# 30 React基本功（一）——图解新旧生命周期

更新时间：2020-05-26 13:57:48

> 宝剑锋从磨砺出，梅花香自苦寒来。——佚名

本节开始，我们进入前端框架的世界。

相信每一个有面试打算的工程师，对React、Vue这样的当红轮子都不会是一无所知的状态。对于大家而言，单纯说轮子的使用，一定不是啥难事儿。在框架这个知识板块，我们强调的是“原理”和“表达”。

原理不必多讲，肯定是我们备考的重点——但凡稍微有点面试经验的同学，都会清楚框架源码、底层机制这些东西在面试官眼里的重要性。我这里想要强调的是“表达”——很多同学认为，有了日常工作中的频繁使用，自己对框架基础已经不能更熟悉，便忽视了对基础知识的准备。结果面试现场，受试者面对框架基础类型的考题，总是噤若寒蝉——这时候才发现，原来会写代码，和能说囫囵是两回事；原来就算工作中写了那么多React代码，可脑子里也只记住了最最高频的那些操作——原来，会用和理解是两回事；原来，你的框架基础并没有想象中扎实。

React开篇，我们不讲虚拟DOM，不讲Fiber架构，我先带大家把React常考的基础知识整个给串一遍：

## React 新旧生命周期

大家知道，随着 React16 的发布和推广，新的 React 生命周期越来越为广大开发者所接受。不过在一些团队，可能因为各种各样的原因，他们并没有进行 React 的版本迁移，因此面试官仍然对老生命周期更感兴趣。

学新还是学旧，这个咱们不纠结——小孩子才做选择，面试的人当然是全都要啊！

## 旧生命周期

先从旧的生命周期说起，旧的 React 包括以下几个主要的生命周期函数：

- **componentWillMount**
- **componentDidMount**
- **componentWillReceiveProps**
- **shouldComponentUpdate**
- **componentWillUpdate**
- **componentDidUpdate**
- **componentWillUnmount**

> 注意： React的生命周期流程从广义上分为三个阶段：**挂载、更新、卸载**。

以上函数各自有着自己的执行时机。关于执行时机，我们需要考虑四种情况：

1. React 初始化应用时 （挂载阶段）
2. props 发生数据更新 （更新阶段）
3. state 发生数据更新 （更新阶段）
4. 卸载应用时 (卸载阶段，比较简单，仅涉及 componentWillUnmount）

### 卸载

我们先来说说最简单的卸载阶段：

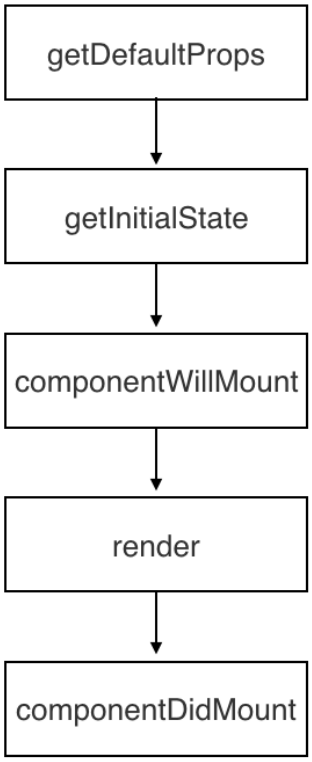
> `componentWillUnmount()` 会在组件卸载及销毁之前直接调用。在此方法中执行必要的清理操作，例如，清除 timer，取消网络请求或清除在 `componentDidMount()` 中创建的订阅等。

