

스터디 발표

계기

사이드 프로젝트를 nextjs13버전으로 진행하고 있는데 동작방식이 이해가 되지 않는 부분이 있어서

자세히 알아보기 위해 조사하였고, 이를 위해 테스트를 진행하거나 혼자 공부 한 것을 기록하기 보다

발표를 하는게 좋겠다고 판단해서 준비하게 되었습니다.

목적

이번 발표의 목적은 csr과 SSR의 차이점과 nextjs에서 왜 SSR을 다시 사용하는지 그리고 새롭게 도입된 RSC에 대해서 알아보는 것이 이번 발표의 목적입니다.

CSR과 SSR이란?

CSR은 클라이언트 사이드 렌더링 SSR은 서버 사이드 렌더링의 줄임말입니다.

간단히 한 줄로 요약하면 페이지가 클라이언트에서 렌더링 되나 서버에서 렌더링 되냐의 차이라고 생각합니다.

nextjs를 사용하는 이유는 너무나도 많지만

꼭 빠지지 않고 나오는 이유는 SSR을 쉽게 할 수 있다는 말이 나오는 것 같습니다.

그런데 왜 SSR이 갑자기 중요하게 됐는지 의문점이 생깁니다.

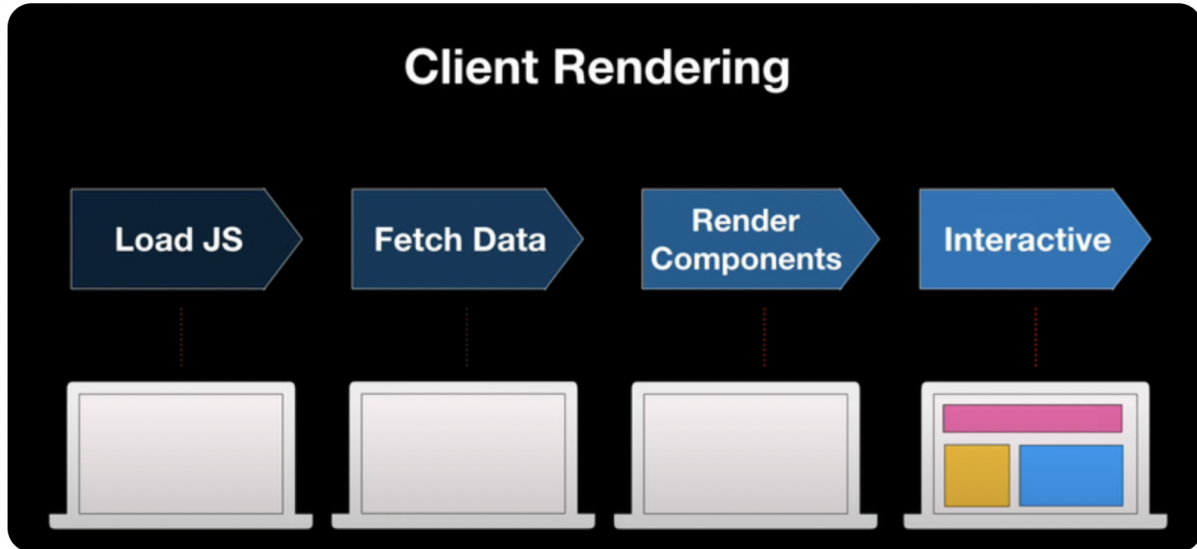
사실 SSR은 새로운 기술이 아니라 기존에 있는 기술이고 SSR에서 CSR로 바뀐건데 왜 갑자기 과거로 돌아가는 것일까요?

제가 생각하는 대표적인 이유는 SEO와 초기 웹페이지 화면 로딩 때문입니다.

CSR

기존의 react는 csr로 동작합니다.

csr은 클라이언트에서 렌더링이 되는 것을 의미합니다.



아래와 같이 처음에 필요한 js를 다운로드 받을 뒤 필요한 데이터를 요청하고 화면에 렌더링을 한뒤 화면이 보이게 됩니다.

장점으로는 하나의 index.html을 사용하기 때문에 페이지 간 이동 시 속도가 빠르고 화면의 깜빡임이 없다는 장점이 있습니다.

하지만 필요한 초기에 데이터와 js를 전부 가져와야 되기 때문에 초기 렌더링이 느리다는 단점이 있으며 때문에 사용자 입장에서는 빈화면을 오래 봐야하며, SEO(검색엔진최적화)가 잘 안된다는 단점이 존재합니다.



