Component (Advanced)

리액트 컴포넌트에서 폼(form) 다루기

ref

- https://reactjs.org/docs/refs-and-the-dom.html
- **임의의 값을 저장하고 있는 객체**를 저장하기 위해서 사용
  - **this.state와는 별개로 동작**하는 상태 저장용 객체
  - **state가 변경되면 render 메소드가 호출되어 re-render 되지만, ref 값은 변경해도 re-render 하지 않음!**
    - state => 각 렌더링 함수 호출 사이에 값을 보존함, **state가 변경되면 다시 렌더링함**
    - ref => 각 렌더링 함수 호출 사이에 값을 보존함, **ref가 변경되어도 다시 렌더링하지 않음**
- 일반적으로 컨트롤러에서 **내부적으로 접근할 DOM 요소를 저장하는 용도**로 사용
  - 아무 DOM 요소나 다 저장하는 것은 아니고, input, canvas, 미디어(audio, video) 요소를 저장하고 조정하는 용도로 사용
    - ref에 저장해놨다가 직접 접근해야 하는 DOM 노드들의 특징 => 반드시 **요소 객체를 통해 접근해야 하는 자체 제공 메소드가 존재**함
      - ex) 캔버스 요소의 그리기 메소드, input 요소의 focus 메소드, 미디어 요소(audio, video)의 play, stop 메소드 등등
      - 리액트만 이용해서 **네이티브 API에 접근할 방법이 없으므로** 필연적으로 ref 값으로 DOM 요소를 저장하고 있어야 할 필요성이 생김
- DOM 요소가 아닌 **컴포넌트도 ref 값을 통해 저장**할 수 있음

component-advanced/ref-demo.js

```
import React, { Component, createRef } from 'react'
import ReactDOM from 'react-dom'

class RefDemo extends Component {
    constructor(props) {
        super(props)

        // 주로 직접 접근이 필요한 DOM 노드를 저장하기 위한 용도로 Ref 사용
        this.inputEl = createRef()
        this.canvasEl = createRef()

        // 그냥 일반적인 값을 저장하기 위해서도 사용 가능 (보통 state 객체를 통해 상태를 저장하므로, 딱히 권장되는 방법은 아님)
        this.value = createRef()

        // 값 자체에는 Ref 객체의 current 속성을 통해서 접근
```

```
        this.value.current = 1

        this.state = { trigger: true }
    }

    render() {
        console.log('render')

        return (
            <>
                {/* ref 속성에 앞서 생성한 ref 객체를 전달하는 방식으로 DOM 요소를 저장
*/}
                <input type='text' ref={this.inputEl} />
                <br />
                <button onClick={() => {
                   this.setState((state) => ({ trigger: !state.trigger}) )
                }}>Trigger re-render {`(${this.state.trigger})`}</button>
                <br />
                <button onClick={() => {
                    // ref 값은 변경해도 re-render 하지 않음을 유의!
                    this.value.current++
                    console.log(this.value.current)
                }}>Update ref {`(${this.value.current})`}</button>
                <br />
                <button onClick={() => {
                    // 값 자체에는 current를 이용하여 접근
                    // ref에 저장된 값은 DOM 노드
                    this.inputEl.current.focus()
                }}>Focus input</button>
                <hr />
                <canvas width="200" height="200" ref={this.canvasEl} />
                <br />
                <button onClick={() => {
                    const c = this.canvasEl.current
                    // 리액트를 통해서 특정 요소만 가지고 있는 고유 속성 및 메소드에는
접근하지 못하므로 ref 값을 통해 직접 DOM 요소 접근
                    const ctx = c.getContext("2d")

                    const grd = ctx.createLinearGradient(0, 0, 200, 0)
                    grd.addColorStop(0, this.state.trigger ? "red" : "white")
                    grd.addColorStop(1, this.state.trigger ? "white" : "red")

                    ctx.fillStyle = grd
                    ctx.fillRect(0, 0, 200, 200)
                }}>Draw gradient</button>
            </>
        )
    }
}

ReactDOM.render(<RefDemo />, document.getElementById("root"))
```