简单说，这个生命周期钩子，就是一个“扫地僧”啊！

相比卸载阶段而言，前三种情形会比较有嚼头，也是面试的重点。它们分别对应了不同的生命周期流程（下面画图说明）：

### 初始化

首先，是初始化应用，也即第一次 render 页面时：

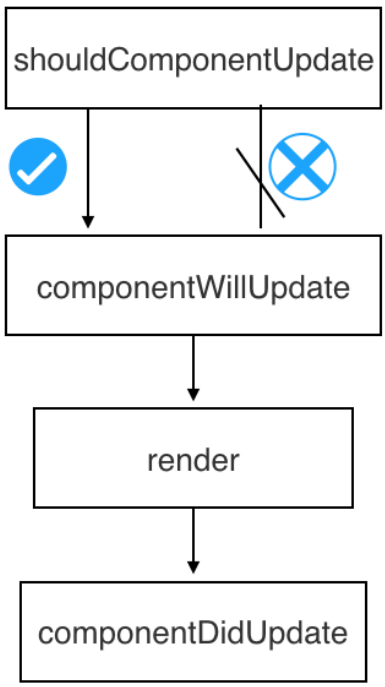



大家看到，这里除了我们提到过的生命周期函数以外，还涉及一些其它的 React 函数。我们一一来看：

1. **getDefaultProps**：这个函数会在组件创建之前被调用一次（有且仅有一次），它被用来初始化组件的 Props；
2. **getInitialState**：用于初始化组件的 state 值；
3. **componentWillMount**：在组件创建后、render 之前，会走到 componentWillMount 阶段。这个阶段我个人一直没用过、非常鸡肋。后来React 官方已经不推荐大家在 componentWillMount 里做任何事情、到现在 **React16 直接废弃了这个生命周期**，足见其鸡肋程度了；
4. **render**：这是所有生命周期中唯一一个你必须要实现的方法。一般来说需要返回一个 jsx 元素，这时 React 会根据 props 和 state 来把组件渲染到界面上；不过有时，你可能不想渲染任何东西，这种情况下让它返回 null 或者 false 即可；
5. **componentDidMount**：会在组件挂载后（插入 DOM 树中后）立即调用，标志着组件挂载完成。一些操作如果依赖获取到 DOM 节点信息，我们就会放在这个阶段来做。此外，这还是 React 官方推荐的发起 ajax 请求的时机。该方法和 componentWillMount 一样，有且仅有一次调用。

初始化完成后，我们需要考虑数据更新的两种情况（下面两张图分别对应 state 和 props的更新）：

## state 更新流程：



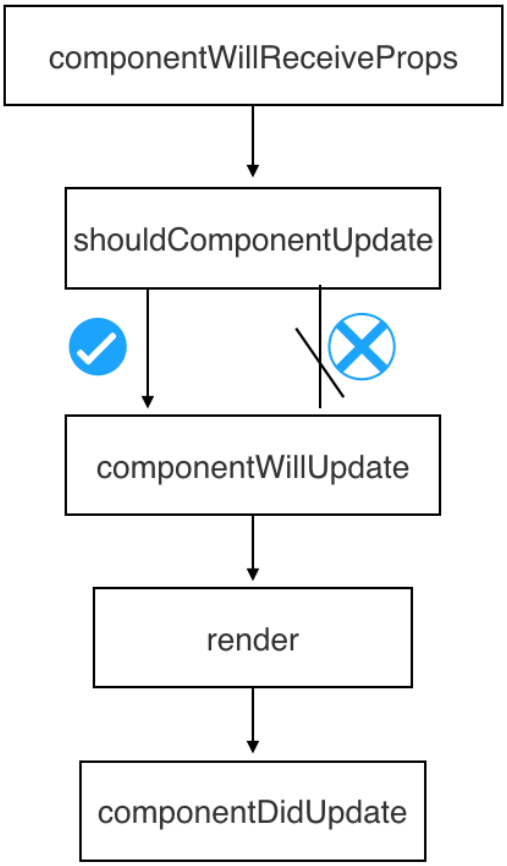

我们一起来看看这个过程当中涉及的函数：

1. shouldComponentUpdate: 当组件的 state 或 props 发生改变时，都会首先触发这个生命周期函数。它会接收两个参数：nextProps, nextState——它们分别代表传入的新 props 和新的 state 值。拿到这两个值之后，我们就可以通过一些对比逻辑来决定是否有 re-render（重渲染）的必要了。如果该函数的返回值为 false，则生命周期终止，反之继续；

> 注意：此方法仅作为**性能优化的方式**而存在。不要企图依靠此方法来“阻止”渲染，因为这可能会产生 bug。你应该**考虑使用内置的 PureComponent 组件**，而不是手动编写 `shouldComponentUpdate()`

2. componentWillUpdate：当组件的 state 或 props 发生改变时，会在渲染之前调用 componentWillUpdate。componentWillUpdate 是 **React16 废弃的三个生命周期之一**。过去，我们可能希望能在这个阶段去收集一些必要的信息（比如更新前的 DOM 信息等等），现在我们完全可以在 React16 的 getSnapshotBeforeUpdate 中去做这些事；
3. componentDidUpdate：componentDidUpdate() 会在UI更新后会被立即调用。它接收 prevProps（上一次的 props 值）作为入参，也就是说在此处我们仍然可以进行 props 值对比（再次说明 componentWillUpdate 确实鸡肋哈）。

