Chapter 6. State와 생명주기

교재: 처음만난 리액트 (저자: 이인제, 한빛출판사)

Contents

- CHAPTER 6: State와 생명주기

6.1 State 개념과 특징

6.2 생명주기(Lifecycle)

6.3 (실습) State와 생명주기 함수 사용하기

- 실습1. State 사용하기
- 실습2. React Developer Tools 설치
- 실습3. 생명주기 함수 사용해 보기

SECTION 6.1 State

State란?

- 리액트 컴포넌트의 상태
- 리액트 컴포넌트의 변경 가능한 데이터
- state는 컴포넌트를 개발하는 개발자가 직접 정의

State 정의할 때 주의사항

- 렌더링이나 데이터 흐름에 사용되는 값만 state에 포함시켜야 함
  - state가 변경될 경우 컴포넌트가 재렌더링 됨
  - 렌더링이나 데이터 흐름과 관련없는 값을 포함하면 불필요한 렌더링이 발생하여 성능 저하
  - 그외 값은 컴포넌트의 인스턴스 필드로 정의

SECTION 6.1 State

- **State 정의**
  - 클래스 컴포넌트 - 생성자(constructor)에서 state 정의
  - 함수 컴포넌트 - useState( )라는 훅을 사용해서 정의 (7장)

```
1 class LikeButton extends React.Component {
2     constructor(props) {
3         super(props);
4
5         this.state = {
6             liked: false
7         };
8     }
9
10    ...
11 }
```

SECTION 6.1 State

State의 특징

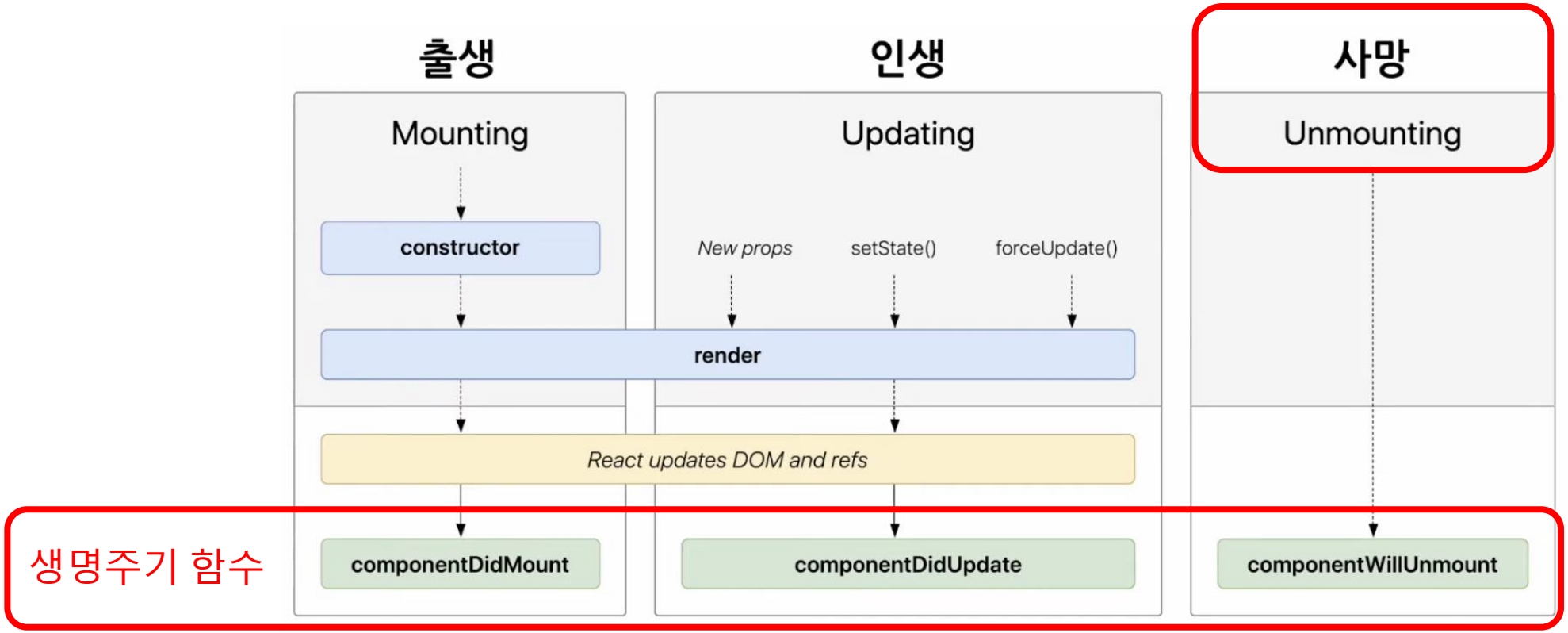
- state는 정해진 형태가 있는 것이 아니라, 일반적인 JavaScript 객체
- state는 정의된 이후 자바스크립트 변수처럼 직접 수정할 수 없음
  - 직접 수정할 경우 리액트가 state 변경을 인지하지 못할 수 있음
- state 변경할 때는 setState( ) 함수를 사용

```
// state를 직접 수정 (잘못된 사용법)
this.state = {
    name: 'Inje'
};

// setState 함수를 통한 수정 (정상적인 사용법)
this.setState({
    name: 'Inje'
});
```