Controlled vs Uncontrolled 컴포넌트

- 리액트에서 폼 관련 요소 다루기

- 폼의 입력 요소 => **자체적으로 상태를 가질 수 있다는 특징**을 가짐
    - ex1) input(type은 text) => 입력창에 쓰여진 텍스트 내용이 상태
    - ex2) input(type은 checkbox) => 체크박스의 체크 여부가 상태
    - ex3) select 요소 => 어떤 항목이 선택되었는지 여부가 상태
    - 즉, **컴포넌트의 this.state 값과 무관한, 개별적으로 상태를 가지는 요소가 있을 수 있음**
- **Controlled 컴포넌트 (제어 컴포넌트)**
    - 이러한 폼의 입력 요소에 따르는 상태를 **컴포넌트의 상태(state)와 동기화**하려는 전략
        - 컴포넌트 상태를 **신뢰 가능한 단일 출처(single source of truth)**로 만들어 두 요소를 결합
            - 즉, **컴포넌트 상태값을 이용하여 폼을 제어**
    - 일반적으로 **제어 컴포넌트 사용이 권장**됨

component-advanced/controlled-component.js

```
import React from 'react'
import ReactDOM from 'react-dom'

class FormControlledComponent extends React.Component {
    constructor(props) {
        super(props);

        // 입력을 받는 태그의 내용과 동기화 할 값을 저장할 수 있도록 state 객체 구성
        this.state = {
            text: '',
            textareaText: '',
            checked: false,
            selected: 'default'
        }
    }
    // 입력을 받는 태그(input, textarea, select 등)의 변화에 반응할 이벤트 핸들러들 정
의
    // e는 이벤트 객체
    handleTextChange = e => {
        this.setState( {
            text: e.target.value
        });
    }
    handleTextareaTextChange = e => {
        this.setState( {
            textareaText: e.target.value
        });
    }
    handleCheckChange = e => {
        this.setState({
            checked: e.target.checked
        });
    }
    handleSelectChange = e => {
        this.setState({
            selected: e.target.value
        });
    }
    render() {
        return (
```

```
            <form>
                <p>text : {this.state.text}</p>
                {/*
                    1. 컴포넌트 상태와 input 입력값을 동기화하기 위해서 value 속성으로
현재 state 값 전달
                    2. 내용이 변경될 때 호출될 메소드를 onChange 속성으로 전달
                */}
                <input type="text" value={this.state.text} onChange=
{this.handleTextChange} /><br />
                <p>textarea text : {this.state.textareaText}</p>
                <textarea value={this.state.textareaText} onChange=
{this.handleTextareaTextChange} /><br />
                <p>checked : {this.state.checked + ""}</p>
                <input type="checkbox" onChange={this.handleCheckChange}
checked={this.state.checked} /><br />
                <p>selected : {this.state.selected + ""}</p>
                <select value={this.state.selected} onChange=
{this.handleSelectChange}>
                    <option value="default">Default</option>
                    <option value="item1">Item 1</option>
                    <option value="item2">Item 2</option>
                    <option value="item3">Item 3</option>
                </select>
                <br />
            </form>
        );
    }
}

ReactDOM.render(<FormControlledComponent />, document.getElementById("root"))
```

- 폼에 포함된 여러 input 요소에 각각의 이벤트 처리 핸들러를 달아주기는 힘들기 때문에 다음 링크의 트릭 활용 가능
    - https://reactjs.org/docs/forms.html#handling-multiple-inputs