## props 更新流程：

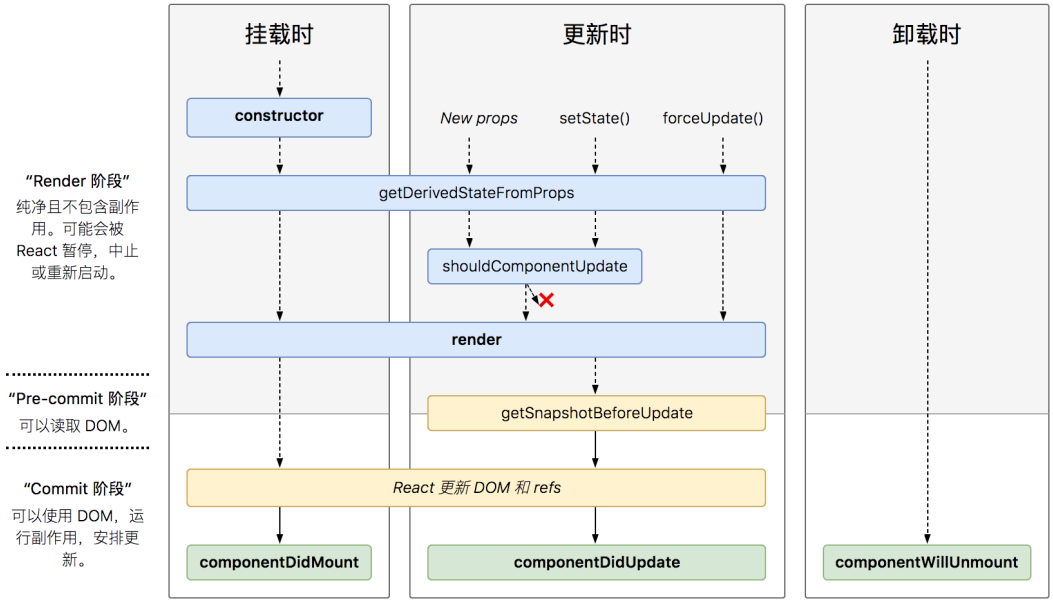



相对于 state 更新，props 更新后唯一的区别是增加了对 componentWillReceiveProps 的调用。关于 componentWillReceiveProps，你需要知道这些事情：

- componentWillReceiveProps：它在Component接受到新的 props 时被触发。componentWillReceiveProps 会接收一个名为 nextProps 的参数（对应新的 props 值）。**该生命周期是 React16 废弃掉的三个生命周期之一**。在它被废弃前，一些同学可能习惯于用它来比较 this.props 和 nextProps 来重新setState。在 React16 中，我们用一个类似的新生命周期 getDerivedStateFromProps 来代替它。

## 新生命周期

关于 React16 开始应用的新生命周期，React 官方（http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/）提供的这张图表很能说明问题：




图中我们可以看出，React16 自上而下地对生命周期做了另一种维度的解读：

- **Render 阶段**：用于计算一些必要的状态信息。这个阶段可能会被 React 暂停，这一点和 React16 引入的 Fiber 架构（我们后面会重点讲解）是有关的；
- **Pre-commit阶段**：所谓"commit"，这里指的是"更新真正的 DOM 节点"这个动作。所谓 Pre-commit，就是说我在这个阶段其实还并没有去更新真实的 DOM，不过 DOM 信息已经是可以读取的了；
- **Commit 阶段**：在这一步，React 会完成真实 DOM 的更新工作。Commit 阶段，我们可以拿到真实 DOM（包括 refs）。

与此同时，新的生命周期在流程方面，仍然遵循"挂载"、"更新"、"卸载"这三个广义的划分方式。它们分别对应到：

挂载过程：

- **constructor**
- **getDerivedStateFromProps**
- **render**
- **componentDidMount**

更新过程：

- **getDerivedStateFromProps**
- **shouldComponentUpdate**
- **render**
- **getSnapshotBeforeUpdate**
- **componentDidUpdate**

卸载过程：

- **componentWillUnmount**

以上这些生命周期方法，红字标注的是咱们到现在为止还没介绍过的，也是 React16 新增的两个方法、需要大家重点关注：

1. **getDerivedStateFromProps**

在 render 方法之前调用，它应返回一个对象来更新 state，如果返回 null 则不更新任何内容。它接收 props 和 state 两个入参，从调用时机和实际场景上，它都微妙地对标了**componentWillReceiveProps** 这个已经被废弃的旧方法。

注意！React官方并不推荐开发者使用该方法：

> 此方法适用于罕见的用例，即 state 的值在任何时候都取决于 props。否则，派生状态会导致代码冗余，并使组件难以维护。

更何况，大部分的派生状态咱们都有更好的替代方案可以解决（当然这不是咱们复习的重点，感兴趣的同学戳更好的派生状态方式了解派生状态新姿势~）。

2. **getSnapshotBeforeUpdate**

这个方法会接受 prevProps、prevState 两个入参

```
getSnapshotBeforeUpdate(prevProps, prevState)
```

**这个生命周期函数必须有返回值**——它的返回值会作为第三个参数传递给 `componentDidUpdate`。

注意，因为走到这一步时，React 已经更新上了所有状态，所以新状态可以通过 this.props、this.state 获取。所以结合两个入参，我们可以拿到所有的新旧状态。不过即便这个方法能提供的信息如此丰富，它也并不常用：

> 此用法并不常见，但它可能出现在 UI 处理中，如需要以特殊方式处理滚动位置的聊天线程等。

因为用得少，所以面试官对于 getSnapshotBeforeUpdate 的询问普遍也是蜻蜓点水。不过这里以防万一，我还是给大家提一嘴：

结合个人经验来看，因为 getSnapshotBeforeUpdate 触发时，真实的 DOM 节点还没有更新；此外，它又可以和 componentDidMount（DOM 节点更新后触发的钩子） 通信。大家知道，一些场景下，我们是需要对更新前后的 DOM 节点信息作一些对比或是处理的。比如说我想知道更新前后，某一个 div 的位置移动了多少，以此来决定是否来把它矫正回原位、或者是直接帮它移动一个更合适的距离呢？这种情况下，用 getSnapshotBeforeUpdate 就再合适不过啦~

}