useSyncExternalStore

- The useSyncExternalStore hook is designed to work with external stores to enable consistent state synchronization between the store and React components.
- It is particularly useful for managing state derived from external sources, such as a Redux store, a global event system, or any subscription-based system.
- useSyncExternalStore provides a clean and efficient way to synchronize React components with external stores.
- It ensures consistent state updates, particularly for SSR and hydration scenarios.
- By using this hook, you can connect components to various subscription-based systems without additional complexity.

Key Concepts

- **External Store:** An external store is any state container or subscription-based system outside React's state management. Examples include Redux, Zustand, or a custom event emitter.
- **Consistent State:** useSyncExternalStore ensures consistent state updates across server and client rendering. This is crucial for React's SSR (Server-Side Rendering) and hydration process.
- **Subscription Mechanism:** The hook provides an API to subscribe to changes in the external store and update the component state when the store changes.
- **Snapshot:** A snapshot is the current value of the external store's state.

Syntax

```
const snapshot = useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot);
```

where

- subscribe: A function to register a listener for store changes.
- getSnapshot: A function to get the current state of the store.
- getServerSnapshot (optional): A function to get the state on the server during SSR.

It Returns the latest snapshot (store state) to be used in the component.

Example: Using useSyncExternalStore with a Custom Store

```
import React, { useState, useEffect } from 'react';
import { useSyncExternalStore } from 'react';
// Create a simple store for managing state
const store = (() => {
 let state = { count: 0 }; // Initial state
 const listeners = new Set();
 return {
  getState: () => state,
  setState: (newState) => {
   state = { ...state, ...newState }; // Update the state
   listeners.forEach((listener) => listener()); // Notify subscribers
  },
  subscribe: (listener) => {
   listeners.add(listener); // Add listener
   return () => listeners.delete(listener); // Return an unsubscribe function
  },
 };
})();
function App() {
 // Subscribes to the store
 const count = useSyncExternalStore(
  store.subscribe,
  () => store.getState().count
 );
 const increment = () => {
  store.setState({ count: count + 1 });
 };
 return (
  <div>
   <h1>Count: {count}</h1>
   <button onClick={increment}>Increment</button>
  </div>
 );
}

export default App;
```

Key Advantages of useSyncExternalStore Over Other Hooks

- **Purpose-built for External Stores:** Unlike other hooks, useSyncExternalStore is explicitly designed to handle state from external, non-React state management systems.
- **Built-in Subscription Handling:** It abstracts away the complexity of manually managing subscriptions with hooks like useEffect.
- **Server-Side Rendering (SSR) Compatibility:** It ensures consistent state during SSR, reducing hydration mismatch risks.
- **Simplicity:** Provides a clean and declarative API to synchronize external store states without requiring additional boilerplate.

When to Use useSyncExternalStore

- **For External State Systems:** Use it when integrating external state managers like Redux, Zustand, or a custom subscription-based store.
- **Avoid for Local Component State:** For internal state, prefer useState or useReducer.
- **Subscription-heavy Use Cases:** Scenarios requiring consistent state synchronization, such as real-time updates (e.g., WebSocket, stock price ticker).

Example: Basic Counter with Redux using reducer

```
import { createStore } from 'redux';
import React from 'react';
import { useSyncExternalStore } from 'react';

// Redux: Reducer
const reducer = (state = { count: 0 }, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    default:
      return state;
  }
};

// Redux: Store
const store = createStore(reducer);

// React Component
function App() {
  const count = useSyncExternalStore(
    store.subscribe,
    () => store.getState().count
  );

  const increment = () => store.dispatch({ type: 'INCREMENT' });

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default App;
```

Example: Todo List with Redux

```
import { createStore } from 'redux';
import React, { useState } from 'react';
import { useSyncExternalStore } from 'react';
// Redux: Reducer
const reducer = (state = { todos: [] }, action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return { todos: [...state.todos, action.payload] };
    default:
      return state;
  }};
const store = createStore(reducer);          // Redux: Store
function App() {
  const todos = useSyncExternalStore(
                    store.subscribe,    () => store.getState().todos
  );
  const [task, setTask] = useState('');
  const addTodo = () => {
    store.dispatch({ type: 'ADD_TODO', payload: task });
    setTask('');
  };
  return (
    <div>
      <h1>Todo List</h1>
      <input        type="text"       value={task}
          onChange={(e) => setTask(e.target.value)}      />
      <button onClick={addTodo}>Add</button>
      <ul>
        {todos.map((todo, index) => (
          <li key={index}>{todo}</li>
        ))}
      </ul>
    </div>
  );
}
export default App;
```