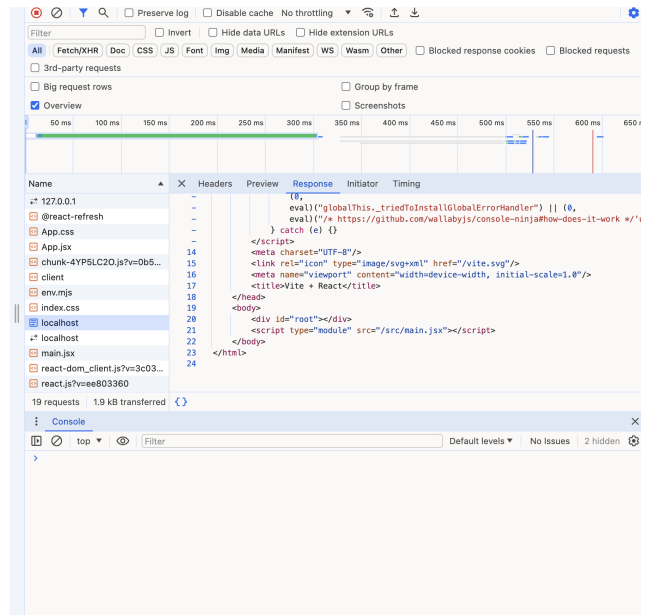
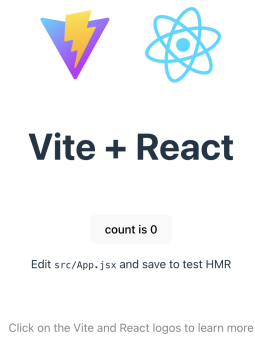
SEO에 대해 약간 설명해보자면

우리는 궁금한게 생길 경우 구글 또는 네이버등으로 검색을 하게 되는데 이때 특정 키워드를 검색하게 됩니다.

이때 대부분의 사람들은 검색결과 상위에 있는 것을 클릭하게 되는데 이때 상위에 노출도를 높이는 것을 SEO라고 합니다.

그럼 react는 왜 SEO가 제대로 안될까요?

그 이유는 react는 초기에 아무것도 없는 div 하나만 있기 때문에 SEO에 노출이 낮은 것입니다.(물론 vite등을 사용하며 ssr을 할 수 있지만 아무것도 하지 않았을 경우는 상정하겠습니다.)



```

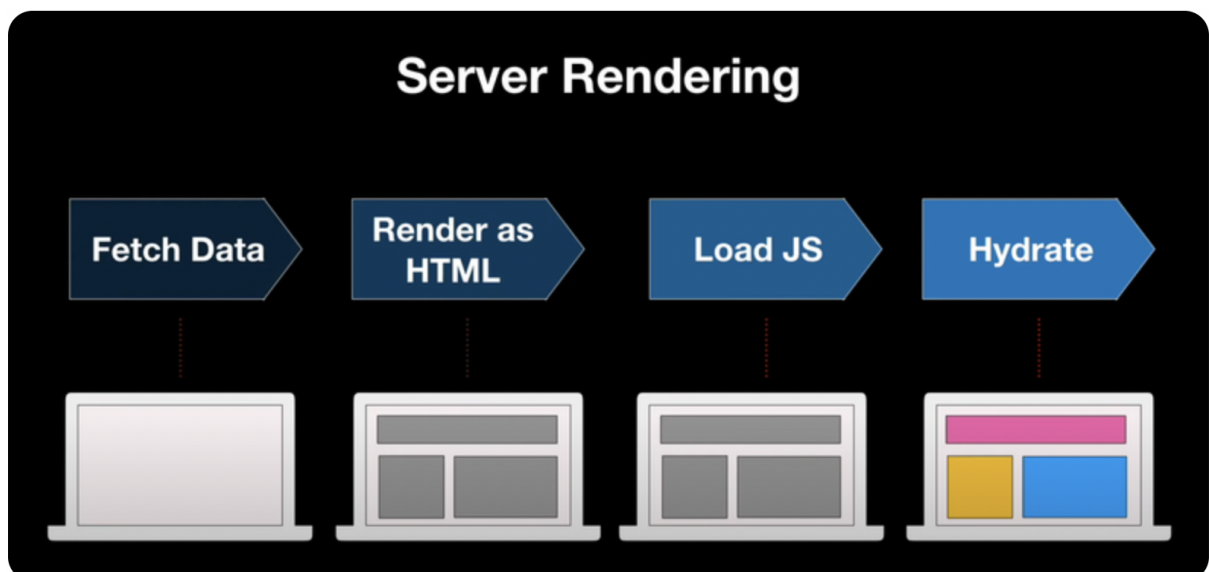
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
</html>

```

(위의 사진을 보면 root 라는 아이디를 가진 div 태그에 아무것도 없는 것을 볼 수 있습니다.)

따라서 다시 ssr이 주목을 받게 된것입니다.

SSR



ssr은 위와 같이 동작합니다.

필요한 데이터를 요청하고 html을 렌더링하게 됩니다.

그럼 사용자는 js를 사용할 수는 없지만 일단 화면에는 html이 보이게 됩니다.

그 후 필요한 js를 다운받고 Hydrate를 하게 됩니다.



Hydrate는 수분을 공급해준다. 즉, 여기서는 메마른 html에 js를 공급해준다는 의미입니다.

<https://github.com/reactwg/react-18/discussions/46#discussioncomment-846714>

따라서 사용자는 초기에 빈화면을 csr과 비교해서 훨씬 짧은 시간만 보게 되며, SEO에도 csr과 비교해서 잘 노출 될것입니다.

