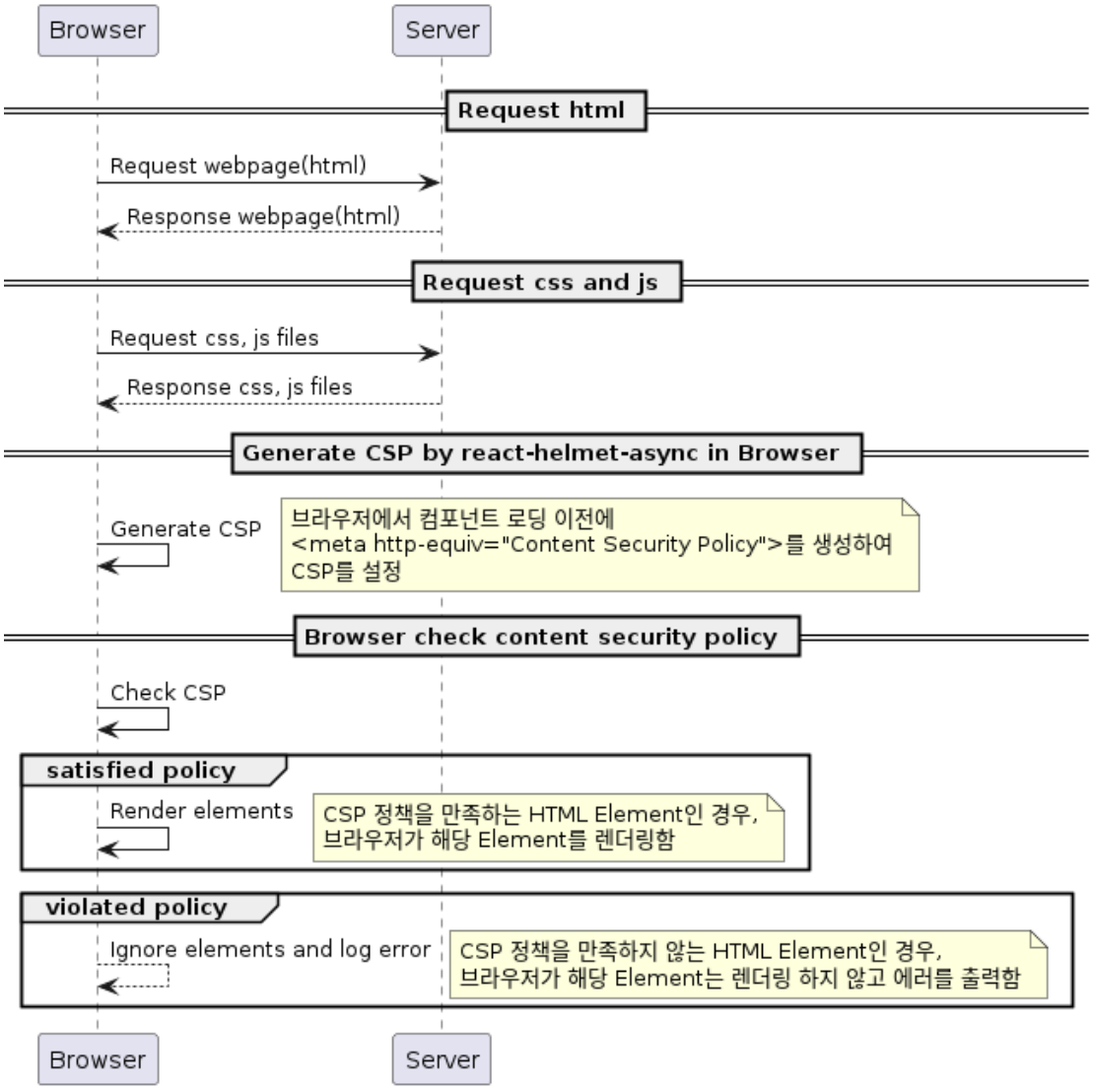
Content Security Policy in React

개요

react-helmet-async를 이용하여 React에서 Content Security Policy(CSP)를 적용하거나, 해제하는 예시를 보여주는 코드와 관련된 글입니다.

