Case1 : JSX를 사용해 생성하는 React Element

케이스 주제

JSX

페이스북의 React 개발팀에서 **React** 를 소개할 때 3가지를 중점으로 설명합니다.

(*선언형, 컴포넌트 기반, 한 번 배워서 어디서나 사용하기*)

그중에서도 **Declarative** (선언형)은 이번 주제와 깊은 연관성이 있는데요

프론트엔드 개발을 하다 보면 유저 인터렉션이 많은 UI를 만들 때 어려움이 발생합니다.

이에 대응하여 React 는 데이터 변경에 알아서 렌더링 할 수 있는 방법을 제공하고 개발자가 각 상태에 대한 뷰만 설계할 수 있도록 돕습니다.

구체적으로 **Declarative (선언형) View** 를 작성하여 예측 가능한 코드로 드러내는 것을 위해 사용할 수 있는 것이 **JSX** 입니다.

JSX(Javascript XML) 는 마치 **HTML** 처럼 보이지만 **JavaScript** 로 인식할 수 있습니다.

이러한 동작이 가능하기 위해 내부적으로 다양한 동작을 하겠지만 기본적으로 **Babel** 이라는 도구를 활용해 트랜스파일링 할 수 있습니다.

React.createElement

React를 처음 접할 때 **JSX** 만으로 컴포넌트를 작성했다면 `React.createElement`를 알지 못하더라도 큰 문제는 없습니다.

사실 **JSX** 는 `React.createElement()`의 **Syntactic Sugar** (문법적 설탕)이기 때문입니다.

기본적으로 `React.createElement()`는 네이밍에서 알 수 있는 의미 그대로 **React Element** 를 생성하고 반환합니다.

또한 **React Element** 는 React에서 가장 작은 단위라고도 할 수 있습니다.

기능요구사항

1. React.createElement를 구현하기
2. JSX Element를 JSON 포맷으로 표시하기

```
// JSX Element
  const element = (
```

```
    <div>
      <h1>Hello World</h1>
    </div>
  )
```

```
// React.createElement()
  const element = React.createElement('div', null, React.createElement('h1', null, 'Hello World'))
```

JSON 포맷

```
{
    "type": {
      "type": "div",
      "props": {
        "children": [
          {
            "type": "h1",
            "props": {
              "children": [
                {
                  "type": "TEXT_ELEMENT",
                  "props": {
                    "nodeValue": "Hello World",
                    "children": []
                  }
                }
              ]
            }
          }
        ]
      }
    },
    "props": {
      "children": []
    }
  }
```

기능작동이미지

{ type: { type: "div", props: { children: [{ type: "h1", props: { children: [{ type: "TEXT_ELEMENT", props: { nodeValue: "Hello World", children: [] } }] } }] } }, props: { children: [] } }

문제

JSX Element를 JSON 포맷으로 표시하기

- createElement 함수 작성하기
- createTextElement 함수 작성하기

문제 풀이를 위해서는 2가지 함수만 작성하면 됩니다.
단순히 2개의 함수를 구현한다고 해서 React의 모든 매커니즘과 동작 방식을 이해할 수는 없습니다.
때문에 React의 내부 동작을 풀어낸다는 생각으로 문제에 접근한다면 React 개발팀이나 해박한 지식을 가진 개발자가 아니라면 풀어내기 어려울 수 있습니다.
`React.createElement()`를 흉내내며 모방한다는 생각으로 해당 케이스의 문제 풀이에 접근하고 시도해봅시다!

```
function createElement(type, props = {}, ...children) {
    // TODO: Write code

    return {
      type,
      props: {},
    }
  }

  function createTextElement(value) {
    // TODO: Write code
  }
```

주요 학습 키워드

1. JSX
2. React.createElement

작성해주셔야하는 question 파일경로

`./question/index.js`

실행 방법 및 의존성 모듈 설치

경로 : `./question/index.html`

`index.html`에 복잡한 **Babel** 설정을 하지 않기 위해 `CDN`이 포함되어 있으니 `live-server` 등을 이용해 확인하시면서 구현할 수 있습니다.

Reference

JSX Live Editor : https://jsx.egoist.sh
Babel REPL : https://babeljs.io/repl/#?presets=react&code_lz=GYVwdgxgLglg9mABACwKYBt1wBQEpEDeAUIogE6pQhlIA8AJjAG4B8AEhlogO5xnr0AhLQD0jVgG4iAXyJA

Case1 : JSX를 사용해 생성하는 React Element - 출제자 해설

HTML

```
<!DOCTYPE html>
  <html lang="ko">
    <head>
      <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width, initial-scale=1" />
      <title>Case1. JSX를 사용해 생성하는 React Element / Solution</title>
      <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/pretty-print-json@1.0/dist/pretty-print-json.css" />

      <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
      <script src="https://cdn.jsdelivr.net/npm/pretty-print-json@1.0/dist/pretty-print-json.min.js"></script>
    </head>

    <body>
      <div id="root"></div>
    </body>

    <script type="text/babel" data-type="module" src="./index.js"></script>
  </html>
```

JS

index.js

```
function createElement(type, props = {}, ...children) {
    return {
      type,
      props: {
        ...props,
        children: children.map((child) => (typeof child === 'string' ? createTextElement(child) : child)),
      },
    }
  }

  function createTextElement(value) {
    return createElement('TEXT_ELEMENT', { nodeValue: value })
  }
```

```
const React = {
  createElement,
}

// 런타임시 각 Node를 트랜스파일러인 Babel에 알려주기 위해 참조합니다.
/** @jsx React.createElement */
const element = (
  <div>
    <h1>Hello World</h1>
  </div>
)

const container = document.getElementById('root')

container.innerHTML = prettyPrintJson.toHtml(React.createElement(element))
```

Case2 : DOM Elements 렌더링

케이스 주제

Case1에서는 `React.createElement()`를 구현해보며 **React** 에서의 **Element** 가 어떻게 구성되고 만들어지는지 이해해보려고 했습니다.
또한 앞서 생성된 **React Element** 는 **React** 에서 가장 작은 단위라고 설명드렸었는데요.
일반적인 객체 (**Plain Object**)라고도 할 수 있습니다.
페이스북의 React 개발팀에서 **React**를 소개할 때 3가지를 중점으로 설명합니다.
(*선언형, 컴포넌트 기반, 한 번 배워서 어디서나 사용하기*)
그중에서도 **Learn Once, Write Anywhere (한 번 배워서 어디서나 사용하기)** 가 이번 케이스에 일부분 연관되어 있습니다.
React 를 기본적으로 사용할때 누구나 습관처럼 당연하게 사용하는 구문이 있을겁니다.

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

ReactDOM.render(element, container)
```

보통 최상위 컴포넌트에서 위의 코드처럼 **ReactDOM** 을 가져온 후 `ReactDOM.render()`을 사용하게 됩니다.
여기서 `ReactDOM.render()`는 최상위에서 사용할 수 있는 몇 안 되는 **메서드** 로 **DOM** 에 특화되어 있으며
첫 번째 인자로 제공된 **React Element** 를 두 번째 인자인 **Container Element** 의 내부에 렌더링시킬 수 있습니다.

기능요구사항

1. ReactDOM.render() 구현하기
2. render() 메서드를 구현하고 DOM 에 렌더링하기

이전 케이스에서 `React.createElement()`를 통해 **React Element** 를 만들었다면,
이번에는 **React Element Tree** 를 생성하는 `render()` 메서드를 구현하고 **DOM** 에 렌더링까지 하는 것이 목표입니다.

```
const element = (
  <div>
    <h1>Hello World</h1>
  </div>
)

const rootElement = document.getElementById('root')
ReactDOM.render(element, rootElement)
```

기능 작동 이미지

문제

아래의 `render()` 함수 구문의 내부를 작성하여 위의 **React Element** 를 `rootElement` 내부에 렌더링해보도록 하겠습니다.

```
function render({ props, type }, container) {
  // TODO: Write code
}
```

React 는 자식 노드들을 **재귀적으로** 렌더링한다는 것을 유념하면 구현시 큰 힌트가 될 수 있습니다.
단순히 `render()`를 구현한다고해서 **ReactDOM** 의 모든 매커니즘과 동작 방식을 이해할 수는 없습니다.
때문에 React의 내부 동작을 풀어낸다는 생각으로 문제에 접근한다면 React 개발팀이나 해박한 지식을 가진 개발자가 아니라면 풀어내기 어려울 수 있습니다.

`ReactDOM.render()`를 흉내내며 모방한다는 생각으로 해당 케이스의 문제 풀이에 접근하고 시도해봅시다!

주요 학습 키워드

> ReactDOM.render()

작성해주셔야 하는 question 파일경로

`./question/index.js`

실행 방법 및 의존성 모듈 설치

경로 : `./question/index.html` `index.html`
에 복잡한 Babel 설정을 하지 않기 위해 CDN이 포함되어 있으니 `live-server` 등을 이용해 확인하시면서 구현할 수 있습니다.

Reference

> Understanding Rendering Behavior in React : https://geekflare.com/react-rendering
> React Components, Elements, and Instances : https://medium.com/@dan_abramov/react-components-elements-and-instances-90800811f8ca
> React (Virtual) DOM Terminology : https://gist.github.com/sebmarkbage/fcb1b6ab493b0c77d589

Case2 : DOM Elements 렌더링 - 출제자 해설

HTML

```
<!DOCTYPE html>
  <html lang="ko">
    <head>
      <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width, initial-scale=1" />
      <title>Case2. DOM Elements 렌더링 / Solution</title>

      <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    </head>

    <body>
      <div id="root"></div>
    </body>

    <script type="text/babel" data-type="module" src="./index.js"></script>
  </html>
```

JS

index.js

```
// 가상 요소
  function createElement(type, props = {}, ...children) {
    return {
      type,
      props: {
        ...props,
        children: children.map((child) => (typeof child === 'string' ? createTextElement(child) : child
      },
    }
  }

  // 가상 텍스트 요소
  function createTextElement(value) {
    return createElement('TEXT_ELEMENT', { nodeValue: value })
```

```
}

function render({ props, type }, container) {
  const v = Object.entries(props).reduce(
    (totalNode, [key, value]) => {
      if (key !== 'children') {
        totalNode[key] = value
      }

      return totalNode
    },
    type === 'TEXT_ELEMENT' ? document.createTextNode('') : document.createElement(type)
  )

  props.children.forEach((child) => render(child, v))
  container.appendChild(v)
}

const React = {
  createElement,
}

const ReactDOM = {
  render,
}

// 런타임시 각 Node를 트랜스파일러인 Babel에 알려주기 위해 참조합니다.
/** @jsx React.createElement */
const element = (
  <div>
    <h1>Hello World</h1>
  </div>
)

const rootElement = document.getElementById('root')
ReactDOM.render(element, rootElement)
```

Case4: React State

케이스 주제

React Component의 상태는 State 라는 개념으로 다룹니다. 이 State는 변경 가능하며 변경될때마다 Component는 리렌더링 됩니다.
Class Component와 Function Component에 따라 State 관련 코드를 작성하는 방식과 내부적으로 동작하는 방식이 조금 다릅니다.
다만 State를 직접 변경하지 않는다는 점과 State가 변경되면 리렌더링이 트리거되는 점은 여전히 같습니다.
Function Component에서의 State는 useState에서 다룰 예정이니 이번 케이스에서는 Class Component에서의 State에 대해서 다룹니다.
State와 리렌더링이 어떻게 동작하는지 알아보기 위해서 간단하게 아래와 같은 형태로 Component 클래스를 작성해봅니다.

