<input>

내장 브라우저 `<input>` 컴포넌트를 사용하면 여러 종류의 폼 입력Input을 렌더링 할 수 있습니다.

```
<input />
```


- 레퍼런스
  - `<input>`
- 사용법
  - 다양한 유형의 입력 표시
  - 입력에 레이블 제공하기
  - 입력에 초기값 제공하기
  - 폼 제출 시 입력 값 읽기
  - State 변수로 입력 제어하기
  - 키보드를 누를 때마다 리렌더링 최적화하기
- 문제 해결
  - 텍스트 입력에 타이핑을 해도 업데이트되지 않습니다
  - 체크박스를 클릭해도 업데이트되지 않습니다
  - 키보드를 누를 때마다 입력 커서가 입력의 처음으로 돌아갑니다
  - 다음과 같은 에러가 납니다. “A component is changing an uncontrolled input to be controlled(컴포넌트가 제어되지 않는 입력을 제어 상태로 변경합니다)”

레퍼런스

`<input>`

입력Input을 표시하려면, `<input>` 브라우저 내장 컴포넌트를 렌더링하세요.

```
<input name="myInput" />
```

아래 예시를 참고하세요.

Props

`<input>` 은 모든 공통 엘리먼트 Props를 지원합니다.

- `formAction` : A string or function. Overrides the parent `<form action>` for `type="submit"` and `type="image"` . When a URL is passed to `action` the form will behave like a standard HTML form. When a function is passed to `formAction` the function will handle the form submission. See `<form action>` .

You can make an input controlled by passing one of these props:

- `checked` : 불리언 타입. 체크박스 입력 또는 라디오 버튼에서 선택 여부를 제어합니다.
- `value` : 문자열 타입. 텍스트 입력의 경우 텍스트를 제어합니다. (라디오 버튼의 경우 폼 데이터를 지정합니다.)

둘 중 하나를 전달할 때는 반드시 전달된 값을 업데이트하는 `onChange` 핸들러도 함께 전달해야 합니다.

다음의 `<input>` Props는 제어되지 않는 입력들에만 적용됩니다.

- `defaultChecked` : 불리언 타입. `type="checkbox"` 와 `type="radio"` 입력에 대한 초기값을 지정합니다.
- `defaultValue` : 문자열 타입. 텍스트 입력에 대한 초기값을 지정합니다.

다음의 `<input>` Props는 제어되지 않는 입력과 제어되는 입력 모두에 적용됩니다.

- `accept` : 문자열 타입. `type="file"` 입력에서 허용할 파일 형식을 지정합니다.
- `alt` : 문자열 타입. `type="image"` 입력에서 대체 이미지 텍스트를 지정합니다.
- `capture` : 문자열 타입. `type="file"` 입력으로 캡처한 미디어(마이크, 비디오 또는 카메라)를 지정합니다.
- `autoComplete` : 문자열 타입. 자동 완성 동작들 중 가능한 하나를 지정합니다.
- `autoFocus` : 불리언 타입. `true` 일 경우 React는 마운트 시 엘리먼트에 포커스를 맞춥니다.