(위의 사진을 보면 아까의 react와 다르게 내부에 html이 서버에서 전달된것을 확인 할 수 있습니다.)

하나의 예시로 넷플릭스를 생각해보면 csr을 사용할 경우 그 많은 프리뷰 비디오가 다운로드 될때까지 빈 화면을 봐야겠지만 ssr을 사용할 경우 빈 화면 대신 콘텐츠 제목, 설명, 메인 이미지가 채워진 페이지를 제공받을 수 있을 것입니다.

이는 페이지를 기다리는 동안 사용자의 시선을 끌 수 있고 곧 사용자 경험의 만족도에 큰 도움이 될 것입니다.

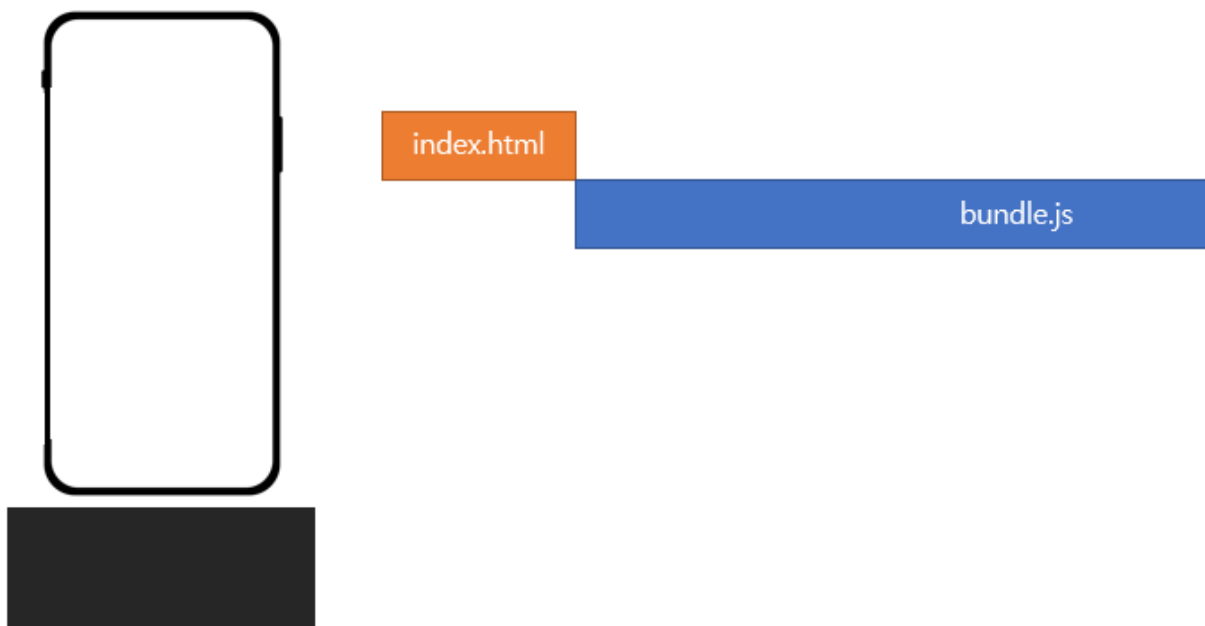
하지만 `ssr`이 장점만 있는 것은 아닙니다. `ssr`은 초기렌더링시 위와 같은 이점이 있지만 페이지 이동시 서버에서 `html`을 받아오기 때문에 화면이 깜빡이는 현상이 발생하게 되고 이는 사용자 경험 만족도에 악영향을 끼치게 됩니다.

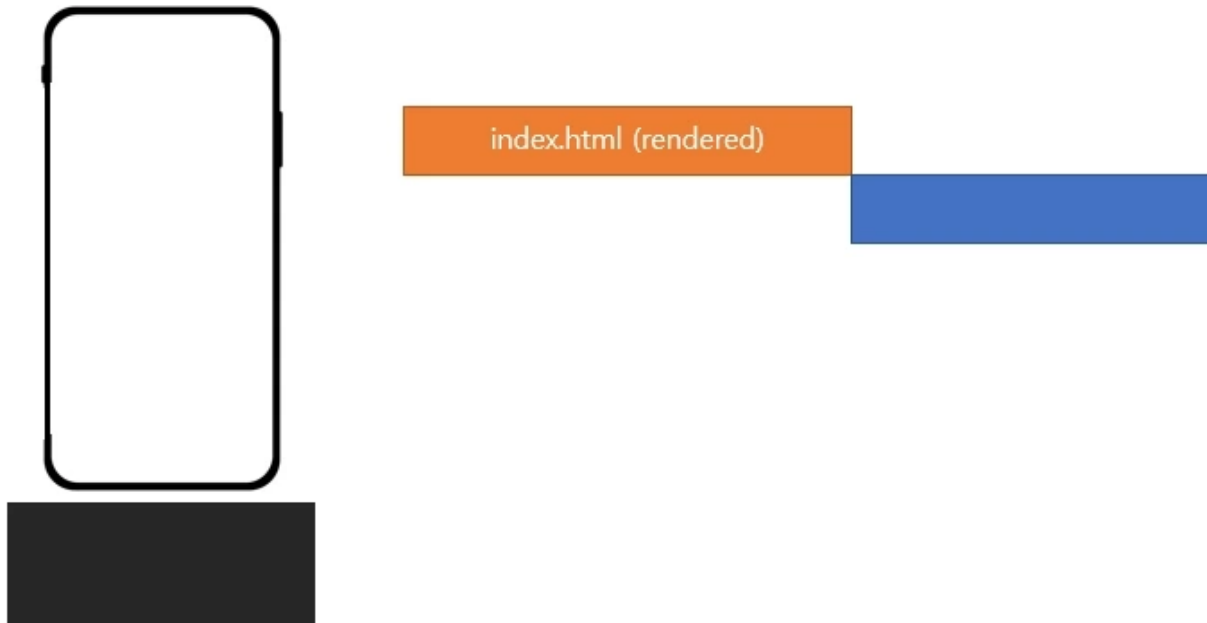
따라서 `nextjs`는 처음에는 `ssr`로 동작하며 이후에는 서버컴포넌트와 클라이언트컴포넌트를 활용하여

