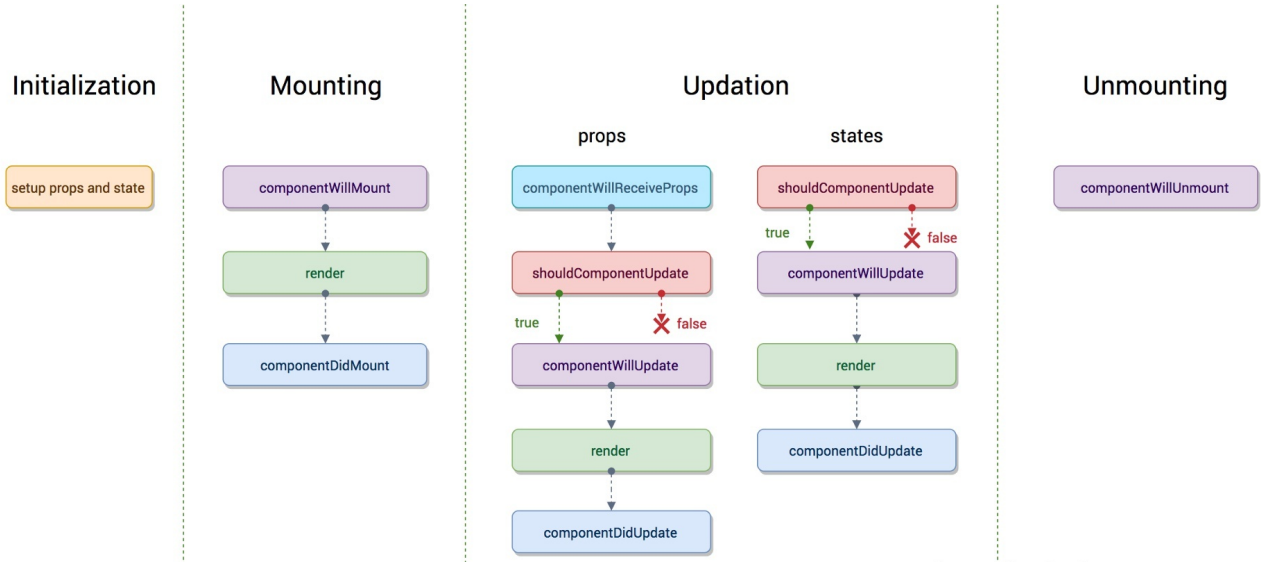
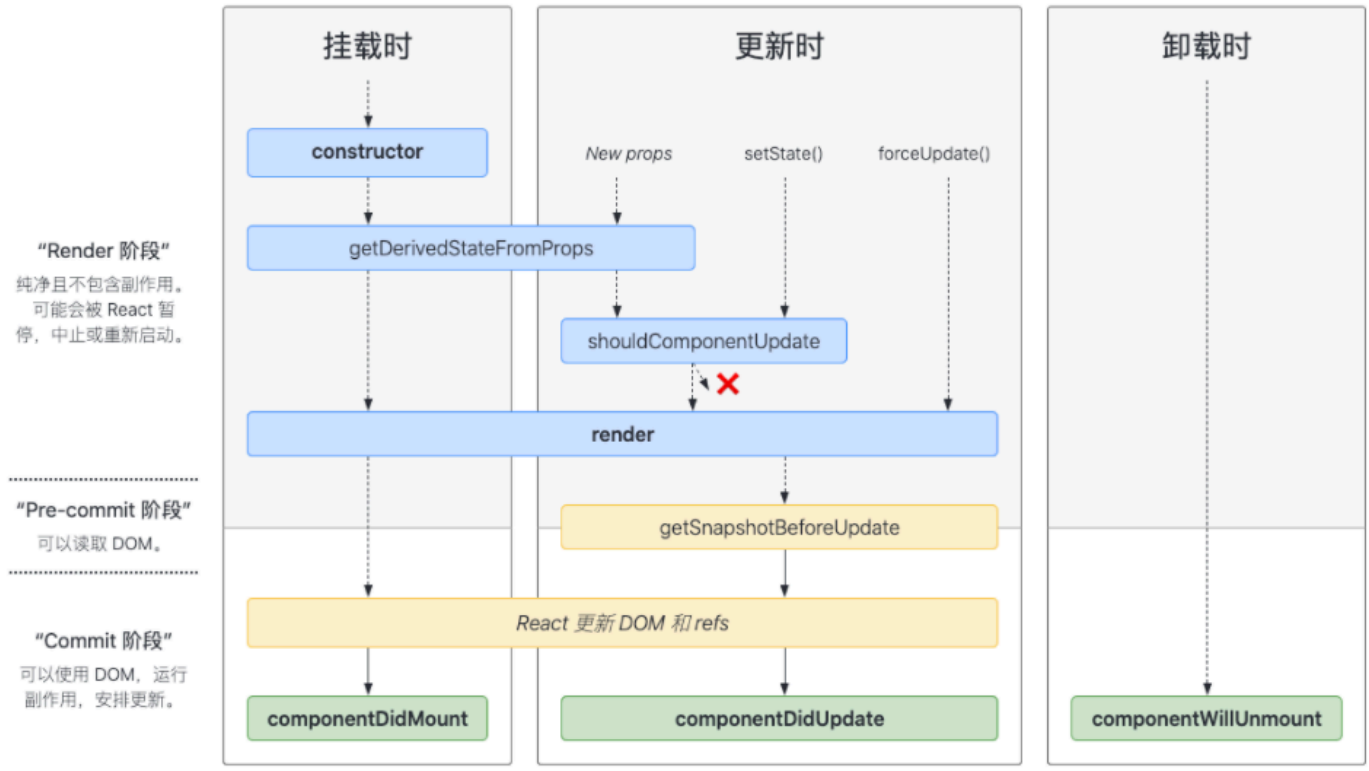
https://react.docschina.org