```
class Component {
  setState(newState) {}
  render() {}
}
```

요구 및 참고사항

setState와 render 메서드를 갖는 Component 클래스를 만듭니다.
setState 메서드는 아래와 같이 작동해야 합니다.

- newState 파라미터에 대한 유효성 검사를 합니다.
    - newState는 객체 또는 함수 타입을 받습니다. 그 외의 타입이 오면 에러를 내도록 합니다.
- newState가 객체일 경우에는 기존의 state와 newState를 병합합니다.
- newState가 함수일 경우에는 newState 함수에 현재 state와 props를 전달해 실행하고 반환된 새 state를 기존의 state와 병합합니다.

- 마지막으로 리렌더링을 합니다.

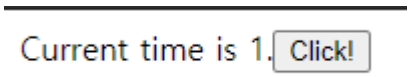
이 Component를 html에 초기 렌더링할 수 있는 render 함수를 만듭니다.

- render 함수는 컴포넌트와 컴포넌트가 렌더링될 공간인 컨테이너 두 파라미터를 받습니다.
- 받은 컴포넌트를 인스턴스화 하고 컴포넌트의 렌더링 결과물을 컨테이너의 child로 붙입니다.

이렇게 만든 Component와 render를 import 해서 활용하는 코드를 만듭니다.

- Component 클래스를 상속해서 만든 커스텀 컴포넌트에서 setState를 호출해서 리렌더링이 되도록 합니다.
- 만약 this.state를 직접 업데이트하면 리렌더링이 안되도록 합니다.
- 마지막으로 이렇게 만든 커스텀 컴포넌트를 import 한 redner 함수를 이용해 렌더링 합니다.

기능 작동 이미지



문제

Q. State에 따라 렌더링을 하는 React Class Component를 단순한 방식으로 모방해서 만들어보세요.

주요 학습 키워드

JavaScript

- Class
- Arrow Function

Function Parameter Validation

작성해주셔야 하는 question 파일 경로

`./question/index.js`

`./question/React.js`

실행 방법

경로 `./question`

index.html 파일을 브라우저로 열거나 로컬 웹 서버로 실행하기

```
$ npx serve -l 3000
```

Case4 : react state - 출제자 해설

HTML

```
<!DOCTYPE html>
  <html lang="en">
    <head>
      <meta charset="UTF-8" />
      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      <title>React State</title>
    </head>
    <body>
      <div id="root"></div>
      <script type="module" src="./index.js"></script>
    </body>
  </html>
```

JS

index.js

```
import { Component, render } from './React.js'

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

    render() {
      const timer = new Timer()
      const div = document.createElement('div')
      div.appendChild(timer.render())
      return div
```

```
    }
  }

  class Timer extends Component {
    constructor(props) {
      super(props)
      this.state = {
        time: 0,
      }

      const intervalId = setInterval(() => {
        this.setState((prevState) => ({
          time: prevState.time + 1,
        }))
      }, 1000)
    }

    render() {
      const textContent = `Current time is ${this.state.time}.`

      // 리렌더링을 간단하게 처리합니다.
      if (this.selfElement) {
        this.selfElement.firstChild.textContent = textContent
        return
      }

      const div = document.createElement('div')
      this.selfElement = div
      const span = document.createElement('span')
      const button = document.createElement('button')

      span.textContent = textContent
      button.textContent = 'Click!'
      button.addEventListener('click', () => {
        // setState를 통해 업데이트 했을 경우
        this.setState((prevState) => ({
          time: prevState.time + 1,
        }))

        // state를 직접 업데이트 했을 경우
        // this.state = {
        //   time: this.state.time + 1,
        // }
      })

      div.appendChild(span)
      div.appendChild(button)

      return div
    }
  }
```

```
render(App, document.getElementById('root'))
```

React.js

```
const ALLOWED_STATE_TYPES = ['object', 'function']

  class Component {
    constructor(props) {
      this.props = props
    }

    setState(newState) {
      // Validation
      if (!ALLOWED_STATE_TYPES.includes(typeof newState)) {
        throw new Error('Type of passed state is not object or function.')
      }

      if (typeof newState === 'object') {
        this.state = {
          ...this.state,
          ...newState,
        }
      }

      if (typeof newState === 'function') {
        this.state = {
          ...this.state,
          ...newState(this.state, this.props),
        }
      }

      this.render()
    }

    render() {}
  }

  function render(ComponentToRender, container) {
    const component = new ComponentToRender()
    container.appendChild(component.render())
  }

  export { Component, render }
```

Case5: React Props

케이스 주제

React Component는 Props 라는 불변(immutable) 객체를 받습니다.
이 Props는 Properties를 의미하며 컴포넌트 트리에서 항상 위에서 아래로만 흐르는 단방향 특성을 갖습니다.
그리고 컴포넌트는 전달받은 Props를 변경할 수 없습니다.
이는 React가 컴포넌트를 Props 라는 arguments를 받아 UI 정보를 반환하는 하나의 단순한 함수로 바라보는 철학에서 비롯된 것입니다.
다시 말해, 어떤 컴포넌트에 동일한 Props를 전달하면 항상 동일한 UI 결과가 나올 것이라고 확신할 수 있고
이 높은 확신에서 UI를 더 선언적으로 바라보게 되는 점, 코드 가독성 향상, 테스트하기 쉬운 점 등의 이점을 얻게 됩니다.

문제

Q. Props에 받아 렌더링을 하는 React Function Component를 만들어서 렌더링 해보세요.

요구 및 참고사항

DogCard 라는 함수 컴포넌트를 독립된 파일로 만듭니다.

- 이 컴포넌트는 name, imageUrl, age, breed, owner라는 Props를 받습니다.
- 한번 렌더링이 되면 props와 만들어진 element를 캐쉬합니다.
- 여러번 렌더링 되었을때 Props가 바뀌지 않았으면(캐쉬한 props와 비교) 리렌더링을 하지 않습니다.
- 전달받은 Props가 immutable 한지 검사하고, immutable 하지 않다면 에러를 방출합니다.

이 DogCard 컴포넌트를 렌더링하는 App 컴포넌트를 만듭니다.
컴포넌트와 컨테이너 두 파라미터를 받아서 id가 root인 div tag에 컴포넌트를 렌더링하는 render 함수를 만들어 활용합니다.
App 컴포넌트 또는 DogCard 컴포넌트를 최소한 두번 리렌더링 합니다.

- DogCard 컴포넌트에 같은 Props를 전달했을때와
- 같은 Props를 전달했을때를 비교합니다. 같은 Props를 전달해서 두번 이상 리렌더링하면 DogCard 컴포넌트는 한번만 렌더링 되어야 합니다.

기능 작동 이미지

주요 학습 키워드

JavaScript

- Function
    - Function Parameter Validation
- Object.seal() & Object.isSealed()
- Object.freeze() & Object.isFrozen()
- Object.getOwnPropertyDescriptor()

참고 문서

Why are React props immutable?:
https://stackoverflow.com/questions/47471131/why-are-react-props-immutable
Props vs. State

- https://lucybain.com/blog/2016/react-state-vs-pros/
- https://github.com/uberVU/react-guide/blob/master/props-vs-state.md

Difference between freeze and seal:
https://stackoverflow.com/questions/21402108/difference-between-freeze-and-seal
Object.getOwnPropertyDescriptor():
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyDescriptor

작성해주셔야 하는 question 파일 경로

`./question/index.js`

`./question/React.js`

`./question/DogCard.js`

실행 방법

경로 `./question`

index.html 파일을 브라우저로 열거나 로컬 웹 서버로 실행하기

```
$ npx serve -l 3000
```

Case5 : react props - 출제자 해설

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React Props</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="./index.js"></script>
  </body>
</html>
```

JS

index.js

```
import React from './React.js'
import DogCard from './DogCard.js'

/**
 * React Component를 Props를 받아 UI를 반환하는 함수로 가정한다.
 */
function App({ propsForDogCard }) {
  const app = document.createElement('div')
  app.style = `
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: 20px;
    padding: 15px;
  `

  const renderedDogCard = DogCard(propsForDogCard)
  app.appendChild(renderedDogCard)

  return app
}

// Object.seal()
// - 객체에 속성을 추가하거나 제거하는 것을 막는다.
// - 객체를 non-configurable로 만든다. descriptor 변경이 불가능해진다.
// - sealed 된 객체의 값을 변경하면 TypeError를 방출한다.
// Object.freeze()
// - seal()이 하는 일을 똑같이 하며
// - 추가로, 기존의 property들을 변경하는 것을 막는다.
const propsForDogCard = Object.freeze({
  name: 'Briton',
  imageUrl:
    'https://mblogthumb-phinf.pstatic.net/MjAxODA3MTNfMjQz/MDAxNTMxNDQ30DcyMTEy.0J440rl_srCV1Cc2YOUsJ4SUlYcEfbSY-ogTzscKOAYg.BFVU24-KEWT1i16UDzeHxP2
  age: 3,
  breed: 'Golden Retriever',
  owner: 'Charles Brown',
})
// freeze를 하지 않으면 configurable과 writable이 모두 true여서 객체의 속성 변경 및 추가가 가능하다. -> props 로서는 적절하지 않다.
// const propsForDogCard = {
```

```
//   name: 'Briton',
//   imageUrl: 'https://ww.namu.la/s/db6df9d8f32a265abab22f79f0e954cc056451a76751d1211516f5c02c9af0a7ff0602e3f327ab1f916c2ad4c8cfc00465003ae8251628b
//   age: 3,
//   breed: 'Golden Retriever',
//   owner: 'Charles Brown',
// }

// 똑같은 props로 세번 렌더링을 해도 DogCard 컴포넌트의 렌더링은 한번만 된다.
React.render(
  () =>
    App({
      propsForDogCard,
    }),
  document.getElementById('root')
)

React.render(
  () =>
    App({
      propsForDogCard,
    }),
  document.getElementById('root')
)

React.render(
  () =>
    App({
      propsForDogCard,
    }),
  document.getElementById('root')
)

// 새로운 props를 전달하면 DogCard 컴포넌트가 두번 렌더링 된다.
React.render(
  () =>
    App({
      propsForDogCard: Object.freeze({
        name: 'Charlie',
        imageUrl:
          'https://img1.daumcdn.net/thumb/R1280x0.fjpg/?fname=http://t1.daumcdn.net/brunch/service/user/3XvY/image/pF-v9rZiGBquMBQh633BmSm7CUY.jpg'
        age: 5,
        breed: 'Basenji',
        owner: 'Jeffrey',
      }),
    }),
  document.getElementById('root')
)
```

