<textarea>

내장 브라우저 `<textarea>` 컴포넌트를 사용하면 여러 줄의 텍스트 입력을 렌더링할 수 있습니다.

```
<textarea />
```


- 레퍼런스
  - `<textarea>`
- 사용법
  - 텍스트 영역 표시하기
  - 텍스트 영역 레이블 제공하기
  - 텍스트 영역 초기값 제공하기
  - 폼 제출 시 텍스트 영역 값 읽기
  - State 변수로 텍스트 영역 제어하기
- 문제 해결
  - 텍스트 영역에 타이핑을 해도 업데이트되지 않습니다
  - 키보드를 누를 때마다 텍스트 커서가 텍스트 영역의 처음으로 돌아갑니다
  - 다음과 같은 에러가 납니다. “A component is changing an uncontrolled input to be controlled(컴포넌트가 제어되지 않는 입력을 제어 상태로 변경합니다)”

레퍼런스

`<textarea>`

텍스트 영역을 표시하려면 내장 브라우저 `<textarea>` 컴포넌트를 렌더링하세요.

```
<textarea name="postContent" />
```

아래 예시를 참고하세요.

Props

`<textarea>` 는 모든 공통 엘리먼트 Props를 지원합니다.

텍스트 영역을 제어하려면 `value` Prop을 전달하세요.

- `value` : 문자열 타입. 텍스트 영역 내부의 텍스트를 제어합니다.

`value` 를 전달할 땐 반드시 해당 값을 업데이트하는 `onChange` 핸들러도 함께 전달해야 합니다.

`<textarea>` 가 제어되지 않는 경우 `defaultValue` Prop을 대신 전달해도 됩니다.

- `defaultValue` : 문자열 타입. 텍스트 영역 초기값을 지정합니다.

다음의 `<textarea>` props는 제어되지 않는 텍스트 영역과 제어되는 텍스트 영역 모두에 적용됩니다.

- `autoComplete` : 'on' 또는 'off'. 자동 완성 동작을 지정합니다.
- `autoFocus` : 불리언 타입. `true` 일 경우 React는 마운트 시 엘리먼트에 포커스를 맞춥니다.
- `children` : `<textarea>` 는 자식을 받지 않습니다. 초기값을 설정하려면 `defaultValue` 를 사용하세요.
- `cols` : 숫자 타입. 평균 문자 너비의 기본 너비를 지정하세요. 기본값은 `20` 입니다.
- `disabled` : 불리언 타입. `true` 일 경우 입력은 상호작용이 불가능해지며 흐릿하게 보입니다.
- `form` : 문자열 타입. 텍스트 영역 입력이 속하는 `<form>` 의 `id` 를 지정합니다. 생략 시 가장 가까운 부모 폼으로 설정됩니다.
- `maxLength` : 숫자 타입. 텍스트의 최대 길이를 지정합니다.
- `minLength` : 숫자 타입. 텍스트의 최소 길이를 지정합니다.
- `name` : 문자열 타입. 폼과 제출되는 입력의 이름을 지정합니다.
- `onChange` : `Event` 핸들러 함수. 제어되는 텍스트 영역 필수 요소로 가령 사용자가 키보드를 누를 때마다 실행되는 방식으로 입력 값을 변경하는 즉시 실행되며 브라우저 `input` 이벤트처럼 동작합니다.
- `onChangeCapture` : 캡처 단계에서 실행되는 `onChange` .
- `onInput` : `Event` 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`: `Event` 핸들러 함수. 폼 제출 시 입력이 유효하지 않을 경우 실행되며 `invalid` 내장 이벤트와 달리 React `onInvalid` 이벤트는 버블링됩니다.
- `onInvalidCapture`: 캡처 단계에서 실행되는 `onInvalid`.
- `onSelect`: `Event` 핸들러 함수. `<textarea>` 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`.
- `placeholder`: 문자열 타입. 텍스트 영역 값이 비었을 때 흐린 색으로 표시됩니다.
- `readOnly`: 불리언 타입. `true` 일 경우 사용자가 텍스트 영역을 편집할 수 없습니다.
- `required`: 불리언 타입. `true` 일 경우 폼이 제출할 값을 반드시 제공해야 합니다.
- `rows`: 숫자 타입. 평균 문자 높이의 기본 높이를 지정하세요. 기본값은 `2` 입니다.
- `wrap`: `'hard'`, `'soft'`, `'off'` 중 하나. 폼 제출 시 텍스트를 감싸는 방식을 지정합니다.

