useSyncExternalStore

useSyncExternalStore 는 외부 store를 구독할 수 있는 React Hook입니다.

```
const snapshot = useSyncExternalStore(subscribe, getSnapshot, getServerSn
```


- 레퍼런스
  - useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot?)
- 사용법
  - 외부 store 구독
  - 브라우저 API 구독
  - custom Hook으로 로직 추출하기
  - 서버 렌더링 지원 추가
- 트러블 슈팅
  - 오류가 발생했습니다: “ getSnapshot 의 결과를 캐시해야 합니다.”
  - 리렌더링할 때마다 subscribe 함수가 호출됩니다.

레퍼런스

useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot?)

컴포넌트의 최상위 레벨에서 useSyncExternalStore 를 호출하여 외부 데이터 저장소에서 값을 읽습니다.

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

```
import { todosStore } from './todoStore.js';

function TodosApp() {
  const todos = useSyncExternalStore(todosStore.subscribe, todosStore.getSnapshot);
  // ...
}
```

store에 있는 데이터의 스냅샷을 반환합니다. 두 개의 함수를 인수로 전달해야 합니다.

1. `subscribe` 함수는 store를 구독하고 구독을 취소하는 함수를 반환해야 합니다.
2. `getSnapshot` 함수는 store에서 데이터의 스냅샷을 읽어야 합니다.

아래 예시 참조

매개변수

- `subscribe` : 하나의 `callback` 인수를 받아 store에 구독하는 함수입니다. store가 변경될 때, 제공된 `callback` 이 호출되어 React가 `getSnapshot` 을 다시 호출하고 (필요한 경우) 컴포넌트를 다시 렌더링하도록 해야 합니다. `subscribe` 함수는 구독을 정리하는 함수를 반환해야 합니다.

- `getSnapshot` : 컴포넌트에 필요한 store 데이터의 스냅샷을 반환하는 함수입니다. store가 변경되지 않은 상태에서 `getSnapshot` 을 반복적으로 호출하면 동일한 값을 반환해야 합니다. 저장소가 변경되어 반환된 값이 다르면 (`Object.is` 와 비교하여) React는 컴포넌트를 리렌더링합니다.

- **optional** `getServerSnapshot` : store에 있는 데이터의 초기 스냅샷을 반환하는 함수입니다. 서버 렌더링 도중과 클라이언트에서 서버 렌더링 된 콘텐츠의 하이드레이션 중에만 사용됩니다. 서버 스냅샷은 클라이언트와 서버 간에 동일해야 하며 일반적으로 직렬화되어 서버에서 클라이언트로 전달됩니다. 이 함수가 제공되지 않으면 서버에서 컴포넌트를 렌더링할 때 오류가 발생합니다.

반환값

렌더링 로직에 사용할 수 있는 store의 현재 스냅샷입니다.

주의 사항

- `getSnapshot` 이 반환하는 store 스냅샷은 불변이어야 합니다. 기본 스토어에 변경 가능한 데이터가 있는 경우 데이터가 변경된 경우 변경 불가능한 새 스냅샷을 반환합니다. 그렇지 않으면 캐시 된 마지막 스냅샷을 반환합니다.

- 리렌더링하는 동안 다른 `subscribe` 함수가 전달되면 React는 새로 전달된 `subscribe` 함수를 사용하여 저장소를 다시 구독합니다. 컴포넌트 외부에서 `subscribe` 를 선언하면 이를 방지할 수 있습니다.

- non-blocking transition 업데이트 중에 스토어가 변경되면, React는 해당 업데이트를 blocking으로 수행하도록 되돌아갑니다. 구체적으로, 모든 Transition 업데이트에 대해 React는 DOM에 변경 사항을 적용하기 직전에 `getSnapshot` 을 한 번 더 호출합니다. 처음 호출했을 때와 다른 값을 반환하면, React는 처음부터 다시 업데이트를 시작하고, 이번에는 blocking 업데이트를 적용하여 화면의 모든 컴포넌트가 같은 스토어 버전을 반영하도록 합니다.

