useReducer

useReducer 는 컴포넌트에 reducer를 추가하는 React Hook입니다.

```
const [state, dispatch] = useReducer(reducer, initialArg, init?)
```

레퍼런스

useReducer(reducer, initialArg, init?)

useReducer 를 컴포넌트의 최상위에 호출하고, reducer를 이용해 state를 관리합니다.

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

function reducer(state, action) {
  // ...
}

function MyComponent() {
  const [state, dispatch] = useReducer(reducer, { age: 42 });
  // ...
```

아래에서 더 많은 예시를 확인하세요.

매개변수

- `reducer` : state가 어떻게 업데이트 되는지 지정하는 Reducer 함수입니다. Reducer 함수는 반드시 순수 함수여야 하며, State와 Action을 인수로 받아야 하고, 다음 State를 반환해야 합니다. State와 Action에는 모든 데이터 타입이 할당될 수 있습니다.
- `initialArg` : 초기 State가 계산되는 값입니다. 모든 데이터 타입이 할당될 수 있습니다. 초기 State가 어떻게 계산되는지는 다음 `init` 인수에 따라 달라집니다.
- **선택사항** `init` : 초기 State를 반환하는 초기화 함수입니다. 이 함수가 인수에 할당되지 않으면 초기 State는 `initialArg` 로 설정됩니다. 할당되었다면 초기 State는 `init(initialArg)` 를 호출한 결과가 할당됩니다.

반환값

`useReducer` 는 2개의 엘리먼트로 구성된 배열을 반환합니다.

1. 현재 state. 첫번째 렌더링에서의 state는 `init(initialArg)` 또는 `initialArg` 로 설정됩니다 (`init` 이 없을 경우 `initialArg` 로 설정됩니다).
2. `dispatch` 함수. `dispatch` 는 state를 새로운 값으로 업데이트하고 리렌더링을 일으킵니다.

주의 사항

- `useReducer` 는 Hook이므로 **컴포넌트의 최상위** 또는 커스텀 Hook에서만 호출할 수 있습니다. 반복문이나 조건문에서는 사용할 수 없습니다. 필요한 경우 새로운 컴포넌트를 추출하고 해당 컴포넌트로 State를 옮겨서 사용할 수 있습니다.
- `dispatch` 함수는 안정된 식별성을 가지고 있기 때문에, 흔히 Effect의 의존성에서 제외하는 것을 볼 수 있습니다. 하지만 포함해도 Effect를 실행하지는 않습니다. 린터에서 의존성을 생

략해도 오류가 발생하지 않는다면, 그렇게 하는 것이 안전합니다. Effect 의존성 제거에 대해 자세히 알아보세요.

- Strict Mode에서는 우연한 비순수성을 찾아내기 위해 Reducer와 `init` 함수를 두번 호출합니다. 개발 환경에서만 한정된 동작이며, 배포Production 환경에는 영향을 미치지 않습니다. Reducer와 `init` 함수가 순수 함수라면(그래야만 하듯이) 로직에 어떠한 영향도 미치지 않습니다. 호출 중 하나의 결과는 무시합니다.

`dispatch` 함수

`useReducer` 에 의해 반환되는 `dispatch` 함수는 state를 새로운 값으로 업데이트하고 리렌더링을 일으킵니다. `dispatch` 의 유일한 인수는 action입니다.

```
const [state, dispatch] = useReducer(reducer, { age: 42 });

function handleClick() {
  dispatch({ type: 'incremented_age' });
  // ...
```

React는 현재 `state` 와 `dispatch` 를 통해 전달된 action을 제공받아 호출된 `reducer` 의 반환값을 통해 다음 state값을 설정합니다.

매개변수

- `action` : 사용자에 의해 수행된 활동입니다. 모든 데이터 타입이 할당될 수 있습니다. 컨벤션에 의해 action은 일반적으로 action을 정의하는 `type` 프로퍼티와 추가적인 정보를 표현하는 기타 프로퍼티를 포함한 객체로 구성됩니다.

반환값

`dispatch` 함수는 어떤 값도 반환하지 않습니다.

주의 사항

- `dispatch` 함수는 **오직 *다음* 렌더링에 사용할 state 변수만 업데이트 합니다.** 만약 `dispatch` 함수를 호출한 직후에 state 변수를 읽는다면 호출 이전의 최신화되지 않은 값을 참조할 것입니다.

- `Object.is` 비교를 통해 새롭게 제공된 값과 현재 `state` 를 비교한 값이 같을 경우, React는 컴포넌트와 해당 컴포넌트의 자식 요소들의 리렌더링을 건너뜁니다. 이것은 최적화에 관련된 동작으로써 결과를 무시하기 전에 컴포넌트가 호출되지만, 호출된 결과가 코드에 영향을 미치지는 않습니다.
- React는 state의 업데이트를 batch합니다. **이벤트 핸들러의 모든 코드가 수행**되고 `set` 함수가 모두 호출된 후에 화면을 업데이트 합니다. 이는 하나의 이벤트에 리렌더링이 여러번 일어나는 것을 방지합니다. DOM 접근 등 이른 화면 업데이트를 강제해야할 특수한 상황이 있을 경우 `flushSync` 를 사용할 수 있습니다.

사용법

컴포넌트에 reducer 추가하기

state를 reducer로 관리하기 위해 `useReducer` 를 컴포넌트의 최상단에서 호출합니다.

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

function reducer(state, action) {
  // ...
}

function MyComponent() {
  const [state, dispatch] = useReducer(reducer, { age: 42 });
  // ...
```

