공부일지

Recoil 공부

https://tech.osci.kr/2022/06/16/recoil-state-management-of-react/ 참고

Frontend 개발 중 state 관리를 어떻게 할지 고민하다가 recoil을 사용하기로 했다.

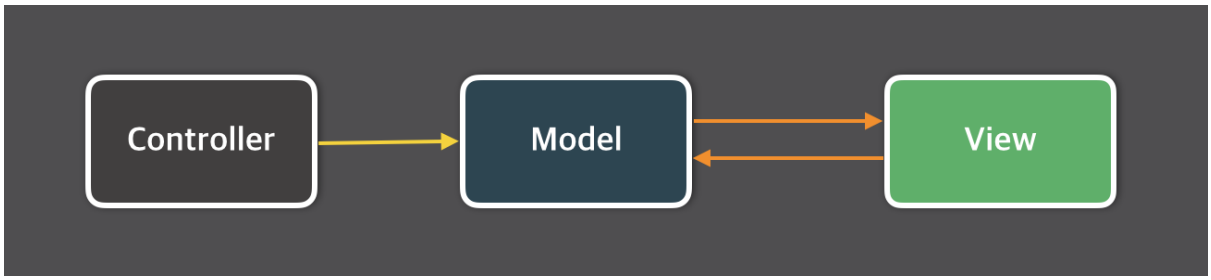
React에서는 공식적으로 ContextAPI사용을 권장하지만 코드량을 줄이고 간결하게 작성하기 위해 Recoil을 사용하기로 했다.

React에서 추구하는 패턴

React에서 데이터는 단방향으로 흐르며, 위에서 아래로 즉 부모에서 자식 컴포넌트로 흐른다. 이러한 방식은 Flux 패턴에 의해서 적용된 방식이다.

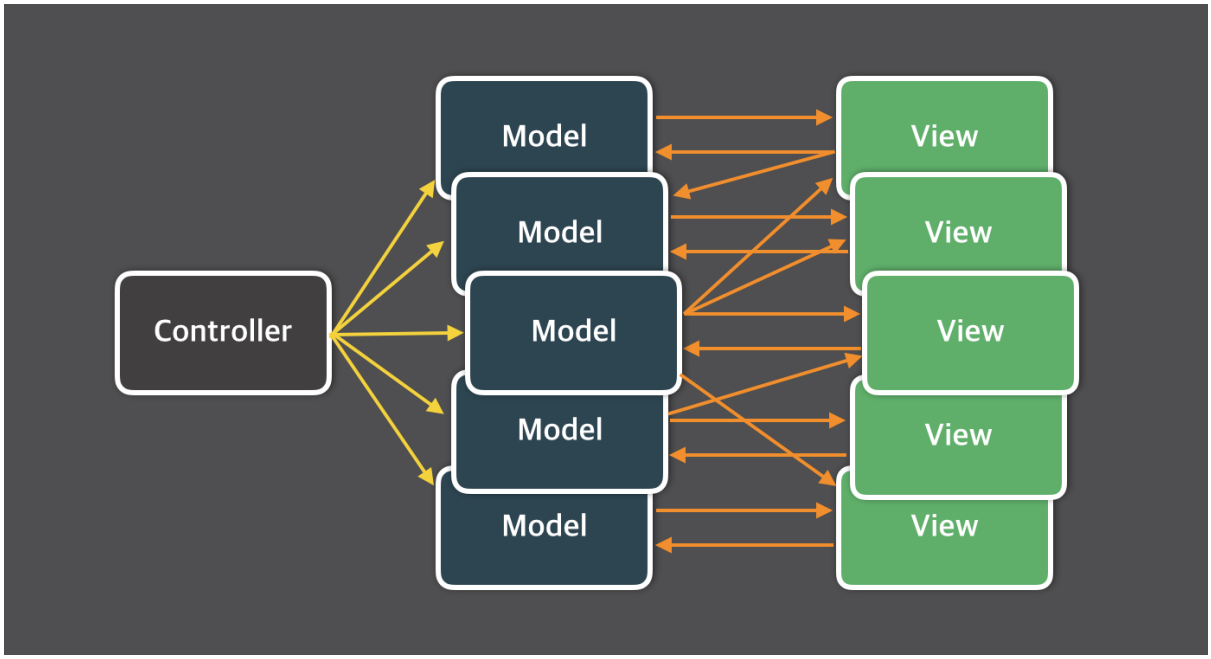
MVC 패턴

Flux패턴은 MVC 패턴이 가진 문제점에서 시작했다.

