

10. 서버 컴포넌트와 클라이언트 컴포넌트

Server 컴포넌트와 Client 컴포넌트의 동작 방식을 이해하려면, 다음 두 가지 웹 기본 개념을 아는 것이 도움이 됩니다:

- 애플리케이션 코드가 실행되는 환경: 서버(Server)와 클라이언트(Client)
 - 서버 코드와 클라이언트 코드를 나누는 네트워크 경계(Network Boundary)
-

서버와 클라이언트 환경

웹 애플리케이션에서:

- **클라이언트(Client)**
사용자의 브라우저를 의미합니다. 브라우저는 서버에 요청을 보내고, 받은 응답을 사용자 인터페이스(UI)로 변환합니다.
- **서버(Server)**
데이터 센터에 있는 컴퓨터입니다. 애플리케이션 코드를 저장하고 클라이언트로부터 요청을 받아 계산을 수행하고 응답을 돌려줍니다.

각 환경은 고유한 기능과 제약 조건이 있습니다.

예를 들어, 렌더링과 데이터 페칭을 서버에서 수행하면, 클라이언트로 전송하는 코드량을 줄일 수 있어 성능이 향상됩니다.

하지만 UI를 상호작용 가능하게 만들려면, 최종적으로는 클라이언트에서 DOM 업데이트가 필요합니다.

결국, 서버용 코드와 클라이언트용 코드는 항상 다를 수밖에 없습니다.

네트워크 경계(Network Boundary)

네트워크 경계란 서버와 클라이언트 간의 논리적 구분선을 의미합니다.

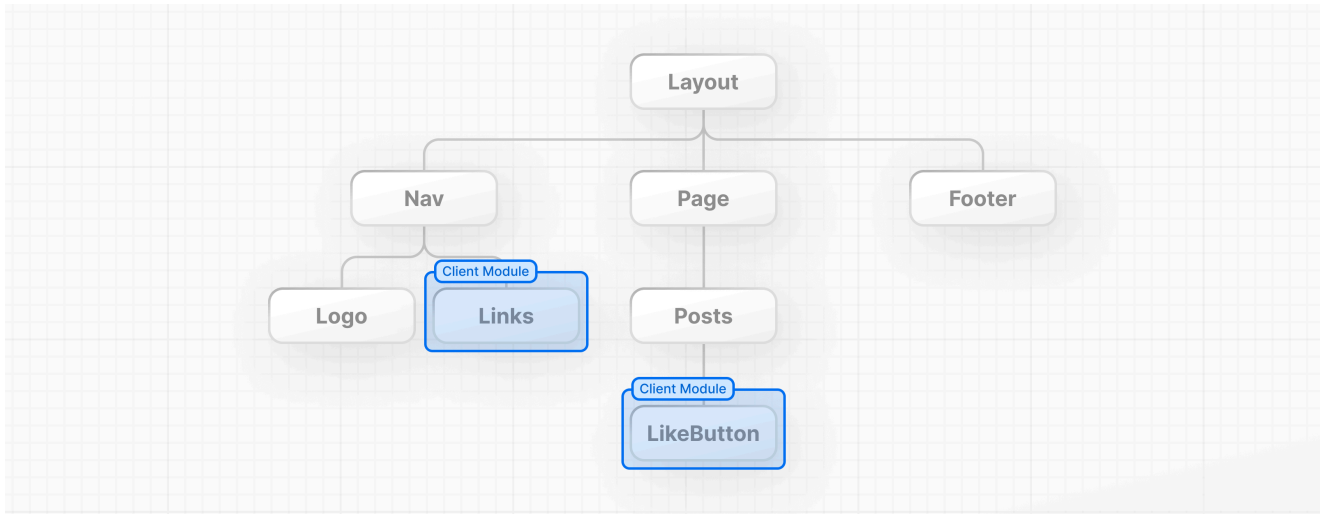
React에서는 컴포넌트 트리 안에서 어디에 이 경계를 둘지 개발자가 결정할 수 있습니다.

