

NEXT.JS



섹션 22. Next.js 요약

2024.07.21
두선아



table of contents

—

Next.js 요약하기

현재의 Next.js

pages & app directory

Next.js 요약하기

3장 요약 해주세요 👁👁



Next.js란 무엇인가요?

“프로덕션 앱 개발을 위한 리액트 프레임워크”

Next.js는 리액트의 기능으로 만들어졌습니다.

프레임워크는 라이브러리보다 범위가 넓고, 명확한 규칙이 있습니다.

NEXT.JS는 리액트에 핵심 기능을 추가합니다.

프로덕션 앱에서 보편적으로 필요한 기능을 추가하기 위해 서드 파티 라이브러리 설치 과정을 줄여줍니다.

요약 시작!



서버 사이드 렌더링(SSR) 지원

클라이언트가 아닌 서버 사이드에서 콘텐츠를 준비하는 기능을 제공합니다.

☞ React 앱의 소스 코드에는 HTML 스켈레톤과 엔트리 포인트가 존재합니다. ex) `<div id="app"></div>`
클라이언트에서 렌더링이 진행되며, 사용자는 데이터 패칭 과정을 경험합니다.

☞ Next.js는 서버 사이드 렌더링을 기본으로 자동으로 진행합니다.
완성된 페이지를 화면에 그리는 유저 경험을 제공하고, 검색이 필요한 콘텐츠는 SSR을 통해 SEO를 최적화할 수 있습니다.

또한 빌드 (SSR) -> 배포 -> 클라이언트에서 hydrate을 거친 후
사용자는 여전히 React 앱의 빠른 상호작용을 경험할 수 있습니다.

파일 기반 라우팅

- ☞ React는 기본적으로 라우팅 기능이 없습니다.
일반적으로 react-router-dom과 같은 서드 파티 라이브러리를 사용합니다.
- ☞ Next.js는 파일과 폴더를 사용하여 라우팅을 표현하여, 코드가 이해하기 쉽고 줄어듭니다.

풀스택, Node.js 서버

서버 사이드(백엔드) 코드를 앱에 쉽게 추가할 수 있습니다.


데이터베이스 및 아키텍처 접근, 인증 처리 등을 위한
서버 사이드 코드를 작성할 수 있습니다.

현재의 Next.js

2024년 7월 20일



server action, app 디렉토리와 Next.js 버전



October 26th, 2023


Next.js 14

As we announced at [Next.js Conf](#), Next.js 14 is our most focused release with:

- **Turbopack**
 - **53% faster** local server startup
 - **94% faster** code updates
- [Server Actions \(Stable\)](#)
- [Partial Prerendering \(Preview\)](#)
- [Next.js Learn \(New\)](#)

[Read More](#)

<2023년 8월 26일>
Next.js 14의 업데이트에서
서버 액션이 안정화되었습니다.



May 23rd, 2024

Next.js 15 RC

The Next.js 15 Release Candidate (RC) is now available. This early version allows you to test the latest features before the upcoming stable release.

- [React RC](#)
- [Caching defaults changes](#)
- [Incremental Partial Prerendering adoption](#)
- [next/after \(Experimental\)](#)
- [New create-next-app design](#)
- [Bundling external packages \(Stable\)](#)

[Read More](#)

<2024년 5월 23일>
Next.js 15 RC가 공개되었습니다.
최종 버전이 아니며
메이저 버전이 바뀌어
브레이킹 체인지가 포함되어 있습니다.

[공홈 블로그 참고](#)

May 23rd, 2024

Next.js 15 RC

The Next.js 15 Release Candidate (RC) is now available. This early version allows you to test the latest features before the upcoming stable release.

- [React RC](#)
- [Caching defaults changes](#)
- [Incremental Partial Prerendering adoption](#)
- [next/after \(Experimental\)](#)
- [New create-next-app design](#)
- [Bundling external packages \(Stable\)](#)

[Read More](#)

April 11th, 2024

Next.js 14.2

Next.js 14.2 includes development, production, and caching improvements.

