hillel IT school

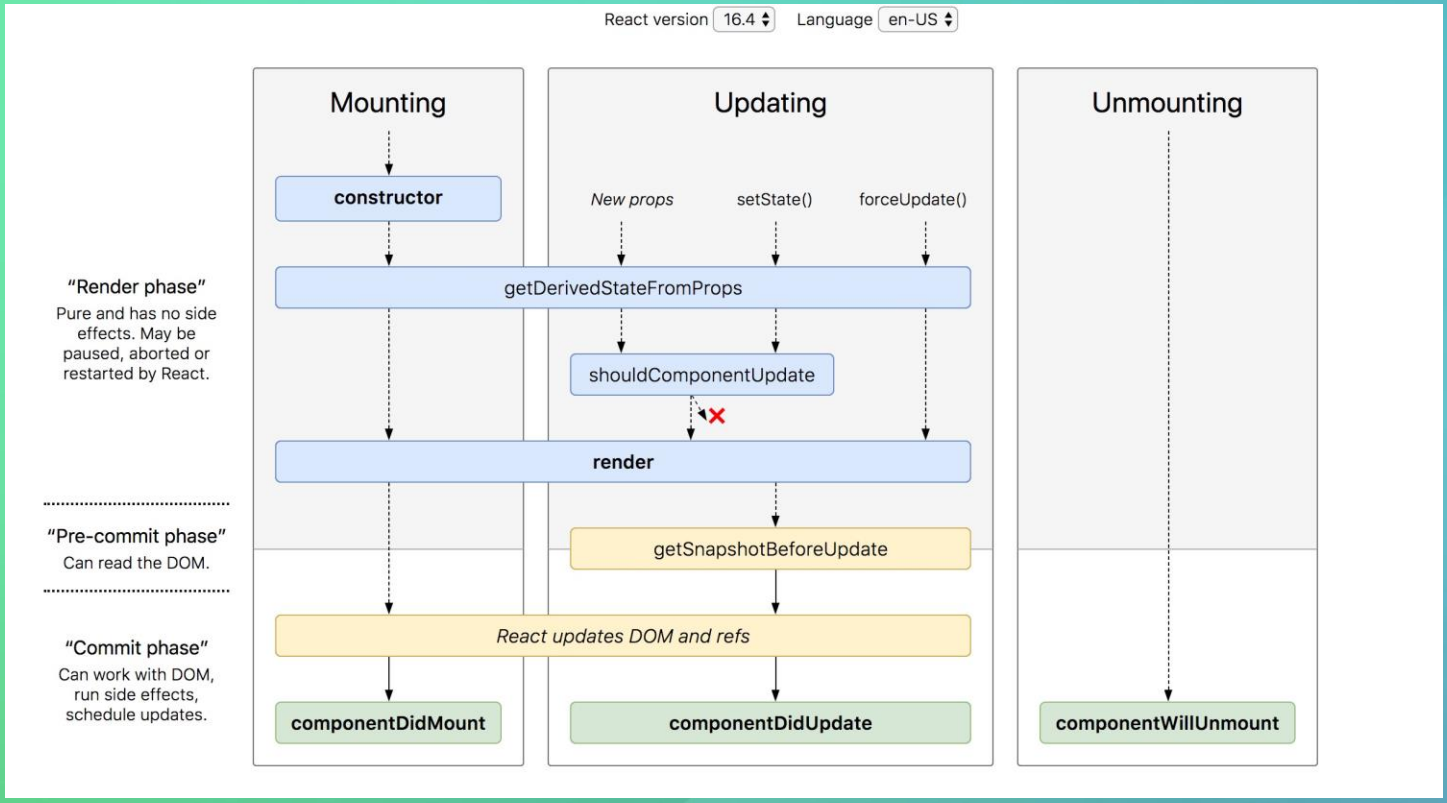
Lesson 31

План заняття

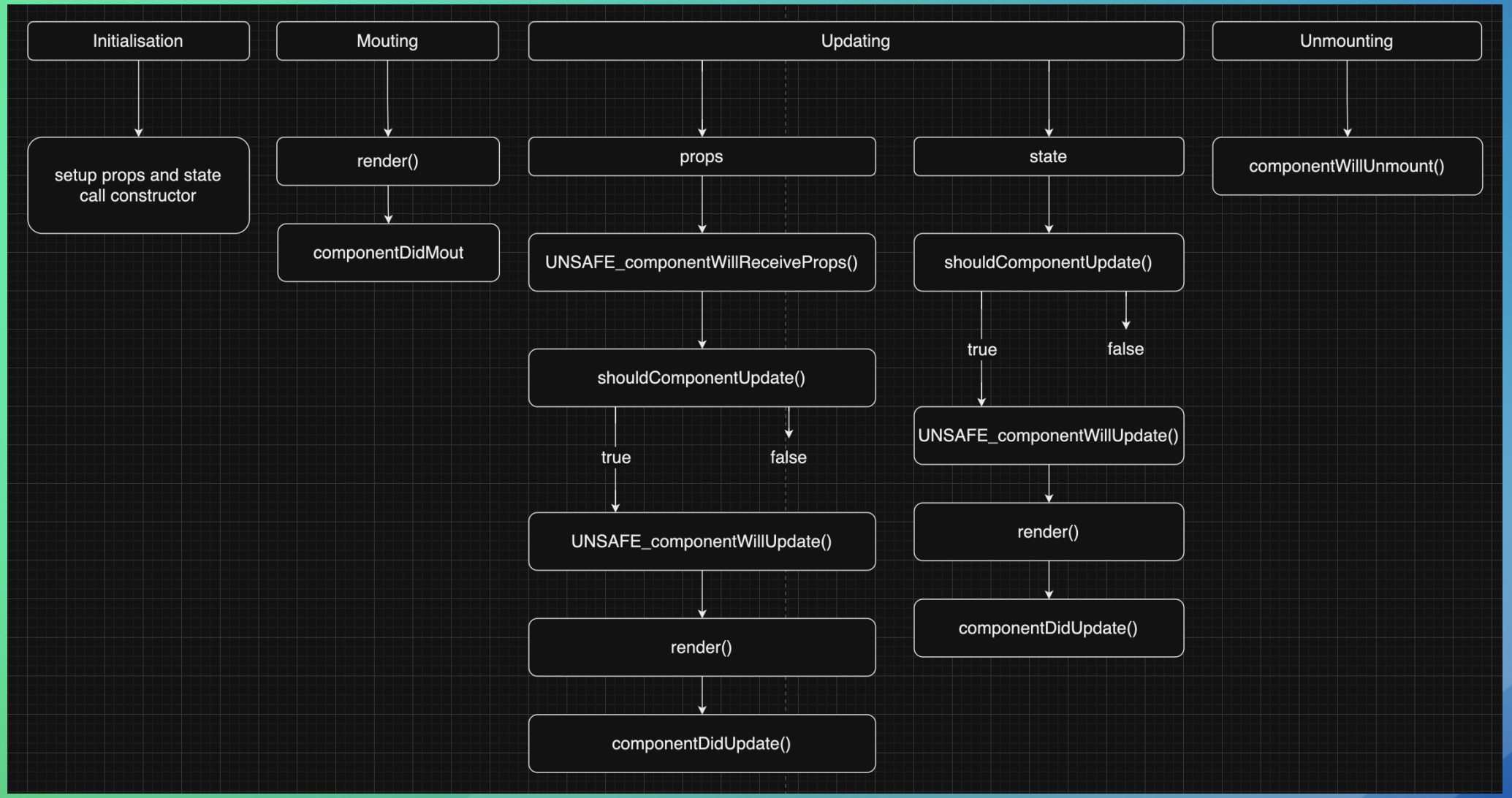
- Життеві цикли у класових компонентах;
- Pure Components;
- Refs

Component Lifecycle

Кожен компонент React проходить кілька стадій у процесі свого життя: він створюється, потім додається до DOM, отримує пропси, і нарешті видаляється з дерева. Цей процес називають життєвим циклом компонента (Component Lifecycle).



Component Lifecycle



constructor

Конструктор запускається перед монтуванням компонента класу (додається на екран). Як правило, конструктор використовується лише для двох цілей у React. Це дозволяє оголошувати стан і прив'язувати методи класу до екземпляра класу.