SECTION 6.2 생명주기(Lifecycle)

- **리액트 컴포넌트의 생명주기**

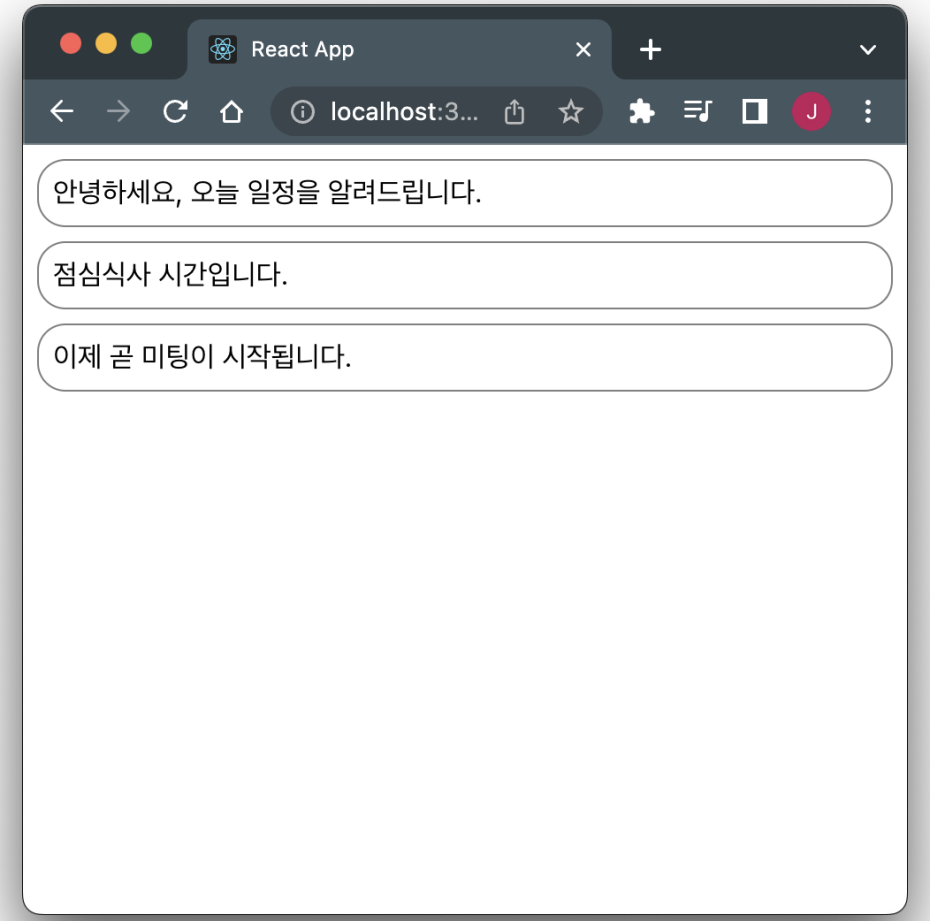
상위 컴포넌트에서 현재 컴포넌트를
더 이상 화면에 표시하지 않게 될 때



- 컴포넌트는 계속 존재하는 것이 아니라,
  시간의 흐름에 따라 생성되고 업데이트 되다가 사라진다.