component-advanced/reservation-form.js

```
import React, { Component, createRef } from 'react'
import ReactDOM from 'react-dom'

class ReservationForm extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            name: '',
            date: '',
            isForeigner: false,
            roomNumber: 'one'
        };
    }
    handleInputChange = (event) => {
        const target = event.target;
        const value = target.type === 'checkbox' ? target.checked :
target.value;
        const name = target.name;
```

```
        // ES6에서 도입된 computed property names 문법 활용
        // https://eloquentcode.com/computed-property-names-in-javascript
        this.setState({
            // 태그의 name 속성값을 속성키로 사용
            [name]: value
        });
    }
    handleSubmit = (e) => {
        alert("submit!");
        // 필요한 네트워크 요청(ex: ajax) 보내기
        // (입력 요소와 상태가 동기화되어 있으므로, 필요한 내용은 전부 state 객체에서 참조 가능)
        e.preventDefault();
    }
    render() {
        return (
            <form>
                <p>{JSON.stringify(this.state)}</p>
                <hr />
                <label>이름 <input value={this.state.name} name="name" type="text" onChange={this.handleInputChange} /></label><br />
                <label>날짜 <input value={this.state.date} name="date" type="date" onChange={this.handleInputChange} /></label><br />
                <label>외국인 여부 <input checked={this.state.isForeigner} name="isForeigner" type="checkbox" onChange={this.handleInputChange} /></label><br />
                <select name="roomNumber" value={this.state.roomNumber} onChange={this.handleInputChange}>
                    <option value="one">1개</option>
                    <option value="two">2개</option>
                    <option value="three">3개</option>
                </select>
                <br />
                <input type="submit" value="제출" onClick={this.handleSubmit} />
            </form>
        );
    }
}

ReactDOM.render(<ReservationForm />, document.getElementById("root"))
```

- **Uncontrolled 컴포넌트 (비제어 컴포넌트)**
    - 폼의 입력 요소들의 개별적 상태를 그대로 두고, 필요한 경우 **DOM 요소에 직접 접근**하여 필요한 정보를 얻어내려는 전략

component-advanced/uncontrolled-component.js

```
import React, { Component, createRef } from 'react'
import ReactDOM from 'react-dom'

class FormUncontrolledComponent extends React.Component {
    constructor(props) {
        super(props);

        // ref 생성
```

```
        this.input = React.createRef();
        // input type="file"의 경우 읽기 전용 요소이므로 비제어 요소로 취급해야 함
        this.fileInput = React.createRef();
    }
    handleSubmit = (e) => {
        // this.input.current => input 요소
        const v = this.input.current.value;
        const file = this.fileInput.current.value; // 파일 경로 및 파일명
        alert(v + " " + file);
        e.preventDefault();
    }
    render() {
        return (
            <form>
                {/* input 요소와 ref 연결 */}
                <input type="text" ref={this.input} /><br />
                <input type="file" ref={this.fileInput} /><br />
                <input type="submit" onClick={this.handleSubmit} />
            </form>
        );
    }
}

ReactDOM.render(<FormUncontrolledComponent />, document.getElementById("root"))
```

- file 타입의 input 태그의 경우 코드를 이용하여 값을 설정할 수 없으므로(값을 읽기만 가능하므로) 제어권을 가질 수 없음, 따라서 이 경우는 반드시 uncontrolled 상태로 가정하고 코드 작성을 해야 함

Forwading Refs (@)

- https://reactjs.org/docs/forwarding-refs.html

```
component-advanced/forwarding-refs.js
```

- 특정 컴포넌트의 내부에 자식 컴포넌트를 두고 싶은데 자식 컴포넌트에 **부모 컴포넌트에서 ref 객체를 통해 참조하고 싶은 요소가 있는 경우** forwarding refs 기법 사용
  - React 클래스의 **forwardRef 함수 사용하여 props 값과 ref 값 전달** 가능
  - 부모 컴포넌트에서 자식 컴포넌트를 렌더링하는 과정에서 ref 요소로 전달한 ref 객체를 통해 참조 가능

함수형 컴포넌트 (권장)

```
const FancyButtonFunc = React.forwardRef((props, ref) => (
    <button ref={ref} className="FancyButton" onClick={props.onClick}>
        {props.children}
    </button>
));
```

