



# Reactjs

---

본 강의교안을 무단 배포 및 게재를 금지합니다.



# Reactjs

---

- Overview
  - What is Reactjs
  - Virtual DOM



# Overview

---

## ■ What is react?

- 사용자 인터페이스를 만들기 위한 JavaScript 라이브러리
- 지난 몇 년간 JavaScript 를 사용하여 UI를 직접 구성하는것에 대한 수요가 증가
- 이를 위해 UI를 개발하기 위해 페이스북이 개발한 오픈소스 라이브러리인 React가 사용되기 시작
- React Native는 앱 개발이 가능, React 360 가상 현실 어플리케이션을 개발하는 데 사용



# DOM

---

## ■ DOM

- HTML 문서의 내용과 구조가 객체 모델로 만든 것
  - HTML(XML) 문서에 접근하기 위한 일종의 인터페이스
  - 트리 형태로 되어 있어서 특정 노드를 찾거나, 수정, 제거 할 수 있다



# DOM

---

## ■ DOM 렌더링 과정

### ■ DOM 트리 생성

- HTML 파서가 전달받은 HTML 파일에 담겨 있는 HTML 코드들을 파싱 하여 메모리 상에 DOM 객체들로 이뤄진 DOM 트리를 생성한다

### ■ 스타일 규칙 생성

- CSS 파서가 전달받은 HTML 파일과 CSS 파일에 담겨 있는 CSS 코드들을 파싱 하여 메모리 상에 스타일 규칙들을 생성

### ■ 렌더 트리 생성

- 앞서 생성한 DOM 트리와 스타일 규칙들의 정보를 바탕으로, 실제로 브라우저의 화면에 노출되어야 하는 노드들에 대한 정보인 렌더 트리를 생성
- 참고로 '실제로 브라우저의 화면에 노출'된다는 것의 의미는, <head> 태그 혹은 display 속성이 none인 태그들과 같이 눈에 보이지 않는 노드들은 렌더 트리를 생성할 때 제외한다는 것을 의미



