07 生命周期：为什么 React v16 产生了新的生命周期函数

更新时间：2020-08-20 10:36:22

> 人不可有傲气，但不可无傲骨。——徐悲鸿

为什么 React v16 版的生命周期函数发生了变更

从v16.3 版本开始，React 引入了两个新的生命周期函数 `getDerivedStateFromProps` 和 `getSnapshotBeforeUpdate`。同时 React 不再建议使用`componentWillMount`，`componentWillReceiveProps`和`componentWillUpdate`三个生命周期函数。**那么，这三个生命周期函数存在什么问题吗？**

componentWillMount 函数存在的问题

在 React 应用中，许多开发者为了避免第一次渲染时页面因为没有获取到异步数据导致的白屏，而将数据请求部分的代码放在了 `componentWillMount` 中，希望可以避免白屏并提早异步请求的发送时间。但事实上在 `componentWillMount` 执行后，第一次渲染就已经开始了，所以如果在 `componentWillMount` 执行时还没有获取到异步数据的话，页面首次渲染时也仍然会处于没有异步数据的状态。换句话说，组件在首次渲染时总是会处于没有异步数据的状态，所以不论在哪里发送数据请求，都无法直接解决这一问题。而关于提早发送数据请求，官方也鼓励将数据请求部分的代码放在组件的 `constructor` 中，而不是 `componentWillMount`。

> 事实上，现在依然有很多很多应用的异步请求写在了 `componentWillMount` 函数中，由于目前的网络性能这么好，我们所担心的白屏带来的用户体验影响也就没那么严重。个人认为异步请求应该写在 `constructor` 中还是 `componentWillMount` 中可根据实际情况而定。

componentWillReceiveProps 函数存在的问题

如果组件自身的某个 state 跟其 props 密切相关（指 state 值可能受到 props 的影响 ）的话，一直都没有一种很优雅的处理方式去更新state，一般的做法是在 componentWillReceiveProps 函数中判断前后两个 props 是否相同，如果不同再将新的 props 更新到相应的state 上去，见代码示例 2.4.1。

```
componentWillReceiveProps(nextProps) {
  // 如果nextProps.a传递的是基本数据类型，可以直接进行相等判断
  if (nextProps.isLogin !== this.props.isLogin) {
    this.setState({
      isLogin: nextProps.isLogin,
    });
  }
  if (nextProps.isLogin) {
    this.handleClose();
  }
}
// 如果nextProps.a传递的是一个引用类型，一般是先将它们转换成字符串，然后进行相等判断
componentWillReceiveProps(nextProps) {
  if (JSON.stringify(nextProps.a) !== JSON.stringify(this.props.a)) {
    this.setState({
      ...
    });
  }
}
```

代码示例 2.4.1 componentWillReceiveProps 函数存在的问题

在 componentWillReceiveProps 函数将 props 映射为对应为 state 一方面会破坏 state 数据的单一数据源，导致组件状态变得不可预测，另一方面也会增加组件的重绘次数。

componentWillUpdate 函数存在的问题

与 componentWillReceiveProps 类似，许多开发者也会在 componentWillUpdate 中根据 props 的变化去触发一些回调。但不论是 componentWillReceiveProps 还是 componentWillUpdate，**都有可能在一次更新中被调用多次，也就是说写在这里的回调函数也有可能会被调用多次，这显然是不可取的**。

另一个常见的 componentWillUpdate 的用例是在组件更新前，读取当前某个 DOM 元素的状态，并在 componentDidUpdate 中进行相应的处理。但在 React 开启异步渲染模式后，render 阶段和 commit 阶段之间并不是无缝衔接的，也就是说在 render 阶段读取到的 DOM 元素状态并不总是和 commit 阶段相同，这就导致在 componentDidUpdate 中使用 componentWillUpdate 中读取到的 DOM 元素状态是不安全的，因为这时的值很有可能已经失效了。