`useReducer` 는 정확히 2개의 항목이 포함된 배열 반환합니다.

1. state 변수의 현재 state . 최초에는 사용자가 제공한 초기 state 로 초기화됩니다.
2. `dispatch` 함수 . 상호작용에 대응하여 state를 변경합니다.

화면을 업데이트하려면 사용자가 수행한 활동을 의미하는 *action* 객체를 인수로하여 `dispatch` 함수를 호출하세요.

```
function handleClick() {
  dispatch({ type: 'incremented_age' });
```

```
}
```

React는 현재 state와 action을 reducer 함수 로 전달합니다. reducer는 다음 state를 계산한 후 반환합니다. React는 다음 state를 저장한 뒤에 컴포넌트와 함께 렌더링 하고 UI를 업데이트 합니다.

App.js

다운로드 새로고침 Clear 포크

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

function reducer(state, action) {
  if (action.type === 'incremented_age') {
    return {
      age: state.age + 1
    };
  }
  throw Error('Unknown action.');
}

export default function Counter() {
```

자세히 보기

useReducer 는 useState 와 매우 유사하지만, state 업데이트 로직을 이벤트 핸들러에서 컴포넌트 외부의 단일함수로 분리할 수 있다는 차이점이 있습니다. 자세한 사항은 useState 와 useReducer 비교하기를 읽어보세요.

---

reducer 함수 작성하기

reducer 함수는 아래와 같이 선언합니다.

```
function reducer(state, action) {
  // ...
}
```

이후 다음 state를 계산할 코드를 작성하고, 계산된 state를 반환합니다. 보통은 컨벤션에 따라 switch 문을 사용합니다. switch 는 각 case 를 이용해 다음 state를 계산하고 반환합니다.

```
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      return {
        name: state.name,
        age: state.age + 1
      };
    }
    case 'changed_name': {
      return {
        name: action.nextName,
        age: state.age
      };
    }
  }
  throw Error('Unknown action: ' + action.type);
}
```

Actions은 다양한 형태가 될 수 있습니다. 하지만 컨벤션에 따라 액션이 무엇인지 정의하는 type 프로퍼티를 포함한 객체로 선언하는 것이 일반적입니다. type 은 reducer가 다음 state를 계산하

는데 필요한 최소한의 정보를 포함해야 합니다.

```
function Form() {
  const [state, dispatch] = useReducer(reducer, { name: 'Taylor', age: 42 });

  function handleButtonClick() {
    dispatch({ type: 'incremented_age' });
  }

  function handleInputChange(e) {
    dispatch({
      type: 'changed_name',
      nextName: e.target.value
    });
  }
  // ...
```

action type 이름은 컴포넌트 내에서 지역적입니다. 각 action은 단일 상호작용을 설명하며, 데이터에 여러 변경 사항을 초래하더라도 하나의 상호작용만을 나타냅니다. state의 형태는 임의적이지만, 일반적으로 객체나 배열일 것입니다.

자세한 내용은 state 로직을 reducer로 작성하기를 읽어보세요.

주의하세요!

state는 읽기 전용입니다. state의 객체나 배열을 변경하지 마세요!

```
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      // 🚩 Don't mutate an object in state like this:
      state.age = state.age + 1;
      return state;
    }
```

대신 reducer에서 새로운 객체를 반환하세요.