- `dirname` : 문자열 타입. 엘리먼트 방향성에 대한 폼 필드 이름을 지정합니다.
- `disabled` : 불리언 타입. `true` 일 경우, 입력은 상호작용이 불가능해지며 흐릿하게 보입니다.
- `children` : `<input>` 은 자식을 받지 않습니다.
- `form` : 문자열 타입. 입력이 속하는 `<form>` 의 `id` 를 지정합니다. 생략 시 가장 가까운 부모 폼으로 설정됩니다.
- `formAction` : 문자열 타입. `type="submit"` 과 `type="image"` 의 부모 `<form action>` 을 덮어씁니다.
- `formEnctype` : 문자열 타입. `type="submit"` 과 `type="image"` 의 부모 `<form enctype>` 을 덮어씁니다.
- `formMethod` : 문자열 타입. `type="submit"` 과 `type="image"` 의 부모 `<form method>` 를 덮어씁니다.
- `formNoValidate` : 문자열 타입. `type="submit"` 과 `type="image"` 의 부모 `<form noValidate>` 를 덮어씁니다.
- `formTarget` : 문자열 타입. `<form target>` for `type="submit"` 과 `type="image"` 의 부모 `<form target>` 을 덮어씁니다.
- `height` : 문자열 타입. `type="image"` 의 이미지 높이를 지정합니다.
- `list` : 문자열 타입. `<datalist>` 의 `id` 를 자동 완성 옵션들로 지정합니다.
- `max` : 숫자 타입. 숫자 와 날짜 입력들의 최댓값을 지정합니다.
- `maxLength` : 숫자 타입. 텍스트와 다른 입력들의 최대 길이를 지정합니다.
- `min` : 숫자 타입. 숫자 와 날짜 입력들의 최솟값을 지정합니다.
- `minLength` : 숫자 타입. 텍스트와 다른 입력들의 최소 길이를 지정합니다.
- `multiple` : 불리언 타입. `type="file"` 과 `type="email"` 에 대해 여러 값을 허용할지 여부를 지정합니다.
- `name` : 문자열 타입. 폼과 제출 되는 입력의 이름을 지정합니다.
- `onChange` : 이벤트 핸들러 함수. 제어되는 입력 필수 요소로 가령 사용자가 키보드를 누를 때마다 실행되는 방식으로 입력 값을 변경하는 즉시 실행되며 브라우저 `input` 이벤트처럼 동작합니다.
- `onChangeCapture` : 캡처 단계에서 실행되는 `onChange`
- `onInput` : 이벤트 핸들러 함수. 사용자가 값을 변경하는 즉시 실행됩니다. 지금까지의 용법에 비춰봤을 때 React에서는 유사하게 동작하는 `onChange` 를 사용하는 것이 관습적입니다.
- `onInputCapture` : 캡처 단계에서 실행되는 `onInput`
- `onInvalid` : 이벤트 핸들러 함수. 폼 제출 시 input이 유효하지 않을 경우 실행되며 `invalid` 내장 이벤트와 달리 React `onInvalid` 이벤트는 버블링됩니다.
- `onInvalidCapture` : 캡처 단계에서 실행되는 `onInvalid`

- `onSelect`: 이벤트 핸들러 함수. `<input>` 내부의 선택 사항이 변경된 후 실행됩니다. React는 `onSelect` 이벤트를 확장하여 선택 사항이 비거나 편집 시 선택 사항에 영향을 끼치게 될 때도 실행됩니다.
- `onSelectCapture`: 캡처 단계에서 실행되는 `onSelect`
- `pattern`: 문자열 타입. `value`가 일치해야 하는 패턴을 지정합니다.
- `placeholder`: 문자열 타입. 입력 값이 비었을 때 흐린 색으로 표시됩니다.
- `readOnly`: 불리언 타입. `true`일 경우 사용자가 입력을 편집할 수 없습니다.
- `required`: 불리언 타입. `true`일 경우 폼이 제출할 값을 반드시 제공해야 합니다.
- `size`: 숫자 타입. 너비를 설정하는 것과 비슷하지만 단위는 제어에 따라 다릅니다.
- `src`: 문자열 타입. `type="image"` 입력의 이미지 소스를 지정합니다.
- `step`: 양수 또는 `'any'` 문자열. 유효한 값 사이의 간격을 지정합니다.
- `type`: 문자열 타입. input types 중의 하나
- `width`: 문자열 타입. `type="image"` 입력의 이미지 너비를 지정합니다.

주의 사항

- 체크박스에는 `value` (또는 `defaultValue`)가 아닌 `checked` (또는 `defaultChecked`)가 필요합니다.
- 텍스트 입력 영역은 문자열 `value` Prop을 받을 경우 제어되는 것으로 취급됩니다.
- 체크박스 또는 라디오 버튼이 불리언 `checked` Prop을 받을 경우 제어되는 것으로 취급됩니다.
- 입력은 제어되면서 동시에 비제어될 수 없습니다.
- 입력은 생명주기 동안 제어 또는 비제어 상태를 오갈 수 없습니다.
- 제어되는 입력엔 모두 백업 값을 동기적으로 업데이트하는 `onChange` 이벤트 핸들러가 필요합니다.

사용법

다양한 유형의 입력 표시

입력을 표시하기 위해 `<input>` 컴포넌트를 렌더링하세요. 기본적으로 텍스트로 입력됩니다. 체크박스에는 `type="checkbox"`, 라디오 버튼에는 `type="radio"`를 전달하거나 다른 입력 유형들 중 하나를 전달할 수 있습니다.

App.js 다운로드 새로고침 Clear 포크

```
export default function MyForm() {
  return (
    <>
      <label>
        Text input: <input name="myInput" />
      </label>
      <hr />
      <label>
        Checkbox: <input type="checkbox" name="myCheckbox" />
      </label>
      <hr />
      <p>
```

자세히 보기

입력에 레이블 제공하기

일반적으로 모든 `<input>` 은 `<label>` 태그 안에 존재하는데, 이렇게 하면 해당 레이블이 해당 입력과 연관됨을 브라우저가 알 수 있습니다. 사용자가 레이블을 클릭하면 브라우저는 입력에 자동적으로 포커스를 맞춥니다. 스크린 리더는 사용자가 연관된 입력에 포커스를 맞출 때 레이블 캡션을 읽게 되므로 이 방식은 접근성을 위해서도 필수입니다.

