React Version 18

source: https://www.youtube.com/watch?v=ytudH8je5ko

리액트 18 버전

1. 리액트 18에는 무엇이 있나요?

2. 리액트 18로 어떻게 업그레이드 할수있나요?

리액트 18 버전에 새로운 것은 무엇인가요 ?!

1. Automatic batching

2. Suspense on the server

3. New APIs for app and library developers

4. Transition

Automatic batching

Automatic batching

https://github.com/reactwg/react-18/discussions/21

배칭(batching)은 업데이트 대상이 되는 상태값들을 하나의 그룹으로 묶어서 한번의 리렌더링에 업데이트가 모두 진행될 수 있게 해주는 것을 의미합니다.

한 함수 안에서 setState를 아무리 많이 호출시키더라도 리렌더링은 단 한번만 발생합니다.

```
function handleClick() {
  setCount(c => c + 1); // Does not re-render yet
  setFlag(f => !f); // Does not re-render yet
  // React will only re-render once at the end (that's batching!)
}
```

리렌더링 1번

함수에 끝에서 업데이트가 되며 리액트는 마지막에 한번만 리렌더링 합니다.
이것은 여러번 리렌더링을 하는것을 막기때문에 성능상 좋은 영향을 줍니다.

batch update를 사용함으로 불필요한 리렌더링을 줄일 수 있어서 퍼포먼스적으로 큰 이점을 얻을 수 있는데요, 이전 버전에서도 이런 batch update가 지원되었지만 클릭과 같은 브라우저 이벤트에서만 적용이 가능하고 api 호출에 콜백으로 넣은 함수나 timeouts 함수에서는 작동하지 않았습니다.

```
function handleClick() {
    fetchSomething().then(() => {
      // React 17 and earlier does NOT batch these because
      // they run *after* the event in a callback, not *during* it
      setCount(c => c + 1); // Causes a re-render
      setFlag(f => !f); // Causes a re-render
    });
  }

setTimeout(() => {
  setCount(c => c + 1); // re-render occurs
  setFlag(f => !f); // re-render occurs again!
}, 1000);
```

리렌더링 1번

리렌더링 2번

리렌더링 1번

리렌더링 2번

batch 처리하지 않으려면 어떻게 합니까?

일반적으로 일괄 처리는 안전하지만 일부 코드는 상태 변경 직후 DOM에서 무언가를 읽는 데 의존할 수 있습니다. 이러한 사용 사례의 경우 ReactDOM.flushSync()를 사용하여 일괄 처리를 옵트아웃할 수 있습니다.

```
import { flushSync } from "react-dom";

function handleClick() {
    flushSync(() => {
        setCount(count + 1);
    })
    // 리액트가 바로 돔 업데이트를 합니다. (re-render)

    flushSync(() => {
        setClicked(true);
    })
    // 리액트가 바로 돔 업데이트를 합니다. (re-render)
}
```

리액트 18 버전에서 Automatic batching

1. 더 나은 성능을 위한 더 적은 리 렌더링을 합니다.

2. 이벤트 핸들러 밖에서도 작동

3. 필요할 때는 제외할 수 있습니다.

Suspense on the server (간단하게)

Suspense on the server

https://github.com/reactwg/react-18/discussions/37

서버 사이드 렌더링

리엑트의 서버사이드 렌더링은 다음의 스텝으로 이뤄집니다.
1. 서버에서 전체 앱에 대한 데이터를 가져옵니다.
2. 그런 다음 서버에서 전체 앱을 HTML로 렌더링하고 응답으로 보냅니다.
3. 그런 다음 클라이언트에서 전체 앱에 대한 JavaScript 코드를 로드합니다.
4. 그런 다음 클라이언트에서 JavaScript 논리를 전체 앱에 대해 서버 생성 HTML에 연결합니다(이것이 "hydration"입니다).

Hydration은 ....
Dry한 HTML에 수분(Javascript) 공급을 하는 것입니다.

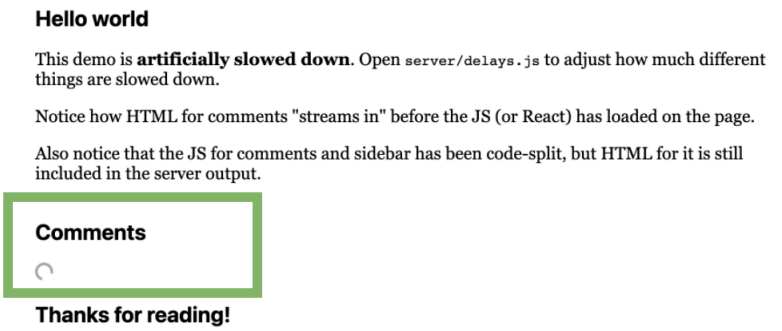
여기에서 문제점은 무엇인가요?!!!

핵심 부분은 다음 단계가 시작되기 전에 각 단계가 전체 앱에 대해 한 번에 완료되어야 한다는 것입니다.(동기 방식 및 Waterfall 방식) 거의 모든 중요하지 않은 앱의 경우와 같이 앱의 일부가 다른 부분보다 느린 경우 이는 효율적이지 않습니다. 왜냐면 한 페이지에 여러 컴포넌트가 있을 때 HTML 생성이 오래 걸리는 컴포너트가 있고, 그렇지 않은 컴포넌트가 있는데 빨리 생성된 컴포넌트가 다른 컴포넌트를 위해서 계속 기다려야 하기 때문에 비효율적이게 됩니다.

리액트 18에서 Suspense로 문제 해결하기

React 18을 사용하면 <Suspense />를 사용하여 앱을 더 작은 독립 단위로 나눌 수 있습니다. 이 단위는 서로 독립적으로 이러한 단계를 거치며 앱의 나머지 부분을 차단하지 않습니다. 결과적으로 앱 사용자는 콘텐츠를 더 빨리 보고 훨씬 빠르게 상호 작용할 수 있습니다. 앱에서 가장 느린 부분은 빠른 부분을 끌어내리지 않습니다. 이러한 개선 사항은 자동이며 작동하기 위해 특별한 조정 코드를 작성할 필요가 없습니다.

```
<Layout>
  <NavBar />
  <aside className="sidebar">
    <Suspense fallback={<Spinner />}>
      <Sidebar />
    </Suspense>
  </aside>
  <article className="post">
    <Suspense fallback={<Spinner />}>
      <Post />
    </Suspense>
    <section className="comments">
      <h2>Comments</h2>
      <Suspense fallback={<Spinner />}>
        <Comments />
      </Suspense>
    </section>
    <h2>Thanks for reading!</h2>
  </article>
</Layout>
```



https://codesandbox.io/s/kind-sammet-j56ro?file=/src/App.js

Hello world

This demo is **artificially slowed down**. Open `server/delays.js` to adjust how much different things are slowed down.

Notice how HTML for comments "streams in" before the JS (or React) has loaded on the page.

Also notice that the JS for comments and sidebar has been code-split, but HTML for it is still included in the server output.

Comments

Wait, it doesn't wait for React to load?

How does this even work?

I like marshmallows