모던리액트 딥 다이브

02장: 리액트 핵심요소 깊게 살펴보기

Date. 2024-03-10

Stud. B조 박민영

1.

JSX란?

JSX의 정의

JSX 변환 방식

2.

가상 DOM과 리액트 파이버

DOM과 가상 DOM 탄생 배경

리액트 파이버 정의/ 역할

리액트 파이버 트리

3.

클래스/함수 컴포넌트

클래스 컴포넌트

클래스의 생명주기 메서드/한계점

둘의 차이점 정리

4.

리액트 렌더링

렌더링은 '언제' 발생할까

렌더링의 실행 방식

5.

성능 최적화 방법: 메모이제이션

언제 사용하는 것이 좋을까?

“브라우저의 DOM을 바로 ‘값’으로 UI를 표현”

화면에 표시되는 UI를 자바스크립트의 문자열, 배열 등과 마찬가지로 값으로 관리하고

이러한 흐름을 효율적으로 관리하기 위한 매커니즘이 리액트의 핵심이다.

1.컴포넌트에 대한 뛰어난 의사결정:

얼마나 작은 단위로 쪼갤 것인지, 어떻게 재사용 컴포넌트를 만들 것인지 아는 개발자.

2.State가 존재하는 위치에 대한 의사 결정:

state는 가상 DOM 내에 여러 곳에 존재할 수 있는데, 이를 어디에 위치 시킬지 아는 개발자.

3.State가 변경될 때, 컴포넌트에 어떤 변화를 줄 것인지에 대한 의사결정:

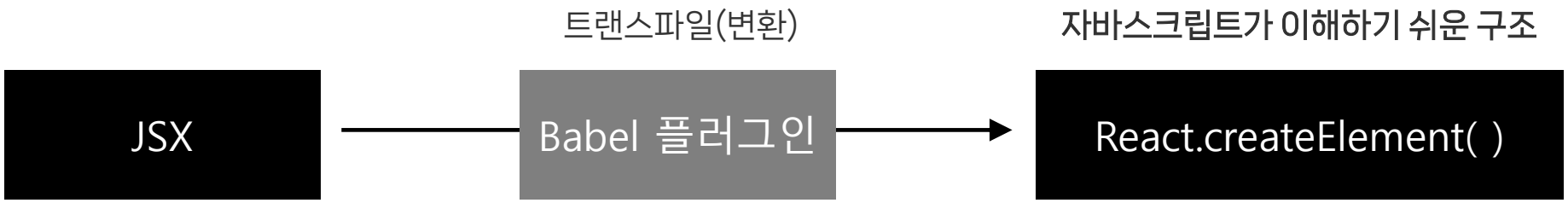
state가 변경되었을 때, 어떤 부분이 re-rendering 되어야 하는지 아는 개발자.

01 JSX란?

JSX: Javascript XML

HTML + XML을 자바스크립트 내부에 표현하는 것 + a
하나의 파일에 자바스크립트와 HTML 을 동시에 작성할 수 있게 한다.

JSXElement	JSXAttributes	JSXChildren	JSXStrings
JSX를 구성하는 가장 기본 요소 **HTML의 요소와 비슷한 역할.**	JSXElement에 부여할 수 있는 속성. (필수값 X).	JSXElement의 자식 값. JSX로 부모와 자식 관계를 나타낼 수 있다.	HTML의 내용을 손 쉽게 가져올 **수 있다.**