동작하게 됩니다.

(기본적으로 `nextjs`는 서버컴포넌트로 동작하며 클라이언트 컴포넌트는 파일 상단에 `'use client'`를 작성해서 사용할 수 있습니다.)

SSR vs CSR의 차이점을 시각적으로 표현한 자료





RSC

위에서 처음에는 `ssr`로 동작하고 그 후 서버 컴포넌트와 클라이언트 컴포넌트로 동작한다고 설명했는데 클라이언트 컴포넌트는 기존의 `react`와 비슷할것 같은데 서버 컴포넌트는 무엇 일까요??

RSC는 nextjs에는 최근에 도입하게 됐는데 간단히 설명하면 서버에서 렌더링 되어 클라이언트에 전달되는 컴포넌트입니다.

전달 예시)

```

1: 1:HL["/_next/static/css/81df82b2dad46e0cd.css"],["$","style"]
2: 0:["LuxH7zUo3e1HSYLIC6pt",["[[["","children":["list","children":["_PAGE_",{}]]],{"$undefined","$undefined",true,"$L2",["[["$","link","0",{"rel":"stylesheet","href":"/_next/s
3: 4:["id":"3673","chunks":["435:static/chunks/435-4f1b69e0eccdbae.js","185:static/chunks/app/layout-4f584ef1afdf37ba.js"],"name":"","async":false
4: 5:["id":"6666","chunks":["435:static/chunks/435-4f1b69e0eccdbae.js","185:static/chunks/app/layout-4f584ef1afdf37ba.js"],"name":"","async":false
5: 6:["id":"7777","chunks":["272:static/chunks/webpack-0f070c5179e9a07e1.js","971:static/chunks/fd9d1056-10ab5691e5ea3b76.js","596:static/chunks/596-7afab56cf14d5cfb.js"],"name":"de
6: 7:["id":"7920","chunks":["272:static/chunks/webpack-0f070c5179e9a07e1.js","971:static/chunks/fd9d1056-10ab5691e5ea3b76.js","596:static/chunks/596-7afab56cf14d5cfb.js"],"name":"de
7: 8:["id":"5201","chunks":["300:static/chunks/app/list/layout-10a3f15652347cb3.js"],"name":"","async":false
8: 9:["id":"4839","chunks":["272:static/chunks/webpack-0f070c5179e9a07e1.js","971:static/chunks/fd9d1056-10ab5691e5ea3b76.js","596:static/chunks/596-7afab56cf14d5cfb.js"],"name":"de
9: a:["id":"5218","chunks":["122:static/chunks/app/list/page-dc058224afdb49.js"],"name":"","async":false
10: 2:["$","html",null,{"lang":"ko","suppressHydrationWarning":true,"children":["$","head",null,{"$","body",null,{"children":["$","$L4",null,{"$","div",null,{"className
11: 3:["$","meta","0",{"charSet":"utf-8"}],["$","title","1",{"children":["ë", "ë" € 8 €ëB "œ$S" todo}],["$","meta","2",{"name":"description","content":["ë", "ë" € 8 €ëB "œ$S"
12

```

(위와 같이 json과 비슷한 형식으로 전달 됩니다.)

리액트 서버 컴포넌트는 서버에서 동작하기 때문에, DB, 파일 등에도 접근할 수 있으며,

```
// Note.server.js - 서버 컴포넌트
import fs from 'react-fs';
import db from 'db.server';

function Note(props) {
  // NOTE: loads *during* render, w low-latency data access o
  const note = db.notes.get(props.id); // 데이터베이스 접근
  const noteFromFile = JSON.parse(fs.readFile(`${id}.json`));

  if (note == null) {
    // handle missing note
  }
  return (/* render note here... */);
}
```

이렇게 서버에서 fetching한 데이터는 클라이언트 컴포넌트에 props로 전달 가능합니다. 다만 한 가지 유의해야 할 점은 json으로 인코딩 가능한 직렬화(serializable) props만 전달 가능하며 function은 전달할 수 없습니다.