Example: User Authentication

```
import { createStore } from 'redux';
import React, { useState } from 'react';
import { useSyncExternalStore } from 'react';
const reducer = (state = { user: null }, action) => {
  switch (action.type) {
    case 'LOGIN':
      return { user: action.payload };
    case 'LOGOUT':
      return { user: null };
    default:
      return state;
  }};
const store = createStore(reducer);
function App() {
  const user = useSyncExternalStore(
            store.subscribe,    () => store.getState().user
  );
  const [username, setUsername] = useState('');
  const login = () => store.dispatch({ type: 'LOGIN', payload: username });
  const logout = () => store.dispatch({ type: 'LOGOUT' });
  return (
    <div>      <h1>Authentication</h1>
      {user ? (        <div>
                  <h2>Welcome, {user}!</h2>
                  <button onClick={logout}>Logout</button>
             </div>      ) : (        <div>
                  <input
                   type="text"           value={username}
                  onChange={(e) => setUsername(e.target.value)}          />
                  <button onClick={login}>Login</button>
        </div>
      )}
    </div>
  );
}
export default App;
```

Example: Toggle Theme

```
import { createStore } from 'redux';
import React from 'react';
import { useSyncExternalStore } from 'react';

const reducer = (state = { theme: 'light' }, action) => {
  switch (action.type) {
    case 'TOGGLE_THEME':
      return { theme: state.theme === 'light' ? 'dark' : 'light' };
    default:
      return state;
  }
};

const store = createStore(reducer);

function App() {
  const theme = useSyncExternalStore(
    store.subscribe,
    () => store.getState().theme
  );

  const toggleTheme = () => store.dispatch({ type: 'TOGGLE_THEME' });

  return (
    <div style={{ backgroundColor: theme === 'light' ? '#fff' : '#333', color: theme
=== 'light' ? '#000' : '#fff' }}>
      <h1>Current Theme: {theme}</h1>
      <button onClick={toggleTheme}>Toggle Theme</button>
    </div>
  );
}

export default App;
```

Example: Product Inventory

```
import { createStore } from 'redux';
import React, { useState } from 'react';
import { useSyncExternalStore } from 'react';

const reducer = (state = { products: [] }, action) => {
  switch (action.type) {
    case 'ADD_PRODUCT':
      return { products: [...state.products, action.payload] };
    case 'REMOVE_PRODUCT':
      return { products: state.products.filter((_, index) => index !== action.payload) };
    default:
      return state;
  }
};

const store = createStore(reducer);

function App() {
  const products = useSyncExternalStore(
    store.subscribe,
    () => store.getState().products
  );

  const [product, setProduct] = useState('');

  const addProduct = () => {
    store.dispatch({ type: 'ADD_PRODUCT', payload: product });
    setProduct('');
  };

  const removeProduct = (index) => store.dispatch({ type: 'REMOVE_PRODUCT',
payload: index });

  return (
    <div>
      <h1>Product Inventory</h1>
      <input
       type="text"        value={product}
       onChange={(e) => setProduct(e.target.value)}      />
      <button onClick={addProduct}>Add Product</button>
      <ul>
       {products.map((p, index) => (
         <li key={index}>
          {p} <button onClick={() => removeProduct(index)}>Remove</button>
         </li>
       ))}
      </ul>
    </div>
  );
}

export default App;
```

Example: Stock Prices

```
import { createStore } from 'redux';
import React, { useEffect } from 'react';
import { useSyncExternalStore } from 'react';

const reducer = (state = { stocks: [] }, action) => {
  switch (action.type) {
    case 'UPDATE_STOCK':
      return { stocks: action.payload };
    default:
      return state;
  }
};

const store = createStore(reducer);

function fetchStocks() {
  setInterval(() => {
    const stocks = [
      { name: 'Friends Corp.', price: Math.random() * 100 },
      { name: 'Farzi Pvt. Ltd.', price: Math.random() * 100 },
    ];
    store.dispatch({ type: 'UPDATE_STOCK', payload: stocks });
  }, 2000);
}

function App() {
  useEffect(() => {
    fetchStocks();
  }, []);

  const stocks = useSyncExternalStore(
    store.subscribe,
    () => store.getState().stocks
  );

  return (
    <div>
      <h1>Stock Prices</h1>
      <ul>
        {stocks.map((stock, index) => (
          <li key={index}>
            {stock.name}: ${stock.price.toFixed(2)}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

Example: Synchronizing a React component with the system clock.

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

// Subscribe to clock updates
function subscribe(callback) {
  const intervalId = setInterval(callback, 1000); // Update every second
  return () => clearInterval(intervalId);
}

// Get the current snapshot of time
function getSnapshot() {
  return new Date().toLocaleTimeString();
}

// Optional server-side snapshot (return a static time)
function getServerSnapshot() {
  return "00:00:00";
}

function App() {
  const time = useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot);

  return (
    <div>
      <h1>Current Time</h1>
      <p>{time}</p>
    </div>
  );
}

export default App;
```

Example: Synchronizing a React component with the browser's network (Online/Offline) status

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

// Subscribe to online/offline events
function subscribe(callback) {
  window.addEventListener('online', callback);
  window.addEventListener('offline', callback);
  return () => {
    window.removeEventListener('online', callback);
    window.removeEventListener('offline', callback);
  };
}

// Get the current network status
function getSnapshot() {
  return navigator.onLine ? "Online" : "Offline";
}