React v16.3 版本新增的两个生命周期函数解决了什么问题呢？

getDerivedStateFromProps 函数解决的问题

React 生命周期函数的命名一直都非常语义化，getDerivedStateFromProps的意思就是从 props 中获取 state ，换句话说就是将传入的props 映射（赋值）到 state 中。getDerivedStateFromProps 函数的使用方式见代码示例 2.4.2。

```
// 使用getDerivedStateFromProps替换componentWillReceiveProps
static getDerivedStateFromProps(nextProps, prevState) {
  if (nextProps.isLogin !== prevState.isLogin) {
    // 注意这里的写法
    return {
      isLogin: nextProps.isLogin,
    };
  }
  return null;
}

componentDidUpdate(prevProps, prevState) {
  if (!prevState.isLogin && this.props.isLogin) {
    // 这里this.props已经是最新的props，prevState是上一版本的state
    this.handleClose();
  }
}
```

代码示例 2.4.2 getDerivedStateFromProps 函数使用方式

通常来讲，在 `componentWillReceiveProps` 中我们一般会做以下两件事。一是根据 props 来更新 state，二是触发一些回调，如动画或页面跳转等。在React v16.3 版本之前，这两件事都需要在 `componentWillReceiveProps` 中去做。而在新版本中，官方将更新 state 与触发回调分配到了 `getDerivedStateFromProps` 与 `componentDidUpdate` 中，使得组件整体的更新逻辑更为清晰。**在 `getDerivedStateFromProps` 中禁止了组件访问 `this.props`**，强制让开发者去比较 `nextProps` 与 `prevState` 中的值，以确保程序在调用 `getDerivedStateFromProps` 这个生命周期函数时是根据当前的 props 来更新组件的 state，而不是去做其他一些让组件自身状态变得更加不可预测的事情。

getSnapshotBeforeUpdate 函数解决的问题

`getSnapshotBeforeUpdate` 函数在（DOM）更新之前被调用（获取一个快照），在该函数中可以访问更新前 DOM 的属性，其返回值将作为第三个参数传递给 `componentDidUpdate` 函数，这就保证了在两个函数中可以访问同一个值，见代码示例 2.4.2。

```
// 针对上面第3个问题使用getSnapshotBeforeUpdate替换componentWillUpdate
class ScrollingList extends React.Component {

  getSnapshotBeforeUpdate(prevProps, prevState) {
    return (
      // 这里可以访问更新前的DOM元素属性
      return this.rootNode.scrollHeight
    );
  }

  componentDidUpdate(prevProps, prevState, snapshot) {
    // snapshot值是在getSnapshotBeforeUpdate函数中返回
    if (snapshot !== null) {
      const curScrollTop= this.rootNode.scrollTop;
   this.rootNode.scrollTop = curScrollTop + (this.rootNode.scrollHeight - snapshot);
    }
  }

  render() {
    return (
      <div>
        {/* ...contents... */}
      </div>
    );
  }
}
```

代码示例 2.4.3 getSnapshotBeforeUpdate 函数使用方式

小结

本文提到的组件的 componentWillMount， componentWillReceiveProps 和 componentWillUpdate 三个生命周期函数在使用过程中存在的一些「问题」是由众多 React 开发者在日常写程序的过程中总结出来，但是这些「问题」是否真的对我们自己的业务产生了影响需要我们根据实际情况进行衡量。React 是一个优秀的前端框架，目前开始建议使用 getSnapshotBeforeUpdate 函数和 getSnapshotBeforeUpdate 函数来完成原本需要在前面三个函数中完成的业务逻辑。但是新增的这两个生命周期函数是否也存在其他的问题呢？这需要我们开发者来一点一点的验证。

现在我们想一想，组件的生命周期函数是怎么调用的呢？

前面文章已经提到过通过组件的实例可以调用生命周期函数，下一节会详细介绍官方文档提到较少的内容 — React 组件实例。

}

06 生命周期：如何理解 React 组件生命周期？

08 组件实例：组件实例到底是什么？