- [Turbopack \(Release Candidate\)](#)
- [Caching Improvements](#)
- [Build and Production Improvements](#)
- [Errors DX Improvements](#)

[Read More](#)

January 18th, 2024

Next.js 14.1

Next.js 14.1 includes developer experience improvements including:

- [Improved Self-Hosting](#)
- [Turbopack Improvements](#)
- [DX Improvements](#)
- [Parallel & Intercepted Routes](#)
- [next/image Improvements](#)

[Read More](#)

October 26th, 2023

Next.js 14

As we announced at [Next.js Conf](#), Next.js 14 is our most focused release with:

- [Turbopack](#)
 - **53% faster** local server startup
 - **94% faster** code updates
- [Server Actions \(Stable\)](#)
- [Partial Prerendering \(Preview\)](#)
- [Next.js Learn \(New\)](#)

[Read More](#)

October 23rd, 2023

How to Think About Security in Next.js

React Server Components (RSC) in App Router is a novel paradigm that eliminates much of the redundancy and potential risks linked with conventional methods. Given the newness, developers and subsequently security teams may find it challenging to align their existing security protocols with this model.

[Read More](#)

September 19th, 2023

Next.js 13.5

Next.js 13.5 improves local dev performance and reliability with:

- **22% faster** local server startup
- **29% faster** HMR (Fast Refresh)
- **40% less** memory usage
- [Optimized Package Imports](#)
- [next/image Improvements](#)
- [And over 438 bugs patched!](#)

[Read More](#)

애플리케이션을 새롭게 개발한다면
Next.js 14.2와 app 디렉토리를
사용하는 것이 좋겠다고 생각합니다🤔

여러분은 어떻게 생각하시나요?



pages & app directory

Next.js 12 -> 13 컴포넌트 변환

<Image>, next/image

(코드 변환에 codemod를 사용합니다.)

next-image-to-legacy-image: 안전하게 next/legacy/image로 변환

next-image-experimental: 위험하게 인라인 스타일 추가하고, 사용하지 않는 props 제거

<Link>에서 <a> 제거

`npx @next/codemod@latest new-link .`

next/font

[구글 폰트 적용기](#) (2023년 3월, velog)

알아두기~



API Route Handler

파일 구조를 `api/요청/route.ts`로 변환해야 합니다.

- `pages/api/hello.ts` → `app/api/hello/route.ts`

라우트 핸들러는 HTTP 메서드별 함수를 export합니다.

`handler()`를 내보내는 것이 아니라, HTTP 메서드에 따라 `GET()`, `POST()` 등을 구분하므로 API 라우트 핸들러를 더 세분화 & 구조화할 수 있습니다.

Routing Hooks

👉 pages: next/router의 `useRouter()`를 사용

👉 app: next/navigation의 `useRouter()`, `usePathname()`, `useSearchParams()` 사용

> <https://nextjs.org/docs/app/building-your-application/upgrading/app-router-migration#step-5-migrating-routing-hooks>

데이터 패칭 메소드

pages 디렉토리의 getServerSideProps, getStaticProps와 같은 데이터 패칭 함수는 fetch 기반 API 요청 기능 및 비동기 서버 컴포넌트로 대체되었습니다.

```
/**
 * 이전 pages/index.tsx에서의 예
 * **/

// 1 `getStaticProps`는 빌드 시 페이지를 사전 렌더링합니다.
// `revalidate` 옵션을 추가하여 일정 시간마다 페이지를 재검토합니다.
export async function getStaticProps() {
  const res = await fetch(`https://api.example.com/data`);
  const data = await res.json();

  return {
    props: { data },
    revalidate: 10, // 페이지는 10초마다 새로 고쳐집니다.
  };
}

// 2 `getServerSideProps`는 매 요청마다 서버에서 데이터를 가져와 페이지에 props로 전달합니다.
export async function getServerSideProps() {
  const res = await fetch(`https://api.example.com/data`);
  const data = await res.json();

  return { props: { data } };
}
```

```
/**
 * app/page.tsx에서의 새로운 접근법
 * **/

export default async function Page() {
  // 이 요청은 수동으로 무효화될 때까지 캐시됩니다.
  // 1 `getStaticProps`와 유사합니다.
  // `force-cache`는 기본값이므로 생략할 수 있습니다.
  const staticData = await fetch(`https://...`, { cache: "force-cache" });

  // 이 요청은 10초 동안 캐시됩니다.
  // `getStaticProps`의 `revalidate` 옵션과 유사합니다.
  const revalidatedData = await fetch(`https://...`, {
    next: { revalidate: 10 },
  });

  // 이 요청은 매 요청마다 다시 가져와야 합니다.
  // 2 `getServerSideProps`와 유사합니다.
  const dynamicData = await fetch(`https://...`, { cache: "no-store" });

  return <div>...</div>;
}
```

