통 합 구 현

대용량 서비스 레퍼런스 아키텍처 2

한국기술교육대학교
온라인평생교육원

학습내용

- Business Layer
- Persistent Layer

학습목표

- Business Layer의 개요 및 요청 처리 방법에 대해 설명할 수 있다.
- Persistent Layer로 사용할 수 있는 요소들에 대해 비교 및 설명할 수 있다.

1. Business Layer

1) 개요

Business Layer란

- 클라이언트로부터 요청을 받아 데이터베이스나 파일에서 데이터를 쓰거나 읽어 비즈니스 로직에 따라 처리하여 응답을 내보내는 계층

1. Business Layer

2) 동기 요청 처리

동기식 처리

- 일반적인 요청(Request) - 응답(Response) 형식의 처리로 응답이 올 때까지 클라이언트가 기다리고 있는 호출 형태

- 클라이언트가 요청
- 서버쪽의 비즈니스 컴포넌트가 요청 처리
- 요청에 대한 처리가 끝나면 클라이언트에 응답을 보냄

[동기 요청 처리 방식]

1. Business Layer

2) 동기 요청 처리

(1) 생산성의 문제

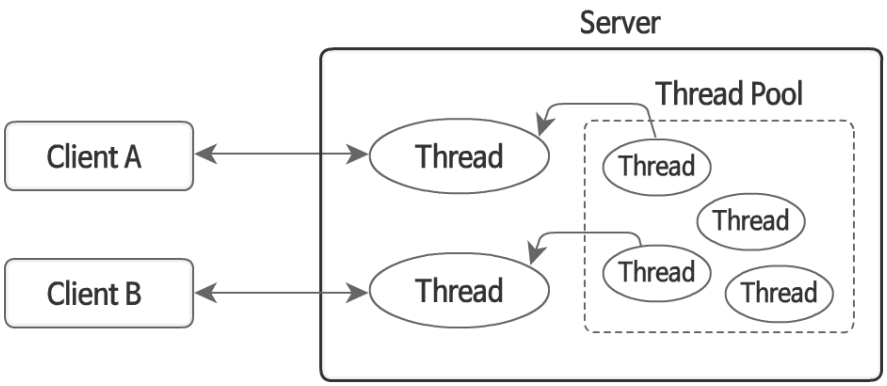
- 자바 플랫폼 이용한 REST API 개발 시 서버에 대한 초기 설정 시 숙련자라도 많은 시간이 소요됨 (스프링 REST 프레임워크, 스프링 프레임워크, myBatis/Hibernate, 톰캣/Jboss, Maven 스크립트)
- 스크립트기반 웹 서비스 플랫폼(python, Ruby on Rails, Node.js 등)을 이용하면 개발 시간이 단축되나 안정성이나 견고성 측면에서 자바 플랫폼보다 열세
- 자바 플랫폼의 경우 XA(eXtended Architecture) 규격으로 지원해 복잡한 트랜잭션 관리가 가능하고 여러 프레임워크의 조합으로 복잡한 비즈니스 로직 구현 가능

1. Business Layer

2) 동기 요청 처리

(2) 용량 문제

- JEE(Java Enterprise Edition)나 자바 스프링 기반 애플리케이션은 Thread Pool 구조 사용
  - Thread Pool : 클라이언트로부터 요청이 올 시 미리 만들어놓은 스레드 풀에서 스레드 꺼내 요청 처리 후, 다시 스레드 풀로 보낸 후 또 다른 요청이 오면 다시 꺼내 사용하는 구조
- 동시에 처리할 수 있는 요청 수 = 스레드 풀 내부의 스레드 개수
  - 톰캣은 500개(최대 2000개) 정도의 스레드를 생성할 수 있음(물리적 한계)



