State와 생명주기

State

- 상태
  - 자바스크립트 객체이다.
  - 리액트 컴포넌트의 변경 가능한 데이터를 다룬다.
  - 컴포넌트 개발자가 직접 정의한다.
- State 사용시 주의 사항
  - 렌더링이나 데이터 흐름에 사용되는 값만 state에 포함시켜야 한다.
  - State가 변경될 경우 컴포넌트는 재렌더링 된다.

State 특징

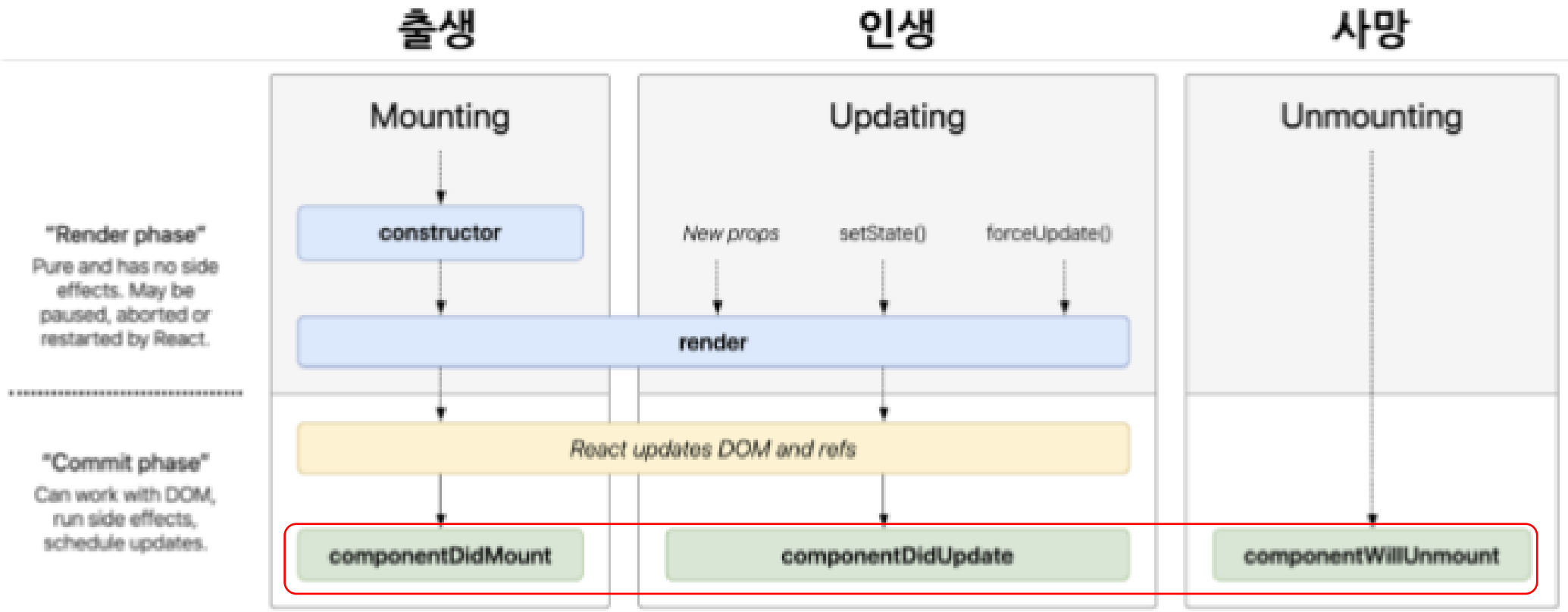
- State는 클래스의 생성자에서 정의된다.
  - 따라서 state는 클래스가 실행될 때 메모리에 존재한다.
  - State는 setter() 함수에 의해 수정 가능하다

```
01  class LikeButton extends React.Component {
02      constructor(props) {
03          super(props);
04
05          this.state = {
06              liked: false
07          };
08      }
09
10      ...
11  }
```

```
01  // state를 직접 수정 (잘못된 사용법)
02  this.state = {
03      name: 'Inje'
04  };
05
06  // setState 함수를 통한 수정 (정상적인 사용법)
07  this.setState({
08      name: 'Inje'
09  });
```