또한, 서버에서 렌더링이 되기 때문에 사용자가 어떤 api에 요청을 하고 있는지 노출시키지 않을 수도 있습니다.

요약

URL: http://localhost:3001/list2?_rsc=70jlc

상태: 200 OK

소스: 네트워크

주소: ::1:3001

요청

Contrary to popular belief, the cat is a social animal. A pet cat will respond and answer to speech , and seems to enjoy human companionship.

list2입니다.

클릭하지마세요

위의 "list2입니다.", "클릭하지마세요"를 제외한 영어 데이터는 api를 요청해서 데이터를 받아왔지만 요청한 api에 대한 url은 노출하지 않습니다.

서버 컴포넌트 실제 예시 : <http://localhost:3001/list2>

또한, 제로 번들 사이즈 컴포넌트가 가능합니다.

프론트엔드 앱을 개발하게된다면 많은 라이브러리를 사용하게 되는데 기존에는 해당 라이브러리때문에 번들 사이즈가 늘어나게 되고, 퍼포먼스에 악영향을 끼치게 되었습니다.

하지만 서버 컴포넌트는 코드는 브라우저에 다운로드 되는것이 아닌 서버에서 미리 렌더링 된 결과를 클라이언트에 전달하기 때문에 번들 사이즈에 영향을 끼치지 않습니다.

예로 api를 통해서 리스트 데이터를 요청할 경우

기존에는 브라우저가 해당 라이브러리의 코드를 알아야 되기 때문에 번들에 추가하여 브라우저에 전달 되었지만 이제는 서버에서 해당 라이브러리의 코드를 읽을 후 결과값을 브라우저에 전달하기 때문에 번들에 추가하지 않아도 됩니다.

하지만 꼭 서버 컴포넌트만 사용하는 것은 아닙니다. 클라이언트 컴포넌트도 존재합니다.

서버 및 클라이언트 구성 요소를 언제 사용합니까?		
다음은 서버 및 클라이언트 구성 요소의 다양한 사용 사례에 대한 간략한 요약입니다.		
월하길 원해?	서버 구성요소	클라이언트 구성요소
데이터 가져오기	✓	×
백엔드 리소스에 (직접) 액세스	✓	×
민감한 정보를 서버에 보관하세요(액세스 토큰, API 키 등)	✓	×
서버에 대한 큰 의존성 유지 / 클라이언트 측 JavaScript 감소	✓	×
상호작용 및 이벤트 리스너(<code>onClick()</code> , <code>onChange()</code> 등) 추가	×	✓
상태 및 수명주기 효과 사용(<code>useState()</code> , <code>useReducer()</code> , <code>useEffect()</code> 등)	×	✓
브라우저 전용 API 사용	×	✓
상태, 효과 또는 브라우저 전용 API에 따라 달라지는 사용자 정의 후크를 사용하세요.	×	✓
React 클래스 구성요소 사용 ^기	×	✓

위의 표를 보고 자신이 필요할 경우 클라이언트 컴포넌트를 사용하면 되고 상단에 'use client'를 작성하면 됩니다.

클라이언트 컴포넌트가 필요한 경우는 위의표에서 나오지만 보통은 react hooks와 브라우저 전용 api를 사용해야 되는 경우가 있습니다.

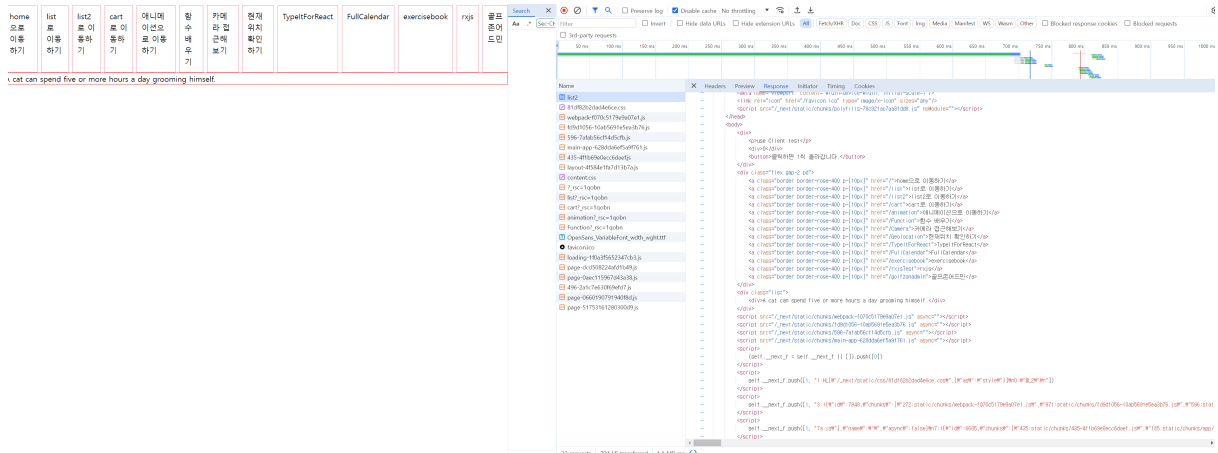
클라이언트 컴포넌트 실제 예시 : <http://localhost:3001/list>

RSC와 SSR의 차이점

서버 컴포넌트와 서버사이드렌더링을 둘다 서버에서 렌더링된다는 점이 비슷한것 같은데 무슨 차이가 있을까요?