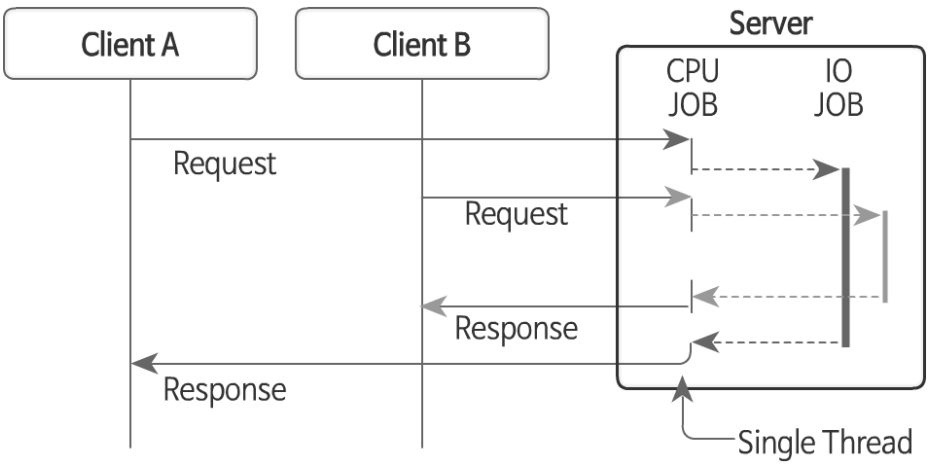
[스레드 풀의 개념]

- I/O 효율 : I/O 작업(DB, 네트워크, 파일)이 있을 경우 I/O 호출 후 스레드는 CPU를 사용하지 않는 대기(wait) 상태로 아무 작업을 하지 않는 비효율성 발생
- 현대의 대용량 서비스는 수만 건의 요청을 동시에 처리(C10K 문제)해야 하기 때문에 근래에는 Node.js, Nginx같은 비동기 서버 사용하기도 함

1. Business Layer

2) 동기 요청 처리

(2) 용량 문제



[비동기 서버의 요청 처리 방식]

- 하이브리드 플랫폼 활용 : 비즈니스 로직이 복잡한 서비스의 경우 중요 API는 자바로 개발하고 일반 API는 스크립트 언어 이용

1. Business Layer

2) 동기 요청 처리

(3) Shared Nothing 아키텍처

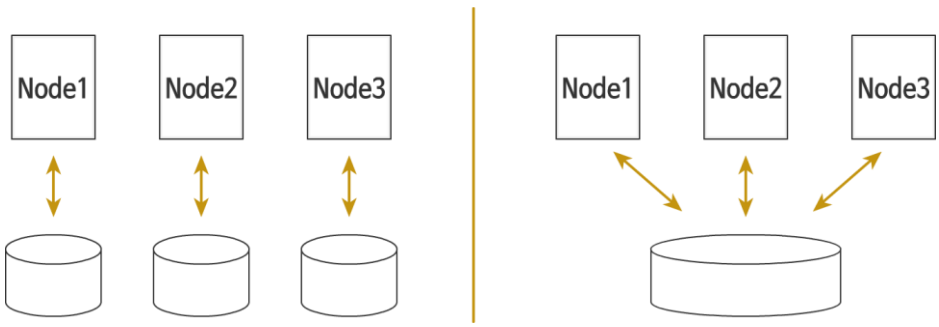
① 정의

Shared Nothing 아키텍처란?

- **분산 처리 시스템을 구성하는 여러 개의 노드가 서로 종속성을 가지지 않고 독립적으로 작동하는 아키텍처**

- 이론적으로 노드를 무제한으로 늘릴 수 있음
- 전체 시스템의 용량이 노드 수에 비례
- 특정 노드 장애가 전체 시스템에 영향을 주지 않음

② 개요



- 상태 정보(state)에 따라 노드 간 종속성 발생 가능

예시

EJB의 Stateful EJB 상태 정보, HTTP 세션 정보 등

- 비즈니스 계층이 아닌 다른 계층에서 상태 정보를 저장하여 서버들이 정보 공유

(4) 상태 정보 저장 방법

- 클라이언트에 상태 정보 저장하는 방법으로 쿠키사용
- 데이터베이스, 캐시, 데이터 그리드 계층, NFS, RDBMS 등을 이용하여 상태 정보 저장

1. Business Layer

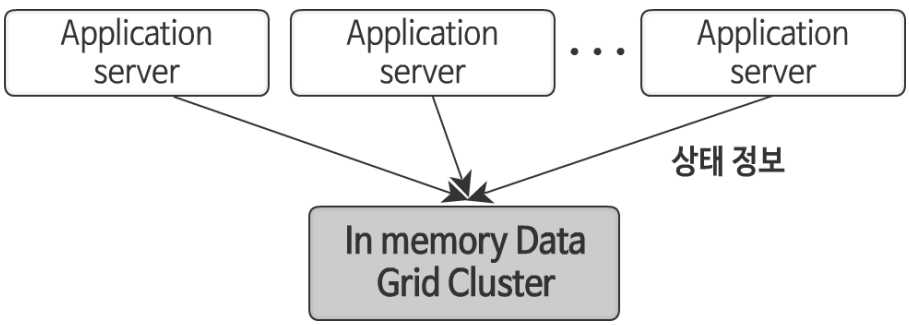
3) 상태 정보 저장소 (메모리 캐쉬)

(1) IMDG(In Memory Data Grid)

① 정의

IMDG(In Memory Data Grid)란?

