


[사용자] useSearchParams() should be wrapped in a suspense boundary at page 에러를 해결하며, 쿼리 스트링에 대해 알아보기

구현 어려움 정도	☆
구현 완료	✓
기간	@December 25, 2024
정리 완료	✓

 **useSearchParams** 오류를 해결하기 전, 쿼리 스트링에 대해 알아보겠습니다.

쿼리 스트링의 정의와 필요성

쿼리 스트링은 URL의 한 부분으로 요청하는 url에 부가 정보를 포함할 때 사용합니다.

기존 url은 단순한 형태의 요청과 응답을 주고 받았지만 쿼리 스트링을 사용하면 조건에 맞게 정렬된 특정 형태의 정보를 요청하고 받을 수 있습니다.

예를 들어, 규모가 크고 복잡한 애플리케이션의 상품 종류가 1000개라면 상품 리스트 페이지에서 1억개의 상품 정보를 불러오는 것은 비효율적 → 1억개의 데이터를 불러오는 시간도 문제지만 실제 유저는 판매량, 최신순, 리뷰 평점순 처럼 특정 기준으로 편집된 정보를 보길 원합니다.

쿼리 스트링의 형태

쿼리 스트링은 문자열의 형태이며 `key=value` 로 표현합니다. url의 일부이므로 `?` 를 통해 여기부터 시작이라고 표시해야 하고 각 페어의 구분은 `&` 로 합니다.

```
// 인기순으로 정렬된 정보
https://www.example.com/products?sort=popular

// 인기순으로 정렬된 정보를 내림차순으로 보고 싶다면
https://www.example.com/products?sort=popular&direction=desc
```

쿼리 스트링을 포함한 라우팅

쿼리 스트링은 url에 부가적인 정보를 포함하는 것이므로 라우터 컴포넌트에도 특별한 설정이 필요 없습니다. 아래처럼 링크 역할을 하는 컴포넌트에 쿼리 스트링이 포함된 주소를 전달하면 됩니다.

```
// Link 컴포넌트
<Link to="/list?sort=popular" />
// useNavigate
navigate("/list?sort=popular")
```

컴포넌트에서 쿼리 스트링 가져오기

- react-router-dom에서 쿼리 스트링 값을 가져올 수 있는 hook으로는 `useLocation`, `useSearchParams` 두 가지가 있습니다.
- `useLocation` hook은 현재의 Location 객체를 반환합니다.
 - 현재 url에 포함된 여러가지 정보

```
// 해당 혹은 호출하고
import { useLocation } from "react-router-dom";

// 컴포넌트 안에서 데이터를 변수에 담고 확인
```

```
const location = useLocation();
console.log(location);
```

이렇게 console.log로 출력해보면, 콘솔창에 여러 객체가 나오고 그 중 search 프로퍼티가 쿼리 스트링 값을 담고 있는 것을 볼 수 있습니다. 이를 활용해 쿼리 스트링 값을 가져와 사용할 수 있습니다.

```
console.log(location.search); // => ?sort=popular
```

이렇게 가져온 값에서 popular만 뽑아서 사용하려면 별도의 작업을 해야하므로 복잡합니다. 페이지가 여러개라면 더 복잡해집니다.

이럴 때, 다양한 메서드를 제공해 원하는 값을 가져올 수 있도록 하는 것이 `useSearchParams` hooks 입니다.

자주 사용하는 메서드를 살펴보겠습니다.

값을 읽어오는 메서드

- `searchParams.get(key)`
 - 특정한 key의 value를 가져오는 메서드
 - 해당 key의 value가 2개라면 제일 먼저 나온 value만 리턴
- `searchParams.getAll(key)`
 - 특정 key에 해당하는 모든 value를 가져오는 메서드
- `searchParams.toString()`
 - 쿼리 스트링을 string 형태로 리턴

값을 변경하는 메서드

- `searchParams.set(key, value)`
 - 인자로 전달한 key 값을 value로 설정
 - 기존에 값이 존재했다면 그 값은 삭제됨
- `searchParams.append(key, value)`
 - 기존 값을 변경하거나 삭제하지 않고 추가하는 방식

코드 예시(Next.js)

1. action에서 signup 성공 시, 원하는 url과 쿼리 스트링으로 리다이렉트 해줍니다.

```
// signup.ts
"use server";

import { db } from "@/db";
import { user } from "@/db/schema";
import { redirect } from "next/navigation";

export const signUp = async (_, { formData }: { formData: FormData }) => {
  // 1. validate Fields 필드 유효성 검사를 서버에서 다시 한 번 검증
  ...

  // 2. 존재하는 사용자인지 체크
  ...

  // 4. 성공/실패처리
  ...
}
```

```
// 3. 성공 시 insert db
...

// 5. 성공 시, 원하는 url과 쿼리 스트링으로 리다이렉트
redirect("/login?signupSuccess=true");
}
```