예시:

- 사용자의 게시글 목록(Post)을 서버에서 렌더링하고 (Server Components)
- 각각의 게시글마다 "좋아요 버튼(LikeButton)"은 클라이언트에서 렌더링 (Client Components)

또 다른 예시:

- Nav 컴포넌트(메뉴)는 서버에서 렌더링하고
- 활성 링크 표시처럼 인터랙션이 필요한 부분은 클라이언트에서 렌더링



(※ 정상 이미지 복원)

서버와 클라이언트 컴포넌트가 동작하는 방식

Next.js는 컴포넌트를 서버용 모듈 그래프와 클라이언트용 모듈 그래프로 나눕니다.

1. 서버 컴포넌트가 렌더링된 후,
2. **React Server Component Payload (RSC Payload)** 라는 특별한 데이터가 클라이언트로 전송됩니다.

RSC Payload는:

- 서버 컴포넌트의 렌더링 결과
- 클라이언트 컴포넌트를 렌더링해야 할 자리(플레이스홀더)와 해당 컴포넌트에 필요한 JS 파일 참조

를 포함합니다.

React는 이 정보를 사용해 클라이언트에서 DOM을 완성합니다.

Client 컴포넌트 사용하기

지난 챕터에서 봤듯이, Next.js는 기본적으로 **Server Component**를 사용합니다.

그런데 `useState` 같은 혹은 **Client Component** 안에서만 사용할 수 있습니다.
(브라우저에서만 동작해야 하니까요)

해결 방법: LikeButton 컴포넌트 분리하기

1. app 폴더 안에 like-button.js 파일을 새로 만듭니다:

/app/like-button.js

```
export default function LikeButton() {}
```

2. 기존의 버튼(button) 요소와 클릭 핸들러를 옮깁니다:

/app/like-button.js

```
export default function LikeButton() {  
  function handleClick() {  
    setLikes(likes + 1);  
  }  
  
  return <button onClick={handleClick}>Like ({likes})</button>;  
}
```

3. likes 상태와 useState import도 옮깁니다:

```
import { useState } from 'react';  
  
export default function LikeButton() {  
  const [likes, setLikes] = useState(0);  
  
  function handleClick() {  
    setLikes(likes + 1);  
  }  
  
  return <button onClick={handleClick}>Like ({likes})</button>;  
}
```

4. 그리고 파일 맨 위에 다음을 추가해 Client Component임을 선언합니다:

```
'use client';  
  
import { useState } from 'react';  
  
export default function LikeButton() {  
  const [likes, setLikes] = useState(0);  
  
  function handleClick() {  
    setLikes(likes + 1);  
  }  
}
```

```
    return <button onClick={handleClick}>Like ({likes})</button>;  
  }  
}
```

5. `page.js`에서는 `LikeButton`을 가져와서 사용합니다:

/app/page.js

```
import LikeButton from './like-button';  
  
function Header({ title }) {  
  return <h1>{title ? title : 'Default title'}</h1>;  
}  
  
export default function HomePage() {  
  const names = ['Ada Lovelace', 'Grace Hopper', 'Margaret Hamilton'];  
  
  return (  
    <div>  
      <Header title="Develop. Preview. Ship." />  
      <ul>  
        {names.map((name) => (  
          <li key={name}>{name}</li>  
        ))}  
      </ul>  
      <LikeButton />  
    </div>  
  );  
}
```

6. 파일을 저장하고 브라우저를 새로고침하면, 더 이상 에러 없이 Like 버튼이 동작합니다!

Fast Refresh 기능


Next.js에는 기본적으로 **Fast Refresh** 기능이 활성화되어 있습니다.
코드를 수정하고 저장하면, 브라우저가 자동으로 변경사항을 즉시 반영합니다.

개발 생산성을 크게 높여주는 기능입니다!

요약

- 서버와 클라이언트 환경을 이해했습니다.
- 서버 컴포넌트는 성능 향상을 위해 기본값입니다.

- 클라이언트 상호작용이 필요한 경우에는 'use client' 를 추가하여 클라이언트 컴포넌트로 선언해야 합니다.

 추가 참고 자료:

- [Server Components 공식 문서](#)
- [Client Components 공식 문서](#)
- [React "use client" 설명 \(react.dev\)](#)