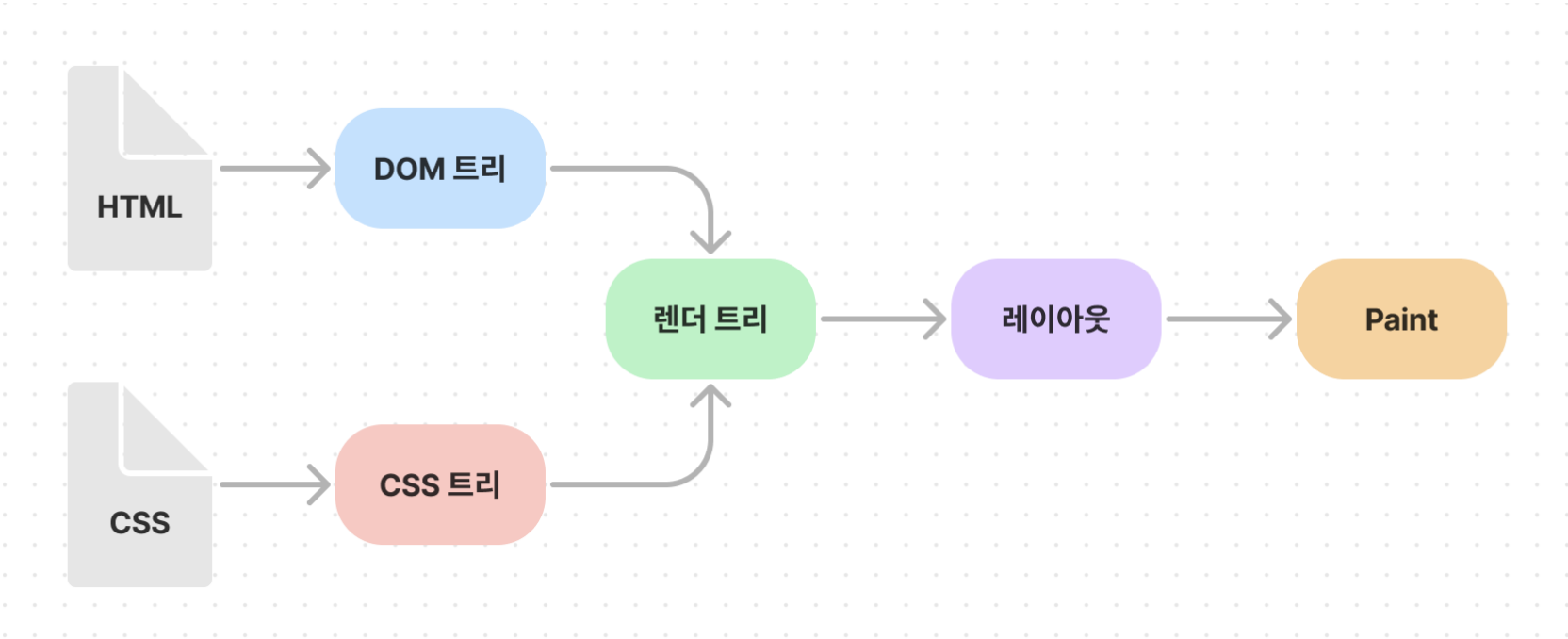


Babel 플러그인 사용하는 이유

- 경우에 따라 다른 JSX Element를 렌더링 해야 할 때, 굳이 요소 전체를 감싸지 않더라도 처리할 수 있다.
- JSX Element만 다르게 처리하여 중복 코드를 최소화할 수 있다.(**속성값, 자식요소는 동일,**)

02 가상 DOM과 리액트 파이버

브라우저 렌더링 과정



가상 DOM 탄생 배경

리페인팅 발생

DOM 변경 일어나는 요소 다수 발생

개발자

DOM의 모든 변경 상황 관리 어려움

싱글 페이지 어플리케이션

하나의 페이지에서 요소 재계산

사용자와 브라우저에게 많은 비용이 발생함.

파이버(자바스크립트 객체)

기존 렌더링 '스택 알고리즘'의 비효율성 문제 해결사.

웹 애플리케이션에서 발생하는 변화요소(애니메이션, 레이아웃, 인터렉션 등)의 반응성 문제 해결.

