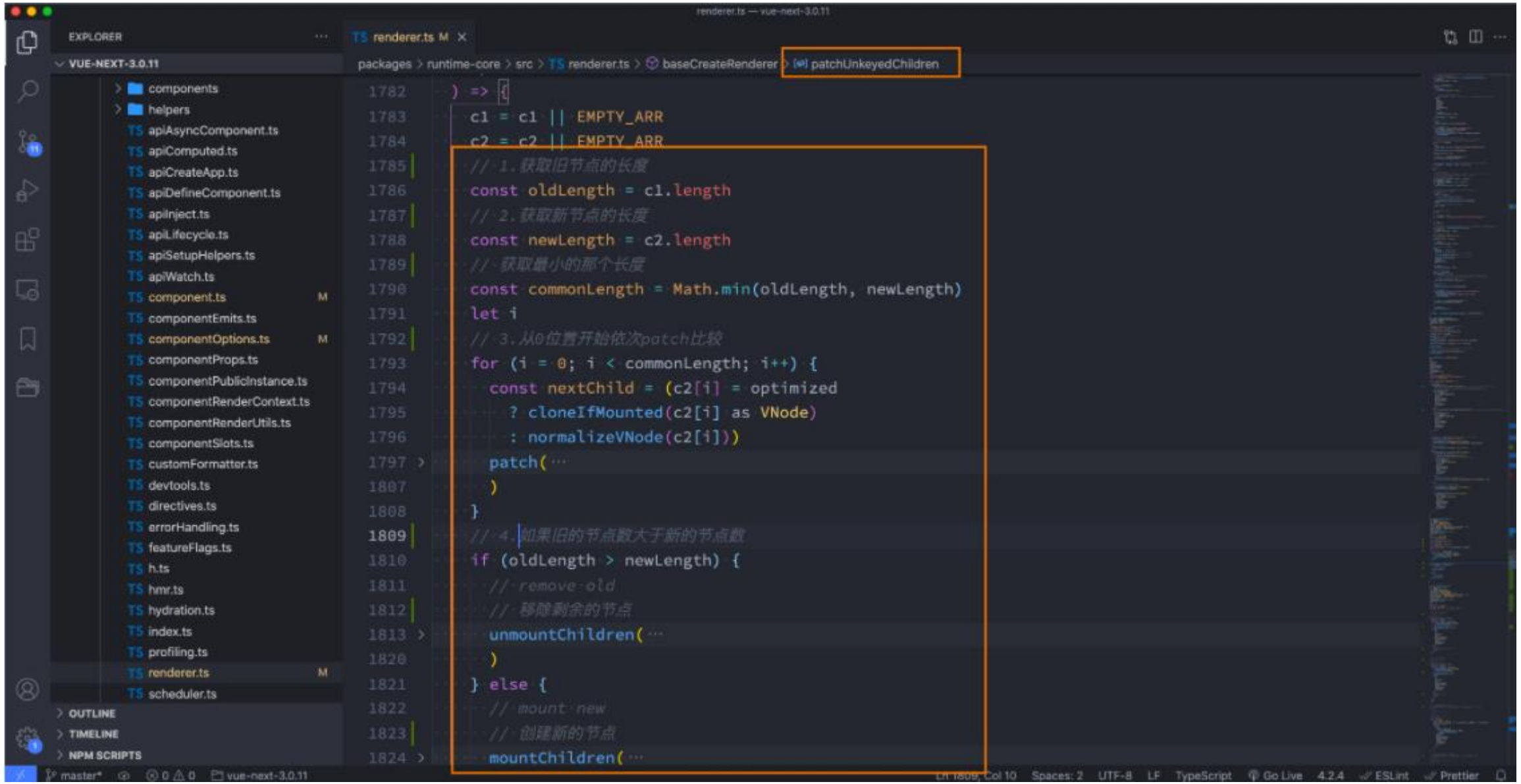
```
<template id="my-app">
  <ul>
    <li v-for="item in letters">{{item}}</li>
  </ul>
  <button @click="insertF">insert f</button>
</template>

<script src="../../dist/vue.global.js"></script>
<script>
  const App = {
    template: '#my-app',
    data() {
      return {
        letters: ['a', 'b', 'c', 'd']
      }
    },
    methods: {
      insertF() {
        this.letters.splice(2, 0, 'f');
      }
    }
  }

  Vue.createApp(App).mount('#app');
```