# DOM

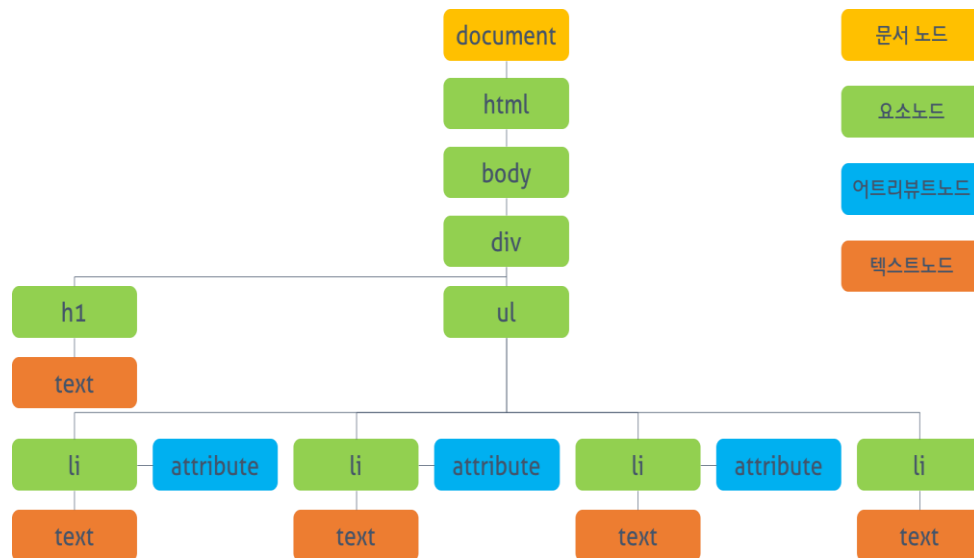
---

- 레이아웃(리플로우)
  - 렌더 트리의 각 노드가 화면의 어느 위치에 어떤 크기로 배치되어야 하는지에 대한 정보를 계산하는 단계
  
- 페인트(리페인트)
  - 렌더 트리의 각 노드를 실제로 브라우저의 화면에 그리는 단계

# DOM

## ■ DOM

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      .red { color: #ff0000; }
      .blue { color: #0000ff; }
    </style>
  </head>
  <body>
    <div>
      <h1>Cities</h1>
      <ul>
        <li id="one" class="red">Seoul</li>
        <li id="two" class="red">London</li>
        <li id="three" class="red">Newyork</li>
        <li id="four">Tokyo</li>
      </ul>
    </div>
  </body>
</html>
```





# Virtual DOM

---

## ■ Virtual DOM

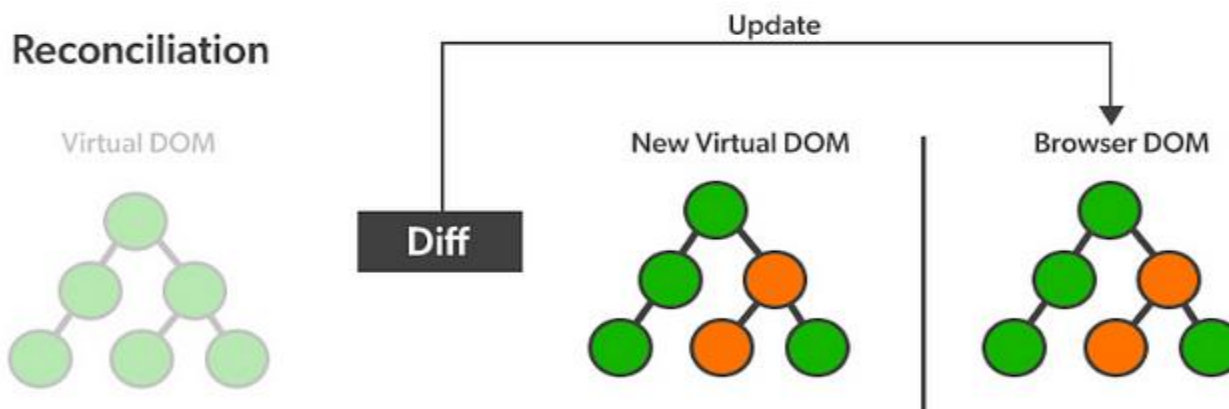
- DOM을 좀더 추상화시킨 자바스크립트 객체
  - 매번 실제 DOM에 접근하여 조작하는 대신에
  - DOM의 상태를 메모리에 저장하고 변경 전과 변경 후의 상태를 비교한 뒤 DOM에 변경이 있을 때만 해당 변경을 반영한다
  - DOM과 다르게 직접적으로 브라우저 화면의 UI를 조작할 수 있게 해주는 API는 제공하지 않는다



# Virtual DOM

## ■ Re-rendering

- 리액트는 STATE가 변경될 때마다 Re-Rendering이 발생
- 실제 브라우저에 렌더링 되기 전에 새로운 내용이 담긴 가상DOM을 생성
- 차이가 발생한 부분만을 (브라우저상의) 실제 DOM에 적용 (Reconciliation)
- 변경된 모든 Element들을 집단화시켜 이를 한번에 실제 DOM에 적용하는 방식





# JSX

## ■ JSX Expression

### ■ Javascript Expression

- React는 많은 패턴에 모던 자바스크립트 특징을 사용
- JSX 구문의 사용
- JSX는 JavaScript의 구문을 확장하여 HTML과 유사한 코드를 함께 사용할 수 있도록 함
- Heading 변수를 h1 태그를 렌더링 할 때 사용 할 수 있다
- 여러 줄을 입력 할 경우 구분을 짓기 위해 괄호를 사용 할 수 있음