- `useSyncExternalStore` 가 반환한 스토어 값을 기반으로 렌더링을 *일시 중단* 하는 것은 권장하지 않습니다. 그 이유는 외부 스토어에 대한 변형을 non-blocking transition 업데이트로 표시할 수 없기 때문에, 가장 가까운 `Suspense` fallback을 트리거해서, 화면에서 로딩 스피너로 대체하여, 일반적으로 UX가 좋지 않기 때문입니다.

  예를 들어, 다음은 권장되지 않습니다.

```
const LazyProductDetailPage = lazy(() => import('./ProductDetailPage.js'));

function ShoppingApp() {
  const selectedProductId = useSyncExternalStore(...);

  // ❌ `selectedProductId`에 종속된 Promise로 `use`를 호출하는 것
  const data = use(fetchItem(selectedProductId))

  // ❌ `selectedProductId`를 기반으로 지연 컴포넌트를 조건부로 렌더링하는 것
  return selectedProductId != null ? <LazyProductDetailPage /> : <FeaturedProducts
}
```

사용법

외부 store 구독

대부분의 React 컴포넌트는 props, state, 그리고 context에서만 데이터를 읽습니다. 하지만 때로는 컴포넌트가 시간이 지남에 따라 변경되는 React 외부의 일부 저장소에서 일부 데이터를 읽어야 하는 경우가 있습니다. 다음이 포함됩니다.

- React 외부에 state를 보관하는 서드파티 상태 관리 라이브러리.
- 변경 가능한 값을 노출하는 브라우저 API와 그 변경 사항을 구독하는 이벤트.

외부 데이터 저장소에서 값을 읽으려면 컴포넌트의 최상위 레벨에서 `useSyncExternalStore`를 호출하세요.

```
import { useSyncExternalStore } from 'react';
import { todosStore } from './todoStore.js';

function TodosApp() {
  const todos = useSyncExternalStore(todosStore.subscribe, todosStore.getSnapshot);
  // ...
}
```

store에 있는 데이터의 `snapshot`을 반환합니다. 두 개의 함수를 인수로 전달해야 합니다.

1. `subscribe` 함수는 store에 구독하고 구독을 취소하는 함수를 반환해야 합니다.
2. `getSnapshot` 함수는 store에서 데이터의 스냅샷을 읽어야 합니다.

React는 이 함수를 사용해 컴포넌트를 store에 구독한 상태로 유지하고 변경 사항이 있을 때 리렌더링합니다.

예를 들어 아래 샌드박스에서 `todosStore`는 React 외부에 데이터를 저장하는 외부 store로 구현되어 있습니다. `TodosApp` 컴포넌트는 `useSyncExternalStore` Hook으로 해당 외부 store에 연결합니다.

App.js todoStore.js 새로고침 Clear 포크

```
import { useSyncExternalStore } from 'react';
import { todosStore } from './todoStore.js';

export default function TodosApp() {
  const todos = useSyncExternalStore(todosStore.subscribe, todosStore.getSnapshot)
  return (
    <>
      <button onClick={() => todosStore.addTodo()}>Add todo</button>
      <hr />
      <ul>
        {todos.map(todo => (
```

자세히 보기

중요합니다!

가능하면 내장된 React state를 `useState` 및 `useReducer`와 함께 사용하는 것이 좋습니다. `useSyncExternalStore` API는 기존 비 React 코드와 통합해야 할 때 주로 유용합니다.

브라우저 API 구독

`useSyncExternalStore`를 추가하는 또 다른 이유는 시간이 지남에 따라 변경되는 브라우저에 노출되는 일부 값을 구독하려는 경우입니다. 예를 들어 컴포넌트에 네트워크 연결이 활성화되어 있는지 여부를 표시하고 싶다고 가정해 보겠습니다. 브라우저는 `navigator.onLine`.이라는 속성을 통해 이 정보를 노출합니다.

이 값은 시간이 지남에 따라 React가 알지 못하는 사이에 변경될 수 있으므로 `useSyncExternalStore`로 값을 읽어야 합니다.

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

function ChatIndicator() {
  const isOnline = useSyncExternalStore(subscribe, getSnapshot);
  // ...
}
```

`getSnapshot` 함수를 구현하려면 브라우저 API에서 현재 값을 읽습니다.