컴포넌트 생명 주기

- 컴포넌트가 생성(Mounting) → 업데이트(Updating) → 제거(Unmounting) 되는 일련의 과정

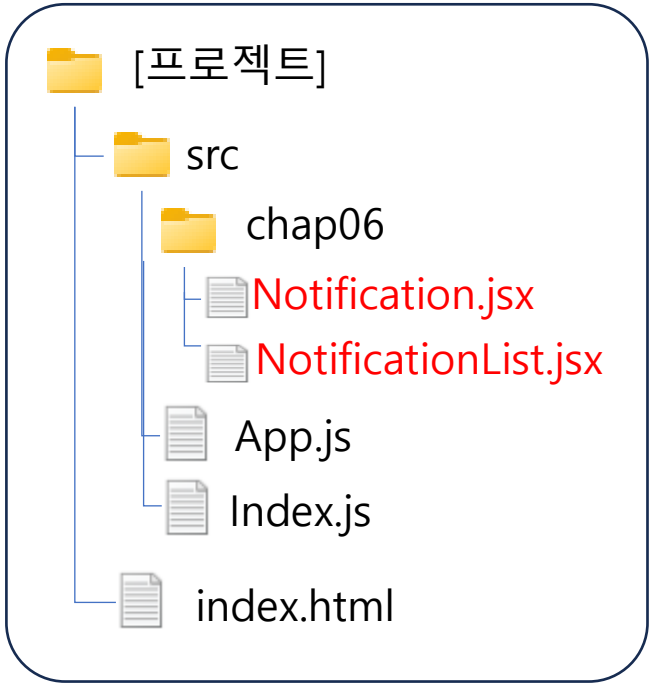


생명주기에 따라 호출되는 함수

컴포넌트 생명 주기

1. 마운트 단계: 컴포넌트 생성자 실행(state 정의), 렌더링 -> componentDidMount() 호출
2. 업데이트 단계: props 변경 또는 setState() 호출에 의한 state 변경 forceUpdate() 호출 등으로 재렌더링 -> componentDidUpdate()
3. 언마운트 단계: 상위 컴포넌트에서 더 이상 호출하지 않을 경우, 언마운트. 언마운트 직전에 componentWillUnmount() 호출

State와 생명주기 함수 사용 실습



State와 생명주기 함수 사용 실습

1. **Notification 컴포넌트 만들기**
   1.1 Notification 클래스 컴포넌트의 스타일 정의

```
const styles = {
    wrapper: {
        margin: 8,
        padding: 8,
        display: "flex",
        flexDirection: "row",
        border: "1px solid grey",
        borderRadius: 16,
    },
    messageText: {
        color: "black",
        fontSize: 16,
    },
};
```

State와 생명주기 함수 사용 실습

1. Notification 컴포넌트 만들기
1.2 Notification 클래스 컴포넌트 코드

```
import React from "react";

class Notification extends React.Component {
  constructor(props) {
    super(props);

    this.state = {};
  }
  render() {
    return (
      <div style={styles.wrapper}>
        <span style={styles.messageText}>{this.props.message}</span>
      </div>
    );
  }
}

export default Notification;
```

State와 생명주기 함수 사용 실습

2. NotificationList 컴포넌트 만들기

```
import Notification from "./Notification";

const reservedNotifications = [
  {
    message: "안녕하세요, 오늘 일정을 알려드립니다.",
  },
  {
    message: "점심식사 시간입니다.",
  },
  {
    message: "이제 곧 미팅이 시작됩니다.",
  },
];

var timer;
```

State와 생명주기 함수 사용 실습

2. NotificationList 컴포넌트 만들기

```
class NotificationList extends React.Component {

        constructor(props) { }

        componentDidMount() { }

        componentWillUnmount() { }

        render() { }


}
```

State와 생명주기 함수 사용 실습

2. NotificationList 컴포넌트 만들기

```
  constructor(props) {
    super(props);

    this.state = {
      notifications: [],  //state 데이터 초기화
    };
  }
```

State와 생명주기 함수 사용 실습

2. NotificationList 컴포넌트 만들기

```
  componentDidMount() {
    const { notifications } = this.state;
    timer = setInterval(() => {
      if (notifications.length < reservedNotifications.length) {
        const index = notifications.length;
        notifications.push(reservedNotifications[index]);

        this.setState({       //state 데이터 업데이트를 위한 setState() 함수 정의
          notifications: notifications,
        });
      } else {
        clearInterval(timer);
      }
    }, 1000); //end of timer
  } //end of componentDidMount
```

State와 생명주기 함수 사용 실습

2. NotificationList 컴포넌트 만들기

```
componentWillUnmount() {
  if (timer) {      //언마운트하기 전에 timer 존재할 경우 제거.
    clearInterval(timer);
  }
}

render() {
    return (
      <div>
        {this.state.notifications.map((notifications) => {
          return <Notification message={notifications.message} />;
        })}
      </div>
    );
}
```

State와 생명주기 함수 사용 실습

3. Index.js에서 NotificationList 컴포넌트 호출하기

```
import NotificationList from "./chap06/NotificationList";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <NotificationList />
  </React.StrictMode>
);
```

안녕하세요, 오늘 일정을 알려드립니다.