`<label>` 안에 `<input>` 을 감쌀 수 없다면, `<input id>` 와 `<label htmlFor>` 에 동일한 ID를 전달해서 연관성을 부여하세요. 한 컴포넌트의 여러 인스턴스 간 충돌을 피하려면 `useId` 로 그러한 ID를 생성하세요.

App.js

다운로드 새로고침 Clear 포크

```
import { useId } from 'react';

export default function Form() {
  const ageInputId = useId();
  return (
    <>
      <label>
        Your first name:
        <input name="firstName" />
      </label>
      <hr />
      <label htmlFor={ageInputId}>Your age:</label>
```

자세히 보기

입력에 초기값 제공하기

입력의 초기값은 선택적으로 지정할 수 있습니다. 텍스트 입력을 위한 `defaultValue` 문자열을 전달하세요. 대신 체크박스와 라디오 버튼은 `defaultChecked` 불리언으로 초기값을 지정해야 합니다.

App.js 다운로드 새로고침 Clear 포크

```
export default function MyForm() {
  return (
    <>
      <label>
        Text input: <input name="myInput" defaultValue="Some initial value" />
      </label>
      <hr />
      <label>
        Checkbox: <input type="checkbox" name="myCheckbox" defaultChecked={true} />
      </label>
      <hr />
      <p>
```

자세히 보기

폼 제출 시 입력 값 읽기

입력과 `<button type="submit">` 바깥을 `<form>` 으로 감싸면 버튼을 클릭했을 때 `<form onSubmit>` 이벤트 핸들러가 호출됩니다. 기본적으로 브라우저는 현재 URL에 폼 데이터를 전송한 후 페이지를 새로고침하며, 이러한 동작은 `e.preventDefault()` 를 호출하여 덮어쓸 수 있습니다. 폼 데이터는 `new FormData(e.target)` 로 읽으세요.

App.js

다운로드 새로고침 Clear 포크

```
export default function MyForm() {
  function handleSubmit(e) {
    // 브라우저가 페이지를 다시 로드하지 못하도록 방지합니다.
    e.preventDefault();

    // 폼 데이터를 읽습니다.
    const form = e.target;
    const formData = new FormData(form);

    // formData를 직접 fetch body로 전달할 수 있습니다.
    fetch('/some-api', { method: form.method, body: formData });
```

자세히 보기

중요합니다!

`<input name="firstName" defaultValue="Taylor" />` 예시와 같이 모든 `<input>`에 `name`을 부여하세요. 해당 `name`은 `{ firstName: "Taylor" }` 예시처럼 폼 데이터의 key로 쓰입니다.

주의하세요!

기본적으로 `<form>` 내부의 *어느* `<button>` 이든 폼을 제출합니다. 뜻밖인가요? 커스텀 `Button` React 컴포넌트의 경우 `<button>` 대신 `<button type="button">` 반환을 고려하세요. 명시성을 부여하기 위해 폼 제출용 버튼으로는 `<button type="submit">`을 사용하세요.
By default, a `<button>` inside a `<form>` without a `type` attribute will submit it. This can be surprising! If you have your own custom `Button` React component, consider using `<button type="button">` instead of `<button>` (with no type). Then, to be explicit, use `<button type="submit">` for buttons that *are* supposed to submit the form.

State 변수로 입력 제어하기

`<input />`과 같은 입력은 *제어되지 않습니다*. `<input defaultValue="Initial text" />`와 같은 초기값을 전달한대도 JSX는 당장의 값을 제어하지 않고 초기값만을 지정합니다.

제어되는 **입력을 렌더링하려면, `value` (또는 체크박스와 라디오에는 `checked`) Prop 을 전달하세요.** React는 전달한 `value`를 항상 갖도록 입력에 강제합니다. 일반적으로 State 변수를 선언하여 할 수 있습니다.

```
function Form() {
  const [firstName, setFirstName] = useState(''); // State 변수를 선언합니다.
```

```
  // ...
  return (
    <input
      value={firstName} // 입력 값이 State 변수와 일치하도록 강제합니다.
      onChange={e => setFirstName(e.target.value)} // 입력을 편집할 때마다 State 변수를 업
    />
  );
}
```

