REDIS

REDIS 개념

- 정의 : 오픈 소스이며, key-value 구조로 데이터 저장되는 in-memory **데이터 저장소**이다.

NoSQL이란,

등장 배경

- 환경의 변화 : 웹 2.0과 빅데이터가 등장하면서, **데이터와 트래픽 양이 기하급수적으로 증가**하였다.
- 한계 : RDB는 많은 데이터를 처리하기 scale-up (성능 향상) 을 할 수 있지만, '관계'형 특성상 scale-out (분산처리)는 복잡성이 너무 높아져 거의 불가능하다.

NoSQL의 등장

- 데이터의 일관성을 포기하고, **작고 값싼 여러대의 컴퓨터로 데이터를 분산저장**하도록 등장하였다.
- 정의 : SQL 외의 데이터 저장소이다. 아직 구체적인 정의는 존재하지 않는다
- 특징
  - (scale out 이 가능한) **클러스터에서 사용할 목적**이므로, 관계형 모델을 사용하지 않는다.
  - 스키마가 없어서, 필드 추가/삭제가 자유롭다
  - SQL을 사용하지 않는다
- 종류
  - key-value 형태 저장소 : **redis**, memcache
  - document 형태 저장소 : mongo
  - graph 모델

REDIS란,

REDIS 사용 이유

애초에 redis의 목적이 저장소와 캐시를 모두 고려하기 위함이기 때문에, 모호하며, 현업에서도 사용하기 쉽고 활용도가 좋아서 이 모저모 사용하고 있는 것으로 보인다.

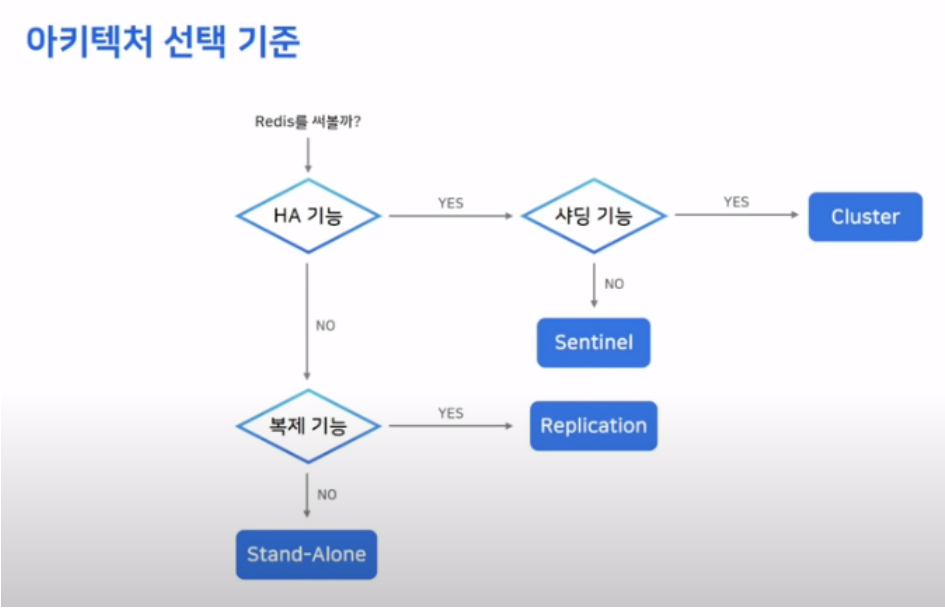
그럼에도 redis를 사용하는 데는 다음과 같은 이유가 있을 수 있다.

1. 데이터와 트래픽양이 많아진 것을 감당하고 싶다. (NoSQL의 특징)
2. 저장해야하는 데이터가 key-value 형태로 표현할 때 편리하다
3. memcache에 비해 10만 tps를 감당할 수 있는 강한 네트워크 대역폭을 제공할 수 있다.