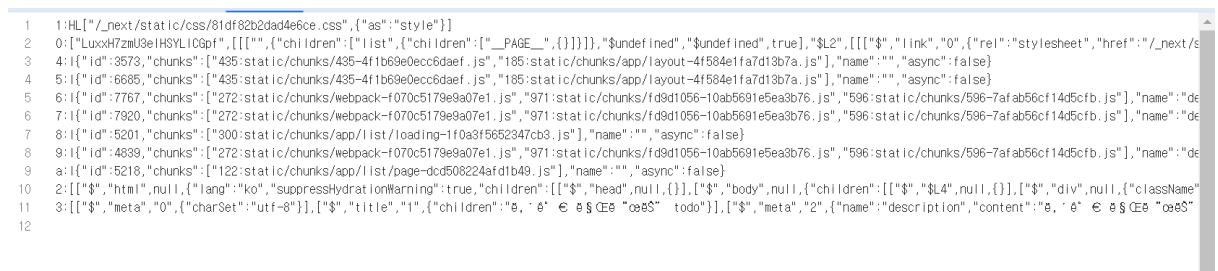
서버 컴포넌트와 서버 사이드 렌더링은 이름에 둘다 서버가 들어가서 비슷하지만 다른점이 존재합니다.

SSR은



위와 같이 html이 전달 되지만

RSC는



위와 같이 전달됩니다.

둘은 전달 되는 방식이 다른데 SSR은 애초에 html자체가 전달 됩니다.

따라서 새로운 refetch가 필요한 경우 HTML 전체를 리렌더링 해야 되기 때문에 클라이언트 상태를 유지할 수 없습니다.

하지만 RSC는 html을 전달하는 것이 아닌 위와 같이 json과 비슷한 형식으로 전달되기 때문에

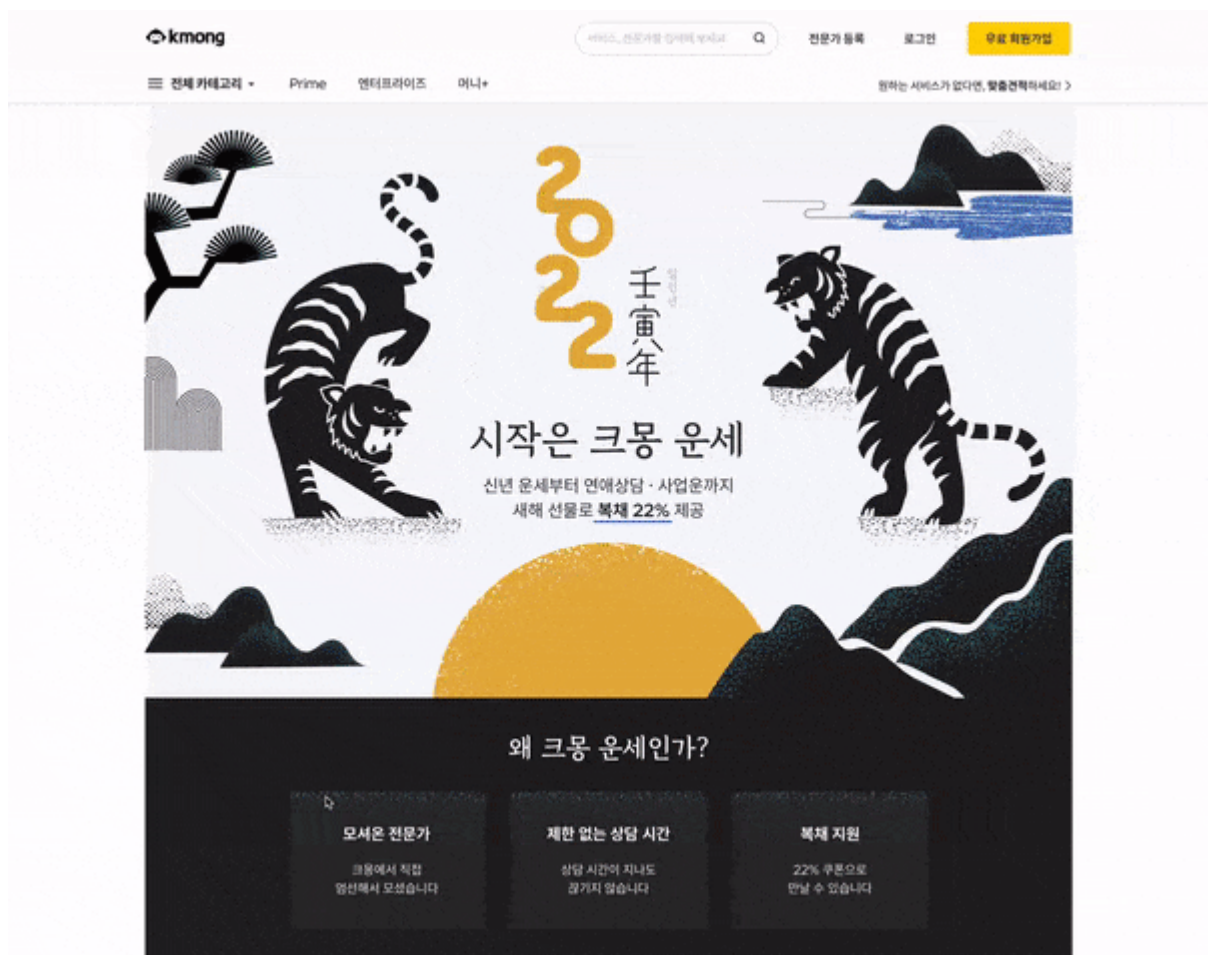
필요한 경우 포커스, 인풋 입력값 같은 클라이언트 상태를 유지하며 여러 번 데이터를 가져오고