클래스 컴포넌트

```
class FancyButtonClass extends Component {
    render() {
```

```
        return (
            <button ref={this.props.forwardRef} onClick={this.props.onClick}>
                {this.props.children}
            </button>
        )
    }
}

const ForwardedFancyButtonClass = React.forwardRef(
    (props, ref) => <FancyButtonClass {...props}
                        forwardRef={ref}
                        onClick={props.onClick}/>)
```

```
class UsingFancyButtonComponent extends Component {
    constructor(props) {
        super(props)

        // 기존 방식과 동일하게 ref 객체 생성
        this.fancyButtonRef1 = React.createRef()
        this.fancyButtonRef2 = React.createRef()
    }

    render() {
        return (
            <div>
                {/* 부모 컴포넌트에서 ref 객체 생성 후 자식 컴포넌트로 전달하기
(forwarding) */}
                <FancyButtonFunc ref={this.fancyButtonRef1} onClick={()=>{
                    /*
                        이후에는 ref 객체를 통해 컴포넌트 내부의 DOM에 직접 접근 가능
                        (단, 여기서 참조하는 current 값은 FancyButton 컴포넌트가 아니
라 FancyButton 내부의 button 요소임을 유의!)
                    */
                    this.fancyButtonRef2.current.disabled = true
                    setTimeout(() => {
                        this.fancyButtonRef2.current.disabled = false
                    }, 1000)
                }}>
                    Disable Button 2
                </FancyButtonFunc>
                <br />
                <ForwardedFancyButtonClass ref={this.fancyButtonRef2} onClick=
{()=>{
                    this.fancyButtonRef1.current.disabled = true
                    setTimeout(() => {
                        this.fancyButtonRef1.current.disabled = false
                    }, 1000)
                }}>
                    Disable Button 1
                </ForwardedFancyButtonClass>
            </div>
        )
    }
}
```

HoC (Higher-order component) (@)

- 고차 컴포넌트는 마치 고차 함수가 함수를 전달받아 함수를 반환하는 것과 같이 **컴포넌트를 전달받아 (더 향상된 기능을 제공하는) 컴포넌트를 반환하는 컴포넌트**
  - 컴포넌트 => HoC로 전달 => 더 기능이 향상된 컴포넌트로 업그레이드
  - 보통 여러 컴포넌트에서 **반복되는 특정 작업(로직)을 추상화**하여 똑같은 반복 코드가 작성되지 않도록 하기 위해 HoC를 작성
  - react-router의 withRouter, redux의 connect도 모두 HoC
- https://reactjs.org/docs/higher-order-components.html

어떠한 기능도 없는, 가장 간단한 형태의 HoC

```
// 컴포넌트를 인자로 제공
const higherOrderComponent = (WrappedComponent) => {
    // 새로운 컴포넌트 정의하고
    class HOC extends React.Component {
        /* 필요한 추가 작업과 관련된 코드를 새 컴포넌트에 추가 */
        render() {
            // render 메소드 내에서 전달받은 컴포넌트를 자식 컴포넌트로 그려줌
            return <WrappedComponent />;
        }
    }
    // 반환
    return HOC;
};
```

컴포넌트에 로깅 기능을 제공하는 HoC

- componentDidUpdate를 재정의하고 해당 메소드 내부에서 수정된 props 값을 출력하도록 설정
- 실질적으로는 **LogProps 클래스가 사용**되므로, LogProps 클래스의 render 메소드가 호출되는 과정에서 전달받은 컴포넌트가 그려지고, 이후 componentDidUpdate 메소드 호출이 이루어지는 과정에서 필요한 추가 로직(로그 출력)이 실행됨

```
component-advanced/hoc-1.js
```

```
// 함수로 전달되는 WrappedComponent는 컴포넌트
function logProps(WrappedComponent) {
    class LogProps extends React.Component {
        // 여기서는 라이프사이클 메소드를 재정의하여 기존 컴포넌트를 강화
        componentDidUpdate(prevProps) {
            console.log('old props:', prevProps)
            console.log('new props:', this.props)
        }

        render() {
            // 전달받은 컴포넌트를 그대로 그려주면서, 전달받은 props도 그대로 전달
            // (즉, 추후 LogProps 클래스를 렌더링하는 시점에 전달한 props 값들이 그대로 자식 컴포넌트로 전달됨)
            return <WrappedComponent {...this.props} />
```

```
        }
    }

    return LogProps
}
```

