React.js 3주차 스터디

김재민 김후정 박지수 송동욱 연주영 오원석 임경섭 정윤지

목차

- React Chapter 5~10
- Redux
- Styled Component
- 앞으로?

React Chapter 5~10

- LifeCycle API
- input 상태 관리하기
- 배열 다루기
- 불변성을 지키는 이유와 업데이트 최적화

LifeCycle API

컴포넌트가 브라우저에서 나타날 때, 사라질 때,
그리고 업데이트 될 때 호출되는 API

- componentDidMount
- shouldComponentUpdate
- getSnapshotBeforeUpdate
- componentDidUpdate
- componentWillUnmount
- componentDidCatch

deprecated된 것들을 제외하고 이렇게 6가지의 API에 대해서 알아보았습니다.

componentDidMount

```
componentDidMount() {
  // 외부 라이브러리 연동: D3, masonry, etc
  // 컴포넌트에서 필요한 데이터 요청: Ajax, GraphQL, etc
  // DOM 에 관련된 작업: 스크롤 설정, 크기 읽어오기 등
}
```

컴포넌트가 화면에 나타날 때 호출되는 API

shouldComponentUpdate

```
shouldComponentUpdate(nextProps, nextState) {
  // return false 하면 업데이트를 안함
  // return this.props.checked !== nextProps.checked
  return true;
}
```

기본적으로 true를 반환

작성한 조건에 따라 false를 반환할 경우 render 함수 호출하지 않음

→ 컴포넌트 최적화에 유용하게 사용됨

getSnapshotBeforeUpdate

```
getSnapshotBeforeUpdate(prevProps, prevState) {
    ...

  componentDidUpdate(prevProps, prevState, snapshot) {
    ...
  }
}
```

1. render()
2. 수정이 발생하기 바로 직전에 호출
   (ex: DOM이 업데이트되기 전)
3. 실제 DOM에 변화 발생
4. 반환값은 세 번째 매개변수로 componentDidUpdate에 전달
5. componentDidUpdate와 함께 실행

componentDidUpdate

```
componentDidUpdate(prevProps, prevState, snapshot) {

}
```

컴포넌트에서 render를 호출한 뒤에 발생하는 API
파라미터를 통해 이전 값인 prevProps와 prevState를 조회할 수 있음

componentWillUnmount

```
componentWillUnmount() {
  // 이벤트, setTimeout, 외부 라이브러리 인스턴스 제거
}
```

등록했던 이벤트 제거

componentDidCatch

```
componentDidCatch(error, info) {
  this.setState({
    error: true
  });
}
```

에러가 발생할 경우 실행

Input 다루기

```
<form>
  <input
    placeholder="이름"
    value={this.state.name}
    onChange={this.handleChange}
    name="name"
  />
  <input
    placeholder="전화번호"
    value={this.state.phone}
    onChange={this.handleChange}
    name="phone"
  />
  <div>{this.state.name} {this.state.phone}</div>
</form>
```

다음과 같이 `name` 값을 부여하여 input 구별

`event.target.name` 을 통하여 조회

부모 컴포넌트에게 정보 전달하기

```
handleSubmit = (e) => {
  // 페이지 리로딩 방지
  e.preventDefault();
  // 상태값을 onCreate 를 통하여 부모에게 전달
  this.props.onCreate(this.state);
  // 상태 초기화
  this.setState({
    name: '',
    phone: ''
  })
}
```

props로 받은 `onCreate`를 호출하고, 이를 통해 상태값을 부모에게 전달

배열 다루기 - 생성과 렌더링

- map 함수

```
const a = [1,2,3,4,5];
const b = a.map(number => number * 2);
```

- 불변성 유지를 위해 고유값을 key로 사용해야 함

```
{
  id: 0,
  name: '김민준',
  phone: '010-0000-0000'
}, {
  id: 1,
  name: '홍길동',
  phone: '010-0000-0001'
}
```

배열 다루기 - 제거와 수정

- filter 함수

```
const arr = [1, 2, 3, 4, 5];
array.filter(num => num !== 3); // [1, 2, 4, 5]
```

- filter를 이용한 정보 제거

```
handleRemove = (id) => {
  const { information } = this.state;
  this.setState({
    information: information.filter(info => info.id !== id)
  })
}
```

배열 다루기 - 제거와 수정(Cont.)