예를 들어 수정할 때마다 UI를 리렌더링 하는 등 State가 필요한 경우 제어된 입력이 적합합니다.

```
function Form() {
  const [firstName, setFirstName] = useState('');
  return (
    <>
      <label>
        First name:
        <input value={firstName} onChange={e => setFirstName(e.target.value)} />
      </label>
      {firstName !== '' && <p>Your name is {firstName}.</p>}
      ...
```

또한 버튼을 클릭하는 등의 입력 State를 조정하는 여러 가지 방법을 제공하려는 경우에도 유용합니다.

```
function Form() {
  // ...
  const [age, setAge] = useState('');
  const ageAsNumber = Number(age);
  return (
    <>
      <label>
        Age:
        <input
          value={age}
          onChange={e => setAge(e.target.value)}
          type="number"
        />
        <button onClick={() => setAge(ageAsNumber + 10)}>
```

```
      Add 10 years
    </button>
```

제어되는 컴포넌트에 전달되는 `value` 는 `undefined` 나 `null` 이 되어서는 안됩니다. 아래의 `firstName` 필드처럼 초기값을 비워두어야 하는 경우 State 변수를 빈 문자열(`''`)로 초기화 하세요.

App.js

다운로드 새로고침 Clear 포크

```
import { useState } from 'react';

export default function Form() {
  const [firstName, setFirstName] = useState('');
  const [age, setAge] = useState('20');
  const ageAsNumber = Number(age);
  return (
    <>
      <label>
        First name:
        <input
          value={firstName}
```

자세히 보기

주의하세요!

입력에 `onChange` 없이 `value`를 전달하면 해당 입력에 타이핑을 할 수 없게 됩니다. `value`를 전달하여 입력을 제어하면 항상 해당 값을 가지도록 *강제합니다*. 그러므로 State 변수를 `value`로 전달해도 `onChange` 이벤트 핸들러 내에서 해당 State 변수를 동기적으로 업데이트하지 않으면 React는 키보드를 누를 때마다 입력을 처음 지정한 `value`로 되돌리게 됩니다.

키보드를 누를 때마다 리렌더링 최적화하기

제어된 입력을 사용할 때는 키보드를 누를 때마다 State를 설정합니다. State를 포함하는 컴포넌트가 큰 트리를 리렌더링할 경우 속도가 느려질 수 있습니다. 리렌더링 성능을 최적화할 수 있는 몇 가지 방법이 있습니다.

예를 들어, 키보드를 누를 때마다 모든 페이지 내용을 리렌더링하는 폼으로 시작한다고 가정해보세요.

```
function App() {
  const [firstName, setFirstName] = useState('');
  return (
    <>
      <form>
        <input value={firstName} onChange={e => setFirstName(e.target.value)} />
      </form>
      <PageContent />
    </>
  );
}
```

`<PageContent />`는 입력 State에 의존하지 않으므로 입력 State를 자체 컴포넌트로 이동할 수 있습니다.

```
function App() {
  return (
    <>
      <SignupForm />
      <PageContent />
    </>
  );
}

function SignupForm() {
  const [firstName, setFirstName] = useState('');
  return (
    <form>
      <input value={firstName} onChange={e => setFirstName(e.target.value)} />
    </form>
  );
}
```

이렇게 하면 키보드를 누를 때마다 `SignupForm` 만 리렌더링하기 때문에 성능이 크게 향상됩니다.

`PageContent` 가 검색 입력 값에 의존하는 경우처럼 리렌더링을 피할 방법이 없는 경우 `useDeferredValue` 를 사용하면 많은 리렌더링 중에도 제어된 입력이 응답하도록 할 수 있습니다.

문제 해결

텍스트 입력에 타이핑을 해도 업데이트되지 않습니다

`onChange` 없이 `value` 만 전달하여 입력을 렌더링하면 콘솔에 오류가 나타납니다.

```
// 🔴 버그: 제어되는 `<input>`에 `onChange` 핸들러가 없습니다.
<input value={something} />
```

Console

- 폼 필드에 onChange 핸들러 없이 value Prop만 전달했습니다. 이렇게 하면 읽기 전용 필드를 렌더링하게 됩니다. 필드가 변경 가능해야 하는 경우 defaultValue를 사용하고 그렇지 않은 경우 onChange 또는 readOnly를 설정하세요.