파이버의 목적

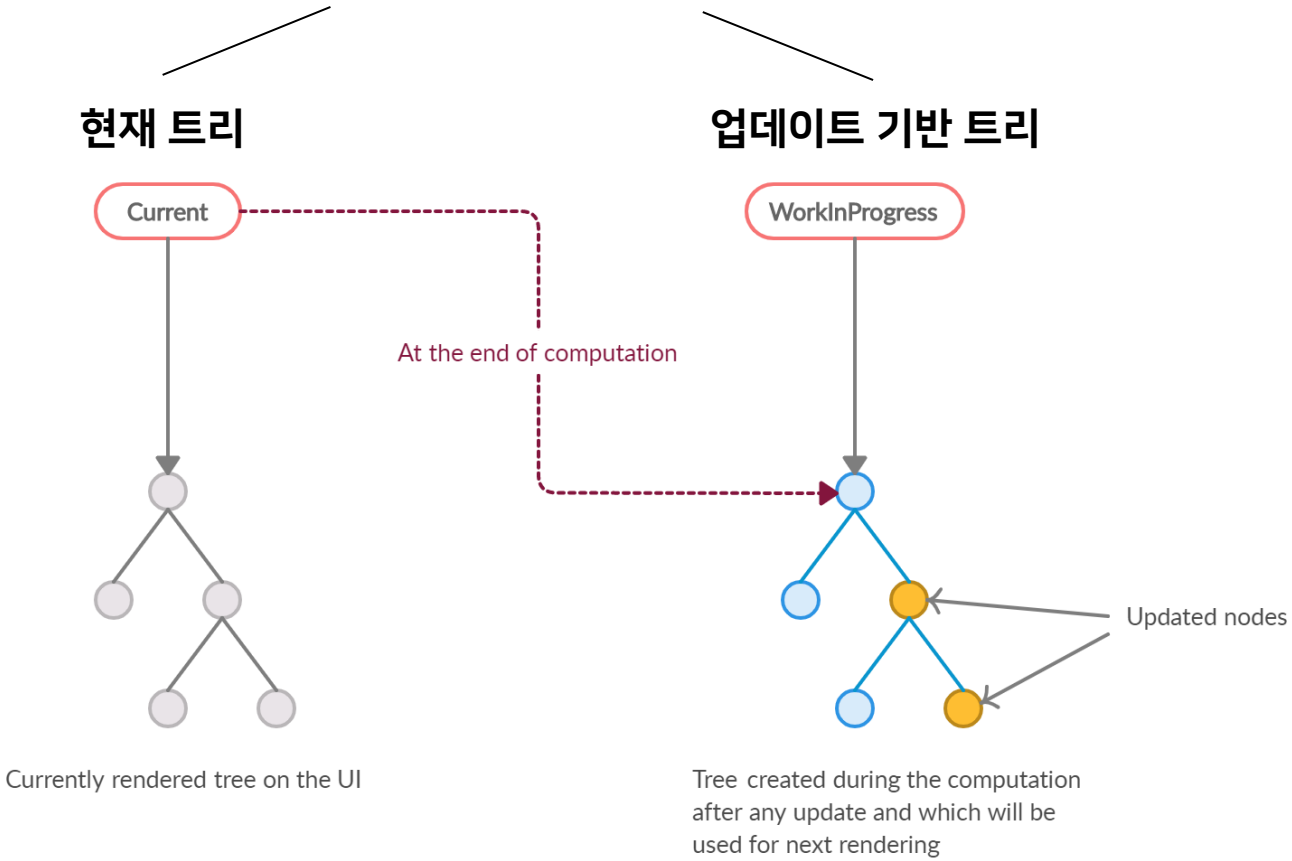
웹 애플리케이션에서 발생하는 변화요소(애니메이션, 레이아웃, 인터렉션 등)의 반응성 문제 해결.

파이버의 역할 (비동기적)

-작업을 작은 단위로 분활하고 쪼갠 다음, 우선순위를 매긴다.
-이러한 작업을 일시 중지하고 나중에 다시 시작할 수 있다.(우선순위가 높은 다른 업데이트가 오는 경우)
-이전에 했던 작업(파이버 트리)을 다시 재사용하거나 필요하지 않은 경우에는 폐기할 수 있다.

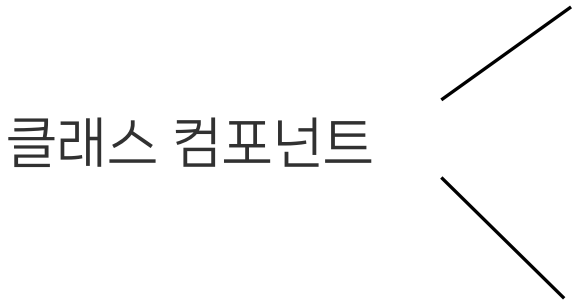
더블 버퍼링 기술

불완전한 트리를 보여주지 않기 위해 사용/ 커밋 단계에서 사용된다.



