**6. Update: 할 일 수정하기**

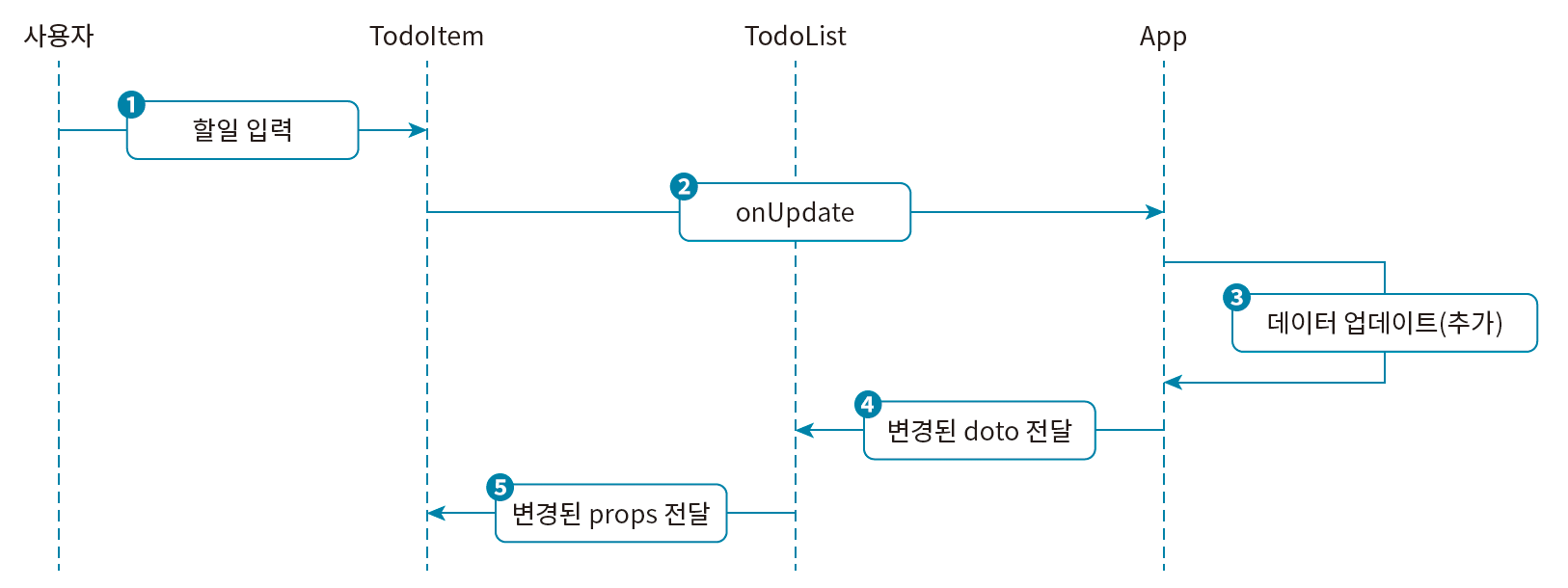
📄

**Table Of Contents**

CRUD의 세 번째 기능은 Update입니 다. [할 일 관리] 앱에서 수정 기능을 만 들겠습니다.

**기능 흐름 살펴보기**

[할 일 관리] 앱에서 할 일 아이템의 수정은 다음 그림과 같이 진행됩니다. 사용자가 TodoItem의 체크박스에 틱(Tik, 체크 표시하는 것)하면 할 일 아이템이 미완료 에서 완료, 완료에서 미완료 상태로 바뀌는 토글 기능이 동작합니다. 이를 위해 다음과 같은 일련의 과정이 필요합니다.

[할 일 관리] 앱의 Update 기능 흐름

1. 사용자가 TodoItem의 체크박스에 틱(체크 표시)합니다.
2. TodoItem 컴포넌트는 함수 onUpdate를 호출하고 어떤 체크박스에 틱이 발생했는지 해당 아이템의 id를 인수로 전달합니다. 물론 그 전에 함수 onUpdate를 App 컴포넌트에서 Props로 TodoItem에 전달해야 합니다.
3. App 컴포넌트의 함수 onUpdate는 틱이 발생한 아이템의 상태(완료 또는 미완료)를 토글하기 위해 State 값을 업데이트합니다.
4. App 컴포넌트의 State 값이 변경되면 TodoList에 전달하는 Props의 값 또한 변경됩니다.
5. TodoList는 변경된 State 값을 다시 리스트로 렌더링합니다. 결과적으로 수정 사항이 반영됩니다.

**아이템 수정 함수 만들기**

할 일 생성을 위해 함수 onCreate를 만들었듯이 수정을 위해 함수 onUpdate를 만듭니다. 그리고 이 함수를 TodoItem 컴포넌트에 전달해야 합니다.