// Optional server-side snapshot
function getServerSnapshot() {
  return "Unknown"; // Assume unknown state for SSR
}

function App() {
  const status = useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot);

  return (
    <div>
      <h1>Network Status</h1>
      <p>{status}</p>
    </div>
  );
}

export default App;
```

Example: Synchronizing a React component with a global authentication state.

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

// Simulate a global auth store
const authStore = {
  isAuthenticated: false,
  listeners: new Set(),
  subscribe(callback) {
    this.listeners.add(callback);
    return () => this.listeners.delete(callback);
  },
  setAuthStatus(status) {
    this.isAuthenticated = status;
    this.listeners.forEach((listener) => listener());
  },
};

// Subscribe to auth changes
function subscribe(callback) {
  return authStore.subscribe(callback);
}

// Get the current authentication status
function getSnapshot() {
  return authStore.isAuthenticated ? "Logged In" : "Logged Out";
}

// Optional server-side snapshot
function getServerSnapshot() {
  return "Unknown"; // Default state for SSR
}

function App() {
  const status = useSyncExternalStore(subscribe, getSnapshot, getServerSnapshot);

  const login = () => authStore.setAuthStatus(true);
  const logout = () => authStore.setAuthStatus(false);

  return (
    <div>
      <h1>Authentication Status</h1>
      <p>{status}</p>
      <button onClick={login}>Login</button>
      <button onClick={logout}>Logout</button>
    </div>
  );
}

export default App;
```

useSyncExternalStore Best Practices

The useSyncExternalStore hook is designed to provide a way to subscribe to external stores and manage state in React applications. It is especially useful for integrating with libraries or systems outside React's state management (e.g., Redux, MobX, or even custom stores).

Below are the best practices of using useSyncExternalStore:

1. **Ensure Consistent Snapshots:** The getSnapshot function must always return a consistent and up-to-date value, as React relies on it to determine whether a component should re-render.
   Avoid returning stale data or values that depend on async computations.
   Example
   ```
   function getSnapshot() {
     return store.getState().count; // Ensure this is always the latest state
   }
   ```

2. **Provide a getServerSnapshot for SSR:** If you use server-side rendering (SSR), always supply a getServerSnapshot function that provides a fallback value when window or other browser-specific APIs are unavailable.
   Example
   ```
   function getServerSnapshot() {
     return 0; // Default count value for SSR
   }
   ```

3. **Unsubscribe Properly:** Ensure that subscribe returns a cleanup function to avoid memory leaks.
   The cleanup function should remove the event listener or unsubscribe from the external store.
   Example
   ```
   function subscribe(callback) {
     store.subscribe(callback); // Register listener
     return () => store.unsubscribe(callback); // Unregister listener
   }
   ```

4. **Avoid Heavy Computation in getSnapshot:** getSnapshot should be synchronous and lightweight. Avoid expensive computations inside it.
   Perform any heavy calculations elsewhere and store the result in the external store.

5. **Handle Edge Cases Gracefully:** Handle scenarios where the external store might not be ready (e.g., when no data is loaded yet).
   Use default values or placeholders in such cases.
   Example
   ```
   function getSnapshot() {
     return store.getState() || { count: 0 }; // Default state if undefined
   }
   ```

6. **Debounce Frequent Updates:** For events like resize or scroll, debounce or throttle updates to avoid excessive re-renders.
   Example
   ```
   import { debounce } from 'lodash';
   function subscribe(callback) {
     const debouncedCallback = debounce(callback, 100); // Debounce
     window.addEventListener('resize', debouncedCallback);
     return () => window.removeEventListener('resize', debouncedCallback);
   }
   ```

useSyncExternalStore Debugging Tips

1. **Verify getSnapshot Outputs:** Log the output of getSnapshot to ensure it reflects the correct state at all times.
   Example
   ```
   function getSnapshot() {
     const state = store.getState();
     console.log('Current snapshot:', state); // Debug output
     return state;          }
   ```

2. **Check Subscriptions:** Ensure the subscribe function is called and properly registers/unregisters listeners.
   Example
   ```
   function subscribe(callback) {
     console.log('Subscribed');
     window.addEventListener('resize', callback);
     return () => {    console.log('Unsubscribed');
       window.removeEventListener('resize', callback);
     };}
   ```

3. **Test Server-Side Rendering:** Use tools like Next.js to simulate SSR and ensure the getServerSnapshot function behaves as expected.
   Example
   ```
   function getServerSnapshot() {
     console.log('Server snapshot invoked');
     return { width: 0, height: 0 };
   }
   ```

4. **Inspect Re-renders:** Use React DevTools to check how often the component re-renders. Excessive renders could indicate a problem with the getSnapshot or subscribe function.

5. **Handle Errors in External Stores:** Ensure the external store is robust and doesn't throw errors during updates. Wrap subscribe and getSnapshot with error handling if needed.
   Example
   ```
   function getSnapshot() {
     try {
       return store.getState().count;
     } catch (error) {
       console.error('Error fetching snapshot:', error);
       return 0; // Default value in case of error
     }
   }
   ```

6. **Use Mock External Stores for Testing:** When testing with libraries like Jest, mock the external store to simulate various states.