React.js

```
function render(ComponentToRender, container) {
  // Validation
  if (!container) {
    throw new Error('Container should be provided to render a component.')
  }

  while (container.firstChild) {
    container.removeChild(container.lastChild)
  }
  container.appendChild(ComponentToRender())
}

export default {
  render,
}
```

DogCard.js

```
let prevProps = null
let prevElement = null

function DogCard(props) {
  // Props immutability validation
```

```
  if (typeof props !== 'object') {
    throw new Error('Props should be an object.')
  }

  const isPropsMutable = Object.keys(props)
    .map((key) => Object.getOwnPropertyDescriptor(props, key))
    .some((descriptor) => descriptor.configurable || descriptor.writable)
  if (isPropsMutable || !Object.isFrozen(props) || !Object.isSealed(props)) {
    throw new Error('Props should be immutable.')
  }

  if (prevProps !== null) {
    const shouldComponentUpdate = Object.keys(prevProps).some((Key) => prevProps[Key] !== props[Key])

    if (!shouldComponentUpdate) {
      return prevElement
    }
  }

  const dogCard = document.createElement('div')
  dogCard.style = `
    display: flex;
    flex-direction: column;
    padding: 10px;
    width: 300px;
    height: 600px;
    background-color: ivory;
    border: 5px solid blue;
    border-radius: 30px;
  `
  dogCard.innerHTML = `
    <h1>Name: ${props.name}</h1>
    </br>
    <img src=${props.imageUrl} alt="dog" >
    </br>
    <h2>Age: ${props.age}</h2>
    </br>
    <h3>Breed: ${props.breed}</h3>
    </br>
    <h4>Owner: ${props.owner}</h4>
    </br>
  `

  prevProps = props
  prevElement = dogCard

  console.log('DogCard rendered.')

  return dogCard
}

export default DogCard
```

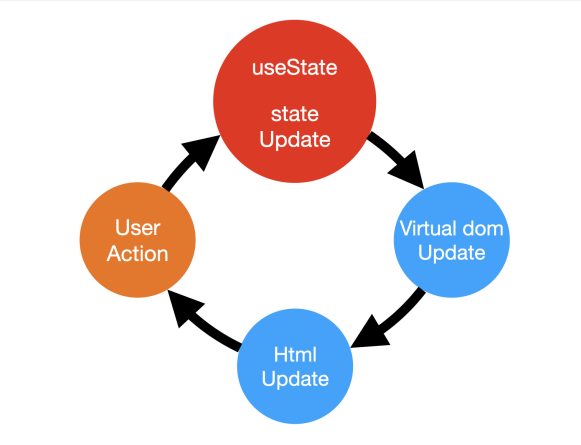
Case6 : useState

케이스 주제

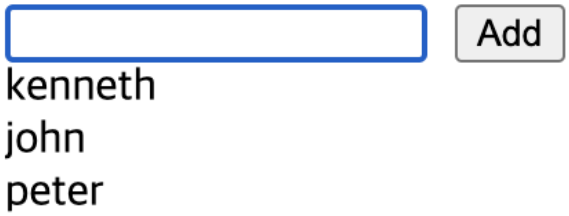
Q. react의 useState함수를 이해하기 위해 구현을 해보자.

기능 요구사항

1. user action을 정의한다.
2. action에 따른 state를 변경한다.
3. state 변경 시 virtual dom tree를 갱신한다.
4. 갱신 된 virtual dom tree로 html node로 생성하여 반영한다.



기능 작동 이미지



문제

q1. useState 함수를 [value, function]을 반환하도록 완성하시오.
q2. state가 업데이트 될 때 re rendering이 되도록 작성하시오.
q3. 데이터 변경을 위한 함수를 완성하시오.

주요 학습 키워드

useState 함수의 동작원리를 알아보자

작성해주셔야 하는 question 파일경로

`./src/question/Members.js`

실행 방법 / 문제 풀이 방법

1. npm install Run `npm install`
2. excution Run `npm run dev`

Case6 : useState - 출제자 해설

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Let's look at React's useState mechanism</title>
  </head>
  <body>
    <div id="result"></div>
  </body>
</html>
```

CSS

```
#result {
  background-color: #fff;
}
```

JS

index.js

```
import './style.css'

import Members from './question/Members'

const excute = () => {
  const targetEl = document.querySelector('#result')

  const member = Members({ list: [] })
  targetEl.appendChild(member)
}

excute()
```

Members.js

```
import { h, create, diff, patch } from 'virtual-dom'

/**
 * @name 생성자 함수
 * @param {Array} list
 * @description 생성자 함수로 최초 데이터를 받아서 virtual tree 구축과 함께 HTML node를 생성 및 저장한다.
 */
function Members({ list }) {
  // virtual dom tree
  let virtualDomTree = null
  // html node
  let htmlNode = null
  // hook이 등록될때마다 state가 저장되는 변수.
  let states = []
  // hook 실행 index
  let currentState = 0

  /**
   * @name render
   * @param {*} newState
   * @description state 변경에 따른 rendering
   */
  const render = function (state) {
    // virtual-dom 라이브러리는 html node가 변경되 전 virtual dom을 메모리에 저장하고
    // 이전 virtual dom과 비교하여 변경된 부분만 html node에 반영해주는 라이브러리 입니다.
    currentState = 0 // 새롭게 렌더링 하므로 초기화
    // 변경된 state 정보를 가지고 tree 를 갱신한다.
    const newVirtualDomTree = setVirtualTreeNode(state)
    // 변경된 정보를 체크한다.
    const changes = diff(virtualDomTree, newVirtualDomTree)

    // 기존 node에 변경된 tree 정보를 patch 한다.
    htmlNode = patch(htmlNode, changes)
    // 변경 된 tree 정보는 다시 새롭게 저장한다.
    virtualDomTree = newVirtualDomTree
  }

  /**
   * @name useState
   * @param {*} value
   * @returns [현재값, 변경 함수]
   * @description react useState함수와 같은 역할
   */
  const useState = function (initialValue) {
    states[currentState] = states[currentState] || initialValue
    // updateState 함수에서 currentState가 덮어 씌워지는 것을 방지.
    const updateStateIndex = currentState
    const updateState = (newState) => {
      states[updateStateIndex] = newState
      // data가 업데이트 되면 re render
      render(states[updateStateIndex])
    }
```

```
    // 두번째 값인 함수로 state를 변경해야만 값이 바뀌도록 한다.
    return [states[currentState++], updateState]
  }

  /**
   * @name setVirtualTreeNode
   * @param {Array} list
   * @returns virtual node
   * @description list를 인자로 받아 data가 binding 된 virtual tree 구축 및 action 함수 정의.
   */
  const setVirtualTreeNode = function (list = []) {
    const [members, setMembers] = useState(list)

    const onKeyUp = (event) => {
      if (window.event.keyCode === 13) {
        members.push(event.target.value)
        event.target.value = ''
        setMembers(members)
      }
    }

    const addRow = () => {
      const inputElement = document.querySelector('#memberInput')
      members.push(inputElement.value)
      inputElement.value = ''
      setMembers(members)
    }

    return h('div', { style: 'width: 100%; margin-top: 10px' }, [
      h('div', { style: 'width: 100%; position: relative;' }, [
        h('input', { id: 'memberInput', type: 'text', onkeyup: (event) => onKeyUp(event) }, []),
        h('button', { style: 'margin-left: 10px;', onclick: () => addRow() }, ['Add']),
      ]),
      h('div', { id: 'list-container', style: 'height: 300px; overflow: auto;' }, [
        members && members.length
          ? members.map((item, index) => h('span', { key: 'item' + index, style: 'display: block;' }, [item
          : [],
      ]),
    ])
  }

  // virtual tree node를 갱신
  virtualDomTree = setVirtualTreeNode(list)
  //virtual tree node를 html node로 생성
  htmlNode = create(virtualDomTree)
  return htmlNode
}

