3.3.1 불변성

'mutate'라는 영어 단어는 변한다는 뜻이다. 따라서 'immutable'이란 단어는 변할 수 없다는 뜻이다. 함수형 프로그래밍에서는 데이터가 변할 수 없다. 불변성 데이터는 결코 바뀌지 않는다.

여러분이 자신의 출생증명서를 공개하고 싶은데 개인 정보는 알아볼 수 없게 만들거나 없애고 싶다면 실질적으로 선택할 수 있는 방법은 2가지뿐이다. 첫 번째 방법은 두꺼운 유성펜을 가지고 원본 출생증명서의 개인 정보를 가리는 것이고, 두 번째 방법은 복사기를 찾는 것이다. 복사기를 찾아서, 출생증명서의 복사본을 만든 다음에 그 복사본을 유성펜으로 지우는 것이다. 물론 두 번째 방식이 더 바람직하다. 두 번째 방법을 택하면 개인 정보를 보호하면서 출생증명을 공개하는 동시에 원본을 안전하게 그대로 유지할 수 있다.

이것이 바로 애플리케이션에서 불변성 데이터가 작동하는 방식이다. 원본 데이터 구조를 변경하는 대신 그 데이터 구조의 복사본을 만들되 그중 일부를 변경한다. 그리고 원본 대신 변경한 복사본을 사용해 필요한 작업을 진행한다.

불변성이 어떻게 작동하는지 이해하기 위해 데이터를 변경한다는 것이 어떤 의미인지를 살펴보자. 잔디 색을 표현하는 객체를 생각해보자.

```
let color_lawn = {
  title: "잔디",
```

```
  color: "#00FF00",
  rating: 0
};
```

색에 평점을 메기는 함수를 만들 수 있을 것이다. 그 함수는 넘겨받은 color 객체의 rating을 변경한다.

```
function rateColor(color, rating) {
  color.rating = rating;
  return color;
}

console.log(rateColor(color_lawn, 5).rating); // 5
console.log(color_lawn.rating);               // 5
```

자바스크립트에서 함수의 인자는 실제 데이터에 대한 참조다. rateColor 함수 안에서 color의 rating을 변경하면 원본 color_lawn객체의 rating도 바뀐다(여러분이 어떤 업체에게 출생증명서에서 개인 정보를 없애는 일을 맡겼는데, 그 업체가 여러분이 제공한 원본 출생증명서의 중요한 개인정보를 모두 다 검은 유성펜으로 가린 후 돌려보냈다고 생각해보라. 여러분은 아마도 업체가 원본은 그대로 남기고 출생 증명서 복사본의 개인정보를 가리는 작업을 진행하는 정도의 상식은 가지고 있으리라 기대할 것이다). rateColor를 다음과 같이 고쳐쓰면 (color 파라미터로 전달 받은) 원본에는 아무런 해가 없이 색깔에 평점을 부여할 수 있다.

```
var rateColor = function(color, rating) {
  return Object.assign({}, color, {rating:rating});
};

console.log(rateColor(color_lawn, 5).rating); // 5
console.log(color_lawn.rating);               // 0
```

여기서는 Object.assign을 사용해 색의 평점을 바꿨다. Object.assign은 복사기와 같다. Object.assign은 빈 객체를 받고, color 객체를 그 빈 객체에 복사한 다음에, 복사본에 있는 rating 프로퍼티의 값을 rating 파라미터의 값으로 변경한다. 이제 우리는 원본은 그대로 남겨둔 채, rating만 변경된 복사본을 손에 쥐게 된다.

화살표 함수와 객체 스프레드 연산자를 활용해 같은 함수를 작성할 수도 있다. 이렇게 만든 rateColor 함수는 스프레드 연산자를 사용해 원본 color를 새로운 객체 안에 복사한 다음에 그 rating 프로퍼티를 덮어쓴다.

```
const rateColor = (color, rating) => ({
    ...color,
    rating
  });
```

이렇게 새로운 버전의 자바스크립트 언어로 작성한 rateColor 함수는 앞에서 본 예제의 rateColor 함수와 똑같은 일을 한다. 이 함수는 color를 변경불가능한 객체로 취급하며, 더 짧고, 약간 더 명확해 보인다. 여기서 반환할 객체를 괄호(())로 감쌌다는 것에 주의하자. 화살표 함수의 본문에서 바로 중괄호({})를 사용해 객체를 반환할 수 없기 때문에, 꼭 괄호가 필요하다.

색의 이름으로 이루어진 배열을 생각해보자.

```
let list = [
  { title: "과격한 빨강"},
  { title: "잔디"},
  { title: "파티 핑크"}
];
```

이 배열에 Array.push를 사용해 색을 추가하는 함수를 작성할 수 있다.

```
const addColor = function(title, colors) {
  colors.push({ title: title });
  return colors;
};

console.log(addColor("화려한 녹색", list).length); // 4
console.log(list.length);                          // 4
```

하지만 Array.push는 불변성 함수가 아니다. 이 addColor 함수는 원본 배열에 새로운 원소를 추가한다. 원래의 list 배열을 변화시키지 않고 유지하기 위해서는 Array.concat을 사용해야 한다.

```
const addColor = (title, array) => array.concat({title});

console.log(addColor("화려한 녹색", list).length); // 4
console.log(list.length);                          // 3
```

Array.concat은 두 배열을 붙여준다. 여기서는 Array.concat이 새로운 객체를 받는다. 그 객체에는 새로운 색의 이름이 title이라는 이름의 프로퍼티로 들어 있이다. Array.concat은 그 객체를 원래의 배열을 복사한 새로운 배열 뒤에 추가한다.

앞에서 객체를 복사할 때 사용했던 방법과 똑같은 방법으로 배열 스프레드 연산자를 사용해 배열을 복사할 수 있다. 다음은 앞의 addColor 함수를 새로운 자바스크립트 버전으로 작성한 코드를 보여준다.

```
const addColor = (title, list) => [...list, {title}];
```

이 함수는 원본 list의 원소들을 새로운 배열에 복사하고, title 파라미터로 받은 값을 title 프로퍼티로 하는 객체를 새 배열 뒤에 추가한다. 이 함수는 인자로 받은 list를 변경하지 않기 때문에 list의 원본인 colorArray의 불변성을 지켜준다.[9]