2. `searchParams.get` hook을 사용해 key와 value가 `signupSuccess === "true"`라면, toast 성공 메시지를 띄워줍니다.

a. 이 때, 상단에 `"use client"` 를 작성해 클라이언트 컴포넌트임을 명시해줘야 합니다.

```
// LoginForm.tsx
"use client";

import toast from "react-hot-toast";
import { useSearchParams } from "next/navigation";

const LoginForm = () => {
  const searchParams = useSearchParams();

  useEffect(() => {
    if (searchParams.get("signupSuccess") === "true") {
      toast.success("회원가입이 완료되었습니다. 로그인 해주세요.");
    }
  }, [searchParams]);

  return (
    <FormCard>
      ...
    </FormCard>
  )
}

export default LoginForm;
```

3. 사용하는 페이지에서 `Suspense`로 감싸줍니다.

```
import { Suspense } from "react";
import LoginForm from "@/components/auth/LoginForm";

const LoginPage = () => {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LoginForm />
    </Suspense>
  );
};

export default LoginPage;
```

Suspense

- `Suspense` 는 React에서 비동기 작업(데이터 로딩, 컴포넌트 동적 로딩 등)을 처리하기 위해 제공되는 기능입니다.
- React 18부터 `Suspense` 는 클라이언트 사이드에서 비동기 로딩을 처리하는데 중요한 역할을 합니다.
- Next.js에서는 Server components와 Client Components를 구분합니다. 서버에서 렌더링되는 컴포넌트에서는 클라이언트 전용 상태나 hook (`useEffect`, `useState`, `useSearchParams`)을 사용할 수 없습니다.

- 이러한 컴포넌트를 서버에서 렌더링할 때 문제가 발생할 수 있는데 Next.js에서는 이를 처리하기 위해 `Suspense` 를 사용합니다.

useSearchParams

```
× useSearchParams() should be wrapped in a suspense boundary at page "/login". Read more: https://nextjs.org/docs/messages/missing-suspense-with-csr-bailout
   at a (/Users/eunjee/Documents/file/crypto/.next/server/chunks/948.js:1:7291)
   at d (/Users/eunjee/Documents/file/crypto/.next/server/chunks/948.js:1:20442)
   at h (/Users/eunjee/Documents/file/crypto/.next/server/app/(auth)/login/page.js:1:4349)
```

- `useSearchParams` 는 클라이언트 컴포넌트에서만 작동합니다.
- Next.js 13 이상 버전에서 `useSearchParams()` 를 사용할 때 **Suspense boundary**로 감싸지 않으면 오류가 발생합니다.
- `Suspense` 는 데이터 로딩을 기다리는 컴포넌트를 안전하게 처리하는 방법으로, 클라이언트에서 데이터를 비동기로 가져오는 경우 유용합니다.

해결 방법

`useSearchParams` 를 사용하는 컴포넌트를 **Suspense**로 감싸주면 됩니다.

```
import { Suspense } from "react";
import LoginForm from "@/components/auth/LoginForm";

const LoginPage = () => {
  return (
    <Suspense fallback=<div>Loading...</div>>
      <LoginForm />
    </Suspense>
  );
};

export default LoginPage;
```

- 이 때, LoginForm 컴포넌트 상단에 `"use client";` 를 작성해 클라이언트 컴포넌트로 설정해야 함
- `Suspense` 는 **React 18**부터 지원되며, **비동기 데이터를** 기다리는 동안 UI를 기다릴 수 있게 도와줌

2. 비동기 로딩이 아니라면, 굳이 `Suspense`를 사용하지 않고, 상단에 `"use client";` 를 명시해주면 됨

```
"use client";

import LoginForm from "@/components/auth/LoginForm";

const LoginPage = () => {
  return (
    <LoginForm />
  );
};

export default LoginPage;
```

위 두가지 경우의 성능차이 비교

1. Suspense

- 비동기적으로 로딩되는 컴포넌트나 데이터를 처리하는 방법입니다. UI를 중단하지 않고 비동기 로딩을 처리하므로 전체 렌더링 성능을 최적화할 수 있습니다.
- 불필요한 렌더링을 방지하고, 비동기 작업이기 때문에 UI가 끊기지 않도록 합니다.

2. use client

- use client는 클라이언트 전용 코드를 구분하고, 서버 사이드 렌더링에서 제외시키는 방식입니다.
- 클라이언트 사이드에서만 실행되므로 서버에서 해당 컴포넌트를 처리할 필요가 없어져 서버 성능에 좋습니다.
- 클라이언트에서 로딩되는 시점에 컴포넌트가 로드되므로 초기 로딩 성능에서 안좋은 수는 있습니다.

use client는 클라이언트 전용 코드를 실행하기 위해, Suspense는 비동기 로딩을 관리하기 위해 사용됩니다. 각각 사용 용도에 맞게 사용하는 것이 바람직하다고 합니다!