```
function getSnapshot() {
  return navigator.onLine;
}
```

다음으로 `subscribe` 함수를 구현해야 합니다. 예를 들어 `navigator.onLine` 이 변경되면 브라우저는 `window` 객체에서 `online` 및 `offline` 이벤트를 실행합니다. `callback` 인수를 해당 이벤트에 구독한 다음 구독을 정리하는 함수를 반환해야 합니다.

```
function subscribe(callback) {
  window.addEventListener('online', callback);
  window.addEventListener('offline', callback);
  return () => {
    window.removeEventListener('online', callback);
    window.removeEventListener('offline', callback);
  };
}
```

이제 React는 외부 `navigator.onLine` API에서 값을 읽는 방법과 그 변경 사항을 구독하는 방법을 알고 있습니다. 네트워크에서 디바이스의 연결을 끊어보면 컴포넌트가 응답으로 리렌더링되는 것을 확인할 수 있습니다.

App.js

다운로드 새로고침 Clear 포크

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

export default function ChatIndicator() {
```

```
  const isOnline = useSyncExternalStore(subscribe, getSnapshot);
  return <h1>{isOnline ? '✅ Online' : '❌ Disconnected'}</h1>;
}

function getSnapshot() {
  return navigator.onLine;
}

function subscribe(callback) {
```

자세히 보기

custom Hook으로 로직 추출하기

일반적으로 컴포넌트에서 직접 `useSyncExternalStore`를 작성하지는 않습니다. 대신 일반적으로 custom Hook에서 호출합니다. 이렇게 하면 서로 다른 컴포넌트에서 동일한 외부 저장소를 사용할 수 있습니다.

예를 들어 이 custom `useOnlineStatus` Hook은 네트워크가 온라인 상태인지 여부를 추적합니다.

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

```
export function useOnlineStatus() {
  const isOnline = useSyncExternalStore(subscribe, getSnapshot);
  return isOnline;
}

function getSnapshot() {
  // ...
}

function subscribe(callback) {
  // ...
}
```

이제 다른 컴포넌트에서 기본 구현을 반복하지 않고도 `useOnlineStatus` 를 호출할 수 있습니다.

App.js useOnlineStatus.js 새로고침 Clear 포크

```
import { useOnlineStatus } from './useOnlineStatus.js';

function StatusBar() {
  const isOnline = useOnlineStatus();
  return <h1>{isOnline ? '✅ Online' : '❌ Disconnected'}</h1>;
}

function SaveButton() {
  const isOnline = useOnlineStatus();

  function handleSaveClick() {
    console.log('✅ Progress saved');
```

자세히 보기

서버 렌더링 지원 추가

React 앱이 server rendering을 사용하는 경우 React 컴포넌트는 브라우저 환경 외부에서도 실행되어 초기 HTML을 생성합니다. 이로 인해 외부 store에 연결할 때 몇 가지 문제가 발생합니다.

- 브라우저 전용 API에 연결하는 경우 서버에 해당 API가 존재하지 않으므로 작동하지 않습니다.
- 서드 파티 데이터 저장소에 연결하는 경우 서버와 클라이언트 간에 일치하는 데이터가 필요합니다.

이러한 문제를 해결하려면 `getServerSnapshot` 함수를 `useSyncExternalStore`의 세 번째 인수로 전달하세요.

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

export function useOnlineStatus() {
  const isOnline = useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot);
  return isOnline;
}

function getSnapshot() {
  return navigator.onLine;
}

function getServerSnapshot() {
  return true; // 서버에서 생성된 HTML에는 항상 "Online"을 표시합니다.
}

