<form>

내장 브라우저 <form> 컴포넌트로 정보 제출을 위한 대화형 컨트롤을 만들 수 있습니다.

```
<form action={search}>
    <input name="query" />
    <button type="submit">검색</button>
</form>
```

레퍼런스

<form>

정보 제출을 위한 대화형 컨트롤을 생성하기 위해, 내장 브라우저 <form> 컴포넌트를 렌더링하세요.

```
<form action={search}>
    <input name="query" />
    <button type="submit">검색</button>
</form>
```

아래 예시를 참고하세요.

Props

`<form>` 은 모든 공통 엘리먼트 Props를 지원합니다.

`action` : a URL or function. When a URL is passed to `action` the form will behave like the HTML form component. When a function is passed to `action` the function will handle the form submission in a Transition following the Action prop pattern. The function passed to `action` may be async and will be called with a single argument containing the form data of the submitted form. The `action` prop can be overridden by a `formAction` attribute on a `<button>` , `<input type="submit">` , or `<input type="image">` component.

주의 사항

- 함수를 `action` 이나 `formAction` 에 전달하면, HTTP 메서드는 `method` 프로퍼티의 값과 관계없이 POST로 처리합니다.

사용법

클라이언트에서 폼 제출 처리하기

폼이 제출될 때 함수를 실행하기 위해, 폼의 `action` 프로퍼티에 함수를 전달하세요. `formData` 가 함수에 인수로 전달되어, 폼에서 전달된 데이터에 접근할 수 있습니다. 이 점이 URL만 받던 기존 HTML action과의 차이점입니다. After the `action` function succeeds, all uncontrolled field elements in the form are reset.

App.js 다운로드 새로고침 Clear 포크

```jsx
export default function Search() {
  function search(formData) {
    const query = formData.get("query");
    alert(`'${query}'을(를) 검색했습니다.`);
  }
  return (
    <form action={search}>
      <input name="query" />
      <button type="submit">검색</button>
    </form>
  );
}
```

검색

서버 함수에서 폼 제출 처리하기

입력 및 제출 버튼과 함께 `<form>` 을 렌더링하세요. 폼을 제출할 때 해당 함수를 실행하기 위해 서버 함수(`'use server'` 가 표시된 함수)를 폼의 `action` 프로퍼티로 전달하세요.

`<form action>` 에 서버 함수를 전달하면 자바스크립트가 활성화되기 전이나 코드가 로드되기 전에 사용자가 폼을 제출할 수 있습니다. 이는 연결 상태나 기계가 느리거나 자바스크립트가 비활성화된 사용자에게 유용하고, `action` 프로퍼티에 URL이 전달될 때와 폼이 동작하는 방식은 비슷합니다.

`<form>` 의 액션에 데이터를 제공하기 위해 폼 필드의 `hidden` 을 사용할 수 있습니다. 서버 함수는 `formData` 대신 `hidden` 이 적용된 폼 필드 데이터를 불러올 수 있습니다.

```
import { updateCart } from './lib.js';

function AddToCart({productId}) {
  async function addToCart(formData) {
    'use server'
    const productId = formData.get('productId')
    await updateCart(productId)
  }
  return (
    <form action={addToCart}>
        <input type="hidden" name="productId" value={productId} />
        <button type="submit">장바구니에 추가</button>
    </form>
  );
}
```

폼 액션에 따른 데이터를 제공하기 위해 `hidden` 폼 필드를 사용하는 대신에 `bind` 를 호출해 추가 인수를 제공할 수 있습니다. 이렇게 하면 함수에 인수로 전달되는 `formData` 외에 새 인수 (`productId`)가 함수에 바인딩됩니다.

```
import { updateCart } from './lib.js';

function AddToCart({productId}) {
  async function addToCart( productId , formData ) {
    "use server";
    await updateCart(productId)
  }
  const addProductToCart = addToCart. bind (null, productId );
  return (
    <form action={addProductToCart}>
      <button type="submit">장바구니에 추가</button>
    </form>
  );
}
```

<form> 이 서버 컴포넌트에 의해 렌더링되고 서버 함수가 <form> 의 action 프로퍼티에 전달되면, 폼은 점진적으로 향상됩니다.

폼이 제출되는 동안 대기 상태 보여주기

폼이 제출되는 동안 대기Pending 상태를 보여주기 위해, <form> 이 렌더링되는 컴포넌트 안에서 useFormStatus Hook을 호출해 반환된 pending 프로퍼티를 읽을 수 있습니다.

여기 폼이 제출되고 있음을 나타내기 위해 pending 프로퍼티를 사용하였습니다.

App.js

새로고침 Clear 포크

```
import { useFormStatus } from "react-dom";
import { submitForm } from "./actions.js";

function Submit() {
  const { pending } = useFormStatus();
  return (
    <button type="submit" disabled={pending}>
      {pending ? "제출중..." : "제출"}
    </button>
  );
}
```

자세히 보기

`useFormStatus` Hook에 대해 더 알고 싶다면, 참고 문서를 확인하세요.