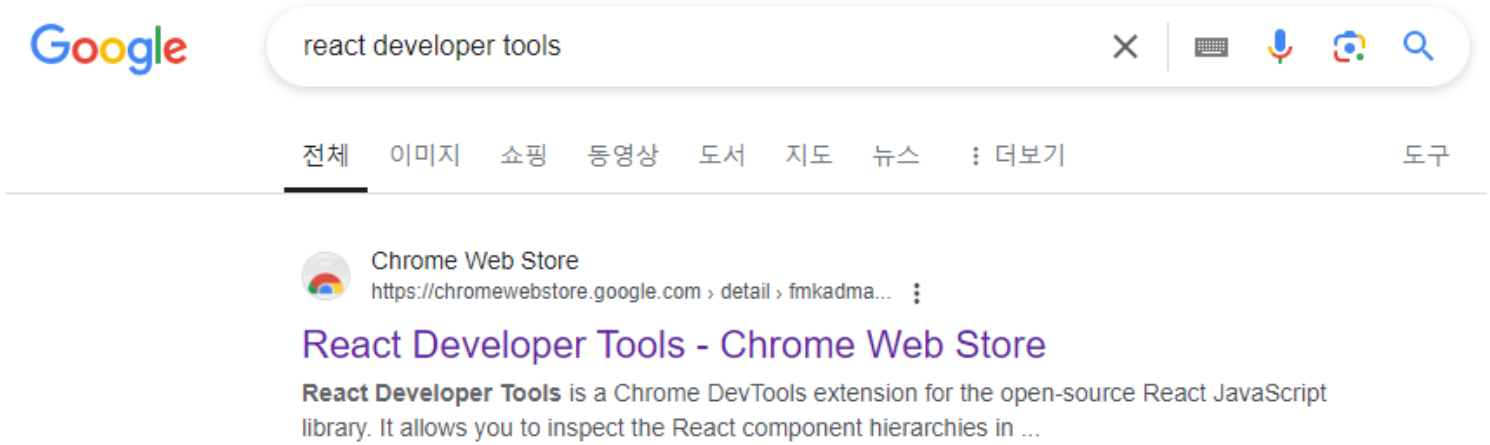
점심식사 시간입니다.

이제 곧 미팅이 시작됩니다.

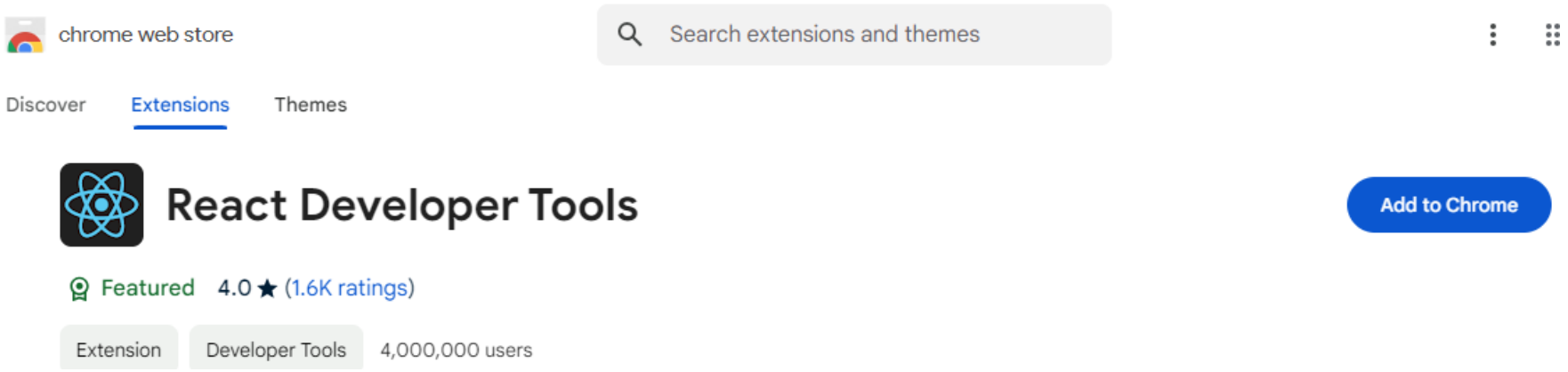
개발자도구의 element 탭 확인:
1초간격으로 메시지 출력 코드가 생성되는 것을 확인할 수 있음