```
const heading = <h1>Mozilla Developer Network</h1>;
```

```
const header = (  
  <header>  
    <h1>Mozilla Developer Network</h1>  
  </header>  
)
```



# React.Component

---

## ■ React.Component

### ■ 개요

- React를 사용할 때는 컴포넌트를 class 또는 함수로 정의할 수 있습니다
- React 컴포넌트 class를 정의하려면 React.Component를 상속해야 함

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```



# React.Component

---

## ■ lifecycle

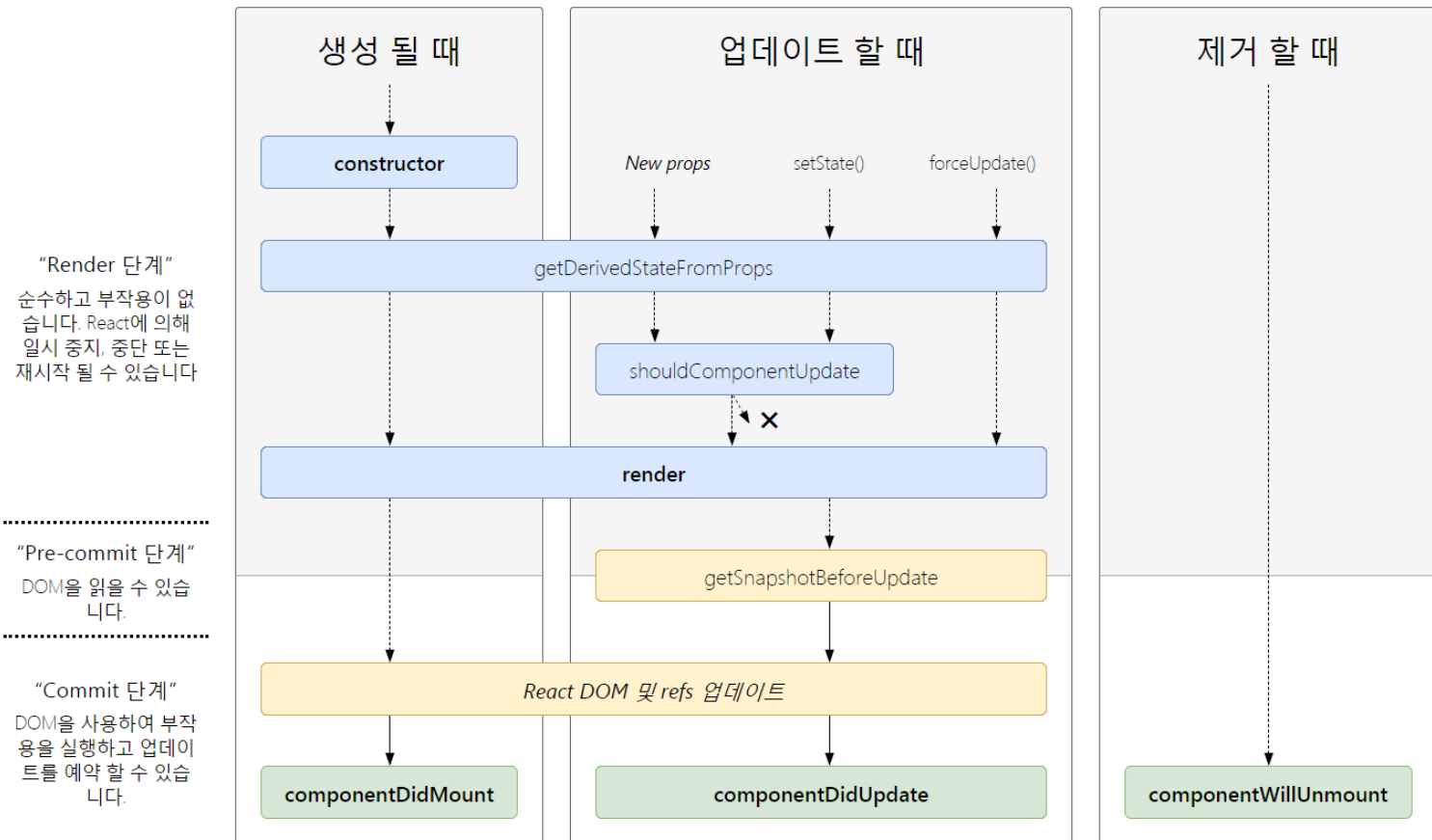
- 모든 컴포넌트는 여러 종류의 “생명주기 메서드”를 가지며, 이 메서드를 오버라이딩하여 특정 시점에 코드가 실행되도록 설정할 수 있습니다.
- React를 사용할 때는 컴포넌트를 class 또는 함수로 정의할 수 있습니다
- 초기화단계, 업데이트단계, 소멸단계

# React.Component

## ■ lifecycle

☑ 덜 일반적인 라이프 사이클 표시

React 버전 ^16.4 언어 KR 한국어



# React.Component

## ■ lifecycle

```
export default class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```



# React.Component

## ■ Props

- props 는 properties 의 줄임말입니다. 우리가 어떠한 값을 컴포넌트에게 전달해줘야 할 때, props 를 사용합니다

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

```
import React from 'react';  
import Hello from './Hello';  
  
function App() {  
  return (  
    <Welcome name="react" />  
  );  
}  
  
export default App;
```



# React.Component

- 객체 비구조 할당문을 통해서
- Props 객체의 property를 개별로 할당 할 수 있음

```
export default class Welcome extends Component {  
  render() {  
    const {name} = this.props;  
    return (  
      <div>  
        { name }  
      </div>  
    );  
  }  
}
```





# React.Component

---

## ■ state

- 컴포넌트 내부에서 변할 수 있는 값
- lifecycle 동안 값을 유지 할 수 있음
- state 값 변경 하면 리렌더링
- setState()는 컴포넌트의 state 객체에 대한 업데이트를 실행
- props는 컴포넌트에 전달되는 반면 state는 컴포넌트 안에서 관리됨
- setState 호출은 비동기, setState 호출 직후 새로운 값이 this.state에 반영 되지 않음

# React.Component

- 이전 state값을 기준으로 값을 계산해야 한다면 객체 대신 updater 함수를 전달

```
export default class Counter extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
    this.increment = this.increment.bind(this);  
  }  
  
  increment() {  
    this.setState({count: this.state.count+1});  
    // this.setState((preState)=>{  
    //   return {count: preState.count+1};  
    // });  
  }  
  
  render() {  
    return (  
      <div>  
        <p>{this.state.count}</p>  
        <button onClick={this.increment}>increment</button>  
      </div>  
    );  
  };  
}
```

# 함수형 컴포넌트

## ■ 함수형 컴포넌트

### ■ 개요

- 자바스크립트의 함수형태로 표현되는 컴포넌트
- this가 없음
- return문이 render 함수를 대신함, jsx 사용

```
import {useState} from 'react';

export default function CounterFuncStyle(props) {
  const [count, setCount] = useState(0);

  const clickHandler = () => {
    setCount((preCount) => preCount+1);
  };

  return(
    <div>
      <p>{count}</p>
      <button onClick={clickHandler}>increase</button>
    </div>
  );
}
```