다음과 같이 App에 할 일 수정 함수 onUpdate를 생성하고 TodoList 컴포넌트에 Props로 전달합니다.

**src/App.js**

**Copy**

(...)

function App() {

(...)

const onUpdate = (targetId) => { ①

setTodo(

todo.map((it) => { ②

if (it.id === targetId) {

return {

...it,

isDone: !it.isDone,

};

} else {

return it;

}

})

);

};

return (

<div className="App">

<Header />

<TodoEditor onCreate={onCreate} />

<TodoList todo={todo} onUpdate={onUpdate} /> ③

</div>

);

}

export default App;

**①** 함수 onUpdate는 TodoItem 체크박스에 틱이 발생했을 때 호출하는 함수입니다. 그런데 어떤 아이템에 틱이 발생했는지 알아야 합니다. 매개변수 targetId로 틱이 발생한 할 일 아이템의 id를 저장합니다.

**②** todo 값을 업데이트하기 위해 함수 setTodo를 호출합니다. 이때 map 메서드를 이용해 배열 todo 에서 id가 targetId와 일치하는 요소를 찾으면, isDone 프로퍼티 값을 토글한 새 배열을 만들어 인수로 전달합니다.

**③** TodoList 컴포넌트에 Props로 함수 onUpdate를 전달합니다.

삼항 연산자를 이용하면 함수 onUpdate를 다음과 같이 훨씬 간결하게 작성할 수 있습니다.

**src/App.js**

**Copy**

(...)

const onUpdate = (targetId) => {

setTodo(

todo.map((it) =>

it.id === targetId ? { ...it, isDone: !it.isDone } : it

)

);

};

(...)

할 일 아이템을 수정하는 함수 onUpdate를 완성했습니다.

이제 TodoList에서 TodoItem 컴포넌트에 함수 onUpdate를 전달해야 합니다. TodoList 컴포넌트를 다음과 같이 수정합니다.

**src/component/TodoList.js**

**Copy**

(...)

const TodoList = ({ todo, onUpdate }) => { ①

(...)

return (

<div className="TodoList">

(...)

<div className="list\_wrapper">

{getSearchResult().map((it) => (

<TodoItem key={it.id} {...it} onUpdate={onUpdate} /> ②

))}

</div>

</div>

);

};

export default TodoList;

**①** Props를 구조 분해 할당합니다.

**②** TodoItem 컴포넌트에 함수 onUpdate를 Props로 전달합니다.

리액트 컴포넌트는 바로 한 단계 아래의 자식 컴포넌트에만 데이터를 전달할 수 있습니다. 따라서 한 단계 이상 떨어져 있는 자식 컴포넌트에 데이터를 전달하려면, 현재로서는 전달에 전달을 반복하는 수밖에 없습니다.

따라서 TodoList 자신은 해당 함수를 사용하지 않지만, TodoItem 컴포넌트에 함수 onUpdate를 전달해야 하므로 Props로 받아 다시 전달하는 일종의 매개 역할을 수행합니다.

이는 리액트에서 State와 Props를 사용할 때 흔히 발생하는 일입니다. 이런 상황을 “Props가 마치 땅을 파고 내려가는 것 같다”라고 하여 ‘Props Drilling’ 이라고 합니다.

Props Drilling은 좋은 구현 방식은 아닙니다만 이것에 대한 해결 방법은 추후 다룰 예정이니 지금은 이 코드가 비효율적이라도 일단 작성합니다.

**TodoItem 컴포넌트에서 아이템 수정 함수 호출하기**

이제 TodoItem 컴포넌트에서 틱 이벤트가 발생하면 함수 onUpdate를 호출합니다.

TodoItem을 다음과 같이 수정합니다.

**src/component/TodoItem.js**

**Copy**

import "./TodoItem.css";

const TodoItem = ({ id, content, isDone, createdDate, onUpdate }) => { ①

const onChangeCheckbox = () => { ②

onUpdate(id);

};

return (

<div className="TodoItem">

<div className="checkbox\_col">

<input onChange={onChangeCheckbox} *// ③*

checked={isDone} type="checkbox" />

</div>

<div className="title\_col">{content}</div>

<div className="date\_col">

{new Date(createdDate).toLocaleDateString()}

</div>

<div className="btn\_col">

<button>삭제</button>

</div>

</div>

);

};