- 我们可以确定的是，这次更新对于ul和button是不需要进行更新，需要更新的是我们li的列表：
  - 在Vue中，对于相同父元素的子元素节点并不会重新渲染整个列表；
  - 因为对于列表中 a、b、c、d它们都是没有变化的；
  - 在操作真实DOM的时候，我们只需要在中间插入一个f的li即可；
- 那么Vue中对于列表的更新究竟是如何操作的呢?
  - Vue事实上会对于有key和没有key会调用两个不同的方法；
  - 有key，那么就使用 patchKeyedChildren方法；
  - 没有key，那么久使用 patchUnkeyedChildren方法；

Vue源码对于key的判断

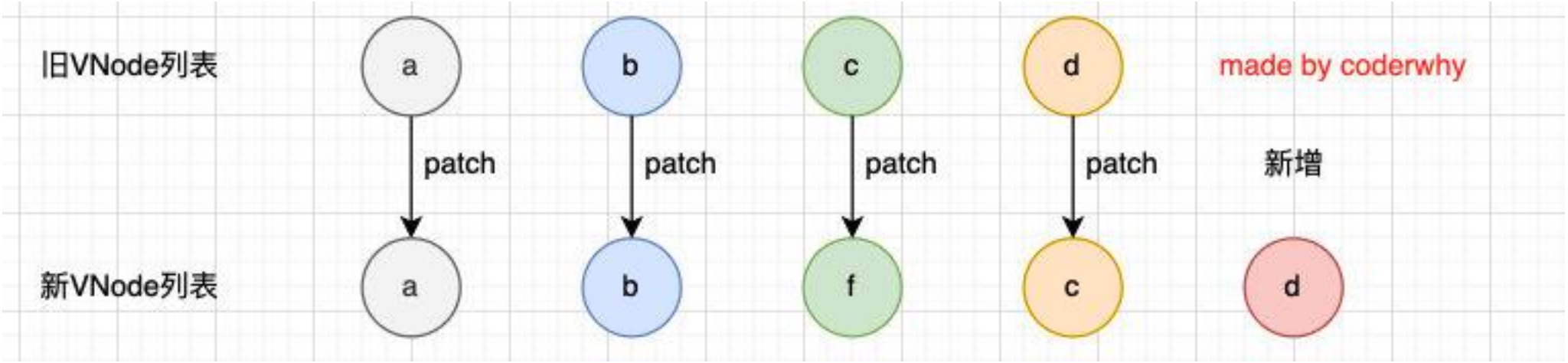


没有key的操作（源码）

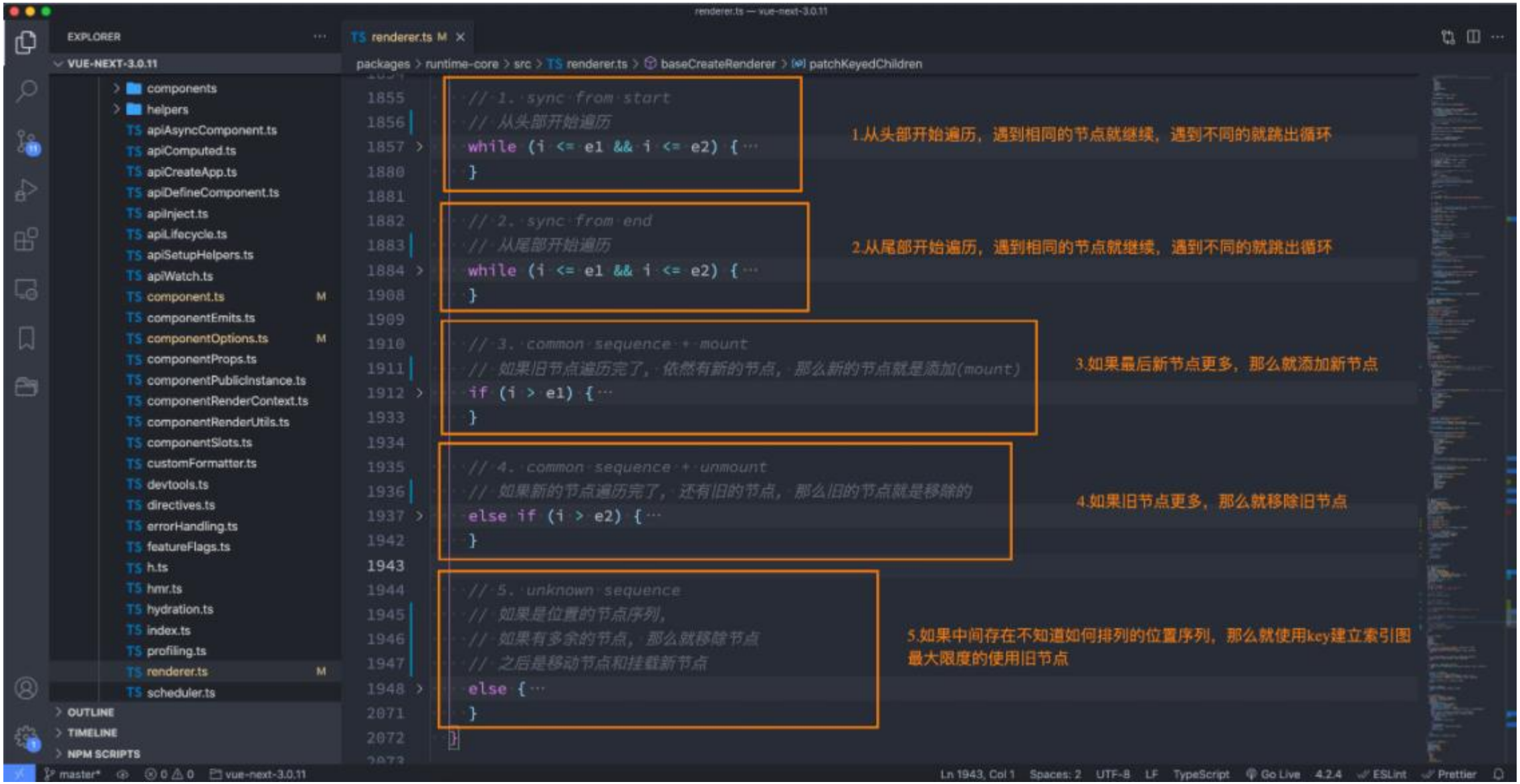

renderer.ts — vue-next-3.0.11
EXPLORER
VUE-NEXT-3.0.11
components
helpers
apiAsyncComponent.ts
apiComputed.ts
apiCreateApp.ts
apiDefineComponent.ts
apiInject.ts
apiLifecycle.ts
apiSetupHelpers.ts
apiWatch.ts
component.ts M
componentEmits.ts
componentOptions.ts M
componentProps.ts
componentPublicInstance.ts
componentRenderContext.ts
componentRenderUtils.ts
componentSlots.ts
customFormatter.ts
devtools.ts
directives.ts
errorHandling.ts
featureFlags.ts
h.ts
hmr.ts
hydration.ts
index.ts
profiling.ts
renderer.ts M
scheduler.ts
OUTLINE
TIMELINE
NPM SCRIPTS
renderer.ts M
packages > runtime-core > src > renderer.ts > baseCreateRenderer > patchUnkeyedChildren
) => {
c1 = c1 || EMPTY_ARR
c2 = c2 || EMPTY_ARR
// 1.获取旧节点的长度
const oldLength = c1.length
// 2.获取新节点的长度
const newLength = c2.length
// 获取最小的那个长度
const commonLength = Math.min(oldLength, newLength)
let i
// 3.从0位置开始依次patch比较
for (i = 0; i < commonLength; i++) {
const nextChild = (c2[i] = optimized
? cloneIfMounted(c2[i] as VNode)
: normalizeVNode(c2[i]))
patch(···
)
}
// 4.如果旧的节点数大于新的节点数
if (oldLength > newLength) {
// remove old
// 移除剩余的节点
unmountChildren(···
)
} else {
// mount new
// 创建新的节点
mountChildren(···
master*
vue-next-3.0.11
Spaces: 2
UTF-8
LF
TypeScript
Go Live
ESLint
Prettier