HoC를 사용하여 Counter 컴포넌트의 기능을 강화

- 여기서 EnhancedComponent는 전달한 Counter 클래스가 아닌, LogProps 클래스임을 유의

```
class Counter extends Component {
    render() {
        return <div>{this.props.count}</div>
    }
}

// HoC를 호출하며 기존에 정의한 컴포넌트 전달
const EnhancedComponent = logProps(Counter)
```

강화된 컴포넌트 사용

```
class App extends Component {
    constructor(props) {
        super(props)

        this.state = {
            count: 0
        }
    }

    componentDidMount() {
        this.state.intervalId = setInterval(() => {
            this.setState((state) => {
                return { count: state.count + 1}
            })
        }, 1000)
    }

    componentWillUnmount() {
        clearTimeout(this.state.intervalId)
    }

    render() {
        // count 값은 먼저 LogProps로 전달되고, 이후 render 메소드 내부에서 Counter로
전달됨
        return <Counter count={this.state.count} />
        // EnhancedComponent를 렌더링한 결과와 비교해보기
        /* return <EnhancedComponent count={this.state.count} /> */
    }
}
```

네트워크 요청을 보낼 특정 prefix 주소가 있고, id를 이용해서 데이터를 가져오는 컴포넌트로 기능을 확장해주는 HoC

- 여기서는 HoC가 상태를 가지고, componentDidMount 메소드를 호출하는 과정에서 (가짜) 네트워크 요청을 보내며 네트워크 요청이 끝나는 시점에 data 값을 설정하며 render 메소드가 호출되는 과정에서 전달받은 컴포넌트를 그리며 data 값을 props를 통해 전달
- 여전히 전달받은 컴포넌트(공통 로직을 추가할 컴포넌트)는 render 메소드에서 그려지고 바깥에서 그려지는 것은 반환되는 HOC임을 유의! (참고로 여기서는 **익명 클래스를 반환**하므로 클래스 이름이 없음)

component-advanced/hoc-2.js

```
function withFetchedData(WrappedComponent, prefix) {
    return class extends React.Component {
        constructor(props) {
            super(props)

            // state 내부에 data에 요청한 데이터를 저장한다고 가정
            this.state = {
                data: null
            }
        }

        componentDidMount() {
            const url = `${prefix}${
                this.props.id === undefined ? '' : '/' + this.props.id }`

            // 여기서 네트워크 요청을 보낸다고 가정하고 setTimeout으로 가짜 요청 처리
            setTimeout(() => {
                // 상태 변경, 상태가 변경되었으므로 render 메소드가 호출되고 그 과정에서
props를 통해 전달받은 컴포넌트로 네트워크 요청 결과 값(data)을 전달함
                this.setState({
                    data: data[url]
                })
            }, 1000)
        }

        render() {
            // https://stackoverflow.com/questions/54824123/can-i-pass-
component-state-to-hoc
            // state를 직접 건드린다기 보다는 props을 통해서 state 값을 전달해야 함
            return <WrappedComponent data={this.state.data} {...this.props} />
        }
    }
}
```

사용할 가짜 데이터

```
const data = {
    'http://api.server.com/users/1': {
        'name': 'John',
        'age': 20
    },
    'http://api.server.com/favorites/2': [
        'Game', 'Movie'
    ],
    'http://api.server.com/todos': [
        '자바스크립트 공부', '리액트 공부'
    ]
}
```