```
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      // ✅ Instead, return a new object
      return {
        ...state,
        age: state.age + 1
      };
    }
```

자세한 내용을 공부하시려면 객체 State 업데이트하기와 배열 State 업데이트하기를 읽어보세요.

기본적인 useReducer 예시

1. 폼 (객체) 2. 투두 리스트 (배열) 3. Immer를 이용해 업데이트 로직을 보다 간결하거

예시 1 of 3: 폼 (객체)

이 예시에서는 reducer를 이용해 `name` 과 `age` 필드를 가진 객체를 state로 관리합니다.

App.js 다운로드 새로고침 Clear 포크

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

function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      return {
        name: state.name,
        age: state.age + 1
      };
    }
    case 'changed_name': {
      return {
```

자세히 보기

다음 예시

초기 state 재생성 방지하기

React는 초기 state를 저장한 후, 다음 렌더링에서는 이를 무시합니다.

```
function createInitialState(username) {
  // ...
}

function TodoList({ username }) {
  const [state, dispatch] = useReducer(reducer, createInitialState(username));
  // ...
```

`createInitialState(username)` 의 반환값이 초기 렌더링에만 사용되더라도 함수는 매 렌더링마다 호출될 것입니다. 함수가 큰 배열이나 무거운 연산을 다룰 경우에는 성능상 낭비가 될 수 있습니다.

이를 해결하기 위한 방법으로는 `useReducer` 의 3번째 인수에 ***초기화 함수*** **를 전달하는 방법**이 있습니다.

```
function createInitialState(username) {
  // ...
}

function TodoList({ username }) {
  const [state, dispatch] = useReducer(reducer, username, createInitialState);
  // ...
```

`createInitialState()` 처럼 함수를 호출해서 전달하는 것이 아니라, `createInitialState` *함수 자체*를 전달해야 한다는 것을 기억하세요. 이 방법을 이용하면 초기화 이후에 초기 state가 다시 생성되는 일은 발생하지 않습니다.

위의 예시에서는 `createInitialState` 함수가 `username` 을 인수로 받습니다. 만약 초기화 함수가 초기 state를 계산하는 것에 어떤 인수도 필요하지 않다면, `useReducer` 의 두번째 인수에 `null` 을 전달할 수 있습니다.

초기화 함수를 전달하는 것과 초기 state를 직접 전달하는 것의 차이점

1. 초기화 함수 전달
2. 초기 state 직접 전달

예시 1 of 2: 초기화 함수 전달

이 예시에서는 초기화 단계에서만 동작하는 함수인 `createInitialState` 를 초기화 함수로 전달합니다. 이 함수는 인풋에 입력 할 때 발생하는 리렌더링 상황 등에서는 호출되지 않습니다.

TodoList.js

새로고침 Clear 포크

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

function createInitialState(username) {
  const initialTodos = [];
  for (let i = 0; i < 50; i++) {
```

```
      initialTodos.push({
        id: i,
        text: username + "'s task #" + (i + 1)
      });
    }
    return {
      draft: ''
```

자세히 보기

다음 예시

트러블 슈팅

dispatch로 action을 호출해도 오래된 state 값이 출력됩니다.

`dispatch` 함수의 호출은 **현재 동작하고 있는 코드의 state를 변경하지 않습니다.**

```
function handleClick() {
  console.log(state.age);  // 42
```

```
  dispatch({ type: 'incremented_age' }); // Request a re-render with 43
  console.log(state.age);  // Still 42!

  setTimeout(() => {
    console.log(state.age); // Also 42!
  }, 5000);
}
```

이러한 현상은 State가 스냅샷으로서 사용되기 때문에 일어납니다. state를 업데이트하면 새로운 state를 이용한 또 다른 렌더링이 요청되지만, 이미 실행중인 이벤트 핸들러 내의 `state` 자바스크립트 변수에는 영향을 미치지 않습니다.

만약 다음 state 값을 알고 싶다면, reducer 함수를 직접 호출해서 다음 값을 계산해볼 수 있습니다.

```
const action = { type: 'incremented_age' };
dispatch(action);

const nextState = reducer(state, action);
console.log(state);     // { age: 42 }
console.log(nextState); // { age: 43 }
```

---

dispatch로 action을 호출해도 화면이 업데이트되지 않습니다.

React는 **이전 state와 다음 state를 비교했을 때, 값이 일치한다면 업데이트가 무시됩니다.** 비교는 `Object.is` 를 통해 이루어집니다. 이런 현상은 보통 객체나 배열의 state를 직접적으로 수정했을 때 발생합니다.

```
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      // 🚩 Wrong: mutating existing object
      state.age++;
      return state;
    }
    case 'changed_name': {
```

```
      // 🚩 Wrong: mutating existing object
      state.name = action.nextName;
      return state;
    }
    // ...
  }
}
```

React는 기존의 `state` 객체가 mutation된 상태로 반환된다면 업데이트를 무시합니다. 이러한 현상을 방지하기 위해서는 객체나 배열을 mutation시키지 않고 객체 state를 변경하거나 배열 state를 변경해야 합니다.

```
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      // ✅ Correct: creating a new object
      return {
        ...state,
        age: state.age + 1
      };
    }
    case 'changed_name': {
      // ✅ Correct: creating a new object
      return {
        ...state,
        name: action.nextName
      };
    }
    // ...
  }
}
```

---

reducer의 state 일부가 dispatch된 이후에 undefined가 할당됩니다.

각각의 `case` 가 **새로운 state를 반환할 때 기존에 있던 필드를 모두 복사하는지** 확인해보세요.