03 클래스 vs 함수 컴포넌트

클래스 컴포넌트



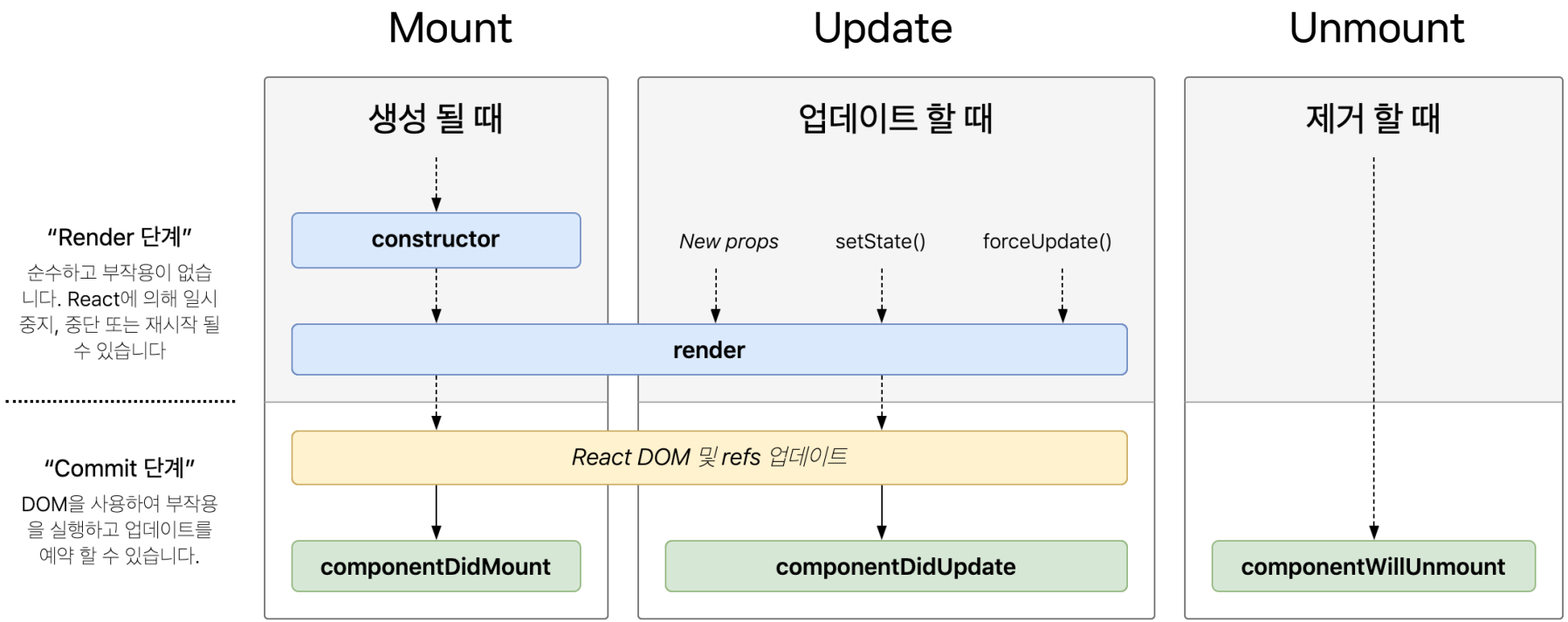
Component

'State가 업데이트 되는 대로' 렌더링 발생.

PureComponent

'얕은 비교'를 수행하여 state 값 결과가 다를 경우만 렌더링 발생.

클래스의 생명주기 메서드/한계점



render(): 컴포넌트가 UI를 렌더링 하기 위해 쓰인다.

componentDidMount(): 마운트 되는 즉시 실행. This.state로 state값을 변경하는 것이 가능하다.

componentDidUpdate(): 컴포넌트 업데이트가 일어난 이후 바로 실행. 일반적으로 state, props 변화에 따라 DOM을 업데이트 하는데 쓰인다.

componentWillUnmount(): 언마운트 되거나 사용되지 않기 직전에 호출. 메모리 누수나 클린 업 함수를 호출하기 위해 사용된다.

shouldComponentUpdate(): 리액트 컴포넌트 리렌더링을 막고 싶은 경우 사용한다.

Static getDerivedStateFromProps(): 반환 객체의 내용이 모두 state로 들어가게 된다.
Static으로 선언되어 있어 this에 접근할 수 없다.

getSnapShotBeforeUpdate(): DOM에 렌더링 되기 전에 윈도우 크기를 조절하거나 스크롤 위치를 조정하는 등 의 작업을 처리하는 데 유용하다.

getDerivedStateFromError(): 자식 컴포넌트에서 에러가 발생했을 때 적절한 에러 처리 로직을 구현할 수 있다.

componentDidCatch(): 자식 컴포넌트에서 에러가 발생했을 때 실행된다. 위의 메서드에서 하지 못했던 부수 효과를 실행할 수 있다.

클래스의 생명주기 메서드/한계점

-데이터의 흐름을 추적하기 어렵다 :
코드를 읽는 과정에서 state가 DJES 식의 흐름으로 변경되서 렌더링이 일어나는 지 판단하기 어렵다.