자세한 설정 방법 등은 레포지토리의 README.md를 참고해 주세요.

원리



react-helmet-async를 이용하여 정적인 CSP를 브라우저에서 설정하는 경우의 sequence diagram

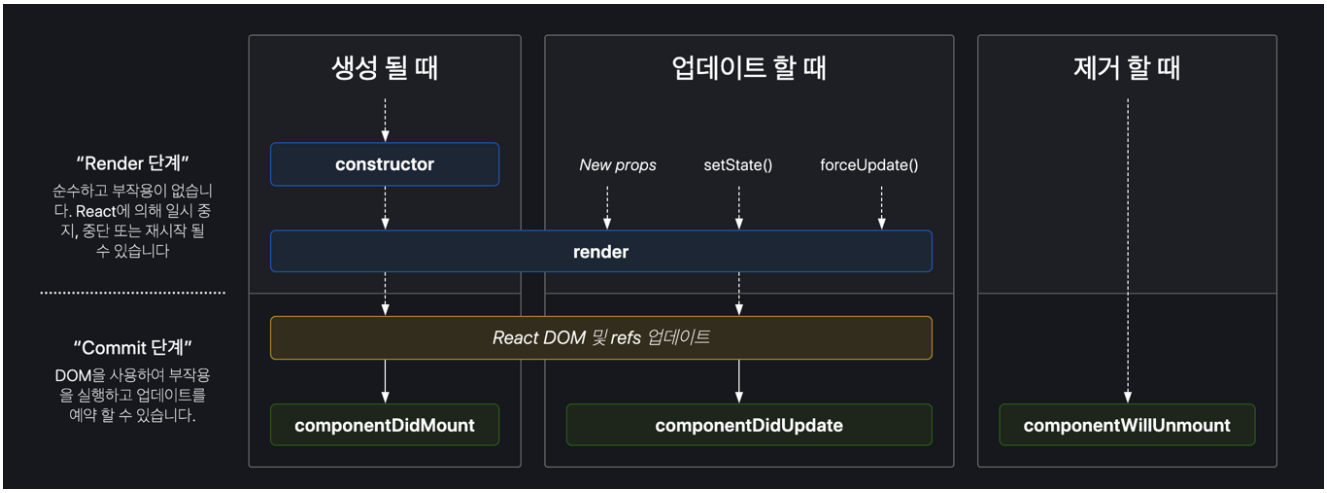
> ℹ️ 위의 예시는 렌더링 이후에 CSP에 정책을 추가하더라도, 추가된 정책이 적용되지 않습니다. 새로운 정책을 적용하기 위해서는 아래의 과정을 거쳐야 합니다.
>
> - 해당 정책이 추가된 CSP를 cookie 등에 저장
> - 페이지가 다시 로딩될 때 추가된 정책이 포함된 CSP로 정책 설정

- Client Side Rendering(CSR)의 경우, HTML `<meta http-equiv="Content-Security-Policy">` 태그를 이용하여 CSP를 설정합니다.
- `<meta>` 태그를 이용한 경우의 제약조건으로 인해, 쿠키와 새로고침을 사용합니다.
  - CSP 관련 데이터는 쿠키에 저장합니다.
  - CSP 관련 데이터가 변경되거나, 페이지를 이동하는 경우 페이지가 새로고침을 진행합니다. 이는 새로운 CSP를 적용시키기 위함입니다.

참고

React의 컴포넌트 렌더링 순서

하위 컴포넌트가 우선 `마운트`, `렌더링` 및 `업데이트`된 이후에, 상위 컴포넌트가 `마운트`, `렌더링` 및 `업데이트`를 진행하게 됩니다.



클래스 기반 React에서의 Component cycle

⭐ `componentDidMount`, `componentDidupdate`, `componentWillUnmount`는 각각 `useEffect(callbackFn, depArr)`에서 다음과 같이 맵핑되며, `마운트`, `업데이트`, `렌더링` 순서대로 실행됩니다.

