폼 제어

2023.06.22.

01. Create 구현
02. Update 구현
03. 함수형 컴포넌트에서 폼 제어

01. Creaet 구현

2023.06.22.

1. onSubmit 이벤트

1.1 <from>의 submit 이벤트에서 양식 요소의 값 얻기(e.target.name.value)

구문

```
<form action=“/” method=“post” onSubmit={function(e){
  e.preventDefault();
  e.target.이름.value //양식 요소의 값
}.bind(this)}>
  <input name="이름" vlaue="값">
  <input type="submit></input>
</form>
```

참고 : e.target : form 요소

예시

```
<form action="/"method="post"onSubmit={function(e){
  e.preventDefault();
  this.props.onSubmit(e.target.title.value, e.target.desc.value);
}.bind(this)}>
  <p><input type="text"name="title"placeholder="title"/></p>
  <p><textarea name="desc"placeholder="desc"></textarea></p>
  <p><button type="submit">제출</button></p>
</form>
```

2. 컨텐츠 변경

2.1 state 객체 속성의 값이 배열일 경우 추가하는 방법

- 배열은 참조형 이므로 배열명을 통해 간접적으로 데이터를 수정할 수 있다. 하지만 react에서 state 값이 배열인 경우 배열명을 통해 state를 변경하였다 하더라도 react는 알지 못한다. 그러므로 setState()를 이용하여 변경되었음을 알려 주어야 한다.

예시

```
<CreateContent onSubmit={function(_title, _desc){
  this.max_content_id++;
  this.state.contents.push({id:this.max_content_id, title:_title, desc:_desc});

  this.setState({contents:this.state.contents});
}.bind(this)}/>
```

- 그러나 위 방식 처럼 push()는 배열에 직접적으로 원소를 추가하는 방식이므로 setState()로 최종 적용전 이미 메모리에서는 변경된 값을 react에 인지시키는 방식이다. 이는 리엑트의 성능을 개선하고자 할 때 까다롭거나 불가능한 상황이 방생할 수 있다.
- 개선방향 : 배열에 concat()로 요소한 추가한 새 배열을 리턴받은 다음 setState()로 state를 변경하는 방법

예시

```
<CreateContent onSubmit={function(_title, _desc){
  this.max_content_id++;
  let _contents = this.state.contents.concat(
    {id:this.max_content_id, title:_title, desc:_desc}
  );

  this.setState({contents:_contents});
}.bind(this)}/>
```

2.2 shouldComponentUpdate(newProps, newState)

- render() 작동전 실행되는 메서드. Component가 Update 가능 여부를 확인

① 매개변수

- newProps : props가 변경될 경우의 props 값
- newState : state가 변경될 경우의 state 값

② 반환값(boolean)

- true : render()가 호출됨
- false : render()가 호출되지 않음

예시

```
class TOC extends Component {
  shouldComponentUpdate(newProps, newState) {
    if(this.props.data === newProps.data) {
      return false;
    }
    return true;
  }

  ...
}
```

- 배열의 push()는 기존배열의 값을 변경하므로 배열명의 참조주소는 동일하다. 즉 shouldComponentUpdate의 newState 매개 변수의 포함되어 들어오는 배열 state 주소값은 this.state의 배열 주소값과 동일한 값을 참조하고 있으므로 변경되었음을 알 수 없다.
- 하지만, concat()는 배열을 결합한 후 새로운 배열을 생성하여 리턴하므로 shouldComponentUpdate()의 입장에서는 newState 매개 변수의 포함되어 들어오는 배열 state 주소값과 this.state의 배열 주소값과 동일한 값을 참조하지 않고 있으므로 다름(변경)되었음을 확인할 수 있다.
- 그러므로 shouldComponentUpdate() 사용시 배열변경이 반영되도록 하려면 새 배열이 러턴되는 형식의 메서드를 활용해야 한다.

2.3 배열의 불변을 유지하면서 setState로 state를 변경(결론)

방법1. 기존 state배열에 concat()로 추가하여 변경된 새 배열을 리턴받은 다음 setState()로 변경

```
let transArray = this.state.array.concat(element);
this.setState({array: transArray});
```

방법2. 기존 state배열을 Array.from()로 새 배열로 복사한 후 concat() 또는 push()로 변경하고 복사및 변경된 배열을 setState()로 변경

```
let copyArray = Array.from(this.state.array);
copyArray.push(element);
this.setState({array: copyArray});
```

02. Update 구현

2023.06.26.

1. Update 구현 : from

props를 form 요소의 value에 적용하면 readonly 상태이다.

1.1 해결방법

① props 값을 state로 설정

② form의 양식요소의 값으로 ①의 state 값을 적용

③ ②의 요소에 onChange 이벤트로 ①의 state 값을 변경

-> state가 변경되면 render()가 호출되고 서식 값이 변경된걸로 보이게 함.