PRACTICE 6.3 State와 생명주기 함수 사용하기

- [실습] State 사용하기 - 알림 목록 앱



PRACTICE 6.3 State와 생명주기 함수 사용하기

- VS Code - 06 폴더 생성
- Notification.jsx – 알림 컴포넌트

```jsx
import React from "react";

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

```jsx
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;
```

PRACTICE 6.3 State와 생명주기

- NotificationList.jsx – 알림 목록 컴포넌트

```jsx
import React from "react";
import Notification from "./Notification";

const reservedNotifications = [
  {
    message: "안녕하세요 오늘 일정을 알려드립니다",
  },
  {
    message: "점심 식사 시간입니다",
  },
  {
    message: "미팅 시작 시간입니다.",
  },
];

let timer;

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

    this.state = {
      notifications: [],
    }
  }

  componentDidMount() {
    console.log('componentDidMount');
    const { notifications } = this.state;
    timer = setInterval(() => {
      if (notifications.length < reservedNotifications.length) {
        const index = notifications.length;
        notifications.push(reservedNotifications[index]);
        this.setState({
          notifications: notifications,
        });
      } else {
        clearInterval(timer);
      }
    }, 1000);
  }

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

export default NotificationList;
```

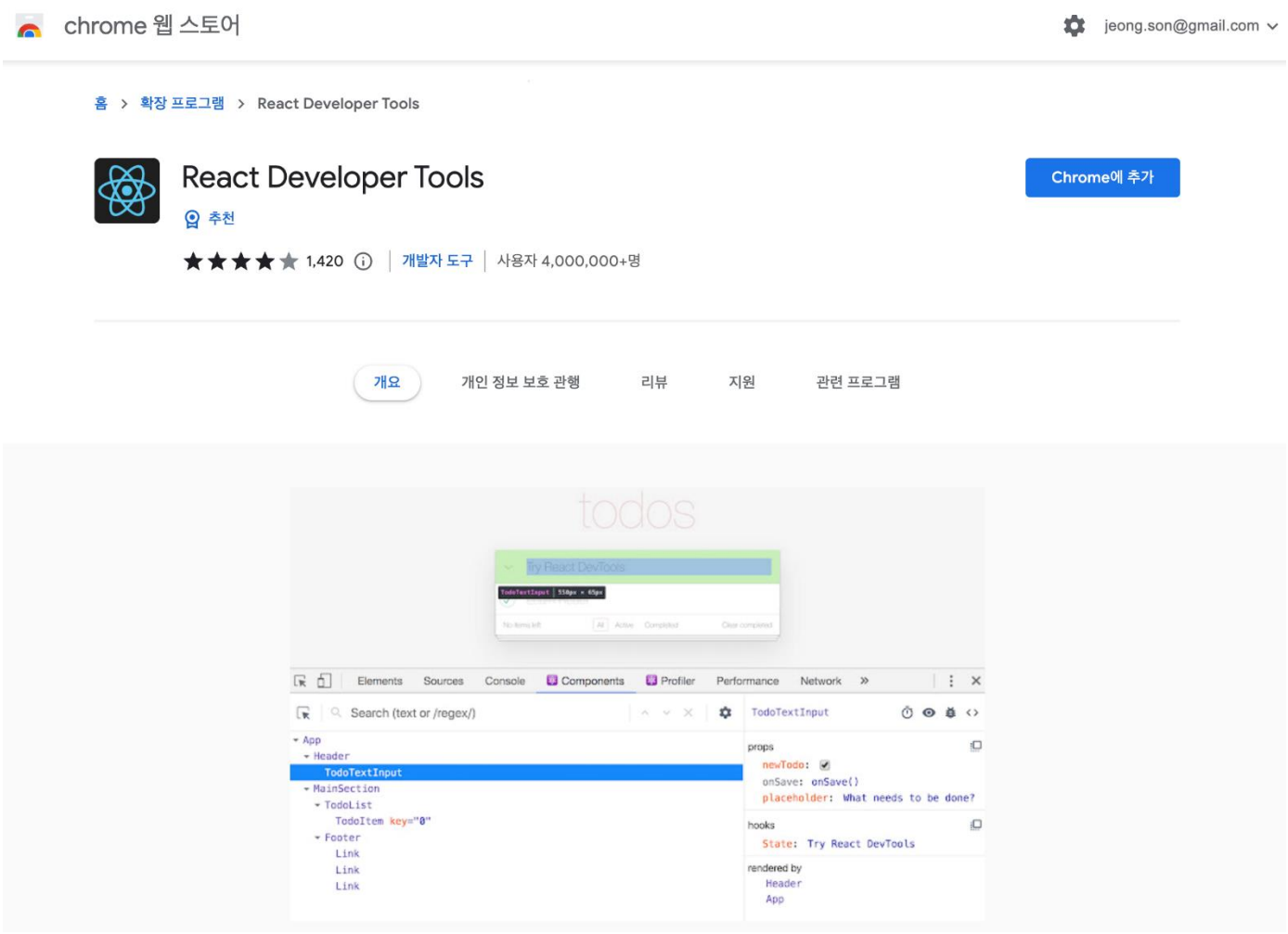
PRACTICE 6.3 State와 생명주기 함수 사용하기

- index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

import NotificationList from './06/NotificationList';

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

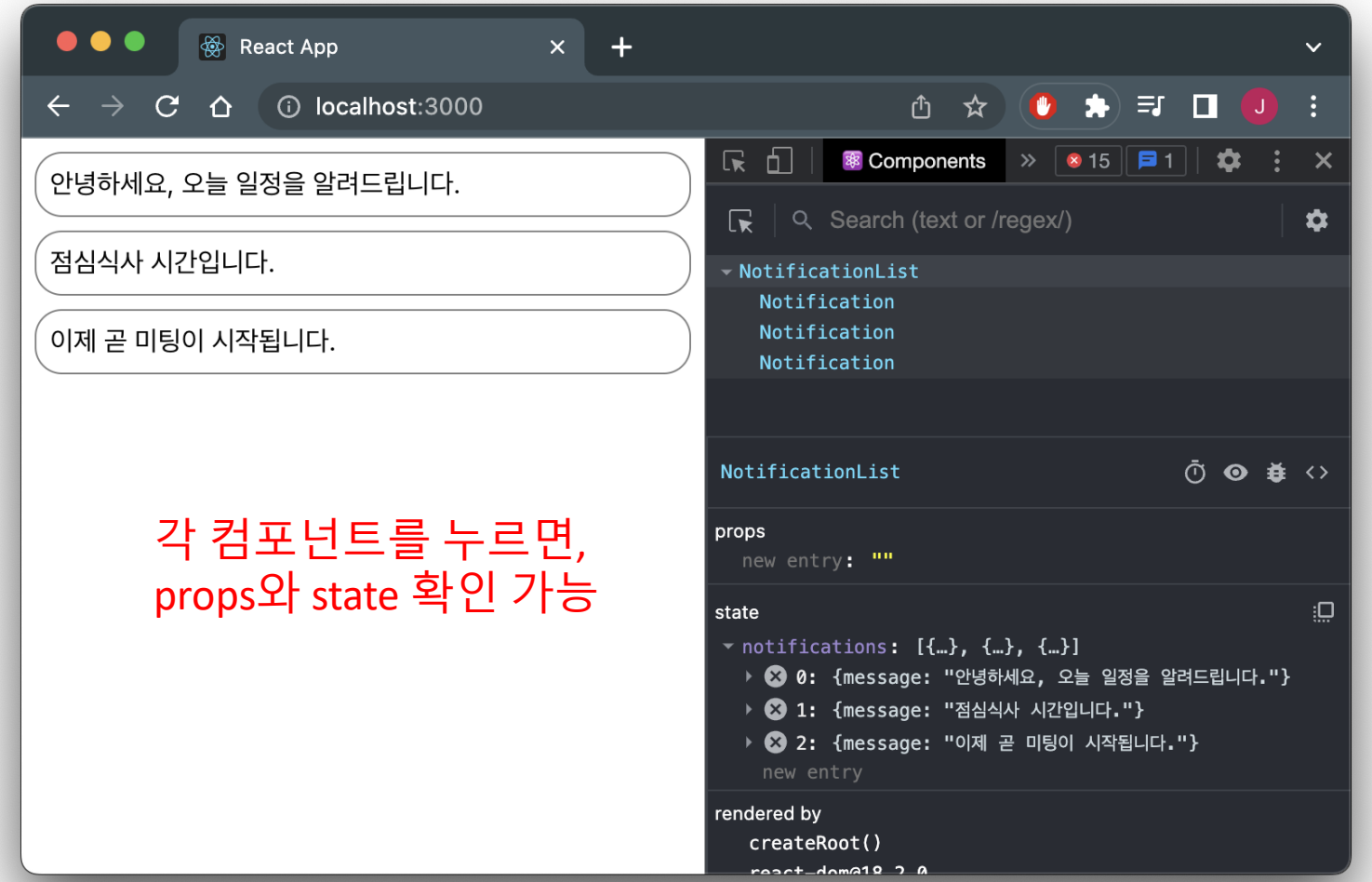
PRACTICE 6.3 State와 생명주기 함수 사용하기

- React Developer Tools 설치



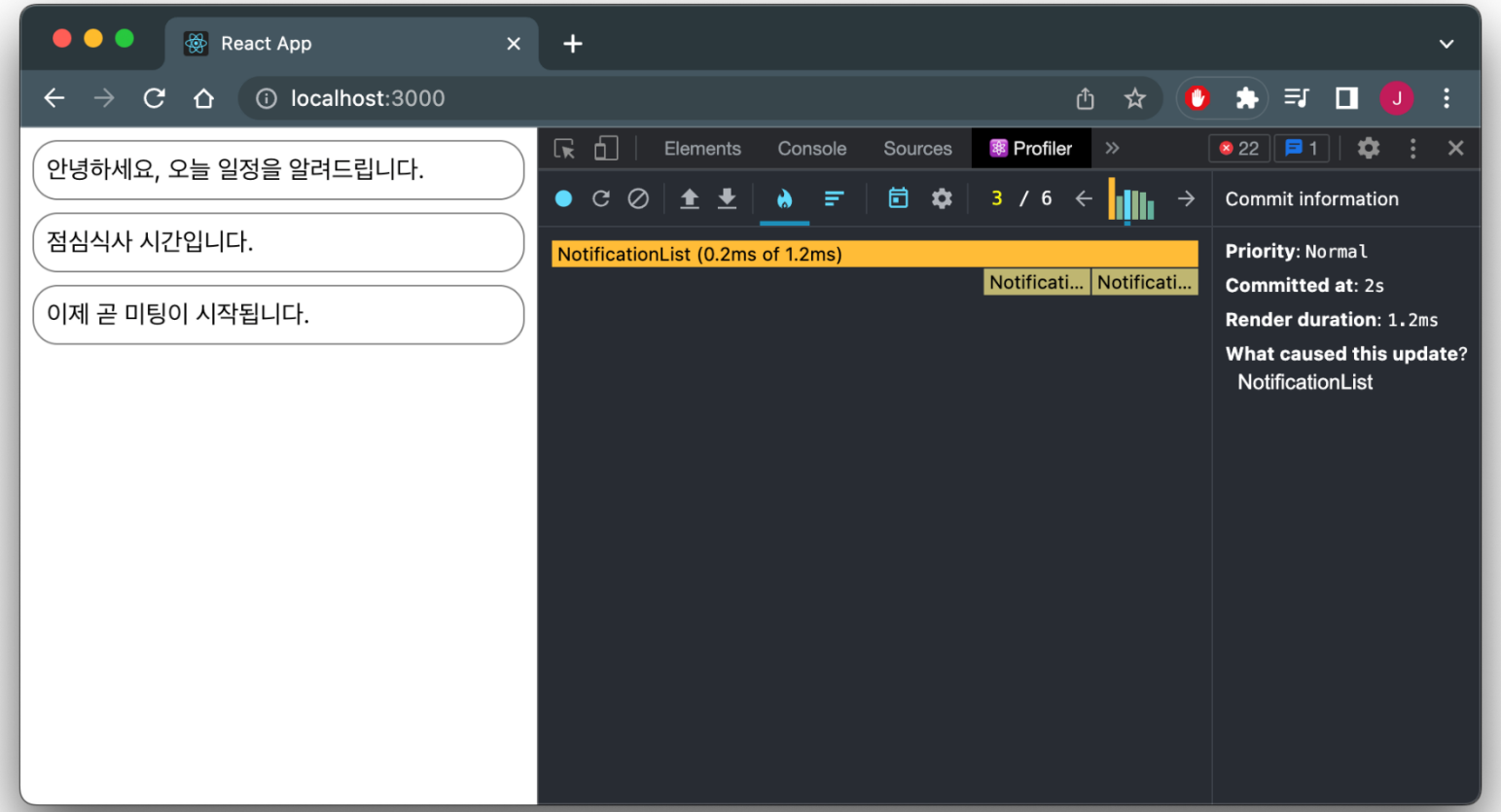
PRACTICE 6.3 State와 생명주기 함수 사용하기

- React Developer Tools 설치
  - 개발자 도구에 Components, Profiler 탭 추가됨
  - Components 탭 - 화면에 존재하는 컴포넌트를 트리형태로 보여줌



PRACTICE 6.3 State와 생명주기 함수 사용하기

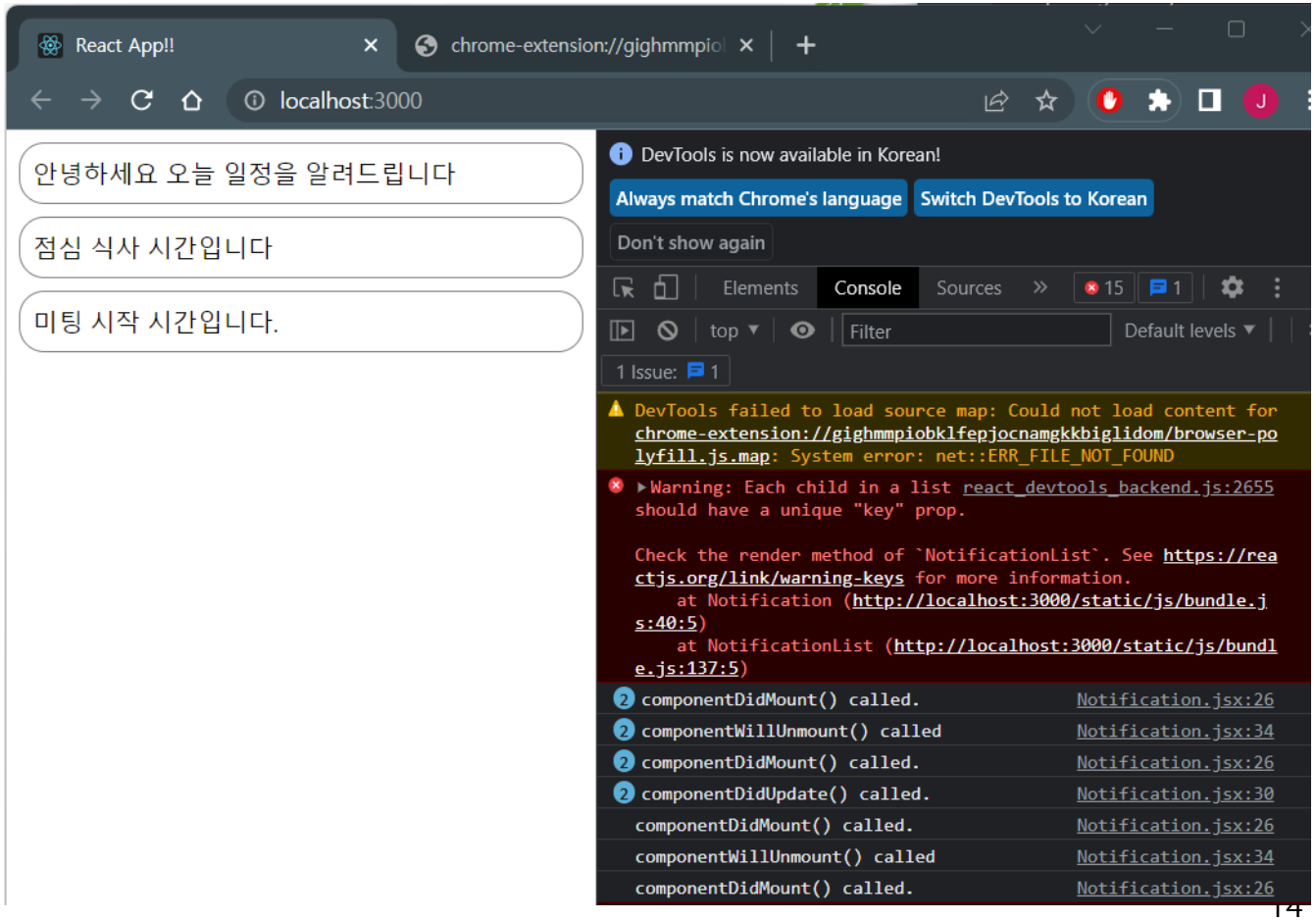
- React Developer Tools 설치
  - Profiler
    - 컴포넌트들이 렌더링되는 과정을 기록하여 단계별로 확인 가능
    - 어떤 컴포넌트가 렌더링되는지, 렌더링 소요 시간 등을 확인하여 최적화



PRACTICE 6.3 State와 생명주기 함수 사용하기

- 생명주기 함수 추가 – Notification.jsx

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

    this.state = {};
  }

  componentDidMount() {
    console.log(`componentDidMount() called.`);
  }

  componentDidUpdate() {
    console.log(`componentDidUpdate() called.`);
  }

  componentWillUnmount() {
    console.log(`componentWillUnmount() called.`);
  }

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



PRACTICE 6.3 State와 생명주기 함수 사용하기

- 생명주기 함수 추가 – props에 key, id 추가

NotificationLIst.jsx