특징

- **key-value** 형태로 저장한다.
  - key 에는 최대 메모리가 있으며, expire 시간을 정할 수 있다.
- value 에는 **다양한 자료구조**를 지원하여, 활용도가 높다.
  - String, List, Hash, Set 등 다양한 자료구조가 존재
- **메모리**에 저장
  - 데이터 처리가 빠르다

    - 서버 재시작시 데이터가 유실된다
- 메모리뿐만 아니라, 디스크에 저장하여 **영속성**을 지원할 수 있다. 따라서, database로서도 사용가능하다. 영속성 레벨에는 다음 3가지 종류가 있다.
    - 비영속성
    - RDB 방식 : 메모리에 있는 내용 전체를 스냅샷으로 전체 저장
    - AOF 방식 : 모든 명령어를 로그 파일에 기록하여 저장
- **싱글쓰레드**를 지원한다.
    - 데이터의 일관성을 유지할 수 있다
    - O(n) 명령어(`keys *`, `hgetall`)를 사용하면 다른 요청은 수행될 수 없으므로 사용하지 않는다.
    - 안정적인 시스템을 위해서 복제 구성을 한다.
- **무중단 서비스** 구성으로 높은 가용성을 구현할 수 있다. 방법은 다음과 같은 4가지 방법이 있다.
    1. Standalone : 래디스 서버 1대로 구성하며, 서버 다운시 AOF 또는 RDB파일로 재시작한다
    2. 복제 : 래디스 서버를 2대(마스터-슬레이브)로 구성한다. 마스터 서버가 다운되면 수동으로 슬레이브 서버를 마스터로 변경시켜야 한다.
    3. 복제 + 센티널 : 마스터-슬레이브 구성에 센티널을 추가하여, 각 서버를 감시하도록 한다. 마스터 서버가 다운되면, 슬레이브를 마스터로 승격시킨다.
    4. 클러스터 : 샤딩 방법을 제공하여, 데이터가 n개의 서버에 나누어 저장된다. 보통, 각 클러스터에 복제와 함께 제공된다.
    - 각 아키텍쳐의 선택 기준은 다음 사진을 참고한다.



REDIS의 활용

1. DB로서 REDIS

활용해도 되는 경우

- scale-out 가능한 대량 데이터 필요할 때
- redis를 DB로 사용해도 되는지 판단 기준

- 지속성보다 성능과 속도가 중요한 데이터
- 데이터가 손실되어도 영향이 적을 때
- 데이터가 비영구적일 때
- 메모리 사용량이 예측 가능할 때

주의사항

1. 복제 구성을 한다.
2. 메모리를 여유있게 사용한다
   - 장기적인 마이그레이션 필요
   - 자동화 툴을 만든다
3. RDB/AOF 등 데이터를 백업하려면, 복제 구성 중 slave에서 구동한다.

2. Cache로서 REDIS

Cache란

- 정의 : 캐시란, 원래 소스보다 더 **빠르고** 효율적으로 접근할 수 있는 임시 데이터 저장소.
- 사용 이유 : 속도 향상을 위해 대부분의 앱에서 사용하고 있다.
- 특징
  1. 원본보다 빠른 접근해야한다
  2. 동일한 데이터에 대해 반복적인 접근이 있을 때 (데이터의 재사용성이 한번 이상이어야
  3. 변하지 않는 데이터여야 한다

캐시로서 래디스가 적합한 이유

1. 사용의 간편함 : key-value 형태로 저장하기 때문에, 어떤 데이터를 쉽게 저장할 수 있다
2. 굉장히 빠르다
   - 모든 데이터를 메모리에 저장하기 때문
   - 평균 읽기/쓰기 작업이 1ms 미만이다. 따라서, 초당 수백만건의 작업이 가능하다