export default Members
```

Case7 : useEffect

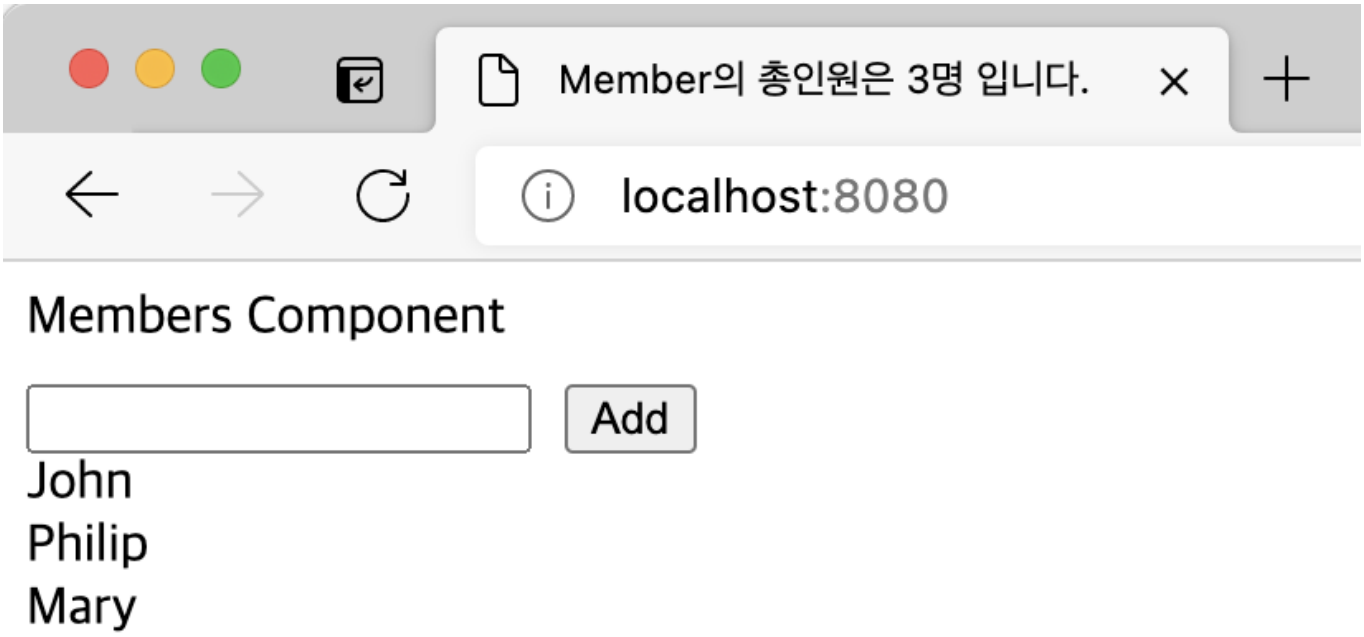
케이스 주제

Q. react의 useEffect함수를 이해하기 위해 구현을 해보자.

기능 요구사항

1. effect를 실행하는 함수와 의존성 데이터를 저장한다.
2. html element를 렌더링 후에 effect를 실행할 수 있도록 한다.
3. 의존성 데이터를 비교하여 effect 실행여부를 결정한다.
4. effect 함수를 통해 document의 title을 변경한다.

기능 작동 이미지



문제

q1. effect를 실행함수와 의존성 데이터를 저장하는 useEffect 함수를 작성하시오.
q2. effect 함수에는 document의 title에 데이터의 갯수를 표현하여 변경을 확인할 수 있도록 작성하시오.
q3. 의존성 데이터 여부에 따라 effect를 실행하는 함수를 작성하시오. case 1) 의존성 데이터가 undefined일 경우에는 effect를 매번 실행 case 2) 의존성 데이터가 있을 경우에는 데이터 변경 여부를 체크하여 effect를 실행 (참고: object-hash라이브러리를 통해서 변경여부를 쉽게 파악할 수 있음)
q4. effect 함수가 반환함수가 있다면 (cleanup) effect 실행전에 실행하도록 작성하시오.

주요 학습 키워드

useEffect 함수의 동작원리를 알아보자

작성해주셔야 하는 question 파일경로

`./src/question/Members.js`

실행 방법 / 문제 풀이 방법

1. npm install Run `npm install`
2. excution Run `npm run dev`

Case7 : useEffect - 출제자 해설

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>React의 useEffect 따라해보기</title>
  </head>
  <body>
    <div>
      <span>Members Component</span>
    </div>
    <div>
      <button id="remove-btn">Remove Component</button>
    </div>
    <div id="result"></div>
  </body>
</html>
```

CSS

```
#result {
  background-color: #fff;
}
```

JS

index.js

```
import './style.css'

import Members from './question/Members'

const excute = () => {
  document.querySelector('#remove-btn').addEventListener('click', () => {
```

```
    targetEl.removeChild(member)
  })

  const targetEl = document.querySelector('#result')

  const member = new Members({ list: [] })
  targetEl.appendChild(member)
}

excute()
```

Members.js

```
import { h, create, diff, patch } from 'virtual-dom'

/**
 * @name 생성자 함수
 * @param {Array} list
 * @description 생성자 함수로 최초 데이터를 받아서 virtual tree 구축과 함께 HTML node를 생성 및 저장한다.
 */
function Members({ list }) {
  // virtual dom tree
  let virtualDomTree = null
  // html node
  let htmlNode = null
  // hook이 등록될때마다 state가 저장되는 변수.
  const states = []
  // hook 실행 index
  let currentState = 0
  // effect hook이 등록될때마다 effect가 저장되는 변수.
  const effects = []
  // 현재 effect index
  let currentEffect = 0

  /**
   * @name render
   * @param {*} newState
   * @description state 변경에 따른 rendering
   */
  const render = function (state) {
    // 새롭게 랜더링 하므로 초기화
    currentState = currentEffect = 0
    // 변경된 state 정보를 가지고 tree 를 갱신한다.
    const newVirtualDomTree = setVirtualTreeNode(state)
    // 변경된 정보를 체크한다.
    const changes = diff(virtualDomTree, newVirtualDomTree)

    // 기존 node에 변경된 tree 정보를 patch 한다.
    htmlNode = patch(htmlNode, changes)
    // 변경 된 tree 정보는 다시 새롭게 저장한다.
    virtualDomTree = newVirtualDomTree
  }

  /**
   * @name useState
   * @param {*} value
```

```
 * @returns [현재값, 변경 함수]
 * @description react useState함수와 같은 역할
 */
const useState = function (initialValue) {
  states[currentState] = states[currentState] || initialValue
  // updateState 함수에서 currentState가 덮어 씌워지는 것을 방지.
  const updateStateIndex = currentState
  const updateState = (newState) => {
    states[updateStateIndex] = newState
    // data가 업데이트 되면 re render
    render(states[updateStateIndex])
  }

  // 두번째 값인 함수로 state를 변경해야만 값이 바뀌도록 한다.
  return [states[currentState++], updateState]
}

/**
 * @name useEffect
 * @param {effect, deps}
 * @returns void
 * @description effect를 실행할 함수와 의존성 데이터를 array로 받는다.
 */
const useEffect = function (effect, deps) {
  // effect를 실행하기전 cleanup이 등록이 되어 있으면 실행 후 초기화 한다.
  if (effects[currentEffect] && effects[currentEffect].cleanup) {
    effects[currentEffect].cleanup()
    effects[currentEffect].cleanup = undefined
  }

  // effect 및 deps를 저장한다.
  if (!effects[currentEffect]) {
    effects[currentEffect] = {
      effect: undefined, // effect 함수
      prevDeps: undefined, // 이전 의존성 데이터.
      newDeps: undefined, // 새로운 의존성 데이터.
      cleanup: undefined, // cleanup을 실행할 함수.
    }
  }
  effects[currentEffect].newDeps = deps
  effects[currentEffect].effect = effect
  // 다음 hook을 위해 증가.
  currentEffect++
}

/**
 * @name executeEffect
 * @param {effect}
 * @returns void
 * @description rendering 후에 effect를 실행.
 */
function executeEffect(effect) {
  // case 1) 의존성 데이터가 undefined일 경우에는 effect를 매번 실행
  // case 2) 의존성 데이터가 있을 경우에는 데이터 변경 여부를 체크하여 effect를 실행
  if (!effect.newDeps) {
    // deps가 undefinde 일 경우 effect를 항상 실행.
    const cleanup = effect.effect()
    effect.cleanup = cleanup
  } else {
```

```
      // 최초 effect 실행
      if (!effect.prevDeps) {
        const cleanup = effect.effect()
        effect.cleanup = cleanup
        effect.prevDeps = effect.newDeps
        return
      }
      const isChangedDeps = !effect.newDeps.every((deps, index) => deps === effect.prevDeps[index])
      // deps가 빈 배열이라면 최초한번만 실행이 된다. why? 체크해야할 의존성 데이터가 없기 때문에
      if (isChangedDeps) {
        const cleanup = effect.effect()
        effect.cleanup = cleanup
        effect.prevDeps = effect.newDeps
      }
    }
  }

  /**
   * @name setVirtualTreeNode
   * @param {Array} list
   * @returns virtual node
   * @description list를 인자로 받아 data가 binding 된 virtual tree 구축 및 action 함수 정의.
   */
  const setVirtualTreeNode = function (list = []) {
    const [members, setMembers] = useState(list)

    useEffect(() => {
      document.title = `Member의 총인원은 ${members.length}명 입니다.`
      return () => {
        document.title = 'React의 useEffect 따라해보기.'
      }
    }, [members.length])

    const onKeyUp = (event) => {
      if (window.event.keyCode === 13) {
        members.push(event.target.value)
        event.target.value = ''
        setMembers(members)
      }
    }

    const addRow = () => {
      const inputElement = htmlNode.querySelector('#memberInput')
      members.push(inputElement.value)
      inputElement.value = ''
      setMembers(members)
    }

    return h('div', { style: 'width: 100%; margin-top: 10px' }, [
      h('div', { style: 'width: 100%; position: relative;' }, [
        h('input', { id: 'memberInput', type: 'text', onkeyup: (event) => onKeyUp(event) }, []),
        h('button', { style: 'margin-left: 10px;', onclick: () => addRow() }, ['Add']),
      ]),
      h('div', { id: 'list-container', style: 'height: 300px; overflow: auto;' }, [
        members && members.length
          ? members.map((item, index) => h('span', { key: 'item' + index, style: 'display: block;' }, [item
          : [],
      ]),
    ])
```

```
  }

  function addEvent(htmlNode) {
    // 해당 이벤트로 dom update 변경을 감지하도록 한다. why? effect는 렌더링후에 실행되기 때문.
    htmlNode.addEventListener('DOMNodeInserted', (event) => {
      effects.forEach((effect) => {
        executeEffect(effect)
      })
    })
    // element가 삭제되면 (unmount 시점) cleanup을 실행한다.
    htmlNode.addEventListener(
      'DOMNodeRemovedFromDocument',
      (event) => {
        effects.forEach((effect) => {
          if (effect.cleanup) {
            effect.cleanup()
          }
        })
      },
      false
    )
  }

  // virtual tree node를 갱신
  virtualDomTree = setVirtualTreeNode(list)
  //virtual tree node를 html node로 생성
  htmlNode = create(virtualDomTree)
  addEvent(htmlNode)
  return htmlNode
}

export default Members
```

Case8: React.useRef

케이스 주제

React의 useRef hook은 기본적으로 원하는 DOM Node에 접근해 reference를 얻는 것에 목적이 있습니다.
그러나 이렇게 DOM Node에 접근하는 용도로도 쓰이지만 useRef는 컴포넌트의 full lifetime 동안 레퍼런스가 유지되는 mutable object를 반환하기 때문에
이전 값을 유지하는 용도로도 사용될 수 있습니다.
즉, useRef는 매 렌더마다 레퍼런스가 고정된 객체를 반환합니다.
useRef는 아래와 같은 function signature를 갖고 있습니다.
특정 타입을 갖는 초깃값을 받아 current 프로퍼티에 그 값을 갖는 리터럴 객체를 반환합니다.
이때 이 객체는 seal이나 freeze가 되어있지 않은 객체여서 기존의 프로퍼티를 수정 및 삭제하거나 새 프로퍼티를 추가할 수 있습니다.

```
function useRef<T>(initialValue: T | null): RefObject<T>
```

이때 React hooks는 내부적으로 memory cells라 불리는 객체를 만들어 각 hook들이 몇번, 어떤 순서로 호출되었는지 저장 및 추적합니다.
그 뒤 컴포넌트의 state 및 props가 변경되어 rerendering이 되면 hook들이 동일한 순서대로 호출되기 때문에
hook을 여러번 사용하더라도 memory cells를 보고 올바른 값을 반환할 수 있습니다.
hooks를 조건문이나 반복문에서 사용할 수 없는 것이 바로 이런 이유에서 입니다.