오류 메시지가 제안하듯 *초기값만 지정*하려면 defaultValue를 대신 전달하세요.

```
// ✅ 좋은 예: 제어되지 않는 `<input>`에 초기값 전달
<input defaultValue={something} />
```

입력을 State 변수로 제어하려면 onChange 핸들러를 지정하세요.

```
// ✅ 좋은 예: 제어되는 `<input>`에 `onChange` 전달
<input value={something} onChange={e => setSomething(e.target.value)} />
```

값이 의도적으로 읽기 전용이라면 오류를 막기 위해 readOnly Prop을 추가하세요.

```
// ✅ 좋은 예: 제어되는 읽기 전용 `<input>`에 `onChange` 생략
<input value={something} readOnly={true} />
```

체크박스를 클릭해도 업데이트되지 않습니다

onChange 없이 checked만 전달하여 체크박스를 렌더링하면 콘솔에 에러가 나타납니다.

```
// 🔴 버그: 제어되는 체크박스에 onChange 핸들러가 없습니다.
<input type="checkbox" checked={something} />
```

> Console

- 폼 필드에 onChange 핸들러 없이 checked Prop을 전달했습니다. 이렇게 하면 읽기 전용 필드를 렌더링하게 됩니다. 필드가 변경 가능해야 하는 경우 defaultChecked를

사용하고 그렇지 않은 경우 `onChange` 또는 `readOnly`를 설정하세요.

오류 메시지가 제안하듯 *초기값만 지정*하려면 `defaultChecked`를 대신 전달하세요.

```
// ✅ 좋은 예: 제어되지 않는 체크박스에 초기값 전달
<input type="checkbox" defaultChecked={something} />
```

체크박스를 State 변수로 제어하려면 `onChange` 핸들러를 지정하세요.

```
// ✅ 좋은 예: 제어되는 체크박스에 onChange 전달
<input type="checkbox" checked={something} onChange={e => setSomething(e.target.checke
```

주의하세요!

체크박스에서는 `e.target.value`가 아닌 `e.target.checked`를 읽어야 합니다.

체크박스가 의도적으로 읽기 전용이라면 오류를 막기 위해 `readOnly` Prop을 추가하세요.

```
// ✅ 좋은 예: 제어되는 읽기 전용 input에 onChange 생략
<input type="checkbox" checked={something} readOnly={true} />
```

키보드를 누를 때마다 입력 커서가 입력의 처음으로 돌아갑니다

입력을 제어할 경우 `onChange` 안에서 State 변수를 DOM에서 받아온 입력 값으로 업데이트해야 합니다.

State 변수는 `e.target.value` (혹은 체크박스에서는 `e.target.checked`) 외의 값으로 업데이트할 수 없습니다.

```
function handleChange(e) {
  // 🔴 버그: 입력을 `e.target.value` 외의 값으로 업데이트합니다.
  setFirstName(e.target.value.toUpperCase());
}
```

비동기로 업데이트할 수도 없습니다.

```
function handleChange(e) {
  // 🔴 버그: 입력을 비동기로 업데이트합니다.
  setTimeout(() => {
    setFirstName(e.target.value);
  }, 100);
}
```

코드를 고치려면 `e.target.value` 로 동기 업데이트하세요.

```
function handleChange(e) {
  // ✅ 제어되는 입력을 `e.target.value`로 동기 업데이트합니다.
  setFirstName(e.target.value);
}
```

이 방법으로 문제가 해결되지 않을 경우 키보드를 누를 때마다 입력이 DOM에서 제거된 후 다시 추가되고 있을 가능성이 있습니다. 실수로 리렌더링마다 State를 재설정하고 있다면 나타날 수 있는 현상입니다. 가령 입력이나 입력의 부모 중 하나가 매번 다른 `key` 어트리뷰트를 받거나 컴포넌트 함수 정의를 중첩시키는 경우(이는 지원되지 않으며 '내부' 컴포넌트가 항상 다른 트리로 간주되도록 합니다)에 해당 문제가 발생할 수 있습니다.

다음과 같은 에러가 납니다. "A component is changing an uncontrolled input to be controlled(컴포넌트가 제어되지 않는 입력을 제어 상태로 변경합니다)"

컴포넌트에 `value` 를 제공할 경우 반드시 생명주기 내내 문자열 타입으로 남아야 합니다.

React는 컴포넌트를 비제어할 것인지 제어할 것인지 의도를 알 수 없기 때문에 처음엔 `value={undefined}` 를 전달했다가 나중에 다시 `value="some string"` 을 전달할 수는 없습니다. 제어되는 컴포넌트는 항상 `null` 이나 `undefined` 가 아닌 문자열 `value` 를 받아야 합니다.

`value` 가 API나 state 변수에서 온다면 `null` 이나 `undefined` 로 초기화할 수 있습니다. 그럴 경우 빈 문자열(`''`)을 초기값으로 설정하거나 `value` 가 문자열임을 보장하기 위해 `value={someValue ?? ''}` 를 전달하세요.

마찬가지로 체크박스에 `checked` 를 전달하는 경우 불리언임을 보장하세요.

이전
<form>

다음
<option>