Якщо ви використовуєте сучасний синтаксис JavaScript, конструктори потрібні рідко. Ви можете використовувати публічні та статичні власивості. Він приймає наачальні пропси, та нічого не повертає. Тому що він під капотом мусить повернути this.

Важливо!

- Не запускайте жодних побічних ефектів або підписок у конструкторі. Натомість використовуйте для цього componenntDidMount.
- Усередині конструктора вам потрібно викликати super(props) перед будь-яким іншим оператором. Якщо ви цього не зробите, this.props буде невизначеним під час роботи конструктора, що може заплутати та викликати помилки.
- Конструктор — це єдине місце, де можна безпосередньо призначити this.state. У всіх інших методах замість цього потрібно використовувати this.setState(). Не викликайте setState у конструкторі.
- Коли ви використовуєте серверний рендеринг, конструктор також запускатиметься на сервері, а потім метод рендерингу. Однак такі методи життєвого циклу, як componentDidMount або componentWillUnmount, не працюватимуть на сервері.
- Коли строгий режим увімкнено, React двічі викличе конструктор під час розробки, а потім викине один із екземплярів. Це допоможе вам помітити випадкові побічні ефекти, які потрібно усунути з конструктора.

render()

Метод render є єдиним обов’язковим методом у компоненті класу. Він має вказувати, що ви хочете відображати на екрані.

React може використовувати рендеринг у будь-який момент, тому ви не повинні знати, що він запускається в певний час. Стандартний метод render має повертати частину JSX, але підтримує деякі інші типи повернення (наприклад, рядки). Щоб обчислити повернутий JSX, метод render може читати this.props, this.state і this.context.

Ви повинні написати метод render як чисту функцію, яку він повинен повернути той самий результат, якщо властивості, стан і контекст однакові. Він також не повинен Публикувати побічних ефектів (наприклад, налаштування підписок) або взаємодіяти з API браузера. Побічні ефекти мають відбуватися або в обробниках подій, або в таких методах, як componentDidMount.

render()

Важливо!

- render має бути написаний як чиста функція пропсів, стану та контексту. Він не повинен мати побічних ефектів.
- render не буде викликано, якщо shouldComponentUpdate визначено та повертає false.
- Коли строгий режим увімкнено, React двічі викличе рендер під час розробки, а потім викине один із результатів. Це допоможе вам помітити випадкові побічні ефекти, які потрібно усунути з методу візуалізації.
- Немає однозначної відповідності між викликом render і наступним викликом componentDidMount або componentDidUpdate. Деякі результати виклику рендерингу можуть бути відкинуті React, коли це буде корисно.

componentDidMount()

Якщо ви визначите метод componentDidMount, React викличе його, коли ваш компонент буде додано (монтовано) на екран. Це звичайне місце для початку отримання даних, налаштування підписок або маніпулювання вузлами DOM.

Якщо ви реалізуєте componentDidMount, вам зазвичай потрібно реалізувати інші методи життєвого циклу, щоб уникнути помилок. Наприклад, якщо componentDidMount зчитує певний стан або властивості, ви також повинні реалізувати componentDidUpdate для обробки їхніх змін і componentWillUnmount для очищення того, що робив componentDidMount.

Ця функція нічого не приймає и нічого не повертає.

Коли строгий режим увімкнено, під час розробки React викличе componentDidMount, потім негайно викличе componentWillUnmount, а потім знову викличе componentDidMount. Це допоможе вам помітити, якщо ви забули реалізувати componentWillUnmount або якщо його логіка не повністю «віддзеркалює» те, що робить componentDidMount.

Хоча ви можете викликати setState відразу в componentDidMount, краще уникати цього, коли є така можливість. Це запустить додаткову візуалізацію, але це станеться до того, як браузер оновить екран. Це гарантує, що навіть незважаючи на те, що візуалізація буде викликана двічі в цьому випадку, користувач не побачить проміжний стан.

shouldComponentUpdate()

Якщо ви визначите shouldComponentUpdate, React викличе його, щоб визначити, чи можна пропустити повторний рендеринг. Якщо ви впевнені, що хочете написати його вручну, ви можете порівняти this.props з nextProps і this.state з nextState і повернути false, щоб повідомити React, що оновлення можна пропустити.

shouldComponentUpdate виконається перед рендерингом, коли React знайде нові властивості або стан. За замовчуванням значення true. Цей метод не викликається для початкової візуалізації або коли використовується forceUpdate.

