apps in toss

Menu On this page

쿼리 파라미터 사용하기

애플리케이션이 URL 스킴으로 실행될 때, 스킴에 포함된 쿼리 스트링 값을 참조할 수 있어요. 스킴으로 애플리케이션을 실행할 때, 필요한 데이터를 전달하거나 특정 기능을 활성화할 수 있어요.

쿼리 파라미터로 데이터 전달하기

애플리케이션을 실행할 때, URL에 데이터의 키-값 쌍을 쿼리 파라미터 형태로 추가할 수 있어요.

예를 들면 다음과 같은 형태에요.

```
intoss://{서비스 이름}?key1=value1&key2=value2
```

이 예시에서는 `test-app` 이라는 이름의 애플리케이션을 실행하고, `name` 과 `age` 데이터를 전달할 수 있어요.

```
intoss://test-app?name=tom&age=10
```

쿼리 파라미터 값 가져오기

`useParams` 훅을 사용하면 애플리케이션이 실행될 때 URL의 쿼리 스트링 값을 쉽게 가져올 수 있어요. 이 훅은 `useParams` 메서드를 통해 특정 키에 해당하는 값을 반환해요.

추가로 `validateParams` 옵션을 활용하면, 화면에서 필요한 쿼리 파라미터를 정의하고 유효성을 검사할 수 있어요. 아래 예제를 참고하세요.

tsx

```tsx
import { BedrockRoute } from "react-native-bedrock";
import { View, Text } from "react-native";
```

```tsx
// 루트 경로('/')에 해당하는 화면 정의
export const Route = BedrockRoute("/", {
  component: Index,
  validateParams: (params) => ({
    // 'name' 키를 필수로 설정하고 타입을 문자열로 변환해요.
    name: params.name as string,
    // 'age' 키를 필수로 설정하고 타입을 숫자로 변환해요.
    age: params.age as number,
  }),
});

function Index() {
  // 'name' 키에 해당하는 쿼리 스트링 값을 가져옵니다.
  const { name, age } = Route.useParams();

  // 또는 다음과 같이 특정 경로에서 값을 가져올 수도 있습니다.
  // const { name, age } = useParams({ from: "/" });

  return (
    <View>
      <Text>이름: {name}</Text>
      <Text>나이: {age}</Text>
    </View>
  );
}
```

쿼리 스트링 값 유효성 검증하기

필수로 포함해야 하는 쿼리 파라미터는 `validateParams` 옵션을 사용해서 유효성을 검사할 수 있어요.

예를 들어, 아래 예시 코드는 `name` 파라미터가 없으면 에러를 발생시켜요.

그래서 필수 쿼리 파라미터가 누락되지 않도록 `validateParams` 옵션을 사용해요.

vanilla valibot zod

tsx

```tsx
import { BedrockRoute } from "react-native-bedrock";
import { View, Text } from "react-native";

export const Route = BedrockRoute("/", {
```

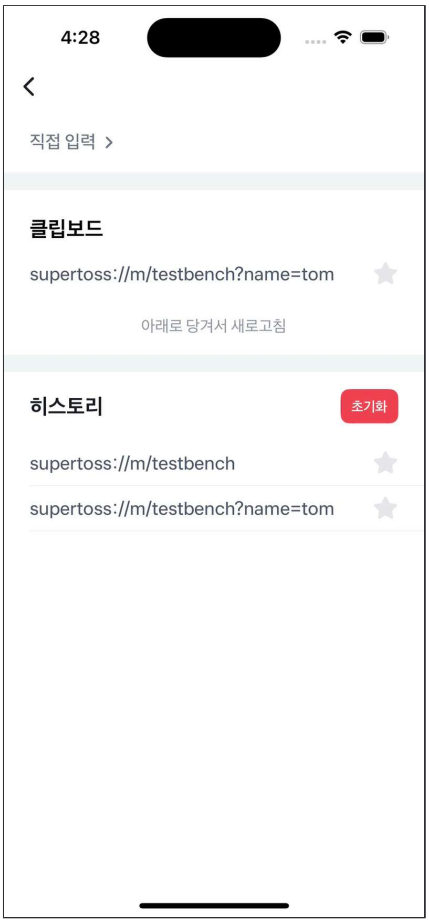
```
  component: Index,
  validateParams: (params) => {
    if (!("name" in params)) {
      throw Error("name is required");
    }
    if (typeof params.name !== "string") {
      throw Error("name must be a string");
    }

    if (!("age" in params)) {
      throw Error("age is required");
    }
    if (typeof params.age !== "number") {
      throw Error("age must be a number");
    }

    return params as {
      name: string;
      age: number;
    };
  },
});

function Index() {
  const { name, age } = Route.useParams();

  return (
    <View>
      <Text>이름: {name}</Text>
      <Text>나이: {age}</Text>
    </View>
  );
}
```



쿼리 파라미터 값 변환하기

`BedrockRoute.parserParams` 옵션을 사용하면 쿼리 스트링으로 전달된 `string` 값을 원하는 타입으로 변환할 수 있어요.

