개발자를 위한 Redis

1장. 마이크로서비스 아키텍처와 레디스

모놀리틱 아키텍처
전통적인 소프트웨어 개발 모델로, 전체 어플리케이션을 하나의 통합된 패키지로 개발, 배포하는 방식. 모든 시스템은 하나로 연결돼 관리된다

마이크로서비스 아키텍처
독립된 각각의 모듈을 조립해 하나의 서비스를 만드는 아키텍처. 업데이트, 테스트, 배포, 확장은 각 서비스별로 독립적으로 수행될 수 있다.

관계형 데이터베이스의 특성
원자성(Atomicity)
- 트랜잭션이 완벽하게 실행되거나 아예 실행되지 않음을 보장

일관성
- 트랜잭션은 실행 전후로도 제약 조건을 만족시킴을 보장

독립성
- 트랜잭션 실행 시 다른 트랙잭션의 개입이 없음을 보장

지속성
- 성공적으로 수행된 트랜잭션은 영원히 반영돼야 함을 보장

NoSQL 특징
실시간응답
- 데이터 저장소에서 데이터를 가져올 때 발생할수 있는 지연은 0~1ms 이내여야 한다. 마이크로서비스 내의 저장소에서는 빠른 응답속도가 더욱 중요. 각각의 개별 서비스가 빠르게 동작하지 않으면 병목 현상 유발

확장성
- 서비스 업그레이드로 인한 계획적인 확장뿐만 아니라, 새해 첫날의 이벤트, 세일등 예상치 못한 이벤트로 인한 트랜잭션의 증가에 유연하게 확장 가능

클라우드 네이티브
- 클라우드 제공 업체에서 제공하는 Dbass를 사용하면 직접 설치 운영할 필요 없이, 설치된 상품을 바로 사용할 수 있고 운영을 위한 모니터링과 알람 또한 제공되기 때문에 데이터 베이스를 사용하기 위해 드는 번거로움 간소화

단순성
- 마이크로서비스 아키텍처와 같이 서비스가 세분화될수록 관리 포인트는 늘어나므로 데이터 저장소를 간단하게 사용하고 싶어한다.

유연성
- 비정형 데이터가 급속하게 많아졌기 때문에 이러한 데이터를 관계형 데이터베이스 보다 다양한 방식으로 저장할 수 있는 방법을 제공한다.

NoSQL 데이터 저장소 유형

그래프 유형
관계를 저장하고 표현할 때 유용하게 사용될 수 있으며, 저장되는 속성의 크기가 크거나 혹은 매우 많은 속성을 저장할 때에는 적합하지 않은 경우가 많음.
추천서비스에서 유용하게 사용될 수 있음. SNS에서 유저(노드) 간 친구 관계(에지)를 이용해 새로운 친구를 추천해주는 서비스나, 쇼핑몰에서 관심 분야나 구매 이력이 비슷한 다른 유저가 구입한 제품을 파악해 관심이 있어 하는 상품을 추천하는 기능, 사기 감지, 소셜미디어, 네트워크 및 IT 운영 등의 상황에서도 유용하게 사용

칼럼유형
데이터는 하나의 열에 중첩된 키-값 형태로 저장될 수 있기 때문의 기존의 관계형 데이터베이스와 비교했을 때보다 유연한 스키마 저장가능
대량의 데이터에 대한 집계 쿼리를 다른 유형보다 훨씬 빠르게 처리할 수 있어, 기업의 BI 분석을 위한 데이터 웨어하우스에서 유용하게 사용될 수 있다.
분석, 보고, 빅데이터 처리에 적합하며 대표적인 데이터베이스로는 Apache Cassandra, Hbase등이 있다.

문서유형
JSON 형태로 데이터가 저장돼 개발자들이 편하게 사용할 수 있는 구조다. 개발자의 프로그램 코드와 동일한 형태로 데이터가 저장돼, 효율적이고 직관적으로 데이터를 사용할 수 있다. 또한 스키마가 따로 정해져 있지 않기 때문에 애플리케이션에 맞게 데이터를 그대로 저장할 수 있어 유연성이 크다.
대표적인 데이터베이스로는 MongoDB, CouchDB, AWS의 DocumentDB등이 있다.

키-값유형
데이터의 저장이 간단하기 때문에 다른 유형보다 수평적 확장이 쉽다. 게임이나 IoT와 같은 실시간 서비스에서는 사용자의 경험을 위해 빠른 응답 속도가 중요하다.
특히 로그를 남기는 작업이나 대규모 세션을 실시간으로 관리해야 하는 상황에서는 지연 시간을 최소화해야 한다.
대표적인 데이터 베이스로는 레디스, AWS의 ElastiCache, AWS의 DynamoDB, Oracle NoSQL Database, Memcached등이 있다.