- map을 이용한 정보 수정

```
const modifiedArray = array.map(item => item.id === 1
  // id 가 일치하면 새 객체를 만들고,
  // 기존의 내용을 집어넣고 원하는 값 덮어쓰기
  ? ({ ...item,. text: 'Korea' })
  : item // 바꿀 필요 없는것들은 그냥 기존 값 사용
```

불변성

```
// 불변성 유지 X
const object = {
  foo: 'hello',
  bar: 'world'
};
const sameObject = object;
sameObject.baz = 'bye';
console.log(sameObject !== object); // false
```

```
// 불변성 유지 O
const object = {
  foo: 'hello',
  bar: 'world'
};
const differentObject = {
  ...object,
  baz: 'bye'
};
console.log(differentObject !== object); // true
```

업데이트 최적화

```
shouldComponentUpdate(nextProps, nextState) {
  return nextProps.data !== this.props.data;
}
```

불변성을 잘 지키면 `shouldComponentUpdate` 로직을 이렇게 간단하게만 작성해도 업데이트 최적화가 끝납니다!!

React 요약

1. 재사용 가능한 컴포넌트를 만듭니다.
2. props 는 부모에게서 전달받는 값입니다.
3. state 는 자기 자신이 지니고 있는 데이터입니다.
4. props 나 state 가 바뀌면 컴포넌트는 리렌더링 합니다.
5. LifeCycle API 를 통해서 컴포넌트 마운트, 업데이트, 언마운트 전후로 처리 할 로직을 설정하거나 리렌더링을 막아줄수도 있습니다.

Redux

리액트 생태계에서 사용되는 상태 관리 라이브러리

- 상태 업데이트 로직의 분리
  : 복잡한 상태 업데이트 로직들을 따로 떼서 모듈화 가능
  → 유지 보수성 ↑
- 더 쉬운 상태 관리

Redux 개념

- 액션(Action) : 상태에 변화가 필요할 때 발생시키는 것. `type` 필드를 필수적으로 가지고 있어야 함.

```
{
  type: "ADD_TODO",
  data: {
    id: 0,
    text: "리덕스 배우기"
  }
}
```

- 액션 생성함수 (Action Creator): 액션을 만드는 함수로 파라미터를 받아와서 액션 객체 형태로 만들어줌.

```
function addTodo(data) {
  return {
    type: "ADD_TODO",
    data
  };
}
```

Redux 개념(Cont.)

- 리듀서(Reducer): 현재 상태와 전달 받은 액션을 참고하여 변화를 일으키는 함수
- 스토어 (Store)
  현재의 앱 상태와 리듀서, 추가적으로 몇가지 내장 함수들이 들어있는 것. 하나의 애플리케이션 당 하나의 스토어를 만듦.
- 디스패치 (dispatch)
  스토어의 내장함수 중 하나로 액션을 발생 시키는 것.
  `dispatch(action)`과 같이 호출을 하면, 스토어는 리듀서 함수를 실행시켜서 새로운 상태를 만듦.
- 구독 (subscribe): 스토어의 내장함수 중 하나로, 특정 함수를 전달해주면 액션이 디스패치 되었을 때 마다 전달해준 함수가 호출됨.

Redux의 3가지 규칙

1. 하나의 애플리케이션 안에는 하나의 스토어만 존재
2. 상태는 읽기 전용 - 기존의 상태는 건드리지 않고, 새로운 상태를 생성하여 업데이트하는 방식으로 진행
3. 리듀서(Reducer)는 순수한 함수 - `new Date()`나 랜덤 숫자 생성처럼 실행할 때마다 다른 결과가 발생할 수 있는 작업 포함 X

Immutable.js

- 내장함수를 사용하여 간단하게 업데이트 가능
- 값이 일반 객체가 아니기 때문에 `counter.color` 대신 `counter.get('color')`과 같이 조회해야한다는 번거로움이 있음

Immer

```
import produce from 'immer';
```

```
const nextState = produce(baseState, draftState => {
  draftState.push({ todo: 'Tweet about it' });
  draftState[1].done = true;
});
```

라이브러리가 알아서 불변성을 유지하면서 업데이트 해줌

리액트 컴포넌트 스타일링

- CSS
- Sass
- styled-components

Sass

- Syntatically Awesome Style Sheets(문법적으로 짱 멋진 스타일 시트)
- .scss/.sass 두 가지 확장자 중 scss 더 많이 사용

```
// .sass
$font-stack:    Helvetica, sans-serif
$primary-color: #333

body
  font: 100% $font-stack
  color: $primary-color
```

```
// .scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

styled-components

- 자바스크립트 파일 안에 스타일 자체를 선언하는 방식으로 스타일 관리
- Tagged Template Literal 문법 사용

```
function myFunction(...args) {
    console.log(args);
}
myFunction`1+1 = ${1+1} and 2+2 = ${2+2}!`
```

```
[
  [
    "1+1 = ",
    " and 2+2 = ",
    "!"
  ],
  2,
  4
]
```

styled-components(Cont.)

```
const Box = styled.div`
  /* props 로 넣어준 값을 직접 전달해줄 수 있습니다. */
  background: ${props => props.color || 'blue'};
  padding: 1rem;
  display: flex;
`;
```

앞으로

- Hook 공부
- 개인 프로젝트 구상 후 진행