실제 활용 예

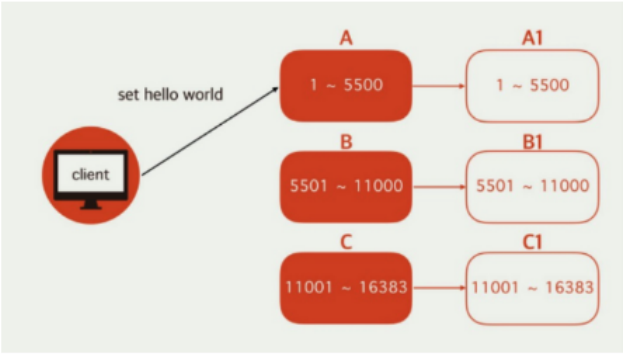
1. 현실 시뮬레이션 : 센서 데이터를 실시간으로 받아서 처리할 때 사용
2. 항공사 비행 일정을 여행사/일반 고객에게 제공 : DB로 구축하여, 빠르게 제공
3. 매신저 : 대화 내용 임시 저장, 사용자 상태 정보
4. IoT : 수많은 센서들로 데이터 받아서 처리 (차가 아파트로 들어오면 난방 가동, ..)
5. 세션 스토어 저장

참고

데이터베이스의 아키텍처

- 초기 구성 : **Stand Alone**
  - 배경 : 1980년대까지 유행했던 아키텍처로, 네트워크로 접속하지 않고 독립하여 동작하는 구성이다. DB서버가 설치된 장소로 물리적인 접근이 필요했다.

    - 특징 : 서버가 1대였으므로 가용성이 낮았고, 확장성 또한 낮았다.
- 그 이후 등장한 구성 : **리플리케이션**과 **클러스터**



	리플리케이션	클러스터
목적	데이터의 안정성	**페일오버** 처리로 높은 가용성
데이터 자징	복제되어 100% 갖고 있다.	서버 n개로 나누어서 저장된다.
노드간 관계	수직관계 (master & slave)	수평관계 (2개이상의 노드)
처리 방식	mater node는 쓰기작업 담당 slave node는 읽기 작업 담당	
동기화 방법	비동기로, 무결성 검사를 하지 않는다	동기
장점	성능을 높일 수 있다. (요청 대부분이 읽기여서) 지연 시간이 없다. (비동기여서)	항상 일관성 있는 데이터 1개 노드가 죽어도, 높은 가용성
단점	동기화가 되지 않을 수 있다 마스터 노드 다운시, 복구 및 대처 어렵다	쓰기 성능 나쁘다(동기화 시간이 필요)확장성에 한계
종류		1개 저장소 공유 - *Active-Active* - *Active-Standby* 2개 이상 저장소 분리 - **Sharding**

> **페일오버란**, (= 시스템 대체 작동, 장애 조치)
> 시스템이 이상이 생겼을 때, 예비 시스템으로 자동 전환되는 기능이다. 높은 가용성과 신뢰성을 요구하는 시스템에서 페일오버 구축은 필수다. 무정지 시스템을 구축할 수 있다.
>
> 반대로 사람이 수동 전환하는 것은 스위치 오버라고 한다.

> **샤딩이란**, 데이터를 다수의 서버에 분산하여 저장하는 방법이다.
> 시스템의 복잡도가 높아지기 때문에, 가능하면 샤딩을 피하거나 지연시켜야 한다.
> 예를 들어, 읽기 부하가 크다면, 캐시나 데이터베이스의 복제를 적용한다. 또한, 테이블의 일부 칼럼만 자주 사용한다면, 파티셔닝을 고려하거나, 데이터를 Cold Data로 분리한다.

References

https://dataonair.or.kr/db-tech-reference/d-lounge/technical-data/?mod=document&uid=236126

http://redisgate.kr/redis/introduction/redis_intro.php

http://redisgate.kr/redis/configuration/redis_overview.php

http://redisgate.kr/redis/sentinel/sentinel.php

https://www.youtube.com/watch?v=92NizoBL4uA&t=1163s

https://www.coovil.net/db-replication/

https://inpa.tistory.com/entry/REDIS-📚-캐시Cache-설계-전략-지침-총정리#

https://velog.io/@wnwl1216/캐시의-모든-것-정리