- 컴포넌트 마운트(`componentDidMount`)

```
useEffect(() => {
  // some logics...
}, []);  // depArr이 비어 있는 경우
```

- 컴포넌트 업데이트(`componentDidUpdate`)

```
useEffect(() => {
  // some logics...
}, [someDeps])  // depArr에 값이 있는 경우
```

- 컴포넌트 렌더링(`componentWillUnmount`)

```
useEffect(() => {
  // 이 부분은 컴포넌트가 렌더링이 완료되는 부분
  // some logics...

  return () => {
    // 이 부분이 componentWillUnmount에 해당됨
    // some cleanups...
  }
})  // depArr이 아예 없는 경우
```

`react-helmet-async`의 경우는 예외적으로 `<Helmet>`의 하위 컴포넌트가 먼저 마운트되고, 이후에 다른 컴포넌트가 마운트됩니다.

```
<Helmet>
  {/* 1. meta 태그가 가장 먼저 마운트됨 */}
  <meta httpEquiv="Content-Security-Policy" id="Content-Security-Policy"
/>
</Helmet>
{/* 2. 이후에 자매 컴포넌트 및 하위 컴포넌트가 마운트됨 */}
<div>
  {/* 다른 컴포넌트들 ... */}
</div>
```

자세한 내용은 useEffect 및 컴포넌트 실행 순서를 참고해 주세요.

> `<meta>`가 먼저 마운트 되는 것이 중요한 이유는 `<meta>`태그보다 먼저 마운트 된 element에는 CSP가 적용되지 않기 때문입니다.
>
> > Authors are *strongly encouraged* to place meta elements as early in the document as possible, because **policies in meta elements are not applied to content which precedes them**. In particular, note that resources fetched or prefetched using the Link
> > HTTP response header field, and resources fetched or prefetched using link and script
> > elements which precede a meta-delivered policy will not be blocked.

`<meta>` 태그 이용 시 제약사항

> Note: Modifications to the content attribute of a meta element after the element has been parsed will be ignored.

> 위 내용으로 인해 `'nonce-…'` 관련 코드는 작성하다가 폐기하고(...), 기본적인 CSP 실습 코드만 작성하였습니다.

실습 내용

> - React 페이지와 개발자 도구의 콘솔 창을 나란히 열어서 CSP 위반 여부가 발생하는지 확인하는 것을 권장합니다.
> - 문제가 발생한 경우, 쿠키를 지우고 루트 페이지로 이동해 보시기 바랍니다.

`Home`

~~아무것도 없습니다.~~

`/XSS` 페이지

`unsafe-inline`을 테스트하는 페이지입니다.

XSS 컴포넌트 아래에 제공한 `<img>`를 `<textarea>`에 붙여놓고, Submit 버튼을 눌러서 `<img>`의 `onerror` 스크립트가 실행되는지 확인합니다.

> CSP 정책에서 `unsafe-inline`이 허용된 경우에는 스크립트가 실행되고, 아닌 경우에는 콘솔에 CSP 오류가 발생하며 스크립트가 실행되지 않습니다.

`Inline-style` 페이지

`unsafe-inline`을 테스트하는 페이지입니다.

> CSP 정책에서 `unsafe-inline`이 허용된 경우에는 inline-style이 적용되고, 아닌 경우에는 콘솔에 CSP 오류가 발생하며 inline-style이 적용되지 않습니다.

`Jodit` 페이지

`https` 및 `unsafe-inline`, `enable hash`를 테스트하는 페이지입니다.

> - `https`가 적용된 경우에는 콘솔에 `ace.js` 관련 오류가 발생하지 않습니다.
> - `unsafe-inline`가 적용된 경우에는 Jodit 에디터의 레이아웃이 올바르게 나옵니다.
> - `enable hash`가 적용된 경우에는 `ace.js`에서 `sha256-` 관련 CSP 오류가 발생하지 않습니다.

Reference

- ContentSecurityPolicy 실습 코드
- useEffect 및 컴포넌트 실행 순서
- [[번역] 리액트 렌더링 동작의 (거의) 완벽한 가이드 A (Mostly) Complete Guide to React Rendering Behavior]
- react-helmet-async
- How to sanitize third-party content with vanilla JS to prevent cross-site scripting (XSS) attacks