React 개발자 도구 설치

1. 구글 검색에서 React Developer Tools 검색
2. 최상단 링크 클릭



3. 크롬 웹스토어 화면에서 'Add to Chrome' 버튼 눌러 설치



State와 생명주기 함수 사용 실습

- Notification 컴포넌트가 호출될 때 생명주기 함수가 어떻게 호출되는지 확인하여 보자.

1. 생명주기 함수가 호출될 때 로그가 출력되도록 코드 작성. 또한 로그 출력시 표출되는 알림 객체의 아이디가 함께 출력되도록 하기 위해 각 알림 객체에 아이디 추가.
   NotificationList 컴포넌트 코드 수정

```
const reservedNotifications = [
  {
    id: 1,
    message: "안녕하세요, 오늘 일정을 알려드립니다.",
  },
```

```
            <Notification
              key={notifications.id}
              id={notifications.id}
              message={notifications.message}
            />
```

State와 생명주기 함수 사용 실습

2. Notification 컴포넌트에 생명주기 함수 및 로그 추가

```
componentDidMount() {
  console.log(`${this.props.id} componentDidMount() called.`);
}

componentDidUpdate() {
  console.log(`${this.props.id} componentDidUpdate() called.`);
}

componentWillUnmount() {
  console.log(`${this.props.id} componentWillUnmount() called`);
}
```

- 개발자 도구의 console 탭을 통해 로그 확인
- componentDidMount()와 componentDidUpdate() componentWillUnmount() 가 생명주기 순서대로 호출되고 있으나 componentWillUnmount() 함수는 최종적으로 호출되지 않음(브라우저 화면에서도 제거되지 않고 있음)

State와 생명주기 함수 사용 실습

2. Notification 컴포넌트가 화면에서 제거될 때 componentWillUnmount() 함수가 호출되는 것을 확인하자. 알림 문자가 모두 출력되었다면, notifications 배열을 비우도록 하여 확인해 보자.

NotificationList 컴포넌트 코드 수정

```
      } else {
        this.setState({
          notifications: [],
        });
        clearInterval(timer);
      }
    }, 1000); //end of timer
```

```
1 componentWillUnmount() called.
2 componentWillUnmount() called.
3 componentWillUnmount() called.
```

- 개발자 도구의 console 탭을 통해 로그 확인하면 최종적으로 componentWillUnmount() 함수가 호출되는 것을 볼 수 있음

정리

- **State**
  - State란?
    - 리액트 컴포넌트의 변경 가능한 데이터
    - 컴포넌트를 개발하는 개발자가 직접 정의해서 사용
    - state가 변경될 경우 컴포넌트가 재렌더링됨
    - 렌더링이나 데이터 흐름에 사용되는 값만 state에 포함시켜야 함
  - State의 특징
    - 자바스크립트 객체 형태로 존재
    - 직접적인 변경이 불가능 함
    - 클래스 컴포넌트
      - 생성자에서 모든 state를 한 번에 정의
      - state를 변경하고자 할 때에는 꼭 setState()함수를 사용해야 함
    - 함수 컴포넌트
      - useState()훅을 사용하여 각각의 state를 정의
      - 각 state별로 주어지는 set함수를 사용하여 state 값을 변경

정리

- **생명주기**
  - 마운트
    - 컴포넌트가 생성될 때
    - componentDidMount()
  - 업데이트
    - 컴포넌트의 props가 변경될 때
    - setState() 함수 호출에 의해 state가 변경될 때
    - forceUpdate()라는 강제 업데이트 함수가 호출될 때
    - componentDidUpdate()
  - 언마운트
    - 상위 컴포넌트에서 현재 컴포넌트를 더 이상 화면에 표시하지 않게 될 때
    - componentWillUnmount()
  - 컴포넌트는 계속 존재하는 것이 아니라 시간의 흐름에 따라 생성되고 업데이트되다가 사라지는 과정을 겪음