- 여러 서버의 메모리를 연결해 수십 기가의 메모리 저장소 만들어놓고 애플리케이션 서버에서 접근해 사용 (메모리 클러스터)



[데이터 그리드를 이용한 공유 정보 처리]

② 용도

- 캐시, HTTP 세션, 사용자 로그인 정보 등의 상태 정보를 저장하여 공유

1. Business Layer

3) 상태 정보 저장소 (메모리 캐쉬)

(1) IMDG(In Memory Data Grid)

③ 특징

고가용성(High Availability)

- 서버 간 클러스터링으로 Fail Over(장애가 나지 않은 서버가 그 역할을 내려받음)가 가능해, 장애에 대해 고가용성을 가짐
  - 고가용성 : 서버와 네트워크, 프로그램 등의 정보 시스템이 상당히 오랜 기간 동안 지속적으로 정상 운영이 가능한 성질 (시스템에 오류가 발생하여도 서비스의 성능은 감소하지만 지속적으로 서비스를 제공)

확장성

- 서버를 추가해 전체 저장 용량 확장 가능

휘발성

- 전체 서버가 다운되면 모든 데이터가 사라질 수 있음

④ 메모리 캐쉬로 활용하는 오픈 소스 솔루션으로 Memcache, Redis 등이 있음

- https://memcached.org/
- http://redis.io/

1. Business Layer

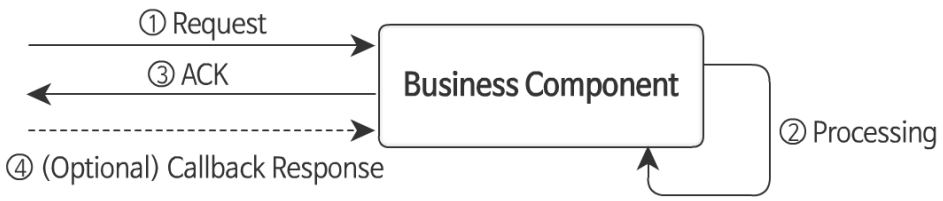
4) 비동기 요청 처리(메시지 큐)

(1) 비동기식 처리

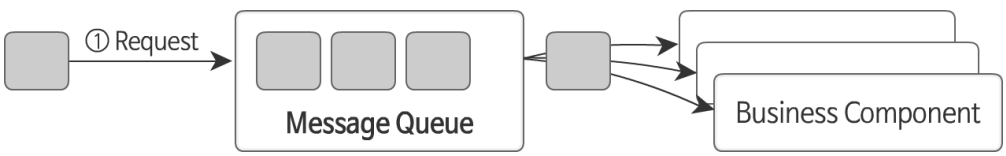
비동기식 처리란?

- 요청을 보낸 후 비즈니스 로직이 처리가 완료되지 않은 상태에서 다음 로직 진행

- 시간이 오래 걸리는 대규모 작업에 유리



- 비동기식 패턴 구현으로 메시지 큐 사용
  - 메시지 큐 : 들어온 요청을 쌓아놓는 임시 공간



[메시지 큐를 이용한 비동기 처리]

1. Business Layer

4) 비동기 요청 처리(메시지 큐)

