[React] 리액트 특) 불변해야 함(하)

https://hi-claire.tistory.com/61
[Javascript] 불변성이란 무엇인가(상)
리액트를 공부해본 당신.... 리액트가 지향하는 개념 중 하나인 "불변성"에 대해 알고 계시나요? 저는 useState, spread 연산자를 사용하면 되는 건줄로만 알고 있었는데... 정확히는 "왜?🤔" 사용하는
hi-claire.tistory.com

이번 포스팅은 위 포스팅에서 이어집니다.

원시 타입과 참조 타입에 대해 알고 포스팅을 읽으면 수월하게 이해하실 수 있습니다!

안녕하세요!

이번 포스팅은 **"리액트의 불변성"**에 대해 알아보도록 하겠습니다.

0. 들어가기 전

저번 포스팅에서 원시 타입과 참조 타입으로 "불변성"이 무엇인지 알아보았습니다.

다시 복습하자면 다음과 같습니다.

> 불변성
>
> **원시타입**
>
> **불변성이 유지되는 데이터 타입참조타입**
>
> **불변성이 유지되지 않는 데이터 타입**

참조 타입은 불변성이 유지되지 않으며 코드가 서로에게 영향을 줄 수 있었기 때문에 저희가 유의해서 코드를 짜야했습니다.

즉, **"불변성을 지킨다"**의 의미는 **메모리 영역에서 데이터를 변경할 수 없게 한다.** 라는 의미입니다.

그렇다면 왜, 리액트에서는 "불변성"을 지켜야 할까요?

1. 불변객체

님들, **"불변객체"** 라고 아십니까?

> 💡
>
> **불변객체**
>
> **생성 후에 상태를 바꿀 수 없는 객체**

그렇다면 상태도 바꿀 수 없는 불변 객체를 도대체 왜 사용할까요?

불변 객체는 다음과 같은 **장점**을 가집니다.

1. 여러 곳에서 사용하더라도 **안전하게 사용 가능**
2. 불변 객체를 사용하는 곳 이외에 **영향을 미치는 것을 고려하지 않아도 됨**
3. 객체 전체가 복사되는 것이 아니기에 **메모리 절약 및 성능 향상**

2. 불변성의 이점

위와 같은 불변객체를 사용하여 불변성을 지킨다면 무엇이 좋을까요?

1. 복잡한 특징들을 단순화시킬 수 있다.

불변성은 복잡한 특징들을 구현하기 쉽게 만들 수 있습니다.

리액트 공식문서(https://ko.legacy.reactjs.org/tutorial/tutorial.html#implementing-time-travel)

의 시간 여행 구현하기 부분을 보면

"되돌아갈 수 있습니다."

버튼만 클릭하면 특정 행동을 취소하고 다시 실행시킬 수가 있습니다!

위와 같은 기능처럼 **직접적인 데이터 변이를 피할 수가 있으며**, 이전 버전의 게임 이력을 유지하고 나중에 **재사용할 수 있게** 만듭니다!

2. 변화를 감지할 수 있다.

불변객체를 사용하지 않는다면, 원본객체가 직접적으로 수정되기 때문에, 복제가 가능한 객체에서 어떤 것이 변화되었는지 감지하기 어렵습니다.

변화의 감지는 복제가 가능한 객체를 이전 사본과 비료하고 전체 객체 트리를 돌아야 합니다.

하지만!!!

불변 객체에서는 변화를 감지하기 상당히 쉽습니다.

참조하고 있는 불변 객체가 이전 객체와 다르다면 객체는 변한 것이겠죠?

3. React에서 다시 렌더링하는 시기를 결정할 수 있다.

리액트에서 불변성의 가장가장 큰 장점은 **"순수 컴포넌트"**를 만드는 데에 도움을 줄 수 있다는 점입니다.

변하지 않는 데이터는 변경이 이루어졌을 때 쉽게 판단할 수 있으며,

이를 바탕으로 컴포넌트가 다시 렌더링할지를 결정할 수 있습니다!