没有key的过程如下

- **我们会发现上面的diff算法效率并不高：**
  - c和d来说它们事实上并不需要有任何的改动；
  - 但是因为我们的c被f所使用了，所有后续所有的内容都要一次进行改动，并且最后进行新增；

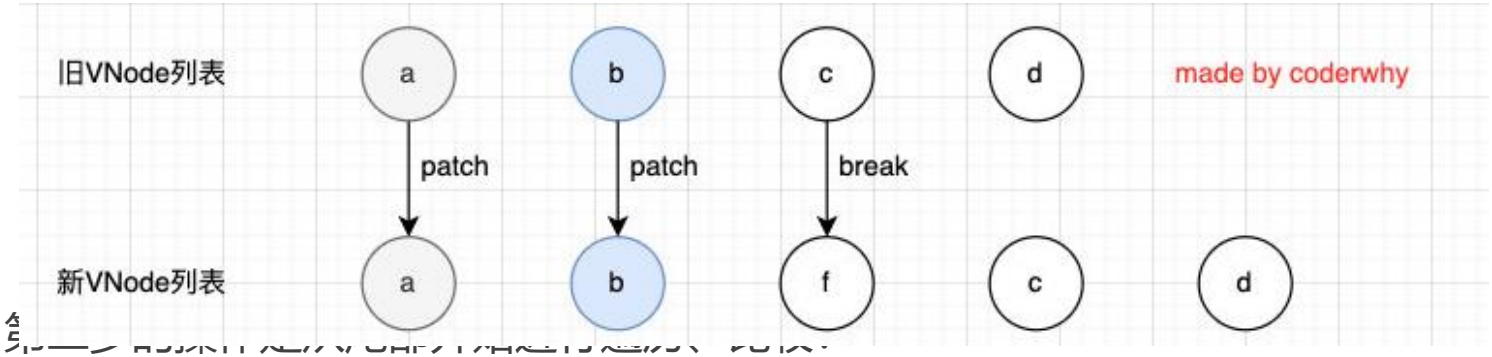


有key执行操作（源码）

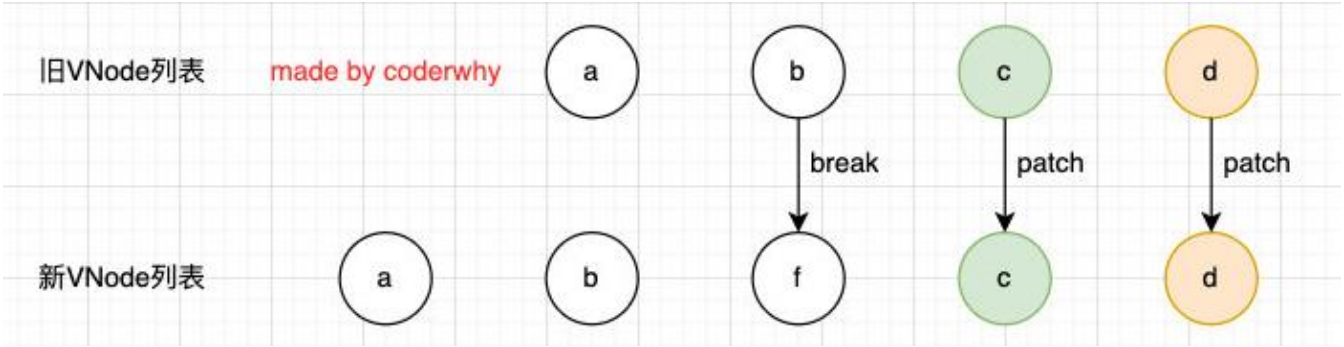

renderer.ts — vue-next-3.0.11
EXPLORER
VUE-NEXT-3.0.11
components
helpers
apiAsyncComponent.ts
apiComputed.ts
apiCreateApp.ts
apiDefineComponent.ts
apiInject.ts
apiLifecycle.ts
apiSetupHelpers.ts
apiWatch.ts
component.ts M
componentEmits.ts
componentOptions.ts M
componentProps.ts
componentPublicInstance.ts
componentRenderContext.ts
componentRenderUtils.ts
componentSlots.ts
customFormatter.ts
devtools.ts
directives.ts
errorHandling.ts
featureFlags.ts
h.ts
hmr.ts
hydration.ts
index.ts
profiling.ts
renderer.ts M
scheduler.ts
OUTLINE
TIMELINE
NPM SCRIPTS
renderer.ts M
packages > runtime-core > src > renderer.ts > baseCreateRenderer > patchKeyedChildren
1855 // 1. sync from start
1856 // 从头部开始遍历
1857 while (i <= e1 && i <= e2) {…
1880 }
1881
1882 // 2. sync from end
1883 // 从尾部开始遍历
1884 while (i <= e1 && i <= e2) {…
1908 }
1909
1910 // 3. common sequence + mount
1911 // 如果旧节点遍历完了，依然有新的节点，那么新的节点就是添加(mount)
1912 if (i > e1) {…
1933 }
1934
1935 // 4. common sequence + unmount
1936 // 如果新的节点遍历完了，还有旧的节点，那么旧的节点就是移除的
1937 else if (i > e2) {…
1942 }
1943
1944 // 5. unknown sequence
1945 // 如果是位置的节点序列，
1946 // 如果有多余的节点，那么就移除节点
1947 // 之后是移动节点和挂载新节点
1948 else {…
2071 }
2072 }
1.从头部开始遍历，遇到相同的节点就继续，遇到不同的就跳出循环
2.从尾部开始遍历，遇到相同的节点就继续，遇到不同的就跳出循环
3.如果最后新节点更多，那么就添加新节点
4.如果旧节点更多，那么就移除旧节点
5.如果中间存在不知道如何排列的位置序列，那么就使用key建立索引图
最大限度的使用旧节点
master*
vue-next-3.0.11
Ln 1943, Col 1
Spaces: 2
UTF-8
LF
TypeScript
Go Live
4.2.4
ESLint
Prettier

