[개발환경] ReactQuery와 Zustand

구현 어려움 정도	개념
구현 완료	☑
기간	@November 29, 2024
정리 완료	☑

- **React Query**: 데이터를 가져오는 로직과 상태를 서버와 동기화하여 최신 상태 유지.
- **Zustand**: 전역 상태 관리를 단순하고 가볍게 관리.

두 라이브러리가 서로 보완적인 역할

---

1. React Query의 역할

React Query는 **서버 상태**를 효율적으로 관리하는 데 초점이 맞춰져 있습니다. 서버 상태는 API를 통해 가져오는 데이터로, 아래와 같은 특징을 가집니다.

- **서버와 동기화**: 서버에서 데이터를 가져오고, 캐싱 및 갱신.
- **비동기 작업 처리**: 데이터 fetching, 로딩 상태 관리, 오류 처리.
- **자동 갱신**: 데이터 갱신 주기 설정(`refetchInterval`)과 오래된 데이터를 자동으로 다시 가져오기(`staleTime`).
- **전역 캐싱**: 동일한 API 호출에 대해 데이터를 공유.
- **초점 이동 시 재요청**: 브라우저 탭 이동 후 다시 돌아올 때 데이터를 자동으로 갱신.

React Query는 서버 데이터를 효율적으로 가져오고 갱신하는 데 매우 적합하지만, **클라이언트에서만 필요한 로컬 상태**를 관리하는 데는 초점이 맞춰져 있지 않습니다.

2. Zustand의 역할

Zustand는 **클라이언트 상태**를 관리하는 데 최적화된 라이브러리로, 아래와 같은 특징이 있습니다.

- **가벼운 상태 관리**: 사용하기 간단하며 React의 Context보다 더 성능이 좋음.
- **동기적 상태 업데이트**: 로컬에서 즉각적인 상태 업데이트 가능.
- **전역 상태 관리**: React의 모든 컴포넌트에서 상태를 쉽게 공유.
- **커스텀 상태 관리**: 다양한 형태의 클라이언트 상태를 쉽게 저장 가능.

로컬 상태란 사용자 입력 값, UI 상태(모달 열림/닫힘), 전역적으로 유지해야 하는 세션 정보 등입니다. 서버와 동기화되지 않아도 되는 상태를 효율적으로 관리합니다.

3. 왜 함께 사용하는가?

`Zustand`와 `React Query`는 서로 다른 목적에 최적화되어 있기 때문에 **서버 상태와 클라이언트 상태를 분리**하여 관리할 수 있습니다.

예: 빗썸 데이터 관리 프로젝트

- **React Query**:
  - 빗썸 API에서 데이터를 주기적으로 가져오고 캐싱.
  - 데이터 로딩 및 에러 상태 관리.
  - 서버와 동기화된 최신 데이터를 제공.
- **Zustand**:
  - 가져온 데이터를 로컬 상태로 저장 (`setMarketData`).
  - 특정 컴포넌트에서 데이터 변형(필터링, 정렬 등).
  - 사용자 인터페이스 상태 관리 (예: 모달, 테마 변경, 현재 활성화된 탭 등).

4. React Query만으로 충분하지 않은 이유

React Query는 서버 상태 관리에는 강력하지만, 클라이언트 상태를 관리하는 데는 제한적입니다:

- 데이터를 캐싱하고 서버와 동기화하는 데 최적화되어 있지만, UI 중심의 상태(예: 모달 상태, 폼 입력 값) 관리에는 비효율적입니다.

- 로컬에서 임시 데이터를 유지하거나, API와 관계없는 전역 상태를 다룰 때는 부적합합니다.

5. Zustand만으로 충분하지 않은 이유

Zustand는 클라이언트 상태 관리에는 최적화되어 있지만, 서버 상태 관리에는 한계가 있습니다:

- 데이터 페칭, 캐싱, 리트라이, 자동 갱신 등의 기능이 없습니다.
- 비동기 데이터 처리 로직을 수동으로 구현해야 하므로 코드가 복잡해질 수 있습니다.

6. 장점 요약

함께 사용할 때의 장점:

1. **역할 분리**:
   - React Query: 서버 데이터 관리 (API 호출, 캐싱, 리트라이).
   - Zustand: UI 상태 및 클라이언트 로컬 상태 관리.
2. **단순화**:
   - 서버 상태와 클라이언트 상태를 명확히 분리해 코드의 가독성과 유지보수성을 높임.
3. **성능 최적화**:
   - React Query는 효율적인 캐싱과 서버 데이터 관리를 제공.
   - Zustand는 빠르고 경량화된 상태 관리를 지원.
4. **유연성**:
   - 필요에 따라 두 라이브러리를 결합해 확장 가능한 상태 관리 솔루션 제공.

7. 언제 이 조합을 사용하는 게 좋은가?

- 프로젝트에서 **서버와 통신하는 데이터**와 **UI 상태**를 모두 관리해야 하는 경우.
- 대규모 애플리케이션에서 **상태 관리와 서버 상태를 분리**하고 싶을 때.
- 상태 관리 라이브러리로 Redux가 과하다고 느껴질 때.

이 조합은 특히 Next.js처럼 서버와 클라이언트 간의 상호작용이 많은 프레임워크에서 잘 어울린다고 합니다.