문제

1. useRef를 여러번 사용하는 코드를 만들어보세요.
    - (option) memoryCells & cursor
2. useRef 함수의 로직을 작성해보세요.
    - (option) mocking react event system

요구 및 참고사항

useRef는 (initialValue) 한개의 파라미터를 받습니다.

- 이 initialValue는 어떤 값을 가질 수도 있고 null 일 수도 있습니다.
- 이 initialValue는 `{ current: ... }`
  와 같이 리터럴 객체의 current 프로퍼티에 대입됩니다. 값이 전달되지 않으면 undefined가 대입됩니다.

useRef는 이렇게 만들어진 리터럴 객체를 반환합니다.
useRef는 컴포넌트가 여러번 리렌더링이 되더라도 캐쉬된 객체를 반환합니다.

- (option) useRef이 여러번 호출되어도 각 호출에 대해 어떤 값을 반환해야 할지 저장할 memoryCells 객체를 만들어 사용합니다.
- hook이 몇번 호출되었는지 추적할 수 있도록 cursor라는 변수를 만들어 사용합니다.
- (option) useRef 호출부에서 rerendering이 되었을때 cursor를 0으로 초기화할 수 있도록 CustomEvent를 사용합니다.

기능 작동 이미지

- App rendering
- App rendering
- Cached object is returned.
- myRef reference is the same.
- normalObject reference is not the same.

주요 학습 키워드

DOM(문서 객체 모델):
https://developer.mozilla.org/ko/docs/Web/API/Document_Object_Model/Introduction

참고 문서

useRef Basics: https://reactjs.org/docs/hooks-reference.html#useref
Introduction to useRef Hook: https://dev.to/dinhhuyams/introduction-to-useref-hook-3m7n

작성해주셔야 하는 question 파일 경로

`./question/index.js`

`./question/useRef.js`

실행 방법

경로 `./question`

index.html 파일을 브라우저로 열거나 로컬 웹 서버로 실행하기

```
$ npx serve -l 3000
```

Case8 : useRef - 출제자 해설

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React.useRef</title>
  </head>
  <body>
    <ul class="info"></ul>
    <script type="module" src="./index.js"></script>
  </body>
</html>
```

JS

index.js

```
import useRef from './useRef.js'

let isRerendered = false
let previousRef = null
let previousObject = null
const infoList = document.querySelector('.info')
infoList.style = `
  margin: 20px auto;
  font-size: 20px;
  padding: 0;
  width: 500px;
`

export function addInfo(message) {
  const li = document.createElement('li')
  li.textContent = message
  infoList.appendChild(li)
```

```
}

// mocking react event system
document.addEventListener('rerender', () => {
  if (isRerendered) {
    return
  }

  isRerendered = true
  document.dispatchEvent(new CustomEvent('initialize'))
  App() // rerender component
})

function App() {
  addInfo('App rendering')

  // 리렌더링 됐을때 useRef로 만든 객체와 일반 객체의 레퍼런스를 비교해보자.
  const myRef = useRef(100)
  const normalObject = {
    current: 100,
  }

  // useRef로 만든 객체는 리렌더링 되어도 레퍼런스가 유지된다.
  if (previousRef === null) {
    previousRef = myRef
  } else {
    addInfo(previousRef === myRef ? 'myRef reference is the same.' : 'myRef reference is not the same.'
  }

  // 일반 객체는 컴포넌트가 리렌더링 되면 새 레퍼런스로 교체된다.
  if (previousObject === null) {
    previousObject = normalObject
  } else {
    addInfo(
      previousObject === normalObject
        ? 'normalObject reference is the same.'
        : 'normalObject reference is not the same.'
    )
  }

  setTimeout(() => {
    document.dispatchEvent(new CustomEvent('rerender'))
  }, 1000 * 2)
}

App()
```

useRef.js

```
import { addInfo } from './index.js'

const memoryCells = {}
let cursor = 0
```

```
document.addEventListener('initialize', () => {
  cursor = 0
})

function useRef(initialValue) {
  if (memoryCells[cursor] !== undefined) {
    addInfo('Cached object is returned.')
    const cachedObject = memoryCells[cursor]
    cursor++

    return cachedObject
  }

  const newObject = {
    current: initialValue,
  }
  memoryCells[cursor] = newObject
  cursor++

  return newObject
}

export default useRef

// debugging 용
window.memoryCells = memoryCells
```

Case9: React.memo

케이스 주제

React에는 어떤 값이나 함수 또는 컴포넌트를 memoize 할 수 있는 방안으로 memo, useMemo, useCallback 등을 제공합니다.
이 중에 이번에 알아볼 memo는 아래의 function signature와 같이 컴포넌트와 memoize 여부를 판단하는 함수를 받아 memoized 된 컴포넌트를 반환합니다.

```
function memo<T extends ComponentType<any>>(
  Component: T,
  propsAreEqual?: (prevProps: Readonly<ComponentProps<T>>, nextProps: Readonly<ComponentProps<T>>) => boolean
): MemoExoticComponent<T>
```

요구 및 참고사항

memo가 받을 컴포넌트는 함수 컴포넌트를 받는 것으로 가정합니다.
memo는 (Component, areEqual) 라는 두 파라미터를 받습니다.

- 이때 areEqual은 optional 입니다.
- areEqual이 전달되지 않았을 경우에는 default compare function이 사용됩니다.
- default compare function 역할을 할 함수를 따로 작성해서 사용합니다.

(Component, areEqual) 두 파라미터에 대한 유효성 검사를 합니다.

- 유효성 검사에 실패할 경우, 각 경우에 알맞는 메세지와 함께 에러를 발생합니다.

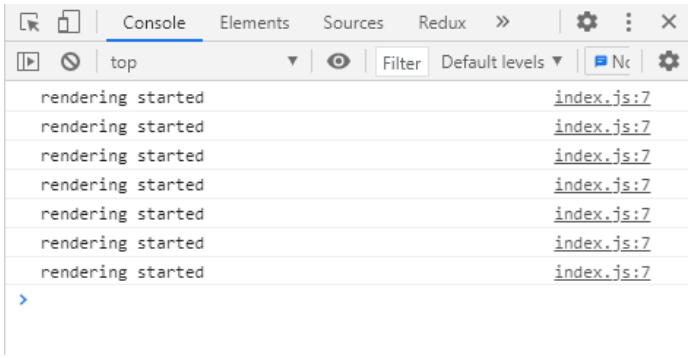
마지막으로 memoized 된 함수 컴포넌트를 반환합니다.

기능 작동 이미지

이번 예제의 결과는 console 창에서만 확인 가능합니다

브라우저 콘솔창 여는 법 (chrome 기준)

> 윈도우 : Ctrl + Shift + J / F12
> 맥 : Command + Option + J



문제

Q. React.memo 함수를 단순한 방식으로 모방해서 만들어보세요.

주요 학습 키워드

JavaScript

- Closure
- Arrow Function
- Default Function Parameter

Function Parameter Validation

작성해주셔야 하는 question 파일 경로

`./question/index.js`

`./question/memo.js`

실행 방법

경로 `./question`

index.html 파일을 브라우저로 열거나 로컬 웹 서버로 실행하기

```
$ npx serve -l 3000
```

Case9 : React.memo - 출제자 해설

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React.memo</title>
  </head>
  <body>
    <script type="module" src="./index.js"></script>
  </body>
</html>
```

JS

index.js

```
import memo from './memo.js'

/**
 * 컴포넌트를 props를 받아서 렌더링 정보를 담은 객체를 반환하는 함수로 가정한다.
 */
const Span = (props) => {
  console.log('rendering started')

  return {
    type: 'span',
    props,
  }
}
```

```
// 렌더링이 세번 일어난다.
Span({
  title: 'This is a title.',
})
Span({
  title: 'This is a title.',
})
Span({
  title: 'This is a title.',
})

const memoizedSpan = memo(Span)

// 같은 props로 컴포넌트를 여러번 호출했을때
memoizedSpan({
  title: 'This is a title.',
})
memoizedSpan({
  title: 'This is a title.',
})
memoizedSpan({
  title: 'This is a title.',
})

// 다른 props로 컴포넌트를 여러번 호출했을때
memoizedSpan({
  title: 'This is a title. 1',
})
memoizedSpan({
  title: 'This is a title. 2',
})
memoizedSpan({
  title: 'This is a title. 3',
})
```

memo.js

```
// memo 함수의 areEqual 파라미터에 default로 사용될 비교 함수
const shallowCompare = (prevProps, nextProps) => {
  for (const key in nextProps) {
    // 비교 메커니즘을 더 엄격하게 할 수록 더 섬세하게 memoize 할 수 있다.
    if (nextProps[key] !== prevProps[key]) {
      return false
    }
    return true
  }
}

const memo = (Component, areEqual = shallowCompare) => {
```

```
  // Validation
  if (typeof Component !== 'function') {
    throw new Error('Only function component can be memoized.')
  }

  if (areEqual !== undefined && typeof areEqual !== 'function') {
    throw new Error('areEqual should be a function.')
  }

  let prevProps = {}
  let memoizedResult = null

  // Clousre를 활용해 memoize 된 결과가 있을 경우 그 결과를 반환하는 함수를 반환한다.
  return (nextProps) => {
    if (memoizedResult !== null && areEqual(prevProps, nextProps)) {
      return memoizedResult
    }

    prevProps = nextProps
    memoizedResult = Component(nextProps)

    return memoizedResult
  }
}

export default memo
```

Case10: React.useCallback

케이스 주제

React에는 어떤 값이나 함수 또는 컴포넌트를 memoize 할 수 있는 방안으로 memo, useMemo, useCallback 등을 제공합니다.
이 중에 이번에 알아볼 useCallback은 hooks로서 Function Component의 body에서 작동합니다.
아래의 function signature와 같이 memoize를 할 대상 함수와 memoize 여부 판단에 사용될 dependency array를 받아 memoized 된 함수를 반환합니다.

```
function useCallback<T extends (...args: any[]) => any>(callback: T, deps: DependencyList): T
```

이때 React hooks는 내부적으로 memory cells라 불리는 객체를 만들어 각 hook들이 몇번, 어떤 순서로 호출되었는지 저장 및 추적합니다.
그 뒤 컴포넌트의 state 및 props가 변경되어 rerendering이 되면 hook들이 동일한 순서대로 호출되기 때문에 hook을 여러번 사용하더라도 memory cells를 보고 올바른 값을 반환할 수 있습니다.
hooks를 조건문이나 반복문에서 사용할 수 없는 것이 바로 이런 이유에서 입니다.