리렌더링하여 전달할 수 있습니다.

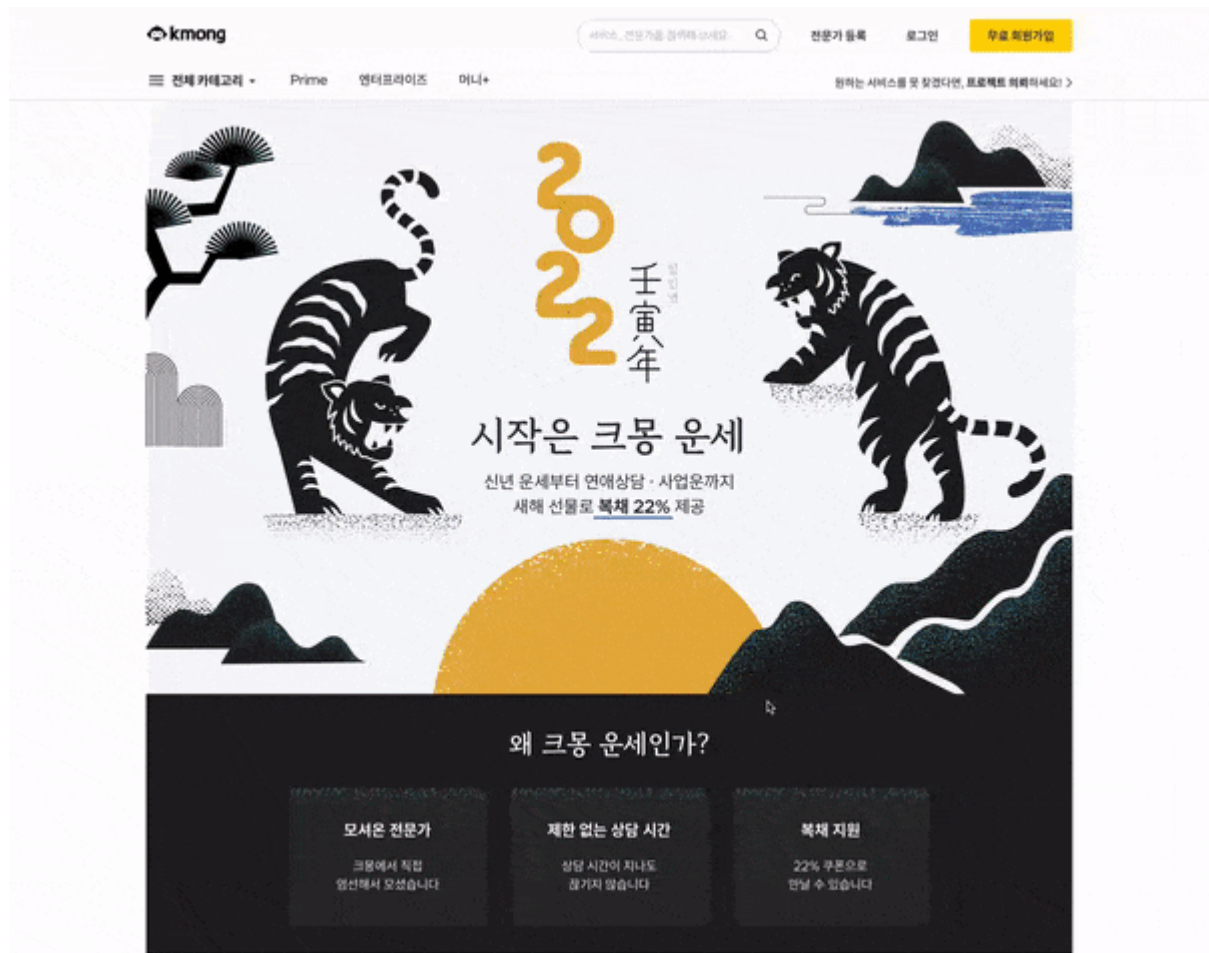
따라서 사용 예시를 들어보자면 초기에는 html페이지를 빠르게 보여주기 위해 SSR을 사용하고, 서버 컴포넌트로는 클라이언트로 전송되는 자바스크립트 번들 사이즈를 감소시킨다면 사용자에게 훨씬 빠르게 페이지를 보여줄 수 있을 것입니다.