Параметри:

- nextProps: наступні властивості, які компонент збирається відобразити. Порівняйте nextProps з this.props, щоб визначити, що змінилося.
- nextState: наступний стан, у якому компонент збирається відобразити. Порівняйте nextState з this.state, щоб визначити, що змінилося.
- nextContext: наступний контекст, у якому компонент збирається відобразити. Порівняйте nextContext з this.context, щоб визначити, що змінилося. Доступно, лише якщо ви вказали статичний contextType (сучасний) або статичний contextTypes (застарілий).

Повертає true || false

shouldComponentUpdate()

Важливо!

- Цей метод існує лише як оптимізація продуктивності.
- Розгляньте можливість використання PureComponent замість написання shouldComponentUpdate вручну. PureComponent поверхнево порівнює властивості та стан і зменшує ймовірність того, що ви пропустите необхідне оновлення.
- React не рекомендує робити глибокі перевірки на рівність або використовувати JSON.stringify у shouldComponentUpdate. Це робить продуктивність непередбачуваною та залежить від структури даних кожного props та state. У найкращому випадку ви ризикуєте створити багатосекундні зупинки вашої програми, а в гіршому – вийти з ладу.
- Повернення false не запобігає повторному рендерингу дочірніх компонентів, коли їхній стан змінюється.
- Повернення false не гарантує, що компонент не буде повторно відтворено. React використовуватиме повернене значення як підказку, але все одно може вибрати повторне відтворення вашого компонента, якщо це має сенс з інших причин.

componentDidUpdate()

Якщо ви визначите метод componentDidUpdate, React викличе його відразу після того, як ваш компонент буде повторно відрендерено з оновленими властивостями або станом. Цей метод не викликається для початкової візуалізації.

Ви можете використовувати його для маніпулювання DOM після оновлення. Це також загальне місце для виконання мережевих запитів, якщо ви порівнюєте поточні властивості з попередніми (наприклад, мережевий запит може не знадобитися, якщо властивості не змінилися). Як правило, ви використовуєте його разом із componentDidMount і componentWillUnmount.

Параметри:

- prevProps: props перед оновленням. Порівняйте prevProps з this.props, щоб визначити, що змінилося.
- prevState: state перед оновленням. Порівняйте prevState з this.state, щоб визначити, що змінилося.
- snapshot: якщо ви реалізували getSnapshotBeforeUpdate, знімок міститиме значення, яке ви повернули з цього методу. В іншому випадку він буде undefined.

componentDidUpdate не повинен нічого повертати.

componentDidUpdate()

Важливо!

- componentDidUpdate не буде викликано, якщо shouldComponentUpdate визначено та повертає false.
- Логіка всередині componentDidUpdate зазвичай повинна бути загорнута в умови, що порівнюють this.props з prevProps, і this.state з prevState. В іншому випадку існує ризик створення нескінченних циклів.
- Хоча ви можете негайно викликати setState у componentDidUpdate, краще уникати цього, коли це можливо. Це запустить додаткову візуалізацію, але це станеться до того, як браузер оновить екран. Це гарантує, що навіть незважаючи на те, що візуалізація буде викликана двічі в цьому випадку, користувач не побачить проміжний стан. Цей шаблон часто спричиняє проблеми з продуктивністю, але він може бути необхідним у рідкісних випадках, як-от модальні та спливаючі підказки, коли вам потрібно виміряти вузол DOM перед відтворенням чогось, що залежить від його розміру чи положення.

Докладніше тут https://react.dev/reference/react/Component#componentdidupdate

componentWillUnmount()

Якщо ви визначите метод componentWillUnmount, React викличе його до того, як ваш компонент буде видалено (демонтовано) з екрана. Це звичайне місце для скасування отримання даних або видалення підписок.