요구 및 참고사항

useCallback은 (targetCallback, dependencyArray) 두 파라미터를 받습니다.
(option) useCallback이 여러번 호출되어도 각 호출에 대해 어떤 값을 반환해야 할지 저장할 memoryCells 객체를 만들어 사용합니다.

- hook이 몇번 호출되었는지 추적할 수 있도록 cursor라는 변수를 만들어 사용합니다.

(option) useCallback 호출부에서 rerendering이 되었을때 cursor를 0으로 초기화할 수 있도록 CustomEvent를 사용합니다.

처음 받은 dependencyArray와 새로 받은 dependencyArray의 element를 각각 비교해 값이 하나라도 변경되었는지 비교하는 로직을 만듭니다.
(targetCallback, dependencyArray) 두 파라미터에 대한 유효성 검사를 합니다.

- 유효성 검사에 실패할 경우, 각 경우에 알맞는 메세지와 함께 에러를 발생합니다.

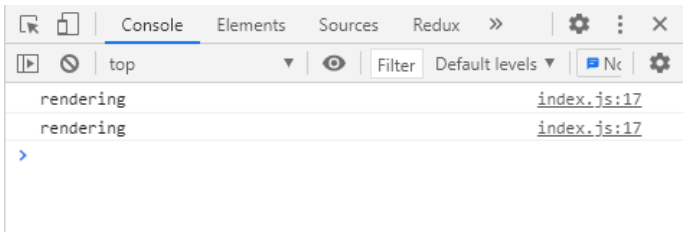
마지막으로 memoized 된 함수를 반환합니다.

기능 작동 이미지

이번 예제의 결과는 console 창에서만 확인 가능합니다

브라우저 콘솔창 여는 법 (chrome 기준)

윈도우 : Ctrl + Shift + J / F12
맥 : Command + Option + J



문제

Q. React.useCallback 함수를 단순한 방식으로 모방해서 만들어보세요.

주요 학습 키워드

JavaScript

- Closure
- Arrow Function
- CustomEvent

Function Parameter Validation

작성해주셔야 하는 question 파일 경로

`./question/index.js`

`./question/useCallback.js`

실행 방법

경로 `./question`

index.html 파일을 브라우저로 열거나 로컬 웹 서버로 실행하기

```
$ npx serve -l 3000
```

Case10 : React.useCallback - 출제자 해설

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React.useCallback</title>
  </head>
  <body>
    <script type="module" src="./index.js"></script>
  </body>
</html>
```

JS

index.js

```
import useCallback from './useCallback.js'

let isRerendered = false

// mocking react event system
document.addEventListener('rerender', () => {
  if (isRerendered) {
    return
  }

  isRerendered = true
  document.dispatchEvent(new CustomEvent('initializeHooks'))
  Div() // rerender component
})

const Div = (props) => {
  console.log('rendering')
  /**
   * useCallback dependency array에 사용될 local state
   */
  let dependencyA = 1
  let dependencyB = 'abc'
```

```
  let dependencyC = false

  // 새로운 레퍼런스 생성
  const normalFunction = () => {
    console.log('normalFunction')
  }

  const memoizedCallback1 = useCallback(() => {
    console.log('memoizedCallback1')
  }, [])

  const memoizedCallback2 = useCallback(() => {
    console.log('memoizedCallback2')
  }, [dependencyA])

  const memoizedCallback3 = useCallback(() => {
    console.log('memoizedCallback3')
  }, [dependencyA, dependencyB, dependencyC])

  setTimeout(() => {
    document.dispatchEvent(new CustomEvent('rerender'))
  }, 1000 * 2)
}

Div()
```

useCallback.js

```
const memoryCells = {}
let cursor = 0

document.addEventListener('initializeHooks', () => {
  cursor = 0
})

// 직접 만들거나 useMemo를 사용하는 방법이 있다.
// useCallback은 useMemo에 function을 넣어 사용하는 것과 동일하다.
const useCallback = (targetFunction, dependencyArray) => {
  // Validation (유효성 검사)
  if (typeof targetFunction !== 'function') {
    throw new Error('Only function can be memoized.')
  }

  if (!Array.isArray(dependencyArray)) {
    throw new Error('dependencyArray should be an array.')
  }

  if (memoryCells[cursor] !== undefined) {
    const [prevFunction, prevDependencyArray] = memoryCells[cursor]
    const isDependencyChanged = prevDependencyArray.some((element, index) => element !== dependencyArray[inde

    if (isDependencyChanged) {
      memoryCells[cursor] = [targetFunction, dependencyArray]
      cursor++

      return targetFunction
    }
```

```
    cursor++

    // memoization
    return prevFunction
  }

  memoryCells[cursor] = [targetFunction, dependencyArray]
  cursor++

  return targetFunction
}

export default useCallback

// debugging 용
window.memoryCells = memoryCells
```

Case11: React State Update & Immutability

케이스 주제

React의 Props와 State는 모두 불변성(immutability)을 유지하는 것이 필요합니다.
불변성을 유지하면서 상태를 다루게 되면 이 상태를 다루는 함수 내부에서 side effect가 발생할 염려가 줄어듭니다.
함수 내부에서 이 상태를 직접 변경할 수 없어서 추적이 용이해지기 때문입니다.
이러한 불변성을 유지함으로써 얻는 이점들 외에도 컴포넌트가 불필요하게 자주 리렌더링 되는 것을 막는 역할을 하는데,
이전 상태와 다음 상태를 비교할때 객체의 레퍼런스만 비교하는(shallow compare) 값싼 연산으로 리렌더링 여부를 결정하는데 큰 도움이 됩니다.
상태를 immutable하게 관리하는 것에 대해 알아보기 위해서 간단하게 아래와 같은 형태로 Component 클래스를 작성해봅니다.

```
class Component {
      setState(newState) {}
      render() {}
    }
```

문제

Q. State를 immutable하게 관리하는 React Class Component를 단순한 방식으로 모방해서 만들어보세요.

요구 및 참고사항

setState와 render 메서드를 갖는 Component 클래스를 만듭니다.
setState 메서드는 아래와 같이 작동해야 합니다.

- newState 파라미터에 대한 유효성 검사를 합니다.
    - newState는 객체 또는 함수 타입을 받습니다. 그 외의 타입이 오면 에러를 내도록 합니다.
- newState가 객체일 경우에는 this.nextState에 newState를 대입합니다.
- newState가 함수일 경우에는 newState 함수에 현재 state와 props를 전달해 실행하고 반환된 새 state를 this.nextState에 대입합니다.
- case4 React State 편에서처럼 state를 병합하지 않는 것은 this.state를 immutable 하게 관리할때와 그렇지 않을때의 컴포넌트 렌더링 작동에 차이를 두기 위함입니다.
- 마지막으로 this.shouldComponentUpdate(this.nextState) 함수를 실행해 true를 반환했을때만 컴포넌트를 리렌더링 하도록 합니다.

이렇게 만든 Component를 import 해서 상속받아 활용하는 코드를 만듭니다.

- 이 상속해서 만든 컴포넌트에서 setState를 호출해서 리렌더링이 되도록 합니다.
- 이 상속해서 만든 컴포넌트의 render 함수를 단순하게 외부에서 호출하는 것으로 body tag 하위에 렌더링이 되도록 합니다.
- 이 컴포넌트에서 shouldComponentUpdate(nextState) {} 메서드를 구현합니다.
- 만약 this.state를 immutable 하게 관리를 하지 않았다면 state를 다른 값으로 업데이트 하더라도 리렌더링이 되지 않도록 합니다.

기능 작동 이미지

버튼클릭 전

버튼클릭 후

주요 학습 키워드

JavaScript

- Class
- Arrow Function

Function Parameter Validation

참고 문서

Using State Correctly: https://reactjs.org/docs/state-and-lifecycle.html#do-not-modify-state-directly
React state가 불변이어야 하는 이유: https://ljs0705.medium.com/react-state가-불변이어야-하는-이유-ec2bf09c1021

작성해주셔야 하는 question 파일 경로

`./question/index.js`

`./question/React.js`

실행 방법

경로 `./question`

index.html 파일을 브라우저로 열거나 로컬 웹 서버로 실행하기

```
$ npx serve -l 3000
```

Case11 : React State Update & Immutability - 출제자 해설

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React State Update & Immutability</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="./index.js"></script>
  </body>
</html>
```

JS

index.js

```
import { Component } from './React.js'

const root = document.getElementById('root')

class Cat extends Component {
  constructor(props) {
    super(props)
    this.state = {
      name: 'Blue',
      age: 2,
      breed: 'Russian Blue',
```

```
    }
  }

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

    const div = document.createElement('div')
    const h1 = document.createElement('h1')
    const info = document.createElement('p')
    const button = document.createElement('button')

    h1.textContent = `Name: ${this.state.name}`
    info.textContent = `
      Age: ${this.state.age}
      Breed: ${this.state.breed}
    `
    button.textContent = 'Update cat state'
    button.addEventListener('click', () => {
      // 불변성을 이용해 상태를 업데이트 했을때
      this.setState((prevCat) => ({
        ...prevCat,
        age: 3,
      }))

      // 불변성을 이용하지 않고 상태를 업데이트 했을때
      // this.setState((prevCat) => {
      //   prevCat.age = 3
      //   return prevCat
      // })
    })

    div.appendChild(h1)
    div.appendChild(info)
    div.appendChild(button)

    while (root.firstChild) {
      root.removeChild(root.lastChild)
    }
    root.appendChild(div)
  }

  shouldComponentUpdate(nextState) {
    // shallow compare
    if (this.state !== nextState) {
      return true
    }

    return false
  }
}
```

```
const cat = new Cat()
cat.render()
```