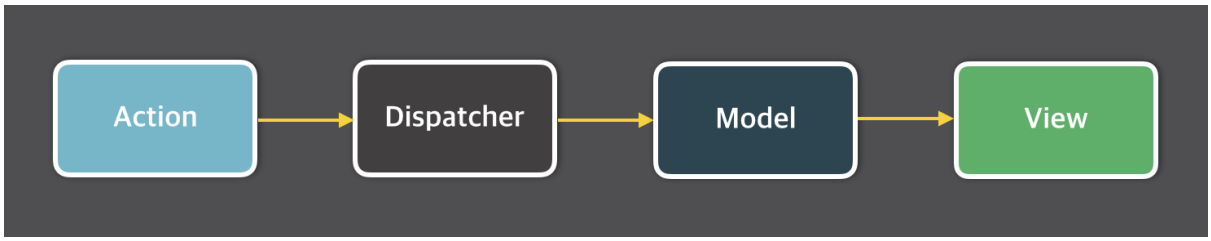


위 사진은 기존의 MVC 패턴이다. Model에 데이터를 정의해 놓고 , Controller를 통해서 Model의 데이터를 CRUD 작업하며, 변경된 데이터를 View에 출력하는 식의 패턴이다.

MVC 패턴은 Web개발을 하는데 가장 많이 사용 되는 패턴이다. MVC 패턴의 문제점은 데이터의 흐름이 정말 많다는 것이다. 어떤 데이터가 변경되었을 시 해당 데이터를 사용하는 모든 곳에서 코드를 작성하고 변경해줘야 한다. 만약 이런 과정을 생략 했을 시 예측하지 못한 부분에서 오류가 발생하거나 다르게 동작하는 등 Side Effect가 발생할 수 있다.



Flux 패턴



페이스북에서 이 같은 문제를 해결하기 위해서 Flux 패턴을 출시 했다.

여기서 Action은 데이터의 상태를 변경하는 명령어 이다. Dispatcher은 Action을 감지하여 Store에 Action을 전달해주는 역할이다. Model은 Store라고도 볼 수 있는데 state가 저장되어 있는 공간이다. Dispatchrer를 통해서 가져온 Action을 확인해 내부에 저장된 데이터를 변경한다. 마지막으로 View는 React를 통해서 만드는 코드들이다. Model에 저장된 데이터를 가져와서 View에 뿌려주고, View는 해당 데이터들을 가지고 와서 화면에 렌더링 한다.

▼ 301과 302의 차이?

301 Moved Permanently

이 응답 코드는 요청한 리소스의 URI가 변경되었음을 의미한다. 새로운 URI가 응답에서 주어질 수 도 있다.

302 Found

이 응답 코드는 요청한 리소스의 URI가 일시적으로 변경되었음을 의미한다. 새롭게 변경된 URI는 나중에 만들어질 수 있습니다. 그러므로, 클라이언트는 향후의 요청도 반드시 동일한 URI로 해야한다.

왜 구분하여 사용해야하는가?

그 이유는 구글과 같은 검색엔진에서 페이지 검색 노출도 등과 관련이 있기 때문

301 리다이렉트는 웹사이트의 도메인을 변경했거나, URL구조가 변경되었을 때 사용할 수 있다.

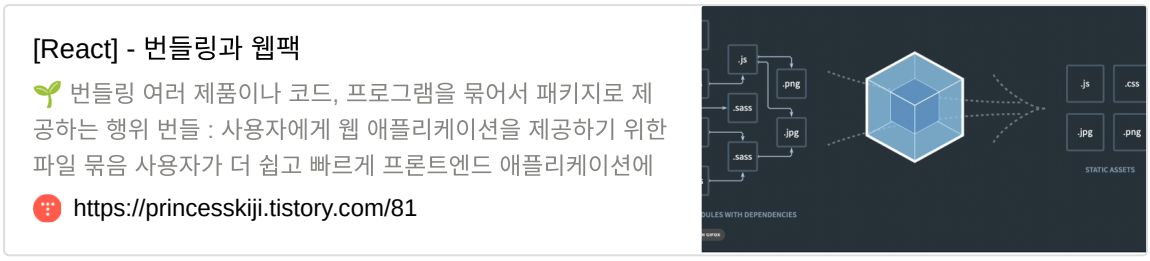
검색엔진은 301요청을 받으면 새로운 URL로 영구적으로 이동했다고 판단한다.

따라서 검색엔진은 과거 URL의 사이트랭크와 평가점수를 새로운 URL로 이동시킨다.

302 리다이렉트는 영구적인 URL이동이 아닌 임시적인 이동으로 검색엔진이 판단한다. 따라서 검색엔진은 사이트랭크나 평가점수를 새로운 URL로 이동시키지 않고, 기존 URL에 유지한다.
기존 URL 사이트랭크나 점수는 유지하면서 새로운 컨텐츠 URL을 조회할 때 유용하다.

▼ 번들링이란?

참고 레퍼런스



- 여러 제품이나 코드, 프로그램을 묶어서 패키지로 제공하는 행위
- 사용자가 더 쉽고 빠르게 프론트엔드 애플리케이션에 접근할 수 있도록 용량을 줄이거나 파일을 최소화하여 유저에게 전달하는 과정
    - 사용자가 브라우저를 열고 주소를 입력하면, 해당 주소에서 개발자가 번들링한 여러 파일을 받음.
    - 이 파일을 브라우저가 실행하여 웹 애플리케이션을 제공

번들링의 필요성? (만약 번들링을 하지 않고 HTML, CSS, JavaScript 파일을 그대로 전송한다면?)