有key的diff算法如下（一）

- **第一步的操作是从头开始进行遍历、比较：**
  - a和b是一致的会继续进行比较；
  - c和f因为key不一致，所以就会break跳出循环；

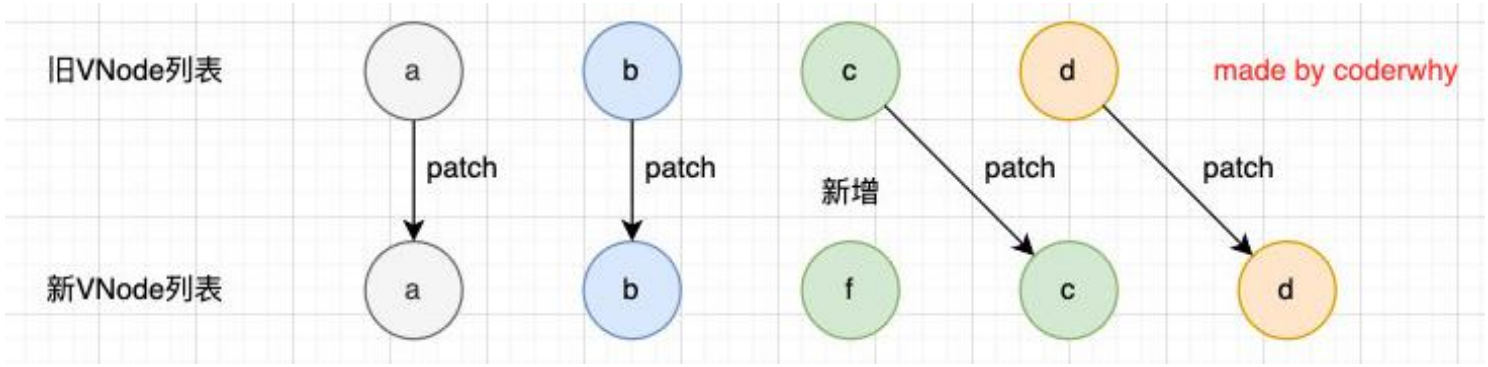


- 第二步的操作是从尾部开始进行遍历、比较：

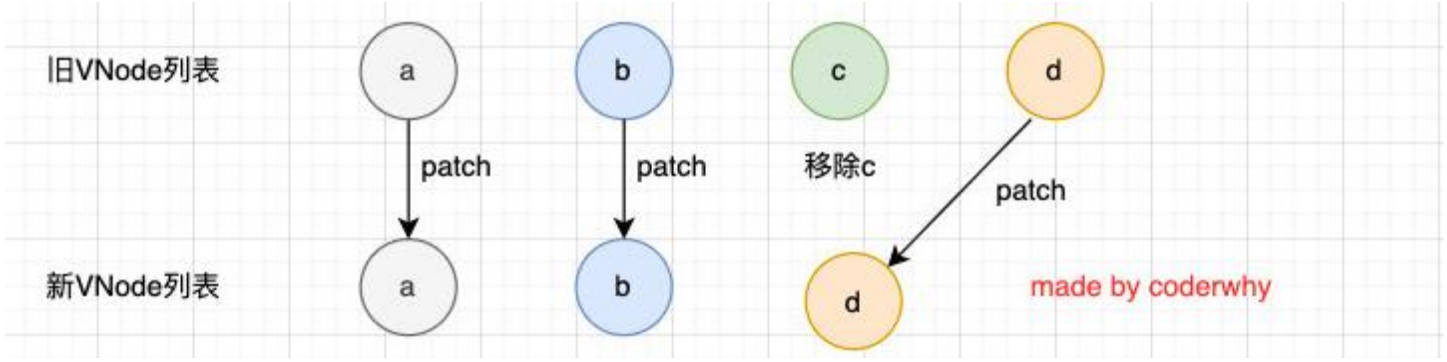


有key的diff算法如下（二）

- 第三步是如果旧节点遍历完毕，但是依然有新的节点，那么就新增节点：