(1) 비동기식 처리

Rabbit MQ

- 지원되는 기능과 모니터링 기능이 탁월 및 매우 편리

Zero MQ

- 새로운 메시지 처리 프레임워크나 미들웨어를 만들기에 좋지만 추가 개발 필요

Active MQ

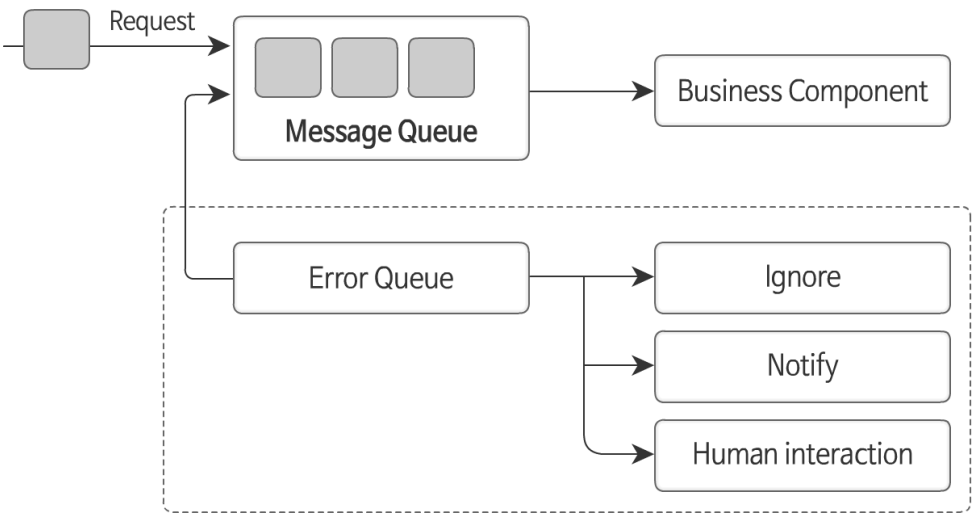
- Apache 재단에서 관리
- 많은 기능과 프로토콜 지원하지만 사용이 복잡함

1. Business Layer

4) 비동기 요청 처리(메시지 큐)

(2) 에러 처리

- 비동기식 구현은 응답을 기다리지 않기 때문에 해당 요청이 제대로 처리 되었는지 보장할 방법이 없음
- 에러 발생 시 에러 큐 라는 재처리용 큐로 전달해 에러 처리 방식 지정
  - 재처리(Retry) : 일시적 장애에 대해 재처리 진행. 대기 시간과 재처리 횟수 지정 필요
  - 무시(Ignore) : 에러가 난 메시지는 무시하고 삭제. 메시지 유실이 허용되는 경우에 사용
  - 알림(Notify) : 관리자에게 통보해 직접 에러에 대한 후처리를 할 수 있도록 함
  - 사람이 처리(Human Interaction) : 관리자가 처리할 수 있는 UI 제공



[메시지 큐를 이용한 비동기 처리에서의 에러 처리 방식]

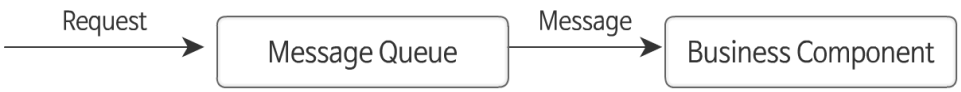
1. Business Layer

4) 비동기 요청 처리(메시지 큐)

(3) 비동기 메시지 패턴

① Fire & Forget 패턴

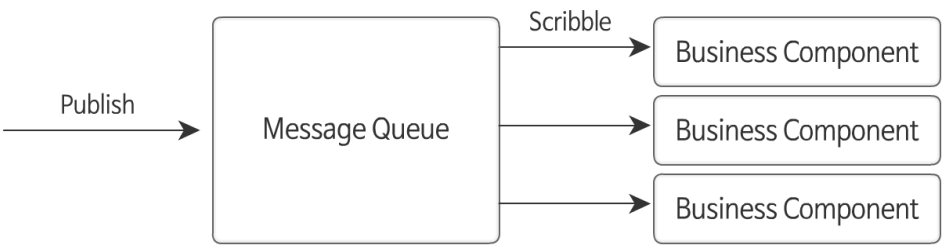
- 가장 일반적인 패턴
- 메시지 큐에 제대로 들어가면 메시지 처리 결과에 관계 없이 바로 반환



[Fire & Forget 패턴]

② Publish & Subscribe 패턴

- 메시지 큐에 구독자를 등록해, 1:N 관계의 비동기 처리 구현



[Publish & Subscribe 패턴]

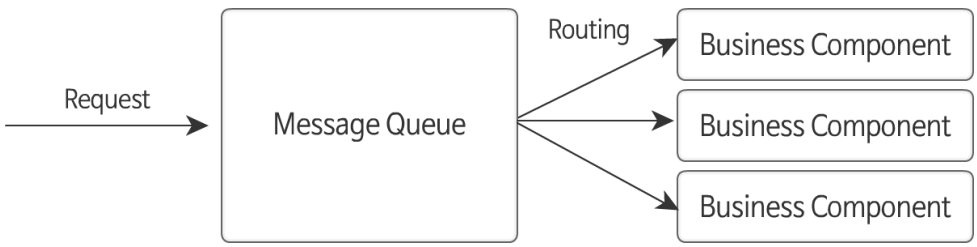
1. Business Layer

4) 비동기 요청 처리(메시지 큐)