HoC를 사용할 컴포넌트들 정의

- HoC에서 수행하는 작업은 **네트워크 요청 이후 전달받은 내용을 data라는 이름으로 접근할 수 있도록 props를 통해 전달하는 것**이므로 이후 각 컴포넌트마다 달리해야 할 작업(ex: 로딩창 표시, 데이터 출력)은 각 컴포넌트에서 알아서 처리하도록 구현해야 함
    - HoC의 목적은 **컴포넌트가 공유해야 하는 공통 로직을 구현**하는 것
        - 무엇이 개별적인 로직이고, 무엇이 공통적인 로직인지 솎아내는 것이 HoC 구현에서 가장 중요한 쟁점

```
const UserInfo = withFetchedData(class extends Component {
    render() {
        // HoC로부터 전달받은 데이터 확인
        const data = this.props.data
        // 아직 전달받고 있는 중이면 로딩창 표시
        if(data === null) return <div>Loading User Info...</div>

        // 전달이 모두 끝난 시점에 내용 표시
        return (
            <div>
                name : {data.name} age : {data.age}
            </div>
        )
    }
}, "http://api.server.com/users")
```

```
const FavoritesInfo = withFetchedData(class extends Component {
    render() {
        const data = this.props.data
        if(data === null) return <div>Loading Favorites Info...</div>

        // 여기서는 리스트 타입 데이터를 전달받으므로 목록 보여주도록 처리
        return (
            <ul>
                {
                    data.map((f, idx) => {
                        return (<li key={idx}>
                            {f}
                        </li>)
                    })
                }
            </ul>
        )
```

```
    }
}, "http://api.server.com/favorites")
```

```
const TodoInfo = withFetchedData(class extends Component {
    render() {
        const data = this.props.data
        if(data === null) return <div>Loading Todo Info...</div>

        return (
            <ul>
                {
                    data.map((todo, idx) => {
                        return (<li key={idx}>
                            {todo}
                        </li>)
                    })
                }
            </ul>
        )
    }
}, "http://api.server.com/todos")
```

ms 단위의 시간을 전달받아 해당 시간만큼 지날때마다 count 상태값을 증가시켜주는 HoC

```
component-advanced/hoc-3.js
```

```
function withCounter(WrappedComponent, ms) {
    class WithCounter extends React.Component {
        constructor(props) {
            super(props)

            this.state = {
                count: 0
            }
        }

        componentDidMount() {
            const intervalId = setInterval(() => {
                // 이 시점에서 상태값(count)이 계속 바뀌게 되며 render 메소드 재호출에 의해서 전달받은 컴포넌트가 그려짐
                // (그 과정에서 자식 컴포넌트로 count값 및 타이머 제어를 위한 콜백 함수(stop, resume)를 전달)
                this.setState((state) => {
                    return { count: state.count + 1 }
                })
            }, ms)
            this.setState({ intervalId: intervalId });
        }

        componentWillUnmount() {
            clearInterval(this.state.intervalId)
        }

        stop = () => {
```

```
            clearInterval(this.state.intervalId)
        }

        resume = () => {
            const intervalId = setInterval(() => {
                this.setState((state) => {
                    return { count: state.count + 1 }
                })
            }, ms)
            this.setState({ intervalId: intervalId });
        }

        render() {
            return <WrappedComponent
                count={this.state.count}
                stop={this.stop}
                resume={this.resume}
                {...this.props} />
        }
    }

    return WithCounter
}
```

HoC 사용 컴포넌트 1

```
const ProgressBar = withCounter(class extends Component {
    render() {
        const progress = (this.props.count % 100)

        return (
            <div style={{
                width: `${progress}%`,
                background: "red",
                height: "10px"
            }}>
            </div>
        )
    }
}, 16)
```

HoC 사용 컴포넌트 2

```
const App = withCounter(class extends Component {
    constructor(props) {
        super(props)
    }

    render() {
        return <div style={{ margin: "0 auto", width: "50%"}}>
            <ProgressBar />
            <h1>{this.props.count}</h1>
            <button onClick={() => {
                this.props.resume()
            }}>resume</button>
            <button onClick={() => {
```

```
                this.props.stop()
            }}>stop</button>
        </div>
    }
}, 1000)
```

리퍼런스

- 폼
  - https://reactjs.org/docs/forms.html
  - https://reactjs.org/docs/uncontrolled-components.html
  - https://itnext.io/controlled-vs-uncontrolled-components-in-react-5cd13b2075f9
  - https://stackoverflow.com/questions/42522515/what-are-react-controlled-components-and-uncontrolled-components
  - https://goshakkk.name/controlled-vs-uncontrolled-inputs-react/
- ref
  - https://blog.logrocket.com/why-you-should-use-refs-sparingly-in-production/
  - https://stackoverflow.com/questions/59522254/how-can-i-store-a-ref-in-an-array
- hoc