주의 사항

- `<textarea>something</textarea>` 와 같이 자식을 전달하는 것은 허용되지 않습니다. 초기 콘텐츠로 `defaultValue` 를 사용하세요.
- 텍스트 영역은 문자열 `value` Prop을 받을 경우 제어되는 것으로 취급됩니다.
- 텍스트 영역은 제어되면서 동시에 비제어될 수 없습니다.
- 텍스트 영역은 생명주기 동안 제어 또는 비제어 상태를 오갈 수 없습니다.
- 제어되는 텍스트 영역엔 모두 백업 값을 동기적으로 업데이트하는 `onChange` 이벤트 핸들러가 필요합니다.

사용법

텍스트 영역 표시하기

텍스트 영역을 표시하려면 `<textarea>` 를 렌더링하세요. `rows` 와 `cols` 어트리뷰트로 텍스트 영역의 기본 크기를 지정할 수도 있지만 기본적으로 사용자가 텍스트 영역의 크기를 조정할 수 있습니다. 크기 조정을 비활성화하려면 CSS에서 `resize: none` 을 지정하세요.

```
export default function NewPost() {
  return (
    <label>
      Write your post:
      <textarea name="postContent" rows={4} cols={40} />
    </label>
  );
}
```

텍스트 영역 레이블 제공하기

일반적으로 모든 `<textarea>` 는 `<label>` 태그 안에 두게 되는데, 이렇게 하면 해당 레이블이 해당 텍스트 영역과 연관됨을 브라우저가 알 수 있습니다. 사용자가 레이블을 클릭하면 브라우저는 텍스트 영역에 포커스를 맞춥니다. 스크린 리더는 사용자가 텍스트 영역에 포커스를 맞출 때 레이블 캡션을 읽게 되므로 이 방식은 접근성을 위해서도 필수입니다.

`<label>` 안에 `<textarea>` 를 감쌀 수 없다면 `<textarea 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 postTextAreaId = useId();
  return (
    <>
      <label htmlFor={postTextAreaId}>
        Write your post:
      </label>
      <textarea
        id={postTextAreaId}
        name="postContent"
```

자세히 보기

텍스트 영역 초기값 제공하기

텍스트 영역 초기값은 선택적으로 지정할 수 있습니다. `defaultValue` 문자열로 전달하세요.

App.js 다운로드 새로고침 Clear 포크

```
export default function EditPost() {
  return (
    <label>
      Edit your post:
      <textarea
        name="postContent"
        defaultValue="I really enjoyed biking yesterday!"
        rows={4}
        cols={40}
      />
    </label>
  );
```

주의하세요!

HTML과 달리 `<textarea>Some content</textarea>` 와 같은 초기 텍스트 전달은 지원되지 않습니다.

폼 제출 시 텍스트 영역 값 읽기

텍스트 영역과 `<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 EditPost() {
  function handleSubmit(e) {
    // Prevent the browser from reloading the page
    e.preventDefault();

    // Read the form data
    const form = e.target;
    const formData = new FormData(form);

    // You can pass formData as a fetch body directly:
    fetch('/some-api', { method: form.method, body: formData });
```

자세히 보기

중요합니다!

`<textarea name="postContent" />` 예시와 같이 `<textarea>` 에 `name` 을 부여하세요. 해당 `name` 은 `{ postContent: "Your post" }` 예시처럼 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.