기본적으로 `useParams` 는 숫자, 문자열, 배열, 객체 같은 대부분의 단순 타입은 자동으로 변환하기 때문

에 파서를 직접 재정의해야 할 일은 많지 않아요.
하지만 복잡한 데이터 구조를 사용해야 할 때나 특정한 params를 지우고 싶을 때는 파서를 직접 정의해서 원하는 타입으로 변환할 수 있어요.

`parserParams` 옵션의 결과가 `validateParams` 옵션에 전달되기 전에 변환됩니다.

기본 파서를 사용한 타입 변환

기본 파서를 활용하면 쿼리 스트링 값이 자동으로 적절한 타입으로 변환돼요. 아래 예제는 쿼리 파라미터를 타입에 맞게 변환하는 방법을 보여줘요.

tsx

```tsx
import { BedrockRoute } from "react-native-bedrock";
import { View, Text } from "react-native";

// URL 예시: intoss://test-app?name=tom&age=10&arr=1,2,3&obj={"name":"jane","age'
export const Route = BedrockRoute("/", {
  component: Index,
  validateParams: (params) => ({
    // 기본 파서로 인해 쿼리 파라미터 값을 올바른 타입으로 자동으로 변환
    name: params.name as string, // 문자열로 변환
    age: params.age as number, // 숫자로 변환
    arr: params.arr as string[], // 배열로 변환
    obj: params.obj as { name: string; age: number }, // 객체로 변환
  }),
});

function Index() {
  const { name, age, arr, obj } = Route.useParams();

  return (
    <View>
      <Text>
        이름: {name}, 타입: {typeof name}
      </Text>
      <Text>
        나이: {age}, 타입: {typeof age}
      </Text>
      <Text>
        배열: {JSON.stringify(arr)}, 타입: {typeof arr}
      </Text>
      <Text>
        객체: {JSON.stringify(obj)}, 타입: {typeof obj}
```

```
        </Text>
      </View>
    );
  }
```

파서 재정의

`parserParams` 옵션을 사용하면 기본 파서로 처리하기 어려운 query parameter를 변환하는 함수를 직접 정의해서 사용할 수 있어요. 예를 들어, 특정 파라미터(`referer`)를 제거하고 나머지 파라미터를 기본 파서로 처리하는 방법을 아래 코드에서 보여줘요.

tsx

```tsx
import { BedrockRoute, defaultParserParams } from "react-native-bedrock";
import { View, Text } from "react-native";

// URL 예시: intoss://test-app?name=tom&age=10&referer=https://google.com
export const Route = BedrockRoute("/", {
  component: Index,

  // 특정 파라미터를 제거하고 나머지를 기본 파서로 처리
  parserParams: (params) => {
    const { referer, ...rest } = params;
    return defaultParserParams(rest);
  },

  validateParams: (params) => {
    // 여기서 `params`는 parserParams 함수에서 변환된 값이에요.
    // 즉, `referer`는 이미 제거된 상태로 전달돼요.
    return {
      name: params.name,
      age: params.age,
    } as {
      name: string;
      age: number;
    };
  },
});

// 컴포넌트에서 파라미터 사용
function Index() {
  const { name, age } = Route.useParams();
```

```
  return (
    <View>
      <Text>
        이름: {name}, 타입: {typeof name}
      </Text>
      <Text>
        나이: {age}, 타입: {typeof age}
      </Text>
    </View>
  );
}
```

중복된 쿼리 파라미터 주의사항

만약 같은 이름의 쿼리 파라미터가 여러 번 사용되면, 해당 값은 배열로 반환돼요. 예를 들어, `age` 파라미터가 두 번 포함되면 다음과 같이 처리돼요.

js

```
// 스킴: `intoss://test-app?name=tom&age=10&age=20`
const params = useParams({
  from: "/",
});

// params
{ name: 'tom', age: [10, 20] }
```

레퍼런스

- useParams

이전 버전 문서가 필요할 때

이전 버전의 문서는 쿼리 파라미터 사용하기에서 확인할 수 있어요.

Previous page
화면이 사용자에게 보이는지 확인하기

Next page
보안이 필요한 화면에서 캡처 차단하기