React.js

```
const ALLOWED_STATE_TYPES = ['object', 'function']

export class Component {
  constructor(props) {
    this.props = props
  }

  setState(newState) {
    // Validation
    if (!ALLOWED_STATE_TYPES.includes(typeof newState)) {
      throw new Error('Type of passed state is not object or function.')
    }

    if (typeof newState === 'object') {
      this.nextState = newState
    }

    if (typeof newState === 'function') {
      this.nextState = newState(this.state)
    }

    if (this.shouldComponentUpdate !== undefined && this.shouldComponentUpdate(this.nextState)
      this.state = this.nextState
      this.render()
    }
  }

  render() {}
}
```

Case14: webpack

케이스 주제

Q. webpack 모듈 번들링의 기능을 구현한다

기능요구사항

1. webpack 사용해보기
2. minipack 으로 구현하기

기능 작동 이미지

문제

Q. dist파일안에 bundle.js를 생성하여 파일 모듈화를 해보세요

```
$ node minipack.js
```

> 위 명령어를 실행하면
> ./src 폴더안의 index.js 를 번들링 하여 ./dist/bundle.js 에 생성
> index.html 브라우저에서 실행

주요 학습 키워드

> minipack 파일을 참고

작성해주셔야 하는 question 파일 경로

`./question/minipack.js` `./question/package.json` `./question/dist/bundle.js`
github 에서 `minipack.js`, `package.json` 파일만 복사해서 붙여주시면 됩니다. (영상참고)

`bundle.js`**는** `index.js`**를 모듈화 시켜서 생성해주시면 됩니다**

실행 방법 및 의존성 모듈 설치

`minipack.js` **와** `pagkage.json` **파일을 작성하신 후 실행하시면 됩니다**

경로 : `./question`

```
$ npm install
```

```
$ node minipack.js
```

`index.html` 파일을 브라우저로 열거나,
에디터에서 제공하는 live-serve로 `index.html` 실행

Case14 : webpack - 출제자 해설

HTML

```
<!DOCTYPE html>
  <html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>webpack</title>
  </head>
  <body>

    <script src="./dist/bundle.js"></script>

  </body>
  </html>
```

JS

minipack.js

```
/**
   * Module bundlers compile small pieces of code into something larger and more
   * complex that can run in a web browser. These small pieces are just JavaScript
   * files, and dependencies between them are expressed by a module system
   * (https://webpack.js.org/concepts/modules).
   *
   * Module bundlers have this concept of an entry file. Instead of adding a few
   * script tags in the browser and letting them run, we let the bundler know
   * which file is the main file of our application. This is the file that should
   * bootstrap our entire application.
   *
   * Our bundler will start from that entry file, and it will try to understand
```

```
 * which files it depends on. Then, it will try to understand which files its
 * dependencies depend on. It will keep doing that until it figures out about
 * every module in our application, and how they depend on one another.
 *
 * This understanding of a project is called the dependency graph.
 *
 * In this example, we will create a dependency graph and use it to package
 * all of its modules in one bundle.
 *
 * Let's begin :)
 *
 * Please note: This is a very simplified example. Handling cases such as
 * circular dependencies, caching module exports, parsing each module just once
 * and others are skipped to make this example as simple as possible.
 */

const fs = require('fs');
const path = require('path');
const babylon = require('babylon');
const traverse = require('babel-traverse').default;
const {transformFromAst} = require('babel-core');

let ID = 0;

// We start by creating a function that will accept a path to a file, read
// its contents, and extract its dependencies.
function createAsset(filename) {
  // Read the content of the file as a string.
  const content = fs.readFileSync(filename, 'utf-8');

  // Now we try to figure out which files this file depends on. We can do that
  // by looking at its content for import strings. However, this is a pretty
  // clunky approach, so instead, we will use a JavaScript parser.
  //
  // JavaScript parsers are tools that can read and understand JavaScript code.
  // They generate a more abstract model called an AST (abstract syntax tree).

  // I strongly suggest that you look at AST Explorer (https://astexplorer.net)
  // to see how an AST looks like.
  //
  // The AST contains a lot of information about our code. We can query it to
  // understand what our code is trying to do.
  const ast = babylon.parse(content, {
    sourceType: 'module',
  });

  // This array will hold the relative paths of modules this module depends on.
  const dependencies = [];

  // We traverse the AST to try and understand which modules this module depends
  // on. To do that, we check every import declaration in the AST.
  traverse(ast, {
```

```
      // EcmaScript modules are fairly easy because they are static. This means
      // that you can't import a variable, or conditionally import another module.
      // Every time we see an import statement we can just count its value as a
      // dependency.
      ImportDeclaration: ({node}) => {
        // We push the value that we import into the dependencies array.
        dependencies.push(node.source.value);
      },
    });

    // We also assign a unique identifier to this module by incrementing a simple
    // counter.
    const id = ID++;

    // We use EcmaScript modules and other JavaScript features that may not be
    // supported on all browsers. To make sure our bundle runs in all browsers we
    // will transpile it with Babel (see https://babeljs.io).
    //
    // The `presets` option is a set of rules that tell Babel how to transpile
    // our code. We use `babel-preset-env` to transpile our code to something
    // that most browsers can run.
    const {code} = transformFromAst(ast, null, {
      presets: ['env'],
    });

    // Return all information about this module.
    return {
      id,
      filename,
      dependencies,
      code,
    };
  }

  // Now that we can extract the dependencies of a single module, we are going to
  // start by extracting the dependencies of the entry file.
  //
  // Then, we are going to extract the dependencies of every one of its
  // dependencies. We will keep that going until we figure out about every module
  // in the application and how they depend on one another. This understanding of
  // a project is called the dependency graph.
  function createGraph(entry) {
    // Start by parsing the entry file.
    const mainAsset = createAsset(entry);

    // We're going to use a queue to parse the dependencies of every asset. To do
    // that we are defining an array with just the entry asset.
    const queue = [mainAsset];

    // We use a `for ... of` loop to iterate over the queue. Initially the queue
    // only has one asset but as we iterate it we will push additional new assets
    // into the queue. This loop will terminate when the queue is empty.
```

```
  for (const asset of queue) {
    // Every one of our assets has a list of relative paths to the modules it
    // depends on. We are going to iterate over them, parse them with our
    // `createAsset()` function, and track the dependencies this module has in
    // this object.
    asset.mapping = {};

    // This is the directory this module is in.
    const dirname = path.dirname(asset.filename);

    // We iterate over the list of relative paths to its dependencies.
    asset.dependencies.forEach(relativePath => {
      // Our `createAsset()` function expects an absolute filename. The
      // dependencies array is an array of relative paths. These paths are
      // relative to the file that imported them. We can turn the relative path
      // into an absolute one by joining it with the path to the directory of
      // the parent asset.
      const absolutePath = path.join(dirname, relativePath);

      // Parse the asset, read its content, and extract its dependencies.
      const child = createAsset(absolutePath);

      // It's essential for us to know that `asset` depends on `child`. We
      // express that relationship by adding a new property to the `mapping`
      // object with the id of the child.
      asset.mapping[relativePath] = child.id;

      // Finally, we push the child asset into the queue so its dependencies
      // will also be iterated over and parsed.
      queue.push(child);
    });
  }

  // At this point the queue is just an array with every module in the target
  // application: This is how we represent our graph.
  return queue;
}

// Next, we define a function that will use our graph and return a bundle that
// we can run in the browser.
//
// Our bundle will have just one self-invoking function:
//
// (function() {})()
//
// That function will receive just one parameter: An object with information
// about every module in our graph.
function bundle(graph) {
  let modules = '';

  // Before we get to the body of that function, we'll construct the object that
  // we'll pass to it as a parameter. Please note that this string that we're
```

```
    // building gets wrapped by two curly braces ({}) so for every module, we add
    // a string of this format: `key: value,`.
    graph.forEach(mod => {
      // Every module in the graph has an entry in this object. We use the
      // module's id as the key and an array for the value (we have 2 values for
      // every module).
      //
      // The first value is the code of each module wrapped with a function. This
      // is because modules should be scoped: Defining a variable in one module
      // shouldn't affect others or the global scope.
      //
      // Our modules, after we transpiled them, use the CommonJS module system:
      // They expect a `require`, a `module` and an `exports` objects to be
      // available. Those are not normally available in the browser so we'll
      // implement them and inject them into our function wrappers.
      //
      // For the second value, we stringify the mapping between a module and its
      // dependencies. This is an object that looks like this:
      // { './relative/path': 1 }.
      //
      // This is because the transpiled code of our modules has calls to
      // `require()` with relative paths. When this function is called, we should
      // be able to know which module in the graph corresponds to that relative
      // path for this module.
      modules += `${mod.id}: [
        function (require, module, exports) {
          ${mod.code}
        },
        ${JSON.stringify(mod.mapping)},
      ],`;
    });

    // Finally, we implement the body of the self-invoking function.
    //
    // We start by creating a `require()` function: It accepts a module id and
    // looks for it in the `modules` object we constructed previously. We
    // destructure over the two-value array to get our function wrapper and the
    // mapping object.
    //
    // The code of our modules has calls to `require()` with relative file paths
    // instead of module ids. Our require function expects module ids. Also, two
    // modules might `require()` the same relative path but mean two different
    // modules.
    //
    // To handle that, when a module is required we create a new, dedicated
    // `require` function for it to use. It will be specific to that module and
    // will know to turn its relative paths into ids by using the module's
    // mapping object. The mapping object is exactly that, a mapping between
    // relative paths and module ids for that specific module.
    //
    // Lastly, with CommonJs, when a module is required, it can expose values by
    // mutating its `exports` object. The `exports` object, after it has been
```

```
  // changed by the module's code, is returned from the `require()` function.
  const result = `
    (function(modules) {
      function require(id) {
        const [fn, mapping] = modules[id];
        function localRequire(name) {
          return require(mapping[name]);
        }
        const module = { exports : {} };
        fn(localRequire, module, module.exports);
        return module.exports;
      }
      require(0);
    })({${modules}})
  `;

  // We simply return the result, hurray! :)
  return result;
}

const graph = createGraph('./src/index.js');
const result = bundle(graph);

fs.writeFile( './dist/bundle.js' , result , function(err){
  if(err === null){
    console.log('success');
  }else{
    console.log('fail');
  }
});
```