구문

```
class 컴포넌트 extends Component{
  constructor(props){
    super(props);
    this.state = {
      스테이트명: this.props.프롭스명
    };
  }

  render(){
    return(
      <from>
        <input value={this.state.스테이트명}
          onChange={funtion(e){
            this.setState({스테이트: e.target.value});
          }.bind(this)}
        />
      </form>
    );
  }
}
```

예시

```
class UpdateContent extends Component {
  constructor(props) {
    super(props);
    this.state = {
      title:this.props.data.title,
    }
  }

  render() {
    return (
      <article>
        <form action="/"method="post">
          <p>
            <input type="text" name="title" placeholder="title" value={this.state.title}
              onChange={function(e){
                this.setState({title:e.target.value});
              }.bind(this)}
            />
          </p>
          <p><button type="submit">수정</button></p>
        </form>
      </article>
    );
  }
}
```

1.2 React에서 <textarea>에 value를 적용하는 방법

잘못된 방법 : <textarea>{값}</textarea>

정확한 방법 : <textarea value={값}></textarea>

예시

```
<textarea name="desc" placeholder="desc" value={this.state.desc}></textarea>
```

1.3 <form>의 양식 요소의 onChange 이벤트를 중복성 제거를 위한 함수화

가정 : 이벤트 함수명은 inputFormHandler라고 하자.

구문

```
class 컴포넌트 extends Component {
  constructor(props){
    super(props);
    this.state = {
      양식요소명 : 값
    }
    this.inputFormHandler = this.inputFornHandler.bind(this);
  }

  inputFormHandler(e) {
    this.setState(
      [e.target.name]: e.target.value
    );
  }

  render(){
    return(
      <form>
        <input name="양식요소명" onChange={this.inputFormHandler} />
      </form>
    );
  }
}
```

03. 함수형 컴포넌트에서 폼 제어

2023.06.19.

1. useRef() 훅

- 컴포넌트나 HTML 요소를 참조로 관리할 수 있다.
- useRef 훅을 사용하면 렌더링 간에 값을 유지할 수 있다.
- 업데이트 시 다시 랜더링 되지 않는 변경 가능한 값을 저장하는데 사용할 수 있다.
- useRef 는 .current 프로퍼티로 전달된 인자로 초기화된 변경 가능한 ref 객체를 반환한다.

구문

```
import { useRef } from “react”;

function 컴포넌트명(...) {
  conset ref명 = useRef();

  return (
    <form action=“/” method=“post” onSubmit = {function(){
        ...
        ref명.current.value  //참조 폼 객체의 값 읽기
        ...
      }}
      <input ref={ref명} type=“...” name=“...” />
    />
  );
}
```

예시

```
import { useRef } from "react";

function CreateContents({onSubmit=f =>f}) {
  const txtTitle = useRef();
  const areaDesc = useRef();

  return (
    <article>
      <h2 className="article_title">Create</h2>
      <form action="/"method="post"onSubmit={function(e){
        e.preventDefault();
        onSubmit(txtTitle.current.value, areaDesc.current.value);
      }}>
        <p><input ref={txtTitle}type="text"name="title"placeholder="title"/></p>
        <p><textarea ref={areaDesc}name="desc"placeholder="desc"></textarea></p>
        <p><button type="submit">제출</button></p>
      </form>
    </article>
  );
}
```

2. useMemo() 훅

- 의존성 배열에 적힌 값이 변할 때만 값, 함수를 다시 정의할 수 있다.(재랜더링시 정의 안함)
- 불필요한 리렌더링이나 재계산을 피하기 위해 비용이 많이 드는 계산이나 함수의 결과를 메모이제이션(특정한 값을 메모리에 저장해둠)하는 데 사용된다.

- 컴포넌트가 다시 렌더링될 때 useMemo를 사용하면 함수의 결과를 캐싱하고 특정 종속성이 변경될 때만 다시 계산할 수 있으므로 응용 프로그램의 전반적인 성능이 향상된다.

구문

```
import { useMemo } from “react”;

function 컴포넌트명(...) {
  conset 메모명 = useMemo(function(){
    의존성배열이 변화될때 적용할 내용;
  }, [의존성배열]);

  return (
    ...
    {메모명}
    ...
  );
}

※ 의존성 배열이 비어있는 경우에는 랜더링 될 때 마다 실행된다.
```

예시

```
import { useMemo } from "react";

function TOC({data, onChangePage = f =>f}) {
  const newData = useMemo(function(){
    let lists = [];
    for(let content of data) {
      lists.push(
        <li key={content.id}>
          <a
            href="#"
            className="content"
            onClick={function(id, e){
              e.preventDefault();
              setActive(id)
              onChangePage(id);
            }.bind(this, content.id)}
          >{content.title}</a>
        </li>
      );
    }

    return lists;
  },[data]);

  return(
    <nav>
      <ul>
        {newData}
      </ul>
    </nav>
  );
}
```