```
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      return {
        ...state, // Don't forget this!
        age: state.age + 1
      };
    }
    // ...
```

위의 코드에서는 `...state` 가 없다면 다음 state는 오로지 `age` 필드만 포함하거나, 아무것도 포함하지 않을 것입니다.

---

reducer의 모든 state가 dispatch가 이루어 진 후 undefined가 할당됩니다.

state에 예기치 않은 `undefined` 가 할당되고 있다면 case 중 하나에 `return` 이 누락되었거나 action의 타입이 `case` 와 짝지어지지 않았을 수 있습니다. 이유를 찾기 위해 switch문 밖에서 에러를 throw 할 수 있습니다.

```
function reducer(state, action) {
  switch (action.type) {
    case 'incremented_age': {
      // ...
    }
    case 'edited_name': {
      // ...
    }
  }
  throw Error('Unknown action: ' + action.type);
}
```

이 외에도 실수를 방지하기 위해 타입스크립트 같은 정적 타입 체커를 사용할 수 있습니다.

”Too many re-renders” 오류가 발생합니다.

`Too many re-renders. React limits the number of renders to prevent an infinite loop.` 라는 에러 메세지를 받을 수 있습니다. 일반적으로는 렌더링 과정에서 dispatch가 실행될 때 이러한 일이 일어납니다. 렌더링은 dispatch를 야기하고, dispatch는 렌더링을 야기하므로 렌더링 무한 루프가 일어납니다. 이러한 상황은 이벤트 핸들러를 잘못 호출할 때 종종 발생합니다.

```
// 🚩 Wrong: calls the handler during render
return <button onClick={handleClick()}>Click me</button>

// ✅ Correct: passes down the event handler
return <button onClick={handleClick}>Click me</button>

// ✅ Correct: passes down an inline function
return <button onClick={(e) => handleClick(e)}>Click me</button>
```

오류의 원인을 찾을 수 없는 경우에는 어느 `dispatch` 함수에서 에러가 생성되는지 확인하기 위해 콘솔창의 오류 옆에 있는 화살표를 클릭한 후 자바스크립트 스택을 찾아보세요.

---

reducer와 초기화 함수가 두번 호출됩니다.

React는 엄격 모드일 때 reducer와 초기화 함수를 두번씩 호출합니다. 이 현상은 코드 실행에 문제가 되지 않습니다.

이러한 현상은 컴포넌트가 순수함수로 유지될 수 있도록 오직 개발 환경에서만 일어나며, 두개의 호출 중 하나는 무시됩니다. 컴포넌트, 초기화 함수, reducer가 순수하다면 로직에 아무런 영향을 미치지 않지만, 순수하지 않다면 실수를 알아챌 수 있도록 알려줍니다.

예시로, 아래의 순수하지 않은 reducer 함수는 state 배열에 mutation을 일으키고 있습니다.

```
function reducer(state, action) {
  switch (action.type) {
    case 'added_todo': {
      // 🚩 Mistake: mutating state
      state.todos.push({ id: nextId++, text: action.text });
```

```
      return state;
    }
    // ...
  }
}
```

React는 reducer 함수를 두 번 호출하므로 todo가 두 개 추가되는 것을 볼 수 있고, 이를 통해 reducer 함수 작성에 실수가 있다는 것을 알아낼 수 있습니다. 이러한 실수는 배열을 mutation 하지 않고 교체하는 방법을 통해 수정할 수 있습니다.

```
function reducer(state, action) {
  switch (action.type) {
    case 'added_todo': {
      // ✅ Correct: replacing with new state
      return {
        ...state,
        todos: [
          ...state.todos,
          { id: nextId++, text: action.text }
        ]
      };
    }
    // ...
  }
}
```

이제 reducer 함수는 순수하므로, 여러번 호출되어도 같은 값을 보장할 수 있습니다. React는 순수성을 보장하기 위해 개발 환경에서 두번씩 호출합니다. **오로지 컴포넌트와 초기화 함수, reducer 함수만 순수할 필요가 있습니다.** 이벤트 핸들러는 순수할 필요가 없습니다. 따라서 이벤트 핸들러는 두 번씩 호출되지 않습니다.

자세한 사항은 컴포넌트를 순수하게 유지하기를 읽어보세요.

이전
useOptimistic

다음
useRef

React 학습하기

빠르게 시작하기

설치하기

UI 표현하기

상호작용성 더하기

State 관리하기

탈출구

API 참고서

React APIs

React DOM APIs

커뮤니티

행동 강령

팀 소개

문서 기여자

감사의 말

더 보기

블로그

React Native

개인 정보 보호

약관