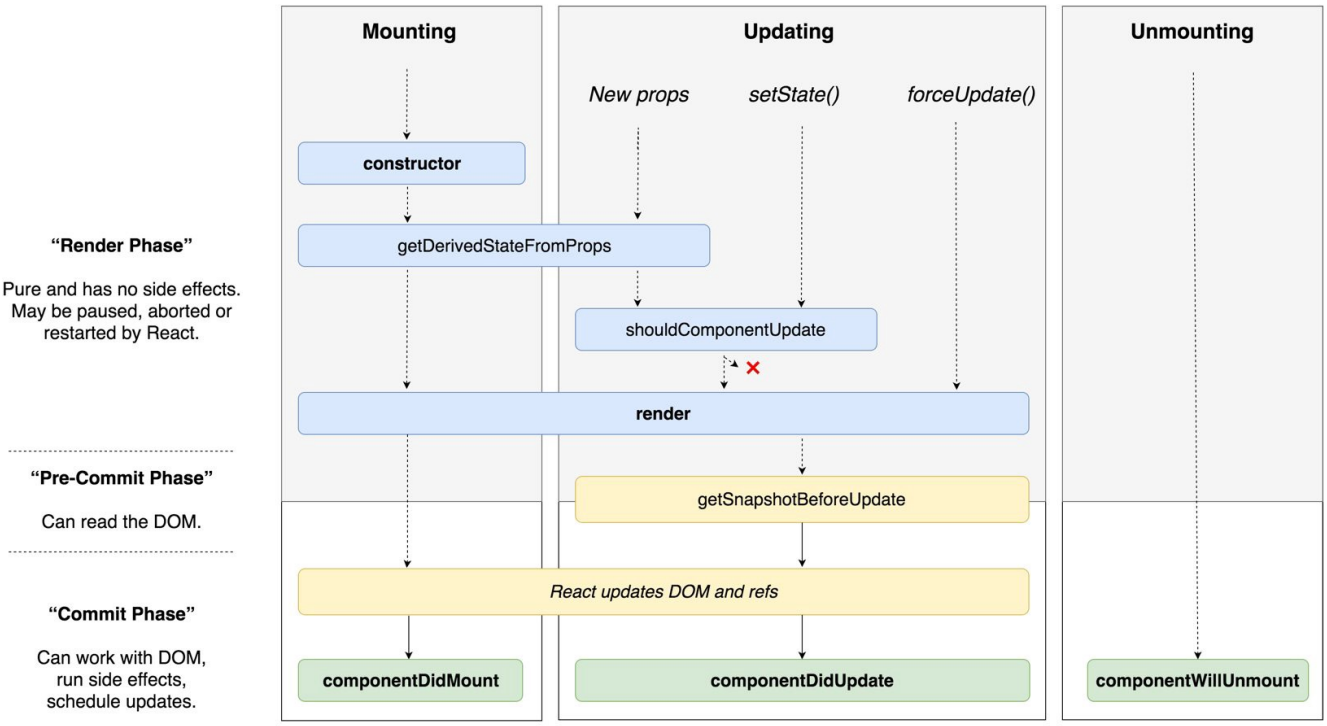
Жизненный цикл компонента

Существует несколько стадий жизненного цикла компонента, каждый из которых вызывает для компонента методы наследуемые от `React.Component`. Мы можем переопределить их поведение добавив необходимый функционал, в рамках установленных правил.

- Методы с префиксом `will` вызываются сразу перед тем, как что-то случится.
- Методы с префиксов `did` вызываются сразу после того, как что-то случилось.



Mounting

Следующие методы вызываются когда React создает экземпляр компонента и добавляет его в DOM.

constructor()

Взывается в момент создания экземпляра компонента, до того как компонент будет помещен в DOM.

- Инициализирует начальное состояние компонента
- Привязывает контекст в методах
- В большинстве случаев явное указание конструтора излишне

static getDerivedStateFromProps(nextProps, prevState)

- Вызывается перед `render()`, при `mount` и перед всеми последующими вызовами `render`, тоесть после обновления `state` или `props`.
- Можно использовать для того чтобы установить `state` в зависимости от `props` при каждом

их изменении.
- Должен вернуть объект, которым будет обновлено состояние, или `null` если ничего обновлять не нужно.