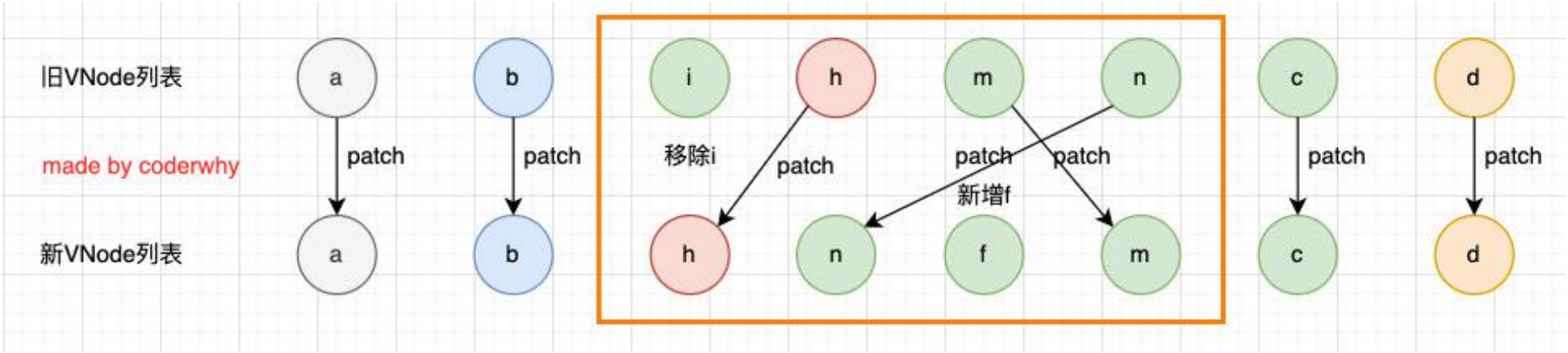


- 第四步是如果新的节点遍历完毕，但是依然有旧的节点，那么就移除旧节点：



有key的diff算法如下（三）

- **第五步是最特色的情况，中间还有很多未知的或者乱序的节点：**



- **所以我们可以发现，Vue在进行diff算法的时候，会尽量利用我们的key来进行优化操作：**
  - 在没有key的时候我们的效率是非常低效的；
  - 在进行插入或者重置顺序的时候，保持相同的key可以让diff算法更加的高效；