bundle.js

```
    (function(modules) {
      function require(id) {
        const [fn, mapping] = modules[id];
        function localRequire(name) {
          return require(mapping[name]);
        }
        const module = { exports : {} };
        fn(localRequire, module, module.exports);
        return module.exports;
      }
      require(0);
    })({0: [
      function (require, module, exports) {
```

```
          "use strict";

  var _varA = require("./varA.js");

  var _varB = require("./varB.js");

  alert((0, _varA.varA)());
  alert((0, _varB.varB)());
        },
        {"./varA.js":1,"./varB.js":2},
      ],1: [
        function (require, module, exports) {
          "use strict";

  Object.defineProperty(exports, "__esModule", {
    value: true
  });
  var text = "varA";

  var varA = exports.varA = function varA() {
    return text;
  };
        },
        {},
      ],2: [
        function (require, module, exports) {
          "use strict";

  Object.defineProperty(exports, "__esModule", {
    value: true
  });
  var text = "varB";

  var varB = exports.varB = function varB() {
    return text;
  };
        },
        {},
      ],})
```

package.json

```
{
    "name": "minipack",
    "version": "1.0.0",
    "description": "",
    "author": "Ronen Amiel",
```

```
  "license": "MIT",
  "dependencies": {
    "babel-core": "^6.26.0",
    "babel-preset-env": "^1.6.1",
    "babel-preset-es2015": "^6.24.1",
    "babel-traverse": "^6.26.0",
    "babylon": "^6.18.0",
    "eslint": "^4.17.0",
    "eslint-config-airbnb-base": "^12.1.0",
    "eslint-plugin-import": "^2.8.0"
  },
  "devDependencies": {
    "eslint-config-prettier": "^2.9.0",
    "eslint-plugin-prettier": "^2.6.0",
    "prettier": "^1.10.2"
  }
}
```

Case15 : 리덕스의 기본 동작 메커니즘

케이스 주제

Q. 리덕스는 애플리케이션의 모든 상태를 중앙 저장소에서 관리하여 리액트의 상태관리를 효율적으로 할 수 있게 도와주는 라이브러리 입니다.
(리액트 뿐만 아니라 다른 뷰 라이브러리와 함께 사용할 수 있습니다.)
이 리덕스의 핵심 코드를 직접 작성해 보면서 스토어(Store)를 직접적인 처리가 아닌 액션(Action) - 리듀서(Reducer) - 스토어(Store) - 뷰(View) 패턴으로 구현하면서 리덕스의 동작 원리를 알아봅니다.

기능 요구사항

1. 화면에는 h3 태그의 count 값에 리덕스의 상태값을 표시합니다.
2. INCREMENT 버튼을 클릭시 count 상태를 1증가 시킵니다.
3. DECREMENT 버튼을 클릭시 count 상태를 1감소 시킵니다.
4. RESET 버튼을 클릭시 count 상태를 0로 초기화 시킵니다.

기능 작동 이미지

문제

q1. redux.js - currentState에 접근할수 있도록 getState 함수를 작성하시오.
q2. redux.js - currentState, action을 파라미터로 currentReducer를 실행하여 새로운 currentState 만드는 dispatch 함수를 작성하시오.
q3. redux.js - 상태 변경시 감지를 위해 listener를 배열로 저장하는 subscribe 함수를 작성하고 dispatch 함수에 currentListeners 의 변경을 알려주는 코드를 추가하시오.

주요 학습 키워드

> Redux

작성해주셔야 하는 question 파일경로

`./question/redux.js`

실행 방법 / 풀이 방법 안내

> 문제 풀기 방식 :
>
> 1. 터미널에서 각 문제 폴더 디렉토리로 이동하여 npx serve로 서버를 실행 (또는 에디터 툴의 Live Server를 활용하여 개발서버 실행)
> 2. http://localhost:5000 접속
> 3. 코드 수정하면서 문제 해결하세요

리액트에서 리덕스를 사용할때와 거의 동일하게 파일을 구성하기 위해 script type="module" 을 활용하여 파일을 분리하였습니다.

`index.html`

을 실행하기 위해 개발서버가 필요합니다. - 문제 디렉토리에서 vscode등 에디터에서 지원해주는

`Live Server`

로 실행하거나

`npx serve`

를 이용하여 개발서버를 실행하세요.

실행 방법 및 의존성 모듈 설치

경로 `./question`

터미널

```
$ npx serve
```

Case15 : 리덕스의 기본 동작 메커니즘 - 출제자 해설

HTML

```
<!DOCTYPE html>
  <html lang="en">
    <head>
      <meta charset="UTF-8" />
      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      <title>Case15 : redux</title>
      <script src="./index.js" type="module"></script>
    </head>
    <body>
      <div id="app">
        <h3>COUNT : <span class="count">0</span></h3>
        <button class="increment">INCREMENT</button>
        <button class="decrement">DECREMENT</button>
        <button class="reset">RESET</button>
      </div>
    </body>
  </html>
```

js

redux.js

```
export const createStore = (reducer, preloadedState) => {
    let currentReducer = reducer
    let currentState = preloadedState
    let currentListeners = []

    const getState = () => {
      return currentState
```

```
    }

    const subscribe = (listener) => {
      currentListeners.push(listener)
    }

    const dispatch = (action) => {
      currentState = currentReducer(currentState, action)

      const listeners = currentListeners
      listeners.forEach((listener) => {
        listener()
      })
    }

    return {
      dispatch,
      subscribe,
      getState,
    }
  }
```

Case18 : 리덕스 미들웨어의 동작 메커니즘

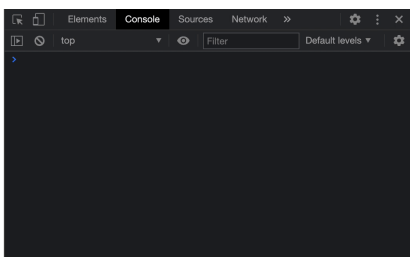
케이스 주제

Q. 리덕스 미들웨어는 액션과 스토어 사이에 임의의 기능을 넣어 확장하는 방법으로 추가적인 작업을 할 수 있습니다.
액션을 콘솔에 출력하여 로깅을 할 수 있고 비동기 작업을 처리할 수도 있습니다.
이 두가지 경우를 직접 구현하면서 리덕스 미들웨어의 동작 원리를 알아봅니다.

기능 요구사항

1. 화면에는 h3 태그의 count 값에 리덕스의 상태값을 표시합니다.
2. INCREMENT/DECREMENT 버튼을 클릭시 count 상태를 1증가/감소 시킵니다. (이때 이전상태와 액션 다음상태를 로깅합니다.)
3. GET COUNT 버튼을 클릭시 getCount 액션을 호출합니다. (이때 h3 태그의 count 값에 loading... 표시를 해줍니다.)
4. getCount 액션호출 2초후에 count 상태를 100으로 초기화 시킵니다. (이때 h3 태그의 count 값에 count 를 표시합니다.)

기능 작동 이미지



문제

q1. middleware.js - 이전상태와 액션 다음상태를 로깅하는 미들웨어 함수를 작성하시오.
q2. middleware.js - action 타입이 function일 경우 action에 store 정보를 넘겨주어 비동기 통신을 할수 있도록 thunk 미들웨어 함수를 작성하시오.
q3. redux.js - 위에서 작성한 미들웨어가 동작할수 있도록 currying을 이용하여 applyMiddleware 부분을 완성하시오. (하단의 compose 함수를 이용하여 배열로 넘어온 미들웨어를 순차적으로 실행할수 있게 합니다.)

주요 학습 키워드

Redux
Redux middleware

작성해주셔야 하는 question 파일경로

`./question/redux.js`

`./question/middleware.js`

실행 방법 / 풀이 방법 안내

문제 풀기 방식 :

1. 터미널에서 각 문제 폴더 디렉토리로 이동하여 npx serve로 서버를 실행 (또는 에디터 툴의 Live Server를 활용하여 개발서버 실행)
2. http://localhost:5000 접속
3. 코드 수정하면서 문제 해결하세요

리액트에서 리덕스를 사용할때와 거의 동일하게 파일을 구성하기 위해 script type="module" 을 활용하여 파일을 분리하였습니다.

`index.html`

을 실행하기 위해 개발서버가 필요합니다. - 문제 디렉토리에서 vscode등 에디터에서 지원해주는

`Live Server`

로 실행하거나

`npx serve`

를 이용하여 개발서버를 실행하세요.

실행 방법 및 의존성 모듈 설치

경로 `./question`

터미널

```
$ npx serve
```

Case18 : 리덕스 미들웨어의 동작 메커니즘 - 출제자 해설

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Case18 : redux middleware</title>
    <script src="./index.js" type="module"></script>
  </head>
  <body>
    <div id="app">
      <h3>COUNT : <span class="count">0</span></h3>
      <button class="increment">INCREMENT</button>
      <button class="decrement">DECREMENT</button>
      <button class="get_count">GET COUNT</button>
    </div>
  </body>
</html>
```

js

redux.js

```
export const createStore = (reducer, preloadedState, middlewares = []) => {
  let currentReducer = reducer
  let currentState = preloadedState
  let currentListeners = []
```

```
  const getState = () => {
    return currentState
  }

  const subscribe = (listener) => {
    currentListeners.push(listener)
  }

  const dispatch = (action) => {
    currentState = currentReducer(currentState, action)

    const listeners = currentListeners
    listeners.forEach((listener) => {
      listener()
    })
  }

  const store = {
    getState,
    dispatch,
    subscribe,
  }

  // applyMiddleware
  const middlewareAPI = {
    getState: store.getState,
    dispatch: (action, ...args) => dispatch(action, ...args),
  }
  const chain = middlewares.map((middleware) => middleware(middlewareAPI))
  let wrapDispatch = compose(...chain)(store.dispatch)

  return {
    ...store,
    dispatch: wrapDispatch,
  }
}

const compose = (...middlewares) => {
  return middlewares.reduce((a, b) => (...args) => a(b(...args)))
}
```

middleware.js

```
// middleware
export const logger = (store) => (next) => (action) => {
  if (typeof action === 'function') return next(action)

  console.group('LOGGER')
  console.log('prev state', store.getState())
```

```
  const result = next(action)

  console.log('action', action)
  console.log('next state', store.getState())
  console.groupEnd()

  return result
}

export const thunk = (store) => (next) => (action) => {
  typeof action === 'function' ? action(store.dispatch, store.getState) : next(action)
}
```