组件的生命周期

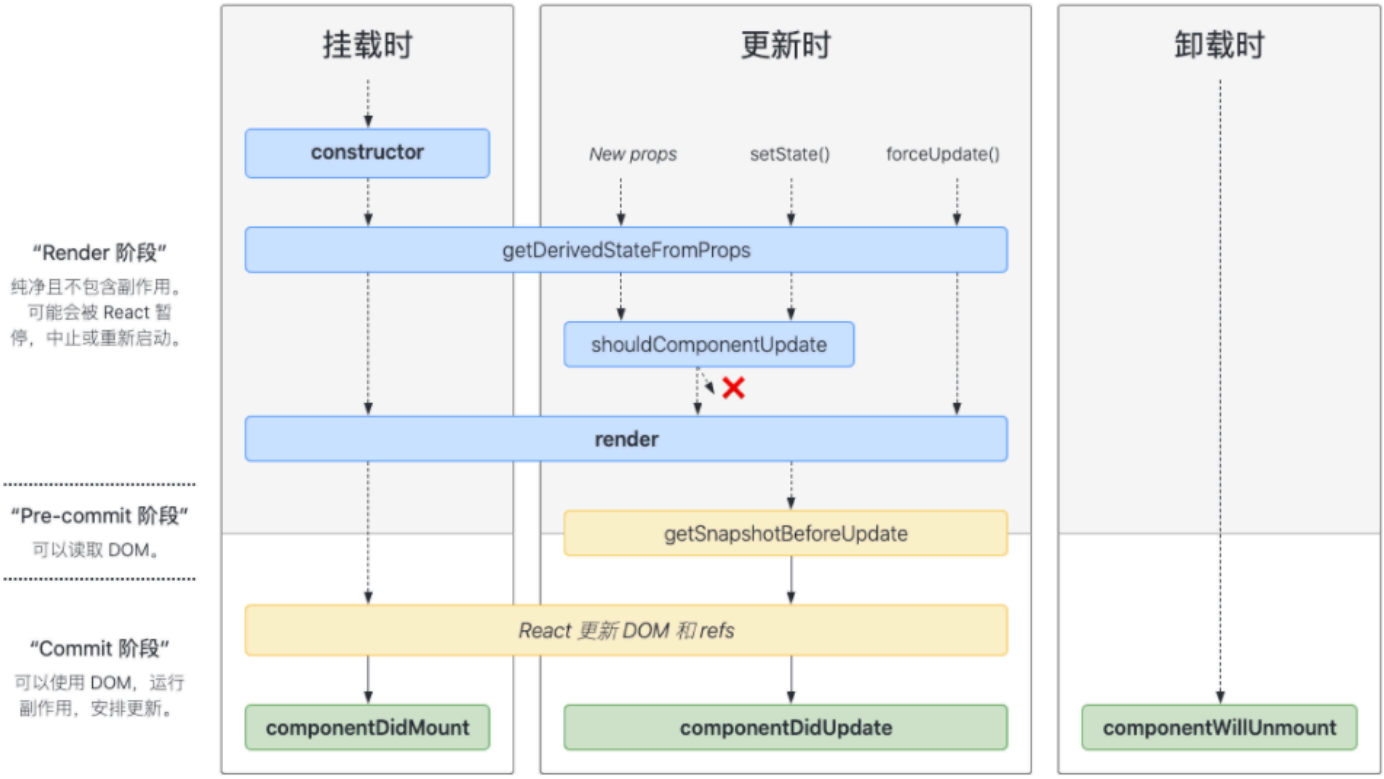
react 16.3 之前



React 16.3



React 16.4之后



生命周期方法介绍

render()

class组件中唯一必须要实现的方法，可以返回

- React 元素
- 数组或者fragments
- Portals
- 字符串或数值类型
- 布尔值或者null

constructor()

如果不初始化 state 或不进行方法绑定，则不需要为 React 组件实现构造函数。

```
constructor(props) {
  super(props);
  // 不要在这里调用 this.setState()
  this.state = { counter: 0 };
  this.handleClick = this.handleClick.bind(this);
}
```

避免将 props 的值复制给 state！这是一个常见的错误。只有在你刻意忽略 prop 更新的情况下使用，此时，应将 prop 重命名为 `initialProp` 或 `defaultProp`

设计组件时，重要的是确定组件是受控组件还是非受控组件，不要直接复制（mirror） props 的值到 state 中。

https://react.docschina.org/blog/2018/06/07/you-probably-dont-need-derived-state.html

componentDidMount()

`componentDidMount()` 会在组件挂载后（插入 DOM 树中）立即调用

componentDidUpdate(prevProps, prevState, snapshot)

`componentDidUpdate()` 会在更新后会被立即调用。首次渲染不会执行此方法。

可以再`componentDidUpdate()` 中**直接调用** `setState()`，但请注意**它必须被包裹在一个条件语句里**，正如上述的例子那样进行处理，否则会导致死循环。

componentWillUnmount()

`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。

shouldComponentUpdate(nextProps, nextState)

当 props 或 state 发生变化时，`shouldComponentUpdate()` 会在渲染执行之前被调用。返回值默认为 true。首次渲染或使用 `forceUpdate()` 时不会调用该方法。你应该**考虑使用内置的** `PureComponent` **组件**，而不是手动编写。

static getDerivedStateFromProps(nextProps, nextState)

`getDerivedStateFromProps` 会在调用 render 方法之前调用，并且在初始挂载及后续更新时都会被调用。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。

派生状态会导致代码冗余，并使组件难以维

- 如果你需要**执行副作用**（例如，数据提取或动画）以响应 props 中的更改，请改用 `componentDidUpdate`。
- 如果只想在 **prop 更改时重新计算某些数据**，请使用 memoization helper 代替。
- 如果你想**在 prop 更改时“重置”某些 state**，请考虑使组件完全受控或使用 `key` 使组件完全不受控 代替。

getSnapshotBeforeUpdate(prevProps, prevState)