```
const reservedNotifications = [
  {
    id: 1,
    message: "안녕하세요 오늘 일정을 알려드립니다",
  },
  {
    id: 2,
    message: "점심 식사 시간입니다",
  },
  {
    id: 3,
    message: "미팅 시작 시간입니다.",
  },
];

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

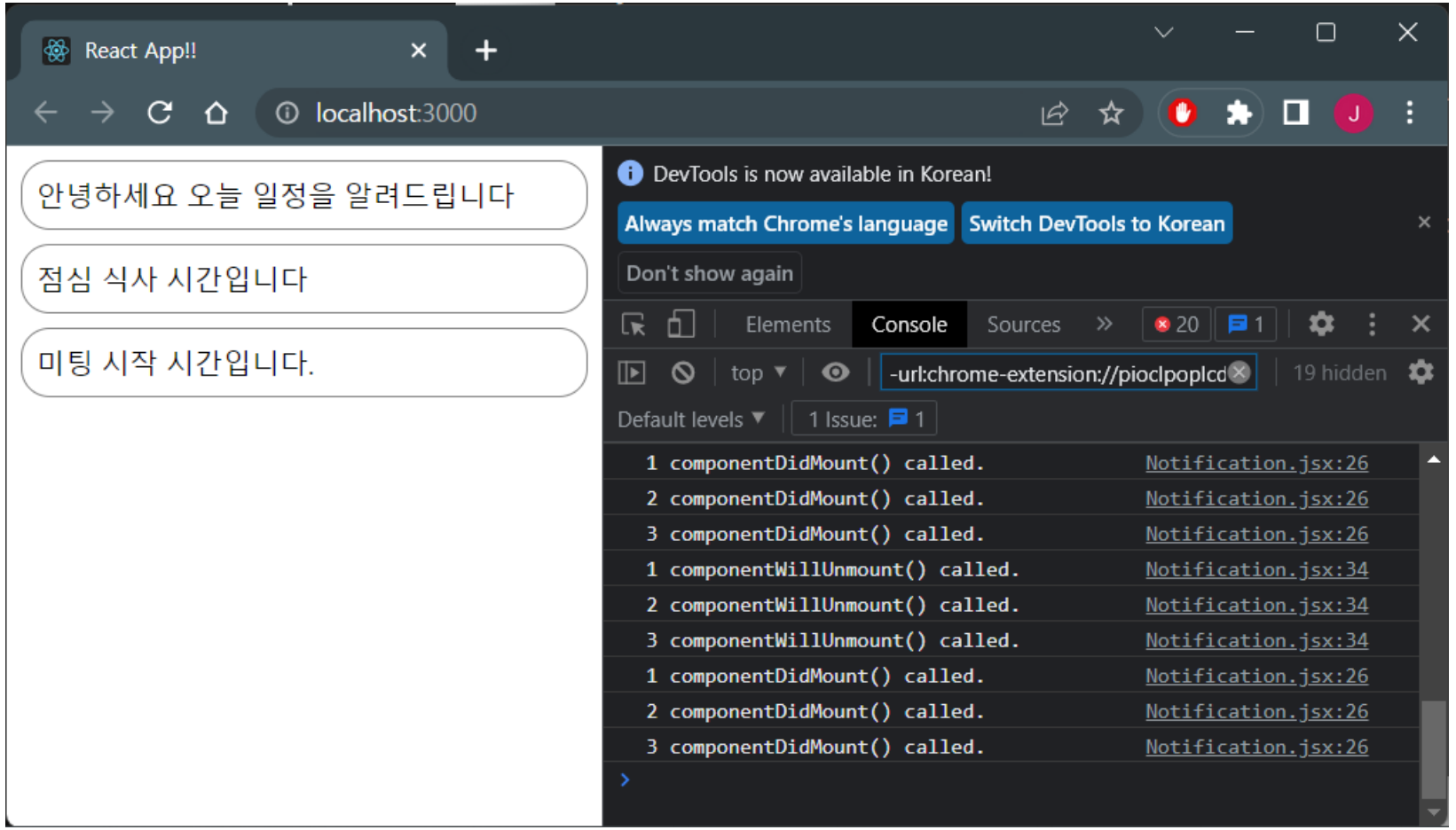
Notification.jsx

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

    this.state = {};
  }

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

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

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

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

PRACTICE 6.3 State와 생명주기 함수 사용하기

- 생명주기 함수 추가 – props에 key, id 추가

실행 결과



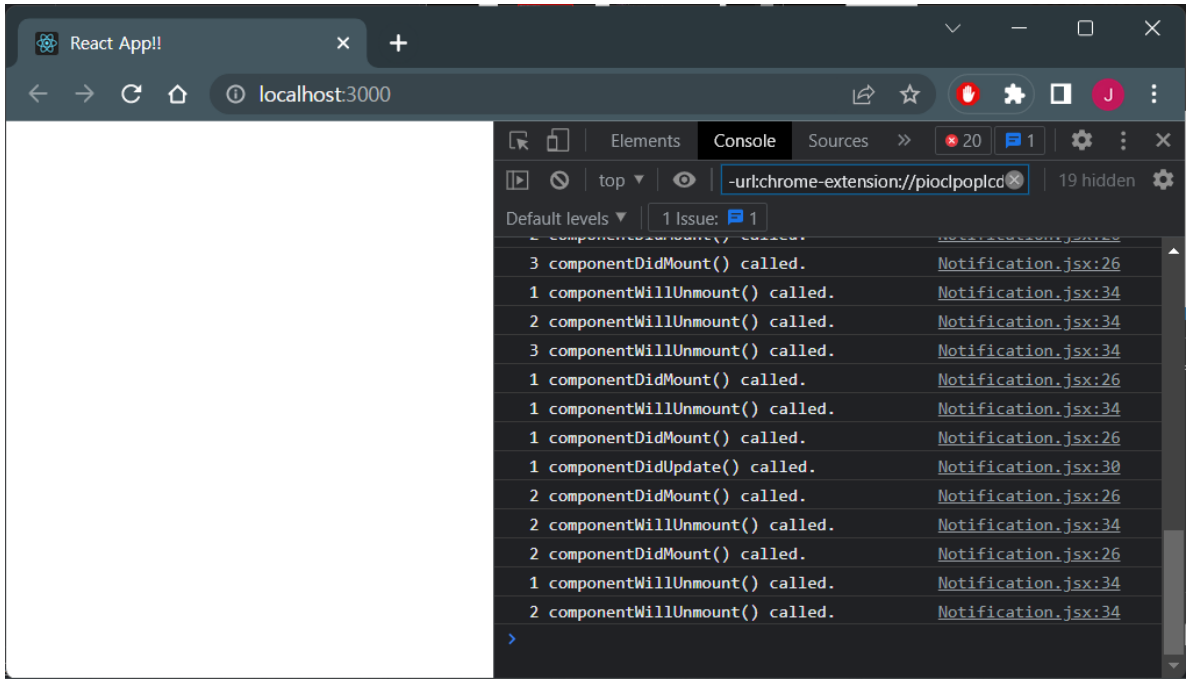
PRACTICE 6.3 State와 생명주기 함수 사용하기

- 생명주기 함수 추가 – 알림이 끝나면 notification 비우기

NotificationLIst.jsx

```
componentDidMount() {
  const { notifications } = this.state;