State 변수로 텍스트 영역 제어하기

`<textarea />` 와 같은 텍스트 영역은 *제어되지 않습니다*. `<textarea defaultValue="Initial text" />` 와 같은 초기값을 전달한대도 JSX는 당장의 값이 아닌 초기값만을 지정합니다.

제어되는 **텍스트 영역을 렌더링하려면** **`value`** **Prop을 전달하세요.** React는 전달한 `value` 를 항상 갖도록 텍스트 영역에 강제합니다. 일반적으로 텍스트 영역은 State 변수를 선언하여 제어합니다.

```
function NewPost() {
  const [postContent, setPostContent] = useState(''); // state 변수를 선언합니다.
  // ...
  return (
    <textarea
      value={postContent} // 입력 값이 state 변수와 일치하도록 강제합니다.
      onChange={e => setPostContent(e.target.value)} // 텍스트 영역을 편집할 때마다 state
```

```
    />
  );
}
```

이 방식은 키보드를 누를 때마다 그에 대한 반응으로 UI 일부를 리렌더링하고자 할 때 유용합니다.

package.json App.js MarkdownPreview.js 새로고침 Clear 포크

```
{
  "dependencies": {
    "react": "latest",
    "react-dom": "latest",
    "react-scripts": "latest",
    "remarkable": "2.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
```

주의하세요!

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

문제 해결

텍스트 영역에 타이핑을 해도 업데이트되지 않습니다

`onChange` 없이 `value`만 전달하여 텍스트 영역을 렌더링하면 콘솔에 오류가 나타납니다.

```
// 🔴 버그: 제어되는 텍스트 영역에 `onChange` 핸들러가 없습니다.
<textarea 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.

에러 메시지가 제안하듯 *초기값만 지정*하려면 `defaultVallue`를 대신 전달하세요.

```
// ✅ 좋은 예: 제어되지 않는 텍스트 영역에 초기값 전달
<textarea defaultValue={something} />
```

텍스트 영역을 State 변수로 제어하려면 `onChange` 핸들러를 지정하세요.

```
// ✅ 좋은 예: 제어되는 텍스트 영역에 onChange 전달
```

```
<textarea value={something} onChange={e => setSomething(e.target.value)} />
```

값이 내부적으로 읽기 전용이라면 오류를 막기 위해 `readOnly` Prop을 추가하세요.

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

키보드를 누를 때마다 텍스트 커서가 텍스트 영역의 처음으로 돌아갑니다

텍스트 영역을 제어할 경우 `onChange` 안에서 State 변수를 DOM에서 받아온 텍스트 영역 값으로 업데이트해야 합니다.

State 변수는 `e.target.value` 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 텍스트 영역이 제거 후 다시 추가되고 있을 가능성이 있습니다. 실수로 리렌더링마다 State를 재설정하고 있다면 나타날 수 있는 현상입니다. 가령 텍스트 영역이나 텍스트 영역의 부모 중 하나가 매번 다른 `key` 어트리뷰트를 받거나 컴포넌트 정의를 중첩시키는 경우(이는 React에서 허용되지 않으며 렌더링마다 '내부' 컴포넌트가 리마운트되는 원인이 됩니다)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 초기화될 수 있습니다. 그럴 경우 빈 문자열(`''`)을 초기값으로 설정하거나 `value` 가 문자열임을 보장하기 위해 `value={someValue ?? ''}` 를 전달하세요.

이전
<select>

다음
<link>

Meta Open Source

Copyright © Meta Platforms, Inc

uwu?

React 학습하기

- 빠르게 시작하기
- 설치하기
- UI 표현하기
- 상호작용성 더하기
- State 관리하기
- 탈출구

API 참고서

- React APIs
- React DOM APIs

커뮤니티

- 행동 강령
- 팀 소개
- 문서 기여자
- 감사의 말

더 보기

- 블로그
- React Native
- 개인 정보 보호
- 약관