(3) 비동기 메시지 패턴

③ Routing 패턴

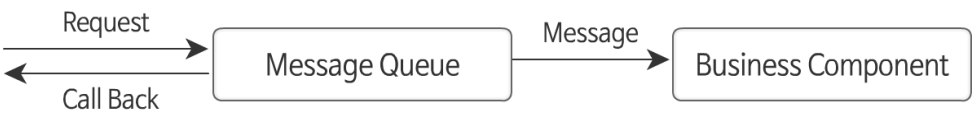
- 큐에 저장된 메시지를 조건에 따라 특정 비즈니스 컴포넌트로 라우팅



[Routing 패턴]

④ Call back 패턴

- 비즈니스 컴포넌트에서 처리가 끝나면 서버가 클라이언트에 처리가 끝났다는 응답과 처리 결과 메시지를 콜백으로 보냄



[Call Back 패턴]

1. Business Layer

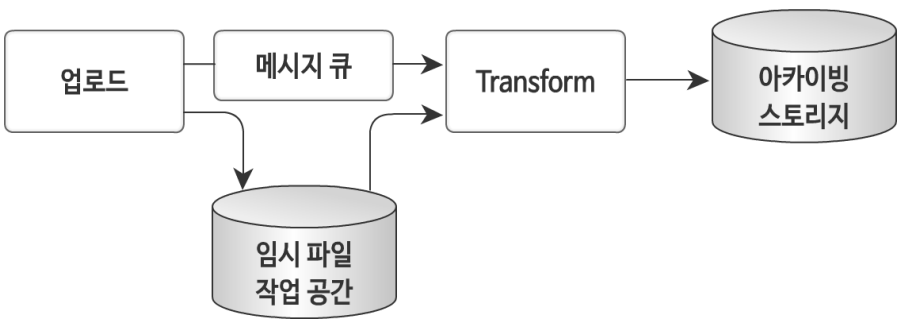
5) 임시 파일 작업 공간

(1) 정의

임시 파일 작업 공간이란?

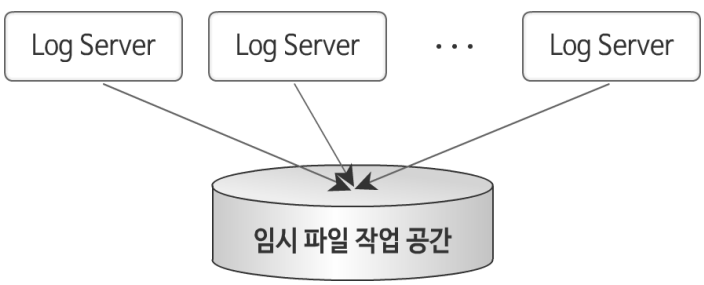
- **비동기 처리 및 분산 환경에서 추가로 필요한 컴포넌트로, 작업을 위해 임시로 파일을 저장하는 공간**

- 직접 어플리케이션 서버에 네트워크 파일 시스템(NFS) 등으로 마운트 될 수 있는 파일 시스템
- 동영상을 변환해서 아카이빙 스토리지에 저장
  - 아카이빙 스토리지에 저장하는 경우 비동기 절차가 길어질 수록 많은 파일 복사가 발생하게 됨
  - 임시 파일 작업 공간을 활용하는 것이 바람직함



[임시 파일 작업 공간 개념]

- 대규모 분산 환경에서는 분산 파일 스토리지를 사용하거나 공유 파일 시스템 사용



[공유 파일 시스템을 이용한 임시 파일 작업 공간]

1. Business Layer

6) 메시징 프로토콜

- 컴포넌트는 프로세스 단위로 분리되어 있어 통신을 하기 위해 리모트 호출 필요
- 근래에는 일반적으로 단순한 텍스트 기반 메시지 포맷을 사용하는 HTTP JSON 기반의 REST 방식을 많이 사용
  - HTTP는 Connectionless 프로토콜로 호출시마다 새로운 호출을 만들기 때문에 네트워크 오버로드가 큼
  - JSON은 일반적인 바이너리 파일에 비해 크기가 크고 파싱에 시간이 오래 소요
  - 가독성 및 사용성 측면에서 외부로 제공하는 API에 적합
- 내부 시스템 간의 호출에는 바이너리 프로토콜을 사용하는 것이 성능 및 용량 면에서 효율적임