`getSnapshotBeforeUpdate()` 在最近一次渲染输出（提交到 DOM 节点）之前调用。它使得组件能在发生更改之前从 DOM 中捕获一些信息（例如，滚动位置）。此生命周期的任何返回值将作为参数传递给 `componentDidUpdate()`。

```
class ScrollingList extends React.Component {
  constructor(props) {
    super(props);
    this.listRef = React.createRef();
  }

  getSnapshotBeforeUpdate(prevProps, prevState) {
    // 我们是否在 list 中添加新的 items ？
    // 捕获滚动••位置以便我们稍后调整滚动位置。
    if (prevProps.list.length < this.props.list.length) {
      const list = this.listRef.current;
      return list.scrollHeight - list.scrollTop;
    }
    return null;
  }

  componentDidUpdate(prevProps, prevState, snapshot) {
    // 如果我们 snapshot 有值，说明我们刚刚添加了新的 items，
    // 调整滚动位置使得这些新 items 不会将旧的 items 推出视图。
    // （这里的 snapshot 是 getSnapshotBeforeUpdate 的返回值）
    if (snapshot !== null) {
      const list = this.listRef.current;
      list.scrollTop = list.scrollHeight - snapshot;
    }
  }

  render() {
    return (
      <div ref={this.listRef}>{/* ...contents... */}</div>
    );
  }
}
```

static getDerivedStateFromError(error)

此生命周期会在后代组件抛出错误后被调用。 它将抛出的错误作为参数，并返回一个值以更新 state

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // 更新 state 使下一次渲染可以显降级 UI
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      // 你可以渲染任何自定义的降级  UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

注意：`getDerivedStateFromError()` 会在`渲染`阶段调用，因此不允许出现副作用。 如遇此类情况，请改用 `componentDidCatch()`。

componentDidCatch(error, info)

此生命周期在后代组件抛出错误后被调用。 它接收两个参数：

1. `error` —— 抛出的错误。
2. `info` —— 带有 `componentStack` key 的对象，其中包含有关组件引发错误的栈信息。

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // 更新 state 使下一次渲染可以显示降级 UI
    return { hasError: true };
  }

  componentDidCatch(error, info) {
    // "组件堆栈" 例子：
    //   in ComponentThatThrows (created by App)
```

```
    //   in ErrorBoundary (created by App)
    //   in div (created by App)
    //   in App
    logComponentStackToMyService(info.componentStack);
  }

  render() {
    if (this.state.hasError) {
      // 你可以渲染任何自定义的降级 UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

注意：如果发生错误，你可以通过调用 `setState` 使用 `componentDidCatch()` 渲染降级 UI，但在未来的版本中将不推荐这样做。 可以使用静态 `getDerivedStateFromError()` 来处理降级渲染。

UNSAFE_componentWillMount()

UNSAFE_componentWillReceiveProps(nextProps)

只要父组件重新渲染就会执行，可以在这里比较组件属性，并设置状态

UNSAFE_componentWillUpdate(nextProps, nextState)

注意，你不能此方法中调用 `this.setState()`；在 `UNSAFE_componentWillUpdate()` 返回之前，你也不应该执行任何其他操作（例如，dispatch Redux 的 action）触发对 React 组件的更新。

通常，此方法可以替换为 `componentDidUpdate()`。如果你在此方法中读取 DOM 信息（例如，为了保存滚动位置），则可以将此逻辑移至 `getSnapshotBeforeUpdate()` 中。

setState()

```
// setState(updater, [callback])
this.setState((state, props) => {
  return {counter: state.counter + props.step};
});
```

forceUpdate(callback)

默认情况下，当组件的 state 或 props 发生变化时，组件将重新渲染。如果 `render()` 方法依赖于其他数据，则可以调用 `forceUpdate()` 强制让组件重新渲染。

defaultProps

```
class CustomButton extends React.Component {
  // ...
}

CustomButton.defaultProps = {
  color: 'blue'
};
```

displayName

`displayName` 字符串多用于调试消息。通常，你不需要设置它，因为它可以根据函数组件或 class 组件的名称推断出来。如果调试时需要显示不同的名称或创建高阶组件.

```
function withSubscription(WrappedComponent) {
  class WithSubscription extends React.Component {/* ... */}
  WithSubscription.displayName =
`WithSubscription(${getDisplayName(WrappedComponent)})`;
  return WithSubscription;
}

function getDisplayName(WrappedComponent) {
  return WrappedComponent.displayName || WrappedComponent.name || 'Component';
}
```

HOOKS

Hook 是 React 16.8 的新增特性。它可以让你在不编写 class 的情况下使用 state 以及其他的 React 特性。

实现定时器