**React 16生命周期函数与其在项目中的应用**

1. **案例描述**

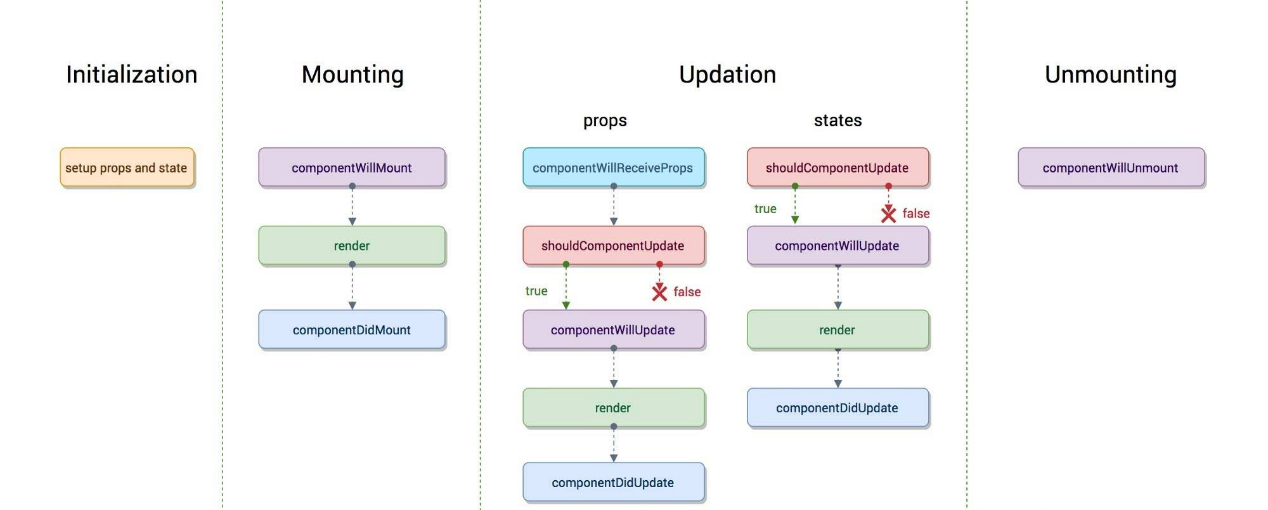
近期因为项目代码中多处使用的旧版React的生命周期函数：componentWillReceiveProps会在浏览器控制台中生成warning。于是重构了项目中所有使用此函数的部分，本文旨在通过这一案例的解决梳理React 16生命周期的函数的功能用法，以及如何改写重构在React 16中各种已废弃的生命周期函数。

1. **案例分析与解决**

React原有生命周期十分完整，基本可以捕捉到组件更新的每一个state/props/ref，但React打算在17版本推出新的Async Rendering，提出一种可被打断的生命周期，而可以被打断的阶段正是实际dom挂载之前的虚拟dom构建阶段，也就是要被去掉的三个生命周期。生命周期一旦被打断，下次恢复的时候又会再跑一次之前的生命周期，因此componentWillMount，componentWillReceiveProps， componentWillUpdate都不能保证只在挂载/获得props/状态变化的时候刷新一次了，所以这三个方法被标记为不安全。而React 16作为中间的过渡，没有强制禁止使用这三个方法，但鼓励使用新的生命周期函数代替，而通过新增的生命周期函数也可以实现所有废弃函数可以完成的功能。

