Redis Base

\*\* Redis

\* Remote Dictionary Server

- Remote : 레디스는 개별 서버에 각각 존재하는 것이 아닌 다수의 서버에서 공통적으로 사용 가능하도록 원격으로 존재함.

- Dictionary : key-value 형태로 상수 시간 복잡도로 사용됨

- 결론 : 레디스란 다수의 서버가 공유하는 해쉬테이블이다.

- 레디스는 표준-C로 작성된 오픈소스 In-Memory 데이터 저장소임(백업을 제외하고 모든 데이터를 램에 저장함) 🡺 기존의 관계형 데이터 베이스와 다름.

- 레디스는 이미 많은 서비스에 사용중인 데이터 베이스임.

\* Redis 특징

1. In-Memory : 모든 데이터를 RAM에 저장함 (백업이나 스냅샷은 제외함)

2. Single Threaded : 단을 스레드에서 모든 작업을 수행한다. 🡺 이러한 단숨함이 뛰어난 성능과 안정성을 제공함.

3. Clustrer Moad : 다중 노드에 데이터를 분산 저장하여 높은 안전성과 고가용성 제공

4. Persistence : Redis는 RDB+AOF를 통해 영속적인 저장 가능

5. Pub/Sub : Pub/Sub 패턴을 통해 실시간 채팅, 알림 서비스를 쉽게 개발 가능.

\* Redis 장점

1. 매우 높은 성능

2. 다양한 Data Type 지원(Data Type이 직관적으로 구성 됨.)

3. Redis는 파이썬, 자바, 자바스크립트등의 클라이언트 라이브러리 지원

4. 많은 관심을 받기 때문에 커뮤니티의 도움을 쉽게 받을 수 있음.

\* Redis 사용 사례

1. caching : 임시 데이터를 redis에 캐싱하여 활용

2. Rate Limiyer : 서버에서 특정 API에 대한 요청 횟수를 제한할 때 사용

3. Message Broker : 레디스의 데이터타입을 활용하여 Message Broker를 구현하여 다양한 서비스 간의 결합을 줄일 수 있음.

4. Redis를 이용하면 다양한 실시간 분석/ 계산이 가능. (실시간 순위표, 좌표 탐색, 실시간 방문자 계산)

5. redis의 pub/sub을 이용하면 실시간 채팅 구현에 매우 용이함.

\* Reids 영속성

- Redis는 주로 캐시로 사용되지만(손실되도 상관없는 데이터를 기록해야함)

- 하지만 실제 상황에서 캐쉬일지언정 보존되어야 하는 데이터가 존재함. 이를 위해서 데이터 영속성을 위한 옵션을 제공함(SSD 활용)

- RDB(Redis Database)

1. Point-in-time Snapshot : 특정 시간대에 스냅샷 생성

2. 장애가 발생 시 재난 복구(Disaster Recovery) 또는 동일한 데이터를 가진 cache를 복제에 사용

3. 스냅샷의 특성상 새로운 스냅샷 생성 이전 데이터 유실이 있을 수 있고 스냅샷 생성 도중엔 성능 저하로 클라이언트 요청이 지연될 수 있음.

-AOF(Append Only File)

1. 레디스에 적용되는 모든 Write 작업을 Log로 작업하기에 데이터 유실없이 거의 모든 데이터의 싱크를 맞출 수 있지만, 재난 복구시 모든 Log를 다시 Write 하기 때문에 스냅샷 방식 보다 복구 속도가 더 느리다.

-따라서 Redis의 경우 영속성 옵션을 사용하지 않거나 두가지 옵션을 각각 사용하거나 두가지 모두 사용하는등 사용자의 필요성에 따라 옵션을 조절할 수 있게 함.

Ex ) RDB + AOF : 둘다 사용하는 옵션

\*Redis Caching

- 캐싱은 데이터를 빠르게 읽고 처리하기 위해 임시로 저장하는 기술임.

- 캐싱은 계산된 값을 임시로 저장하고, 동일한 계산이나 요청이 발생한다면 저장된 값을 바로 사용한다.

- 캐시 사례

1. CPU 캐시 : CPu와 Ram의 속도 차이로 발생하는 지연을 줄이기 위해 L1, L2, L3 캐시 사용

2. 웹 브라우저 캐싱 : 웹 브라우저가 웹 페이지를 데이터를 로컬 저장소에 저장하여 재방문시 사용한다.

3. DNS 캐싱 : 이전에 조회한 도메인 이름과 해당하는 IP 주소(이런 IP 주소는 잘 안바뀌기 때문에 캐싱을 사용함)를 저장하여 재요청 발생 시 사용

4. 데이터베이스 캐싱 : 데이터 베이스 조회나 계산 결과를 저장하여 재요청시 사용(MySql과 같은 데이터 베이스는 버퍼풀에 자주 사용되는 데이터를 메모리에 미리 올려놓고 사용하는데 이런것도 캐싱임)

5. CDN : 이미지나 동영상과 같은 큰파일은 CDN에 캐싱하는데 용량의 큰 파일을 요청할때 사용한다. 즉 사용자와 원본 서버의 물리적 거리가 클 경우 해당 파일은 네트워크를 따라 전송되며 많은 시간이 소요되는데 이때 CDN을 이용하면 Pop서버에 미리 해당 파이을 올려놓고 사용자가 파일 요청시 가장 가까운 pop 서버에서 파일을 응답해줌으로써 네트워크 지연시간을 줄인다.

6. 어플리케이션 캐싱 : 우리가 Redis를 이용해서 백엔드 서비스 구현시 사용하게 될 캐싱을 의미하며 어플리케이션에서 데이터나 계산 결과를 캐싱하여 반복 작업을 최적화 하는 기술이다.

7. 외에도 다양한 캐싱 기술이 있기에 잘 찾아보며 익힐 수 있도록 하자.

\*Cache Hit/ Cache Miss

1. 우리가 redis 캐싱서버에 특정 키를 가진 해쉬를 요청시 정상적인 응답이 오면 Cache Hit, 그리고 키가 잘못되었거나 데이터 만료로 인해 응답이 안되면 cache miss라고 한다.

\*Cache-Aside Pattern(가장 많이 사용하는 방식)

- 어플리케이션에서 클라이언트 요청 처리를 위해 캐싱을 조회하고 hit시 그대로 캐싱을 사용하여 요청을 처리하고 miss면 원본 스토리지에서 데이터를 조회하여 데이터를 캐싱하는 패턴

- “읽기 최적화”방식, 캐시에 없으면 DB에서 읽고 캐시에 채워 넣음

\*Write-Through Pattern

- 애플리케이션이 데이터를 갱신할 때 캐시와 원본 저장소에 동시에 쓰는 방식

- 클라이언트->애플리케이션->캐시+DB동시에 Write

- “쓰기 시 즉시 동기화” : 캐시와 DB에 동시에 씀

\*Write-Aside(Lazy Write) Pattern

- 애플리케이션이 데이터를 갱신할 때 DB에만 기록하고, 캐시는 무효화 시키는 방식. 이후 데이터가 다시 필요할 때 캐시를 채워 넣음(Cache-Aside 패턴과 조합)

- 클라이언트->애플리케이션->DB만 Write->캐시(삭제)

- “쓰기 시 캐시 무효화” : DB만 갱신하고 캐시는 필요할 때 다시 채움.