function subscribe(callback) {
  // ...
}
```

`getServerSnapshot` 함수는 `getSnapshot`과 유사하지만 두 가지 상황에서만 실행됩니다.

- HTML을 생성할 때 서버에서 실행됩니다.
- hydration 중 즉 React가 서버 HTML을 가져와서 인터랙티브하게 만들 때 클라이언트에서 실행됩니다.

이를 통해 앱이 상호작용하기 전에 사용될 초기 스냅샷 값을 제공할 수 있습니다. 서버 렌더링에 의미 있는 초기값이 없다면 컴포넌트가 클라이언트에서만 렌더링되도록 강제 설정할 수 있습니다.

중요합니다!

`getServerSnapshot`이 초기 클라이언트 렌더링에서 서버에서 반환한 것과 동일한 정확한 데이터를 반환하는지 확인하세요. 예를 들어 `getServerSnapshot`이 서버에서 미리 채워진 store 콘텐츠를 반환한 경우 이 콘텐츠를 클라이언트로 전송해야 합니다. 이를 수행하는 일반적인 방법 중 하나는 서버 렌더링 중에 `window.MY_STORE_DATA`와 같은 글로벌을 설정하는 `<script>` 태그를 생성한 다음 클라이언트에서 `getServerSnapshot`에서 해당 글로벌을 읽어오는 것입니다. 외부 스토어에서 이를 수행하는 방법에 대한 지침을 제공해야 합니다.

트러블 슈팅

오류가 발생했습니다: "getSnapshot의 결과를 캐시해야 합니다."

이 오류가 발생하면 `getSnapshot` 함수가 호출될 때마다 새 객체를 반환한다는 의미입니다.

```
function getSnapshot() {
  // 🔴 getSnapshot에서 항상 다른 객체를 반환하지 마세요.
  return {
    todos: myStore.todos
  };
}
```

React는 `getSnapshot` 반환 값이 지난번과 다르면 컴포넌트를 리렌더링합니다. 그렇기 때문에 항상 다른 값을 반환하면 무한 루프에 들어가서 이 오류가 발생합니다.

실제로 변경된 사항이 있는 경우에만 `getSnapshot` 객체가 다른 객체를 반환해야 합니다. store에 변경 불가능한 데이터가 포함된 경우 해당 데이터를 직접 반환할 수 있습니다.

```
function getSnapshot() {
  // ✅ 불변 데이터를 반환할 수 있습니다.
  return myStore.todos;
}
```

store 데이터가 변경 가능한 경우 `getSnapshot` 함수는 해당 데이터의 변경 불가능한 스냅샷을 반환해야 합니다. 즉 새 객체를 생성해야 하지만 매번 호출할 때마다 이 작업을 수행해서는 안 됩니다. 대신 마지막으로 계산된 스냅샷을 저장하고 저장소의 데이터가 변경되지 않은 경우 지난번과 동일한 스냅샷을 반환해야 합니다. 변경 가능한 데이터가 변경되었는지 확인하는 방법은 변경 가능한 저장소가 구현된 방식에 따라 다릅니다.

리렌더링할 때마다 `subscribe` 함수가 호출됩니다.

subscribe 함수는 컴포넌트 내부에 정의되므로 리렌더링할 때마다 달라집니다.

```
function ChatIndicator() {
  // 🚩 항상 다른 함수를 사용하므로 React는 렌더링할 때마다 다시 구독합니다.
  function subscribe() {
    // ...
  }

  const isOnline = useSyncExternalStore(subscribe, getSnapshot);

  // ...
}
```

리렌더링 사이에 다른 `subscribe` 함수를 전달하면 React가 store를 다시 구독합니다. 이로 인해 성능 문제가 발생하고 store 재구독을 피하고 싶다면 `subscribe` 함수를 외부로 이동하세요.

```
// ✅ 항상 동일한 함수이므로 React는 다시 구독할 필요가 없습니다.
function subscribe() {
  // ...
}

function ChatIndicator() {
  const isOnline = useSyncExternalStore(subscribe, getSnapshot);
  // ...
}
```

또는 일부 인수가 변경될 때만 다시 구독하도록 `subscribe` 을 `useCallback` 으로 래핑합니다.

```
function ChatIndicator({ userId }) {
  // ✅ userId가 변경되지 않는 한 동일한 함수입니다.
  const subscribe = useCallback(() => {
    // ...
  }, [userId]);

  const isOnline = useSyncExternalStore(subscribe, getSnapshot);

  // ...
}
```


이전
useState

다음
useTransition

UI 표현하기

상호작용성 더하기

State 관리하기

탈출구

커뮤니티

행동 강령

팀 소개

문서 기여자

감사의 말

더 보기

블로그

React Native

개인 정보 보호

약관