(부모 컴포넌트가 리렌더링 하면 자식 컴포넌트도 함께 리렌더링이 되는 것과 같이)

3. 리액트에서의 불변성

그렇다면 왜 리액트에서 불변성을 지켜야 할까요?

어느정도 위의 장점들을 가지고 대~충은 알겠지만... 가려운 곳을 충분히 긁어주지 않는 것 같습니다...

리액트에서 불변성을 지켜주는 이유는 **리액트가 상태 업데이트를 하는 원리 때문**입니다!

리액트는 상태값을 업데이트 할 때 ***"얕은 비교"를 수행*** 합니다.

즉, 배열이나 객체의 속성을 하나하나 다 비교하는 것이 아니라, **이전 참조 값과 현재 참조 값만을 비교하여 상태 변화를 감지**합니다.

(Javascript의 참조타입과 같죠?)

이러한 이유로 배열이나 객체를 업데이트 할 때

```
setState([...state, newState])// Array
setState({...state, [key]: value})// Object
```

위와 같은 방식으로 새로운 참조값을 가진 배열이나 객체를 생성하는 것입니다.

즉, 다시 말하자면, **불변성을 지킴으로써 리액트는 상태변화를 감지**할 수 있습니다.

또한, **불변성을 지킴으로써 발생되는 사이드 이팩트를 방지**할 수 있습니다.

저번 시간에 봤듯, 원시타입은 애시당초 불변성이란 특징을 가지고 있지만, 참조타입인 객체나 배열의 경우 값을 변경할 때 원본 데이터가 변경될 여지가 있습니다.

이렇게 원본 데이터가 변경될 경우, 이 원본 데이터를 참조하고 있는 다른 객체에서 예상치 못한 오류가 발생할 수 있겠죠..?

이러한 일말의 가능성들을 불변성을 지킴으로써 방지할 수 있습니다.

4. 그렇다면 React에서 어떻게 불변하게 만드는가?

크게 다음과 같은 두 가지 방법을 사용합니다.

1. assign() 메서드 이용

```
const origin = {a: 1, b: 2};
const copy = Object.assign({}, origin);

cosole.log(origin === copy)// false
```

Object.assign() 은 첫 번째 객체 인자 이후의 모든 객체 인자를 합쳐주는 메서드입니다.

첫 번째 인자인 빈 객체({})에 origin 의 프로퍼티들을 담아 새로운 객체를 반환하기 때문에 origin 객체와 copy 객체가 다른 것을 알 수 있습니다.

2. spread 연산자 이용

```
const origin = {a: 1, b: 2};
const copy = {...origin};

cosole.log(origin === copy)// false
```

마찬가지로 새로운 객체에 origin의 프로퍼티를 담았기 때문에 기존 origin과 copy가 다른 참조를 가진 것을 확인할 수 있습니다.

5. 정리

불변성이란 메모리 영역에서 값을 변경할 수 없게 하는 것!

리액트에서 **상태 변화 감지 기준은 "콜 스택의 주소값"**으로 이루어집니다.

리액트는 콜 스택의 주소값만을 비교해 상태 변화를 감지하는데, 이를 **얕은 비교**라고 하죠?

이것이 리액트가 빠르게 상태 변화를 감지할 수 있는 장점이자 우리가 불변성을 유지해야하는 이유였음을 알 수 있었습니다!

또한, 객체와 배열은 실제로 **콜스택의 주소값의 변경이 이루어지지 않기 때문에 상태 변화를 감지할 수 없어 리렌더링이 되지 않습니다**.

불변성을 가진 원시타입과 달리 참조타입의 경우에는 의도적으로 불변성을 지켜주어야 합니다.

이 때 새로운 주소값을 가진 객체를 생성하여 상태를 업데이트 해주기 위하여, 우리는 spread 연산자와 assign과 같은 메소드를 잘 활용해야 합니다!