React Query

타입스크립트 해적단
정세희

Returns

```
data,
dataUpdatedAt,
error,
errorUpdatedAt,
failureCount,
failureReason,
fetchStatus,
isError,
isFetched,
isFetchedAfterMount,
isFetching,
isInitialLoading,
isLoading,
isLoadingError,
isPaused,
isPending,
isPlaceholderData,
isRefetchError,
isRefetching,
isStale,
isSuccess,
promise,
refetch,
status,
```

Options

```
queryKey,
queryFn,
gcTime,
enabled,
networkMode,
initialData,
initialDataUpdatedAt,
meta,
notifyOnChangeProps,
placeholderData,
queryKeyHashFn,
refetchInterval,
refetchIntervalInBackground,
refetchOnMount,
refetchOnReconnect,
refetchOnWindowFocus,
retry,
retryOnMount,
retryDelay,
select,
staleTime,
structuralSharing,
subscribed,
throwOnError,
```

useQuery

- 3개의 인자를 받는다
  - queryKey (필수)
  - queryFn (필수)
  - options (옵셔널)

```
const { isPending, error, data } = useQuery({
  queryKey: ['repoData'],
  queryFn: () =>
    fetch('https://api.github.com/repos/TanStack/query').then((res) =>
      res.json(),
    ),
})
```

staleTime

- **staleTime: number | 'static' ((query: Query) ⇒ number | 'static')**
  - 0 (기본값):
    데이터를 가져오자마자 즉시 stale 상태가 된다.
    즉, 다시 마운트될 때마다 refetch가 발생할 수 있다.
  - Infinity (무한):
    절대 stale 상태가 되지 않는다.
    즉, 수동으로 invalidateQueries를 호출하지 않는 한, 데이터를 새로 가져오지 않는다.
  - 함수(function)으로 설정:
    쿼리 객체(query)를 받아서 동적으로 staleTime 값을 계산할 수 있다.
    예: 사용자 역할에 따라 staleTime을 다르게 줄 수 있음.
  - 'static'으로 설정:
    데이터를 항상 fresh 상태로 간주한다.
    Infinity와 유사하지만, 일부 특수한 내부 정책에서 'static'이라는 구분이 쓰일 수 있다.

select

- **select: (data: TData) ⇒ unknown**
  - 선택값
  - select 옵션은 쿼리 함수(queryFn)가 반환한 전체 데이터 중에서 일부를 선택하거나 변형(transform) 하기 위해 사용한다.
  - 이 옵션은 useQuery 훅에서 반환되는 데이터 값만 변경하며, 쿼리 캐시에 저장되는 원본 데이터에는 영향을 주지 않는다.

```
const { data } = useQuery(['users'], fetchUsers, {
  select: (data) => data.map((user) => user.name),
});
```

Retry

- **retry: boolean | number | (failureCount: number, error: TError) ⇒ boolean**
  - false로 설정하면
    → 실패한 쿼리는 기본적으로 재시도하지 않는다.
  - true로 설정하면
    → 실패한 쿼리는 무한히 재시도한다.
  - 숫자(예: 3)로 설정하면
    → 실패한 쿼리는 지정된 횟수만큼 재시도한 후 멈춘다.

```
useQuery(['todos'], fetchTodos, {
  retry: 2, // 최대 2회 재시도 (최대 3번까지 요청)
});
```

refetchOnWindowFocus

- **refetchOnWindowFocus: boolean | "always" | ((query: Query) ⇒ boolean | "always")**
  - true로 설정하면
    → 데이터가 stale 상태인 경우 브라우저 창에 포커스가 돌아올 때 자동으로 쿼리를 refetch
    → 기본 동작
  - false로 설정하면
    → 창에 포커스가 돌아와도 쿼리를 재요청하지 않는다.
  - "always"로 설정하면
    → 데이터의 상태(stale 여부)와 관계없이 창에 포커스가 돌아올 때마다 쿼리를 항상 refetch
    → 단 staleTime: 'static'이 설정된 경우는 예외로, refetch가 발생하지 않는다.
  - 함수(function)로 설정하면
    → 쿼리 객체를 인자로 받아 동적으로 refetch 여부를 계산하는 함수로 동작한다.

```
useQuery(['profile'], fetchProfile, {
  refetchOnWindowFocus: "always",
});
```

refetchOnWindowFocus

- **refetchOnMount: boolean | "always" | ((query: Query) ⇒ boolean | "always")**
  - true로 설정하면
    → 데이터가 stale 상태인 경우에 한해, 컴포넌트가 마운트될 때 쿼리를 refetch
    → 기본 동작이다.
  - false로 설정하면
    → 컴포넌트가 마운트되더라도 쿼리를 다시 요청하지 않는다.
  - "always"로 설정하면
    → 데이터의 상태와 관계없이, 컴포넌트가 마운트될 때마다 항상 쿼리를 refetch
    → 단, staleTime: 'static'으로 설정된 경우는 예외이며, refetch되지 않는다.
  - 함수(function)으로 설정하면
    → 쿼리 객체를 인자로 받아 동적으로 refetch 여부를 계산하는 함수로 동작한다.

```
useQuery(['user'], fetchUser, {
  refetchOnMount: "always",
});
```

gcTime

- **gcTime: number | Infinity**
  - 쿼리가 더 이상 사용되지 않거나 활성 상태가 아니게 되면,
    해당 쿼리의 캐시는 cacheTime이 지난 후 자동으로 제거된다.
  - 여러 쿼리에서 서로 다른 cacheTime이 설정된 경우,
    가장 긴 시간이 우선 적용된다.
  - Infinity로 설정하면
    → 캐시가 절대 제거되지 않으며, 가비지 컬렉션이 비활성화된다.

```
useQuery(['posts'], fetchPosts, {
  cacheTime: 1000 * 60 * 10, // 10분 동안 메모리에 유지
});
```

감사합니다