Статья о getDerivedStateFromProps

render()

- Позволяет декларативно описать интерфейс
- Возвращает результат JSX-выражений

componentDidMount()

- Вызывается когда компонент был отрендерен в DOM
- Делаем AJAX-запросы, вешаем кастомные слушатели событий и делаем операции с DOM (инициализация сторонних библиотек)
- Вызов `setState()` в этом методе, или после, вызове ре-рендер

Updating

Обновление может быть вызвано измением `state` самого компонента или передаваемых ему `props`. При обновлении необходимо перерендерить компонент, что ведет к вызову следующих методов.

shouldComponentUpdate(nextProps, nextState)

- Не вызывается при инициализации компонента
- Вызывается перед ре-рендером уже существующего компонента
- Необходим исключительно для оптимизации процесса рендеринга
- По умолчанию `render` происходит каждый раз при новых `props` или `state`
- Позволяет сравнить текущие и предыдущие `state` и `props`, вернув `true` или `false` указывая React есть ли необходимость обновлять компонент
- Если вернет `false` то не произойдет `render()` и `componentDidUpdate`
- React не на 100% "слушается" возвращаемого значения и может произвести ре-рендер компонента даже если будет возвращено `false`
- Нельзя вызывать `setState()`
- Использовать необходимо очень окуратно, только после замеров производительности, в противном случае может привести к обратному эффекту
- Возможно стоит заменить на `React.PureComponent`, который будет делать поверхностное сравнение `props`. Но только после замеров производительности

Репозиторий why-did-you-update

getSnapshotBeforeUpdate(prevProps, prevState)

- Вызывается перед тем как все изменения готовы к отображению в DOM
- Можно использовать для получения DOM-значений перед обновлением, к примеру

текущую позицию скрола
- Все что вернет этот метод будет передано как третий параметр в `componentDidUpdate()`

componentDidUpdate(prevProps, prevState, snapshot)

- Вызывается сразу после обновления компонента
- Не вызывается при первоначальном рендере компонента
- Можно вызывать `setState()`
- Можно делать AJAX-запросы, сравнивая `this.props` и `prevProps`, если они не изменились то и запрос делать нет смысла
- Можно передавать сторонним библиотекам новые данные
- Если в компоненте есть `getSnapshotBeforeUpdate()`, то значение возвращаемое им будет передано третим аргументом `snapshot`, в противном случае его значением будет `undefined`

Unmounting

В какой-то момент компоненты будут удалены из DOM. При этом вызывается следующий метод.

componentWillUnmount()

- Вызывается перед удалением компонента из DOM
- Отлично подходим для уборки за собой: слушатели, таймеры, AJAX-запросы. В противном случае будут утечки памяти
- Вызывать `setState()` нет смысла, компонент никогда не перерендерится

Отлов ошибок

В React 16 появился новый метод, срабатывающий при ошибке в дочернем компоненте.

```
componentDidCatch(error, info) {}
```

React очень любит класть все приложение при любой ошибке. Этот метод позволяет родительским компонентам отлавливать ошибки в детях, отображая запасной интерфейс, в результате, при ошибке, интерфейс не падает.

- Используется для контроля ошибок
- Ловит ошибки только в детях, но не в самом родителе
- `error` - результат `toString()` объекта ошибки
- `info` - объект описывающий stack trace

```
class ErrorBoundary extends React.Component {
  state = { hasError: false };
```

```
  componentDidCatch(error, info) {
    // Если метод был вызван значит есть ошибка!
    // Устанавливаем состояние
    this.setState({ hasError: true });
    // Также можно отправить отчет об ошибке вашему аналитическому сервису
    // logErrorToMyService(error, info);
  }

  render() {
    // Если есть ошибка...
    if (this.state.hasError) {
      // Рендерим fallback UI
      return <h1>Something went wrong, please try again later :(</h1>;
    }

    // Если все ок, рендерим детей
    return this.props.children;
  }
}
```

Error Handling in React 16

Дополнительные материалы

- The Component Lifecycle
- Понимание жизненного цикла React-компонента (перевод)
- Deep dive into the React Lifecycle Methods

Licence