-애플리케이션 내부 로직의 재사용이 어렵다 :
재사용을 위해 Props를 넘겨주거나, 고차 컴포넌트로 감싸는 방법으로 재사용 시, 래퍼 지옥에 빠져들 위험성이 커진다.

-기능이 많아질수록 컴포넌트의 크기가 커진다:
내부에 로직이 많아질 수록, 내부에서 처리하는 데이터 흐름이 복잡해져 컴포넌트의 크기가 기하 급수적으로 커지는 문제 발생.

-클래스는 함수에 비해 상대적으로 어렵다: 많은 자바스크립트 개발자는 클래스 보단 함수에 더 익숙하다.

-코트 크기를 최적화하기 어렵다: 사용하지 않는 메서드 발생한다 즉, 트리 쉐이킹이 되지 않는다. -> 번들링 최적화에 불리하다.

-핫 리로딩을 하는 데 상대적으로 불리하다:
클래스 컴포넌트는 최초 렌더링 시에 instatnce를 생성하고 그 내부에서 state 값을 관리하기 때문에 instatnce를 새로 만들어야 한다. 새로 만든 instance 값은 초기화 되어 설계 상 핫 리로딩에 불리하다.

03 클래스 vs 함수 컴포넌트

둘의 차이점 정리

클래스 컴포넌트	함수 컴포넌트
코드 안에 라이프 사이클 기능과 state 기능 구현이 가능하다.	Hook을 사용하여 라이프 사이클 기능과 state 기능을 '비슷하게' 구현할 수 있다.
this에 바인딩 된 props를 사용한다.	Props를 인수로 받아 단순히 리액트 요소만 반환하는 함수이다.
핫 리로딩이 일어나면 다시 기본값으로 돌아간다.	핫 리로딩이 일어나도 변경된 상태 값이 유지된다.

04 리액트 렌더링

- **최초 렌더링**
- **리 렌더링**

클래스 컴포넌트

- 클래스 컴포넌트의 setState가 실행되는 경우
- 클래스 컴포넌트의 forceUpdate가 실행되는 경우

함수 컴포넌트

- 함수 컴포넌트의 useState()의 두번째 배열 요소인 sette가 실행되는 경우
- 함수 컴포넌트의 useReducer()의 두번째 배열 요소인 dispatc가 실행되는 경우

- 컴포넌트의 key props가 변경되는 경우
- Props가 변경되는 경우
- 부모 컴포넌트가 렌더링 될 경우

렌더 단계

컴포넌트를 렌더링하고 변경 사항(type, props, key 등)을 계산하는 모든 작업.

커밋 단계

렌더 단계의 변경 사항을 **실제 DOM에 적용**하여 사용자에게 보여주는 과정.
해당 과정이 끝나야 브라우저의 렌더링이 발생.

※ 렌더링이 실행된다고 해서 무조건 DOM 업데이트가 일어나는 것은 아님.
변경 사항 감지되지 않는 경우 커밋 단계 생략 가능함.

05 성능 최적화 방법: 메모이제이션

05 성능 최적화 방법: 메모이제이션

언제 사용하는 것이 좋을까?

메모이제이션: 리액트에서 발생하는 렌더링을 최소화하기 위해 사용된다.

하지만 정확히 언제 사용되는 것인지에 대해 명확하게 답변이 어려움.

신중히 사용하자

"메모이제이션도 비용이 발생한다":

-렌더링 또는 재계산이 필요한지 확인 비용.

-이전 결과물을 저장해 두었다가 다시 꺼내 와야 하는 비용.

가벼운 작업인 경우, 차라리 매번 작업을 수행해 결과를 반환하는 것이 더 빠를 수 있음.

일단 애플리케이션을 어느정도 만든 후, 개발자 도구나 useEffect를 사용해 실제로 어떻게 렌더링이 일어나고 있는지 확인하고 필요한 곳에서만 최적화 하는 것이 옳다.

모조리 사용하자

"현실적으로 최적화나 성능 향상에 쏟을 시간이 적다":

-애플리케이션 규모가 커질 경우.

-개발자와 컴포넌트의 복잡성이 증가하는 경우.

일단 memo로 감싼 뒤에 생각해보는 것이 더 나을 수 있음.

지불해야 하는 비용의 경우, 어차피 리액트의 기본적인 알고리즘으로 이전 결과물은 저장되어 있기 때문에 우리가 memo로 지불해야 하는 비용은 props에 대한 얕은 비교 뿐이다.

E.O.D

감사합니다