- 두 개의 js 파일에서 같은 변수를 사용하고 있는 경우 → 변수 간 충돌이 일어남
- 딱 한 번 불러오는 프레임워크 코드가 8MB → 인터넷 속도가 느린 국가의 모바일 환경에서는 사용이 불편
- 번들 파일 사이즈를 줄이기 위해 파일의 공백을 모두 지운다면? → 가독성이 너무 떨어져 코딩하기 어려움
- 배포 코드가 너무 읽기 쉬워 개발을 할 줄 아는 사용자가 프론트엔드 애플리케이션을 임의로 자작하여 피해 발생
- 번들링 작업에서는 필연적으로 용량을 줄이고 파일을 통합하는 툴링 작업이 필요하다.
  → 소프트웨어를 잘 만들어도 사용자에게 배포하기 위해 번들링이 꼭 필요하다.

웹팩

- 현재 프론트엔드 애플리케이션 배포를 위해 가장 많이 사용하는 번들러
- 여러 개의 파일을 하나의 파일로 합쳐주는 모듈 번들러
  - 모듈 번들러란? HTML, CSS, JavaScript 등의 자원을 전부 각각의 모듈로 보고 이를 조합해 하나의 묶음으로 번들링하는 도구

모듈 번들러의 등장

- 모던 웹으로 발전하면서 JavaScript 코드의 양이 증가 + 대규모의 의존성 트리를 가진 대형 웹 애플리케이션의 등장으로 세분화된 모듈 파일이 폭발적으로 증가
  → 모듈 단위의 파일들을 호출해 브라우저에 띄우려면?
  - 자바스크립트 언어의 특성에 따라 발생하기 쉬은 각 변수들의 스코프 문제 해결
  - 각 자원을 호출할 때 생겨나는 네트워크 쪽의 코스트도 신경써야함

  → 이러한 복장성에 대응하기 위해 하나의 시작점(ex. React App 의 index.js)으로부터 의존성을 가지는 모듈을 모두 추적하여 하나의 결과물을 만들어내는 모듈 번들러가 등장하게되었다.

빌드와 번들링

- 빌드
  - 개발이 완료된 앱을 배포하기 위해 하나의 폴더로 구성하여 준비하는 작업
  - React앱에서 npm run build 실행 시 React build작업이 진행되고, index.html 파일에 압축되어 배포에 최적화된 상태를 제공

- 번들링
  - 파일을 묶는 작업, 모듈간의 의존성 관례를 파악해 그룹화하는 작업
    - 파일은 의존적 관계에 있는 파일들(import, export) 그 자체 혹은 내부적으로 포함되어 있는 모듈을 의미

Webpack의 필요성

웹팩이 필요한 가장 큰 이유는 웹 애플리케이션의 빠른 로딩 속도와 높은 성능 때문

- 웹팩으로 같은 타입의 파일들을 묶어 요청 및 응답을 받을 수 있다.
  → 네트워크 코스트가 획기적으로 줄어듬
- webpack loader를 사용하면 일부 브라우저에서 지원하지 않는 JavaScript ES6의 문법들을 ES5로 변환해주는 babel-loader를 사용할 수 있게된다.
  - Vue의 경우 vue-loader
  - scss파일의 경우 css파일로 변환해주는 scss-loader
    → 개발자가 원하는 최선의 개발 방식을 선택해 개발할 수 있다.
- Webpack 4버전 이상부터는 Development, Production 두 가지의 모드를 지원한다.
  - Production 모드로 번들링을 진행하는 경우, 코드 난독화, 압축, 최적화 작업을 지원
    → 상용화된 프로그램을 사용자가 느끼기에 더욱 쾌적한 환경 및 보안까지 신경쓰면서 노출시킬 수 있다.

▼ URL 포맷 구조?

URL 포맷은 크게 세 부분으로 이루어져 있다.

1. scheme(스킴) : 리소스에 접근할 때 사용하는 프로토콜을 명시한다.
   ex) http:, ftp:, mailto: 등
   - scheme에는 +, -, . 그리고 문자만 사용할 수 있다.
   - scheme은 대소문자를 구별하지 않지만 보통 소문자로 쓴다.
2. host(호스트) : 서버의 인터넷 주소
3. resource(리소스)

▼ http의 기본 포트가 80, https의 기본 포트가 443인 이유는 무엇일까?

http 기본포트 : 80 / https 기본포트 : 443

둘의 기본포트는 80, 443으로 주어지며 https://www.tistory.com/ 처럼 생략하여도 https://www.tistory.com:443/ 뒤의 URL 접속과 동일하다.

보안상으로 서비스마다 다른 포트를 임의로 지정하여 접근을 제한할 수 있다.

▼ 메타태그란?

메타태그란 해당 웹 문서에 대한 정보를 알리기 위한 태그이다.

메타태그를 이용하여 검색엔진에 잘 노출될 수 있도록 할 수 있다.

프론트엔드 아키텍쳐에 대해서 생각해보자