  timer = setInterval(() => {
    if (notifications.length < reservedNotifications.length) {
      const index = notifications.length;
      notifications.push(reservedNotifications[index]);
      this.setState({
        notifications: notifications,
      });
    } else {
      this.setState({
        notifications: [],
      });
      clearInterval(timer);
    }
  }, 1000);
}
```

실행 결과



[요약]

- State
  - State란?
    - 리액트 컴포넌트의 변경 가능한 데이터
    - 컴포넌트를 개발하는 개발자가 직접 정의해서 사용
    - state가 변경될 경우 컴포넌트가 재랜더링됨
    - 랜더링이나 데이터 흐름에 사용되는 값만 state에 포함시켜야 함
  - State의 특징
    - 자바스크립트 객체 형태로 존재
    - 직접적인 변경이 불가능 함
    - 클래스 컴포넌트
      - 생성자에서 모든 state를 한번에 정의
      - state를 변경할 때는 setState( ) 함수를 사용해야 함

[요약]

- State
  - 함수 컴포넌트
    - useState() 훅을 사용하여 각각의 state를 정의
    - 각 state별로 주어지는 set함수를 사용하여 state 값을 변경

[요약]

- 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 시간의 흐름에 따라 생성되고 업데이트되다가 사라지는 과정을 겪음