1. **React生命周期函数特性与在项目代码重构中的应用**
   1. **新旧React生命周期函数对比**

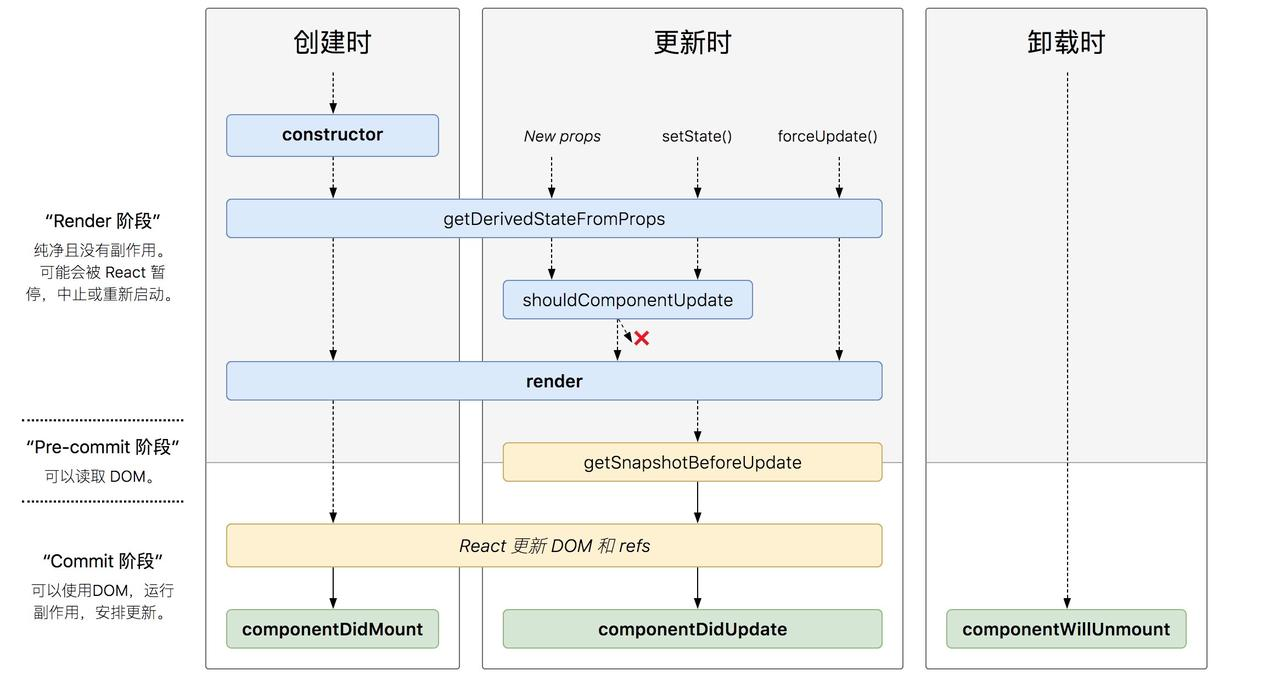
旧版React的生命周期函数，在组件的初始化，加载，更新，卸载阶段的流程如下图所示：



在React 16版本中有3个包含will的函数都已经被弃用，即componentWillMount, componentWillReceiveProps, componentWillUpdate这三个函数。一般通常认为 componentWillReceiveProps 只会在父组件传递来的 props 发生变化的时候触发，因此通常我们可能会在 componentWillReceiveProps 中进行下面两种操作。第一种操作是，接收从父组件传来的 props，来更新子组件自身的 state。 第二种操作是：进行一些有副作用的操作，比如发送请求。而componentWillReceiveProps 会在父组件每次 render 导致子组件 re-render 的时候执行，就算此时 props 没有发生改变也会导致该方法执行。一种常见的用法是判断当前props 和 nextProps 是否相等，再决定是否执行从 props 更新到 state，例如下面代码：



而不同于 componentWillReceiveProps 只在每次父组件 render 导致子组件 re-render 的时候触发， getDerivedStateFromProps 这个生命周期会在每次组件 setState 的时候也会触发，如下图所示，组件的Mounting 和Updating 的任何一个阶段都会触发这个生命周期。React新的生命周期如下图所示。



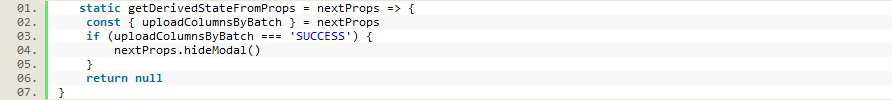
如上图所示，new props, setState, this.forceUpdate()都会触发getDerivedStateFromProps。这个方法接收父组件传递过来的 props 和组件之前的状态，返回一个对象来更新 state 或者返回 null表示不需要更新 state。此生命周期钩子的作用：将父组件传递过来的 props 映射 到子组件的 state 上面，这样组件内部就不用再通过 this.props 获取属性值了，统一通过 this.state获取。映射相当于拷贝了一份父组件传过来的 props作为子组件自己的状态，配合 componentDidUpdate，可以覆盖 componentWillReceiveProps 的所有用法

* 1. **新版React生命周期函数在项目具体实践中的重构**

首先一个比较简单的例子，改写如下图所示，这是一个批量上传的弹窗的文件代码，hideModal()这一方法是从父组件继承，调用后会取消渲染此弹窗。所以无需担心在此弹窗组件一个生命周期内调用多次的问题。希望实现的功能是在收到异步请求成功的结果后，关闭弹窗，也就是取消渲染此组件。