요청 접근하기

- 👉 pages: req 객체를 가져와서 요청의 쿠키와 헤더를 조회
- 👉 app: request의 데이터를 읽기위한 read-only 함수 존재
 - headers(): Web Headers API를 기반으로 하며,
서버 컴포넌트 내에서 요청 헤더를 가져오는 데 사용
 - cookies(): Web Cookies API를 기반으로 하며,
서버 컴포넌트 내에서 쿠키를 가져오는 데 사용

```
// `pages` 디렉토리
export async function getServerSideProps({ req, query }) {
  const authHeader = req.getHeaders()['authorization'];
  const theme = req.cookies['theme'];

  return { props: { ... } };
}

export default function Page(props) {
  return ...
}
```

```
// `app` 디렉토리
import { cookies, headers } from 'next/headers'

async function getData() {
  const authHeader = headers().get('authorization')

  return '...'
}

export default async function Page() {
  // 서버 컴포넌트 내에서 `cookies()` 또는 `headers()`를
  // 직접 사용하거나 데이터 가져오기 함수에서 사용할 수 있습니다.
  const theme = cookies().get('theme')
  const data = await getData()
  return '...'
}
```

정적 생성

```
// `app` 디렉토리
import PostLayout from "@components/post-layout";

export async function generateStaticParams() {
  return [{ id: "1" }, { id: "2" }];
}

async function getPost(params) {
  const res = await fetch(`https://.../posts/${params.id}`);
  const post = await res.json();

  return post;
}

export default async function Post({ params }) {
  const post = await getPost(params); // path가 아니라 params

  return <PostLayout post={post} />;
}
```

👉 pages: getStaticProps 및 getStaticPaths 사용

getStaticProps 함수를 사용하여
페이지를 빌드 시 사전 렌더링

getStaticPaths 함수를 사용하여
빌드 시 사전 렌더링할 동적 경로를 정의

👉 app: getStaticPaths -> generateStaticParams

getStaticProps와 getServerSideProps가
더 이상 필요하지 않기 때문에, 단독으로 사용하기 위한 함수

반환하는 경로 매개변수의 API가 단순화되어 있으며
레이아웃 내에서 사용할 수 있다.

generateStaticParams의 반환 형태는
중첩된 매개변수 객체 배열이 아니라 세그먼트 배열!

fallback -> dynamicParams

```
// `app` 디렉토리
export const dynamicParams = true;

export async function generateStaticParams() {
  return [...]
}

async function getPost(params) {
  ...
}

export default async function Post({ params }) {
  const post = await getPost(params);

  return ...
}
```

👉 pages:

getStaticPaths에서 반환되는 fallback 속성을 사용하여 빌드 시 사전 렌더링되지 않은 페이지의 동작을 정의

👉 app:

dynamicParams를 true로 설정하면(기본값) 생성되지 않은 경로 세그먼트가 요청될 때 서버 렌더링되고 캐시

false라면 generateStaticParams에 포함되지 않은 경로는 404

증분 정적 생성, ISR (Incremental Static Regeneration)

👉 pages: getStaticProps의 revalidate 필드 사용

👉 app: fetch()에서 revalidate 옵션 사용

API route

👉 app: Web Request, Response API를 사용한 커스텀 핸들러

클라이언트에서 외부 API를 숨겨서 호출하기 위해 이전 API 라우트를 사용했던 경우, 서버 컴포넌트를 사용하면 데이터를 안전하게 가져올 수 있습니다.

Styling

👉 pages: 전역 스타일시트를 pages/_app.js에서만 사용할 수 있습니다.

👉 app: 전역 스타일을 레이아웃, 페이지, 또는 컴포넌트에 추가할 수 있습니다.

<https://nextjs.org/docs/app/building-your-application/upgrading/app-router-migration#step-7-styling>

Thank You!

—

reference

유데미 강의: <https://www.udemy.com/course/nextjs-react-incl-two-paths>

Next.js 공식문서: <https://nextjs.org/docs>