낙관적으로 폼 데이터 업데이트하기

`useOptimistic` Hook은 네트워크 요청과 같은 백그라운드의 작업이 끝나기 전에 사용자 인터페이스에 낙관적으로 업데이트하는 방법을 제공합니다. 폼의 맥락에서 이 기술은 앱을 더욱 반응형으로 느끼게 해줍니다. 사용자가 폼을 제출하면 인터페이스는 사용자가 기대하는 결과물로 즉시 업데이트됩니다.

예를 들어, 사용자가 폼에 메시지를 입력하고 "전송" 버튼을 클릭하면 `useOptimistic` Hook은 "전송중..." 라벨과 함께 메시지가 서버에 보내지기 전에 리스트에 즉시 보입니다. 이러한 '낙관적인' 접근 방식은 속도와 반응성이 뛰어나다는 인상을 줍니다. 그다음 폼은 실제로 백그라운드에 메시지 보내기를 시도합니다. 서버에 메시지가 잘 도착하면, "전송중..." 라벨은 사라집니다.

App.js actions.js 새로고침 Clear 포크

```
import { useOptimistic, useState, useRef } from "react";
import { deliverMessage } from "./actions.js";

function Thread({ messages, sendMessage }) {
  const formRef = useRef();
  async function formAction(formData) {
    addOptimisticMessage(formData.get("message"));
    formRef.current.reset();
    await sendMessage(formData);
  }
  const [optimisticMessages, addOptimisticMessage] = useOptimistic(
    messages,
```

자세히 보기

폼 제출 오류 처리하기

`<form>` 의 `action` 프로퍼티로 전달된 어떤 함수는 오류를 던지기도 합니다. 이런 오류를 `<form>` 에 에러 바운더리를 감싸 해결할 수 있습니다. 만약 `<form>` 의 `action` 프로퍼티에서 호출된 함수가 오류를 던진다면 에러 바운더리의 Fallback이 보이게 됩니다.

App.js

새로고침 Clear 포크

```
import { ErrorBoundary } from "react-error-boundary";

export default function Search() {
  function search() {
    throw new Error("search error");
  }
  return (
    <ErrorBoundary
      fallback={<p>폼 제출 중에 오류가 발생했습니다.</p>}
    >
      <form action={search}>
        <input name="query" />
```

자세히 보기

자바스크립트 없이 폼 제출 오류 보여주기

점진적 향상을 위해 자바스크립트 번들이 로드되기 전 오류 메시지를 보여주기 위해 다음 요소들이 지켜져야 합니다.

1. `<form>` be rendered by a Client Component
2. the function passed to the `<form>`'s `action` prop be a Server Function
3. the `useActionState` Hook be used to display the error message

`useActionState` 는 서버 함수와 초기 State라는 두 개의 매개변수를 가집니다. `useActionState` 는 State 변수와 액션이라는 두 개의 값을 반환합니다. `useActionState` 를 통해 반환된 액션은 폼의 `action` 프로퍼티에 전달될 수 있습니다. `useActionState` 를 통해 반환된 상태 변수는 오류 메시지를 보여주는 데 사용됩니다. `useActionState` 에 전달된 서버 함수에서 반환된 값은 State 변수를 업데이트하는 데 사용됩니다.

App.js

새로고침 Clear 포크

```
import { useActionState } from "react";
import { signUpNewUser } from "./api";

export default function Page() {
  async function signup(prevState, formData) {
    "use server";
    const email = formData.get("email");
    try {
      await signUpNewUser(email);
      alert(`"${email}"을 등록했어요`);
    } catch (err) {
      return err.toString();
```

자세히 보기

`useActionState` 문서를 통해 폼 작업에서 상태를 업데이트하는 방법에 대해 자세히 알아보세요.

다양한 제출 타입 처리하기

사용자가 누른 버튼에 따라 여러 제출 작업을 처리하도록 폼을 설계할 수 있습니다. 폼 내부의 각 버튼은 `formAction` 프로퍼티를 설정하여 고유한 동작 또는 동작과 연결할 수 있습니다.

사용자가 특정 버튼을 클릭하면 폼이 제출되고 해당 버튼의 속성 및 동작으로 정의된 해당 동작이 실행됩니다. 예를 들어, 폼은 기본적으로 검토를 위해 문서를 제출하지만 `formAction` 이 설정된 별도의 버튼이 있어 문서를 초안으로 저장할 수 있습니다.

App.js

다운로드 새로고침 Clear 포크

```
export default function Search() {
  function publish(formData) {
    const content = formData.get("content");
    const button = formData.get("button");
    alert(`'${button}' 버튼으로 '${content}'가 발행되었습니다.`);
  }

  function save(formData) {
```

```
    const content = formData.get("content");
    alert(`'${content}' 초안이 저장되었습니다!`);
  }
```

자세히 보기

이전
공통 컴포넌트 (예:)

다음
<input>

State 관리하기

탈출구

커뮤니티

행동 강령

팀 소개

문서 기여자

감사의 말

더 보기

블로그

React Native

개인 정보 보호

약관