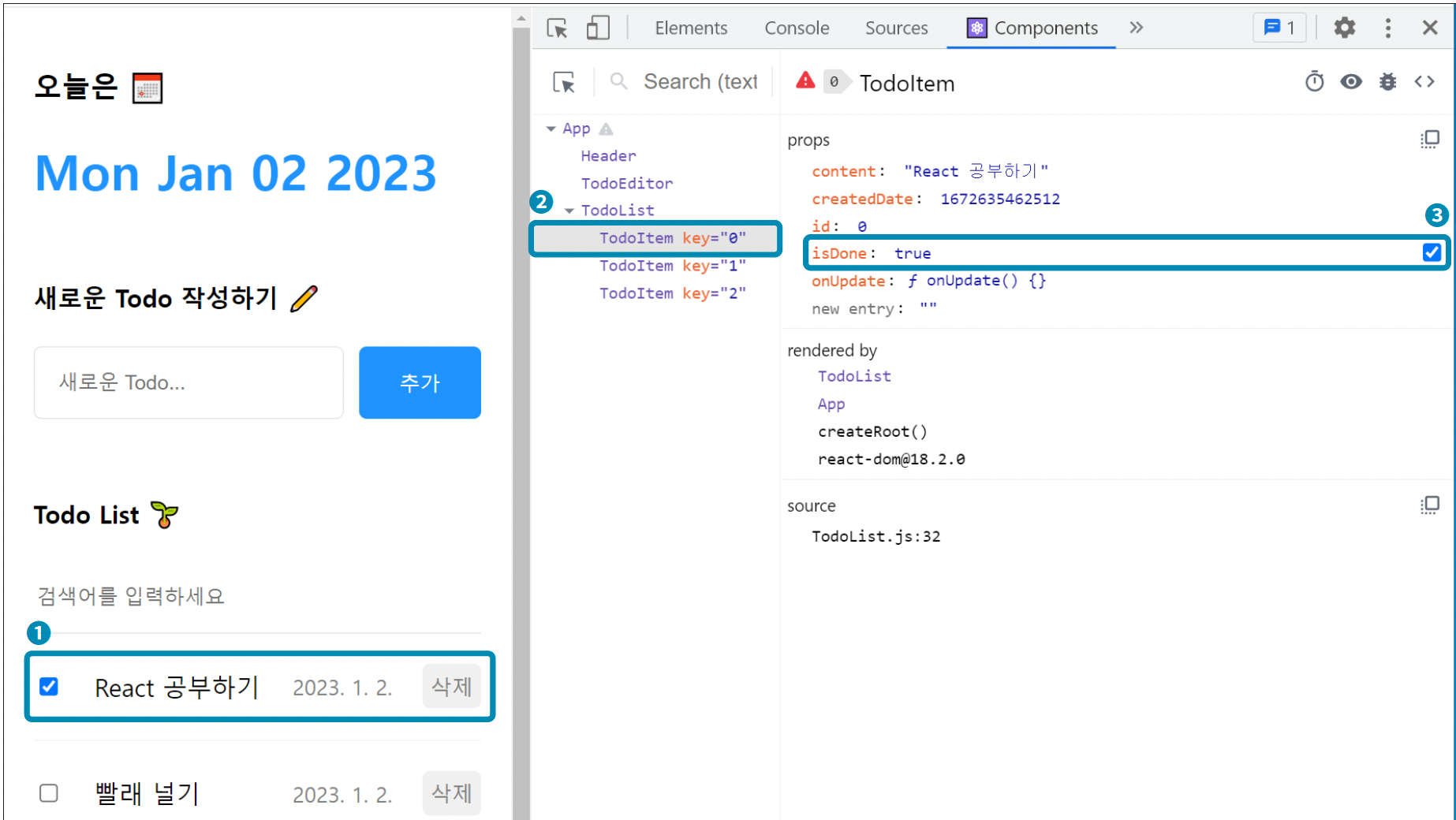
export default TodoItem;

**①** Props를 구조 분해 할당합니다. 함수 onUpdate를 추가합니다.

**②** 체크박스를 틱 했을때 호출할 함수 onChangeCheckbox를 만듭니다. 이 함수는 onUpdate를 호출 하고 인수로 현재 틱이 발생한 할 일 아이템의 id를 전달합니다.

**③** 체크박스 입력 폼의 onChange 이벤트 핸들러를 함수 onChangeCheckbox로 설정합니다.

코드 작성을 모두 완료했다면 결과를 확인합니다. ‘React 공부하기’의 체크박스를 틱 했을 때 완료 여부를 표시하는 체크 표시가 나타나는지 확인합니다. 그리고 [Components]탭을 열고 TodoItem 컴포넌트에서 이 아이템의 isDone 프로퍼티가 true로 변경되는지도 확인합니다.

할 일 완료 여부를 확인하는 체크박스 틱하기

정상적으로 업데이트된 것을 확인할 수 있습니다.