Логіка всередині componentWillUnmount має «дзеркалювати» логіку всередині componentDidMount. Наприклад, якщо componentDidMount встановлює підписку, componentWillUnmount має очистити цю підписку. Якщо логіка очищення у вашому компонентіWillUnmount зчитує деякі властивості або стан, зазвичай вам також потрібно буде реалізувати componentDidUpdate, щоб очистити ресурси (наприклад, підписки), що відповідають старим властивостям і стану.

componentDidCatch(error, info)

Якщо ви визначите componentDidCatch, React викличе його, коли якийсь дочірній компонент (включно з віддаленими дочірніми компонентами) видає помилку під час візуалізації. Це дає змогу зареєструвати цю помилку в службі звітування про помилки у виробництві.

Як правило, він використовується разом зі статичним getDerivedStateFromError, який дозволяє оновлювати стан у відповідь на помилку та відображати повідомлення про помилку для користувача. Компонент із цими методами називається межею помилки.

Параметри:

- error: помилка, яка була викинута.
- info: об'єкт, що містить додаткову інформацію про помилку. Його поле componentStack містить трасування стека з компонентом, який викинув, а також імена та вихідні розташування всіх його батьківських компонентів.

Цей метод нічого не повертає.

Ще немає прямого еквівалента для componentDidCatch у функціональних компонентах. Також ви мможете використовувати бібліотеку https://github.com/bvaughn/react-error-boundary

Error boundary

За замовчуванням, якщо ваша програма видає помилку під час візуалізації, React видалить свій інтерфейс користувача з екрана. Щоб запобігти цьому, ви можете загорнути частину свого інтерфейсу користувача в межу помилки. Границя помилки — це спеціальний компонент, який дозволяє відображати резервний інтерфейс користувача замість частини, яка вийшла з ладу, наприклад, повідомлення про помилку.

Щоб реалізувати компонент межі помилки, вам потрібно надати статичний getDerivedStateFromError, який дозволяє оновлювати стан у відповідь на помилку та відображати повідомлення про помилку для користувача. Ви також можете додатково реалізувати componentDidCatch, щоб додати додаткову логіку, наприклад, щоб зареєструвати помилку в службі аналітики.

Вам не потрібно загортати кожен компонент в окрему error boundary. Коли ви думаєте про деталізацію меж помилок, подумайте, де має сенс відображати повідомлення про помилку.

Ви не можете виявити помилки під час компіляції, error boundary призначені для помилок під час виконання в інтерфейсі користувача.

У development mode ви завжди будете бачити overlay помилок, якщо не вимкнете його або не закриєте за допомогою кнопки X.

PureComponent

PureComponent схожий на Component, але він пропускає повторне рендеринг для тих самих атрибутів і стану. Компоненти класу все ще підтримуються React, але React не рекомендує використовувати їх у новому коді.

React повторно рендорить компонент кожного разу, коли його батьківський рендеринг виконується повторно. В якості оптимізації ви можете створити компонент, який React не буде повторно рендерити, коли його батьківський рендеринг буде повторно рендеритися, якщо його нові властивості та стан будуть такими ж, як старі властивості та стан.

Компонент React завжди повинен мати чисту логіку відтворення. Це означає, що він повинен повертати той самий вихід, якщо його властивості, стан і контекст не змінилися. Використовуючи PureComponent, ви повідомляєте React, що ваш компонент відповідає цій вимозі, тому React не потрібно повторно рендерити, доки його властивості та стан не змінилися. Однак ваш компонент все одно відтворюватиметься повторно, якщо контекст, який він використовує, зміниться.

createRef()

createRef створює об’єкт ref, який може містити довільне значення.

createRef не приймає параметрів.

createRef повертає об’єкт з єдиною властивістю:

- current: спочатку встановлено значення null. Пізніше ви можете встановити для нього щось інше. Якщо ви передасте об’єкт ref до React як атрибут ref до вузла JSX, React встановить його поточну властивість.

Важливо!

- createRef завжди повертає інший об’єкт. Це еквівалентно самому написанню { current: null }.
- У функціональному компоненті вам, ймовірно, потрібен useRef, який завжди повертає той самий об’єкт.
- const ref = useRef() еквівалентно const [ref, _] = useState(() => createRef(null)).

Дякую за увагу