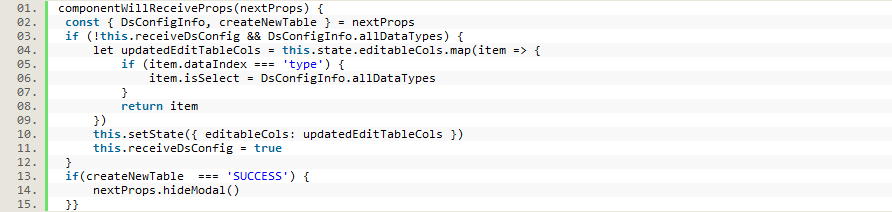
下面第一张图，是旧版React生命周期函数componentWillReceiveProps的实现，第二张是用React 16中的getDerivedStateFromProps函数的实现，从代码可以看出getDerivedStateFromProps是静态方法，函数名前要加static关键字，另外一点是getDerivedStateFromProps函数必须要返回改变的state。 如果没有改变任何状态的话，也要return null。

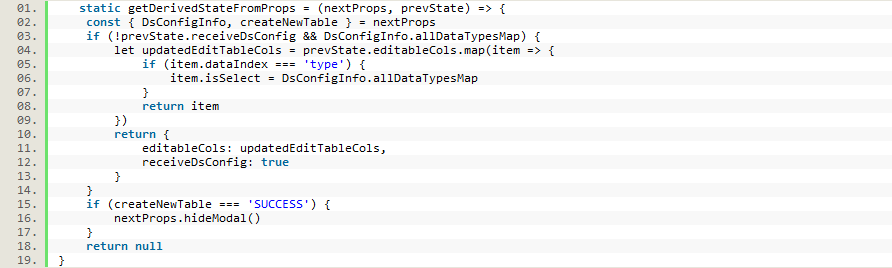




另一个复杂一点的例子是涉及到state变化以及实例属性的变化的。 原来的方法如第一张图，利用getDerivedStateFromProps方法改写为第二张图中代码。一个问题在于，getDerivedStateFromProps 是一个 static 方法，意味着拿不到实例的this，原来getDerivedStateFromProps方法中的的this.receiveDsConfig，需要改写成prevState.receiveDsConfig才可以将新状态保存。

第二张图可见getDerivedStateFromProps函数有两个参数，nextProps以及prevState。为了避免反复调用，判断条件为if (!prevState.receiveDsConfig && DsConfigInfo.allDataTypesMap) 而在调用之后返回的{editableCols: updatedEditTableCols, receiveDsConfig: true}即是要变更的状态state。这样第二次state.receiveDsConfig变为true即防止了反复调用。需要注意的是，使用 getDerivedStateFromProps 派生状态时，不需要把组件自身的状态也设置进去，只需设置变化的状态。





另外一个常见的用法是利用getDerivedStateFromProps + componentDidUpdate 加载数据， 如下图代码所示，希望实现的功能是当收到父组件传的新的id时，利用新的id请求异步数据。实现的流程为，首先当收到与state中保存的prevId不同的id时, 先在getDerivedStateFromProps函数中清空原数据，并将state中的旧id替换为新id。然后通过componentDidUpdate生命周期函数判断当this.state.externalData为null的时候，请求加载异步数据。



* 1. **React新生命周期的特性**

componentWillMount，componentWillReceiveProps，componentWillUpdate 这三个生命周期因为经常会被误解和滥用，所以被称为不安全（不是指安全性，而是表示使用这些生命周期的代码，有可能在未来的 React 版本中存在缺陷，可能会影响未来的异步渲染） 的生命周期。

由于 React 未来的版本中推出了异步渲染，在 dom 被挂载之前的阶段都可以被打断重来，导致 componentWillMount、componentWillUpdate、componentWillReceiveProps 在一次更新中可能会被触发多次，因此那些只希望触发一次的副作用应该放在 componentDidMount 中。例如在实际项目中存在组件挂载时请求异步数据，之前有一些写在componentWillMount中的功能，都可以改写到componentDidMount函数中。

另外一个值得一体的新的生命周期函数: getSnapshotBeforeUpdate(prevProps, prevState)替换了旧版的 componentWillUpdate，触发时机点是在数据状态已更新，最新VNode已生成，但 真实元素还未被更新；可以用来在更新之前从真实元素或状态中获取计算一些信息，便于在更新后使用。

这个方法在render之后，componentDidUpdate之前调用，有两个参数prevProps和prevState，表示之前的属性和之前的state，这个函数有一个返回值，会作为第三个参数传给componentDidUpdate，如果不想要返回值，要返回null，否则控制台会有警告，这个方法还一定要和componentDidUpdate一起使用，否则控制台也会有警告。

1. **经验总结，预防措施核对规范的建议等**

本文从工作中对React生命周期函数的更新重构出发，分析了React 16生命周期函数的主要变化以及实际用法。以期更好应对前端项目因为React版本升级所带来对应的调整和更新。

1. **从本文可导出的检查项（checklist）**

无