레디스
Remote dictionary server의 약자로 고성능 키-값 유형의 인메모리 NoSQL 데이터 베이스로, 오픈 소스 기반의 데이터 저장소다.

레디스 특징
실시간 응답(빠른성능)
 - 인메모리 데이터베이스로 디스크에 접근하는 과정이 필요 없기 때문에 처리 성능이 굉장히 빠르다.

단순성
 - 키-값 형태로 데이터를 관리할 수 있고, 문자열 뿐만 아니라 해시, 셋등 복잡하고 다양한 데이터 구조를 저장할 수 있도록 지원한다. 임피던스 불일치란 기존 관계형 데이터베이스의 테이블과 프로그래밍 언어간 데이터 구조, 기능차이로 인해 발생하는 충돌을 의미하는데 레디스는 내장된 다양한 자료구조를 통해 이를 해소한다. 레디스는 싱글 스레드로 동작한다.

고가용성
 - 자체적으로 HA기능을 제공한다. 복제를 통해 데이터를 여러 서버에 분산시킬 수 있으며, 센티널은 장애 상황을 탐지해 자동으로 페일오버를 시켜준다.

확장성
 - 클러스터 모드를 사용하면 손쉬운 수평적 확장이 가능하다. 데이터는 레디스 클러스터 내에서 자동으로 샤딩된 후 저장되며, 여러 개의 복제본이 생성될 수 있다.
클러스터 구조에서는 모든 레디스 인스턴스는 클러스터 버스라는 프로토콜을 이용해 서로 감시하고 있으며 이를 이용해 클러스터의 마스터 노드에 문제가 발생하면 자동으로 페일오버를 시켜 고가용성을 유지할 수 있다.

클라우드 네이티브 – 멀티 클라우드
 - 클라우드 네이티브는 클라우드 환경에 특화된 애플리케이션의 개발 및 운영 방식을 의미한다. 이 방식은 마이크로서비스, 컨테이너, 오케스트레이션, 데브옵스와 같은 현대의 개발 운영 및 패러다임을 포용하며, 빠른 배포와 확장성, 높은 복원력을 중심으로 한 애플리케이션을 추구한다. 레디스는 이러한 클라우드 네이티브 환경에서 빠른 데이터 액세스 및 처리를 지원하는 구조로 인해 마이크로서비스 아키텍처와의 연계에서 큰 장점을 지닌다.

데이터 저장소로서의 레디스
마이크로서비스 아키텍처에서 각 서비스별 개별 저장소로 사용하기에 알맞다.
설치가 간편하고 최소한의 리소스로 막대한 처리량을 낼 수 있으며, 다양한 자료구조를 제공하면서도 사용이 간단하기 때문이다.
또한 고가용성을 위해 로드밸런스나 프록시 등 추가적인 서비스를 설치할 필요가 없다.
메모리에 있는 데이터가 영구 저장되지 않기 때문에 데이터 저장소로 사용하기 위해 레디스 도입을 고민할 때 AOF 와 RDB 형식으로 디스크에 주기적으로 저장할 수 있다.

메시지 브로커로서의 레디스
레디스의 pub/sub 기능은 가장 간단한 메시징 기능으로, 굉장히 빠르게 동작하며 간단하게 사용할 수 있다. 1개의 채널에 데이터를 던지면 이 채널을 듣고 있는 모든 소비자는 데이터를 빠르게 가져갈 수 있다. 모든 데이터는 전달된 뒤 삭제되는 일회성으로 간단한 알림 서비스에서는 유용하게 사용 가능하다.

레디스의 list 자료 구조는 메시징 큐로 사용하기 알맞다. 애플리케이션은 list에 데이터가 있는지 매번 확인할 필요 없이 대기하다가 list에 새로운 데이터가 들어오면 읽어 갈 수 있는 블로킹 기능을 사용할 수 있다.

레디스의 stream 자료 구조를 이용하면 레디스를 완벽한 스트림 플랫폼으로 사용할 수 있다. 아파치 카프카 시스템에서 영감을 받아 만들어진 자료 구조로 데이터를 계속해서 추가하는 방식으로 저장된다. 카프카처럼 저장되는 데이터를 읽을 수 있는 소비자와 소비자 그룹이 존재해 데이터의 분산 처리도 가능하며, 저장된 데이터를 시간대별로 검색하는 것도 가능하다.