아래의 gif를 보면 react에서 nextjs로 마이그레이션을 했을 경우 얼마나 초기렌더링속도차이가 나는지 알 수 있습니다.

react에서 nextjs로 마이그레이션한 크몽예시)



마이그레이션 전 렌더링



마이그레이션 후 렌더링

위의 차이점이 작다고 생각할 수도 있지만

<https://zdnet.co.kr/view/?no=20190418142445>

위의 링크 자료를 보면

"49%의 고객이 2초 이하의 로딩 속도를 기대하고, 18%의 고객은 1초도 지체없는 로딩 속도를 기대한다는 2년 전 자료가 있다"며 "로딩 속도 1초가 빨라지면 아마존 판매량이 1% 증가하고 구글 검색량 0.2% 증가, 월마트의 전환율이 2% 증가한다" 라고 합니다.

작은 숫자 처럼 보이지만 돈으로 환산할 경우

"아마존 판매량을 환산하면 68억달러, 구글의 검색량 증가로 4억5천만달러의 광고 노출, 월마트의 2억4천400만달러의 매출이 늘어나는 셈" 이라고 합니다.

따라서 초기화면에 동작하지 않더라도 시각적인 자료가 보이는게 굉장히 중요하다는 것을 알 수 있습니다.

발표를 준비하다 발견한 이상한 점

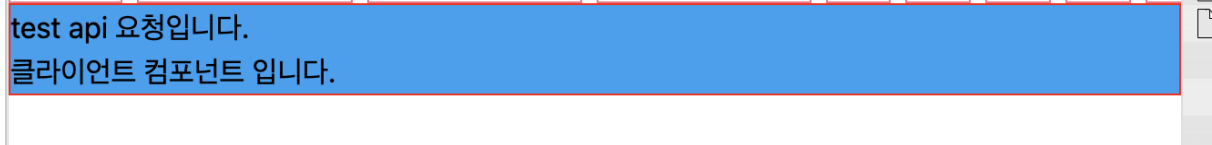
이상하게 테스트를 하는데 계속 클라이언트 컴포넌트에서 html이 채워진 상태로 오는 것을 발견했습니다.

클라이언트에서 렌더링을 진행하기 때문에 빈 html이 올 것이라고 예상 했던 것과 다른 결과가 나와서 계속

구글링을 해본결과 이런 글을 읽게 됐습니다.

✳ "use client"를 사용한다고 모든 것을 다 브라우저에서 실행하지 않고, 서버 측에서 데이터를 받아와 props로 넘기는 등, 서버 내에서 처리가 가능한 부분은 처리한 HTML 파일을 보내준다.

즉, 클라이언트 컴포넌트라고 해서 모든 것을 클라이언트 단에서 해결하는 게 아니라 서버에서 처리가 가능한 부분은 처리한 후 html파일을 전달해준다는 것을 알게 되었습니다.



test api 요청입니다.
클라이언트 컴포넌트 입니다.

(클라이언트 컴포넌트 화면 결과값)

```
<div class="list bg-[#1da1f2]">
  <div></div>
  <div>클라이언트 컴포넌트 입니다.</div>
</div>
```

(네트워크에서 확인한 받아온 결과값)

위의 사진을 보면 api를 요청한 "test api 요청입니다"라는 내용이 네트워크에는 없는 것을 확인할 수 있었고

그냥 하드코딩으로 적은 "클라이언트 컴포넌트 입니다." 라는 내용을 처음부터 서버에서 받아오는 것을 확인할 수 있습니다.

결론

nextjs를 사용하는 이유는 너무나도 많지만

(폴더기반 자동라우팅, 새로디자인한 서버API 기능, 쉬운 DB연결, 직관적인 rendering 전략 선택기능, hydration없는 server-side rendering, 파워풀한 캐싱, 이미지와 폰트 최적화)

이번에는 제가 생각하는 react에서 nextjs로 바뀌어야 되는 대표적인 이유와 그 차이점에 대해서 알아봤고 이 발표로 인해 여러분의 개발지식에 도움이 됐으면 좋겠습니다. 감사합니다.

참고자료 :

<https://velog.io/@syoun9125/Next.js-기본-개념-1-Next.js-란-Next.js를-왜-사용할까-Next.js의-장점은>

<https://yeongchan1228.tistory.com/132>

<https://www.youtube.com/watch?v=qghtdTV7Kqk>

<https://haesoo9410.tistory.com/m/404>

<https://velog.io/@okko8522/RSC와-SSR의-차이>

<https://tech.kakaopay.com/post/react-server-components/#no-client-server-waterfall>

<https://blog.kmong.com/크몽-프론트엔드팀에서-next-js를-사용하는-이유-5474e10bc72b>

<https://solo5star.tistory.com/44>

<https://velog.io/@tnrud4685/Next.js-13버전>

<https://adjh54.tistory.com/53>