React.js 小书

<-- 返回首页

# 3. 前端组件化（二）：优化 DOM 操作

- 作者：胡子大哈
- 原文链接：http://huziketang.com/books/react/lesson3
- 转载请注明出处，保留原文链接和作者信息。

看看上一节我们的代码，仔细留意一下 `changeLikeText` 函数，这个函数包含了 DOM 操作，现在看起来比较简单，那是因为现在只有 `isLiked` 一个状态。由于数据状态改变会导致需要我们去更新页面的内容，所以假想一下，如果你的组件依赖了很多状态，那么你的组件基本全部都是 DOM 操作。

一个组件的显示形态由多个状态决定的情况非常常见。代码中混杂着对 DOM 的操作其实是一种不好的实践，手动管理数据和 DOM 之间的关系会导致代码可维护性变差、容易出错。所以我们的例子这里还有优化的空间：如何尽量减少这种手动 DOM 操作？

## 状态改变 -> 构建新的 DOM 元素更新页面

这里要提出的一种解决方案：**一旦状态发生改变，就重新调用 `render` 方法，构建一个新的 DOM 元素。**这样做的好处是什么呢？好处就是你可以在 `render` 方法里面使用最新的 `this.state` 来构造不同 HTML 结构的字符串，并且通过这个字符串构造不同的 DOM 元素。页面就更新了！听起来有点绕，看看代码怎么写，修改原来的代码为：

```
class LikeButton {
  constructor () {
    this.state = { isLiked: false }
  }

  setState (state) {
    this.state = state
    this.el = this.render()
  }

  changeLikeText () {
    this.setState({
      isLiked: !this.state.isLiked
    })
  }

  render () {
    this.el = createDOMFromString(`
```

```
        <button class='like-btn'>
          <span class='like-text'>${this.state.isLiked ? '取消' : '点赞'}</span>
          <span>👍</span>
        </button>
      `)
      this.el.addEventListener('click', this.changeLikeText.bind(this), false)
      return this.el
    }
  }
```

其实只是改了几个小地方：

1. `render` 函数里面的 HTML 字符串会根据 `this.state` 不同而不同（这里是用了 ES6 的模版字符串，做这种事情很方便）。
2. 新增一个 `setState` 函数，这个函数接受一个对象作为参数；它会设置实例的 `state`，然后重新调用一下 `render` 方法。
3. 当用户点击按钮的时候，`changeLikeText` 会构建新的 `state` 对象，这个新的 `state` ，传入 `setState` 函数当中。

这样的结果就是，用户每次点击，`changeLikeText` 都会调用改变组件状态然后调用 `setState` ；`setState` 会调用 `render` ，`render` 方法会根据 `state` 的不同重新构建不同的 DOM 元素。

也就是说，**你只要调用 `setState`，组件就会重新渲染。**我们顺利地消除了手动的 DOM 操作。

## 重新插入新的 DOM 元素

上面的改进不会有什么效果，因为你仔细看一下就会发现，其实重新渲染的 DOM 元素并没有插入到页面当中。所以在这个组件外面，你需要知道这个组件发生了改变，并且把新的 DOM 元素更新到页面当中。

重新修改一下 `setState` 方法：

```
...
    setState (state) {
      const oldEl = this.el
      this.state = state
      this.el = this.render()
      if (this.onStateChange) this.onStateChange(oldEl, this.el)
    }
...
```

使用这个组件的时候：

```
const likeButton = new LikeButton()
wrapper.appendChild(likeButton.render()) // 第一次插入 DOM 元素
likeButton.onStateChange = (oldEl, newEl) => {
  wrapper.insertBefore(newEl, oldEl) // 插入新的元素
  wrapper.removeChild(oldEl) // 删除旧的元素
}
```

这里每次 `setState` 都会调用 `onStateChange` 方法，而这个方法是实例化以后时候被设置的，所以你可以自定义 `onStateChange` 的行为。**这里做的事是，每当 `setState` 中构造完新的 DOM 元素以后，就会通过 `onStateChange` 告知外部插入新的 DOM 元素，然后删除旧的元素，页面就更新了。**这里已经做到了进一步的优化了：现在不需要再手动更新页面了。

非一般的暴力，因为每次 `setState` 都重新构造、新增、删除 DOM 元素，会导致浏览器进行大量的重排，严重影响性能。不过没有关系，这种暴力行为可以被一种叫 Virtual-DOM 的策略规避掉，但这不是本文所讨论的范围。

这个版本的点赞功能很不错，我可以继续往上面加功能，而且还不需要手动操作DOM。但是有一个不好的地方，如果我要重新另外做一个新组件，譬如说评论组件，那么里面的这些 `setState` 方法要重新写一遍，其实这些东西都可以抽出来，变成一个通用的模式。下一节我们把这个通用模式抽离到一个类当中。

> 因为第三方评论工具有问题，对本章节有任何疑问的朋友可以移步到 React.js 小书的论坛 发帖，我会回答大家的疑问。

下一节：4. 前端组件化（三）：抽象出公共组件类

上一节：2. 前端组件化（一）：从一个简单的例子讲起