5장. 레디스를 캐시로 사용하기

레디스를 어떤 상황에서 잘 캐시로 사용할 수 있는지 주의할 점과 함께 알아봅시다.

캐시란?

원본 데이터보다 빠르고 효율적으로 액세스할 수 있는 임시 데이터 저장소

언제 도입하면 좋을까?

- 원본 데이터 저장소에서 검색하는 시간이 오래 걸리는 경우
- 매번 계산을 통해 데이터를 가져와야 하는 경우
- 캐시에서 데이터를 가져오는 것이 원본 데이터 저장소를 거치는 것보다 빠른 경우
- 잘 변하지 않는 데이터인 경우
- 자주 검색되는 데이터인 경우

왜 좋은가?

- 애플리케이션 `응답 속도를 줄일` 수 있다.
- 원본 데이터 저장소에서 `데이터를 읽는 커넥션을 줄일` 수 있다.
- 같은 계산을 하지 않게 되어 `리소스를 최적화`할 수 있다.

캐싱 전략

look aside

데이터가 캐시에 있는지 먼저 확인한 후 있으면 캐시에서, 없으면 원본 저장소에서 읽는다.

- **캐시 히트**: 원하는 데이터가 캐시에 있는 경우
- **캐시 미스**: 원하는 데이터가 캐시에 없는 경우

단점

- 처음 캐시를 도입한 경우 엄청난 캐시 미스로 지연이 발생하여 성능에 영향을 미칠 수 있다.
    - **캐시 워밍**: `원본 저장소 → 캐시`로 데이터를 미리 밀어 넣어주는 작업

쓰기 전략

데이터 변경 시 원본 데이터베이스에만 업데이트 한다면 데이터 간 불일치가 발생한다. (**캐시 불일치**)

write through

데이터를 업데이트할 때마다 캐시에도 함께 업데이트

단점

- 데이터를 쓸 때마다 시간이 오래 걸림

cache invalidation

데이터를 업데이트할 때마다 캐시 데이터를 삭제

write behind (write back)

캐시에 먼저 업데이트 한 뒤 비동기적으로 원본 저장소에도 업데이트

- 쓰기가 빈번하게 발생하는 경우
- 원본 저장소의 데이터가 실시간으로 정확한 데이터가 아니어도 되는 경우
- ⚠️ 캐시에 문제가 생겨 데이터가 날아가는 경우 위험

why Redis?

1. 키-값 형태의 단순한 방식으로 다양한 자료 구조를 제공하여 애플리케이션에서 사용하던 데이터를 변환하는 과정 없이 바로 저장할 수 있다.
2. 인메모리 데이터 저장소로 데이터를 검색하고 반환하는 것이 매우 빠르다.
3. 센티널, 클러스터 기능으로 마스터 노드의 장애를 자동 감지해 페일오버를 발생시킴으로써 가용성을 높인다.

만료 시간 (TTL)

캐시는 가득차지 않게 일정 양의 데이터를 유지하는 게 좋다.

```
# 데이터 추가
> set a 100
"OK"
# 만료 시간 지정 (ms)
> EXPIRE a 60
(integer) 1
# 만료 시간 확인
> TTL a
(integer) 58
```

TTL 명령어를 사용하는 경우…

1. 만료 시간이 남아 있다면 `남은 시간`을 반환
2. 키가 존재하지 않다면 `-2`를 반환
3. 만료 시간이 지정되지 않았다면 `-1`을 반환

주의점

1. `INCR`, `RENAME` 등의 커맨드로 키 또는 데이터가 변경되는 경우 만료 시간은 변경되지 않는다.
2. `SET` 등으로 기존 키에 새로운 값을 덮어쓰는 경우 만료시간은 사라진다.

```
# 만료 시간 변경되지 않음
> INCR a
(integer) 101
> TTL a
(integer) 51
> RENAME a apple
"OK"
> TTL apple
(integer) 41

# 만료 시간 사라짐
> set b 100
"OK"
> EXPIRE b 60
(integer) 1
```

```
> TTL b
(integer) 57
> set b banana
"OK
> TTL b
(integer) -1
```

3. 키가 만료되어도 메모리에서 바로 삭제되지 않는다.
    - **passive 방식**: 사용자가 키에 접근하고자 할 때 키가 만료됐다면 수동적으로 삭제한다.
    - **active 방식**: TTL 값이 있는 키 20개를 랜덤하게 뽑아서 만료된 키를 모두 삭제한다.
4. **캐시 스탬피드**: 특정 키가 만료되는 시점에 여러 애플리케이션이 해당 키를 바라보면 한꺼번에 데이터에 가서 데이터를 읽어오는 과정을 거침으로써 중복 읽기와, 중복 쓰기가 발생하는 현상
    - 만료 시간을 충분히 길게 설정한다.
    - 키가 실제로 만료되기 전에 값을 미리 갱신한다. (PER 알고리즘)

메모리 관리

만료 시간을 설정해도 너무 많은 키가 저장되면 메모리가 가득 찰 수도 있다 !

- **maxmemory**: 데이터의 최대 저장 용량 설정
- **maxmemory-policy**: 최대 저장 용량 초과 시 처리 방식 설정

maxmemory-policy

- **noeviction**: 데이터를 저장할 수 없다는 에러를 반환 (권장하지 않음)
- **LRU eviction**: 가장 최근에 사용되지 않은 데이터부터 삭제
    - volatile-lru: 만료 시간이 설정된 키에 한해서 LRU 방식으로 삭제 (모두 만료 시간이 없다면 noeviction과 동일)
    - 👍 allkeys-lru: 모든 키에 대해 LRU 방식으로 삭제
- **LFU eviction**: 가장 자주 사용되지 않은 데이터부터 삭제
    - volatile-lfu: 만료 시간이 설정된 키에 한해서 LFU 방식으로 삭제 (모두 만료 시간이 없다면 noeviction과 동일)

    - 👍 `allkeys-lfu`: 모든 키에 대해 LFU 방식으로 삭제
- **`RANDOM eviction`**: 키를 임의로 골라내서 삭제, 레디스의 부하를 줄이지만 오히려 불필요하다.
    - `volatile-random`: 만료 시간이 설정된 키에 한해서 랜덤하게 삭제 (모두 만료 시간이 없다면 noeviction과 동일)
    - 👍 `allkeys-random`: 모든 키에 대해 랜덤하게 삭제

⚠️ **주의 사항**

- 레디스에서의 LRU, LFU 알고리즘은 모두 근사 알고리즘으로 특정 키를 근사치로 찾아내서 효율적으로 데이터를 삭제하는 방식으로 작동한다.

🎁 레디스, 세션 스토어로 사용하기

캐시 사용법과 다른 점

- 캐시: 데이터베이스의 서브셋으로 동작 (캐시가 가진 데이터는 데이터베이스에 모두 있음)
- 세션: 사용자 간 공유되지 않고 세션 스토어에만 데이터를 두었다가 로그아웃 시 DB에 넣을 지 판단함

⚠️ **주의 사항**

- 장애 발생 시 내부 데이터 손실 가능성이 있어 캐시로 사용할 때보다 더 신중한 운영이 필요함