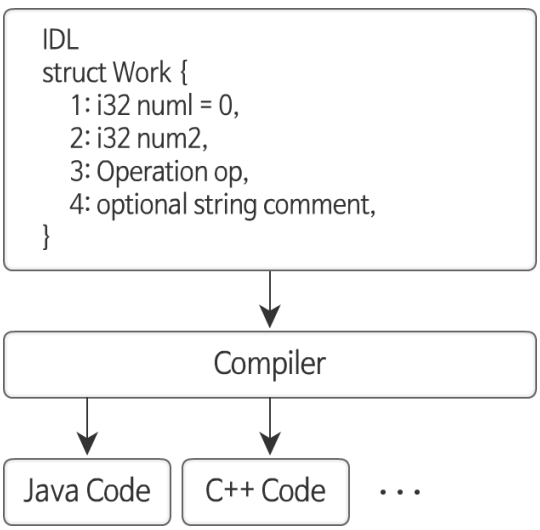
1. Business Layer

6) 메시징 프로토콜

(1) IDL(Interface Definition Language: 인터페이스 정의 언어)

① 메시지 포맷을 정의하고 전용 컴파일러를 통해 IDL로 정의된 메시지 포맷을 사용하고자 하는 언어에 맞는 코드로 변경

- 단점 : 데이터 포맷 변경에 대한 반영이 유연하지 않음 (재컴파일)
- 장점 : REST에 비해 성능이 매우 빠르고 용량이 작음



2. Persistent Layer

1) 개요

Persistent Layer란

- 처리할 데이터를 저장하는 공간

(1) 종류

- 관계형 데이터베이스(RDBMS)
- 파일 시스템
- LDAP(Lightweight Directory Access Protocol)
  - 인증을 위한 사용자 정보와 같이 개수가 많고 레코드별 용량이 적거나, 데이터 구조가 디렉터리 구조인 경우 사용
- NoSQL
  - 대용량의 데이터를 키(key)/값(value) 형태의 단순한 구조로 저장한 저장소로 성능 및 확장성에 중점

2. Persistent Layer

2) RDBMS

(1) 분류

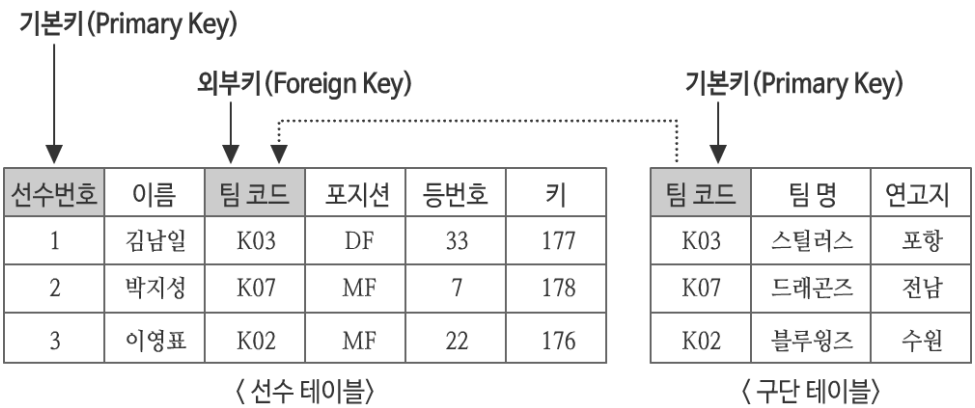
① OLTP(On-Line Transaction Processing)

- 요청을 처리하는 트랜잭션 처리용 → Persistent Layer

② OLAP(On-Line Analytical Processing)

- 데이터를 모아서 분석하고 리포팅 → Analysis Layer

- Persistent Layer의 RDBMS는 OLTP 성격의 데이터베이스
- 데이터베이스는 2차원 테이블 구조



선수번호	이름	팀 코드	포지션	등번호	키
1	김남일	K03	DF	33	177
2	박지성	K07	MF	7	178
3	이영표	K02	MF	22	176

〈선수 테이블〉

팀 코드	팀 명	연고지
K03	스틸러스	포항
K07	드래곤즈	전남
K02	블루윙즈	수원

〈구단 테이블〉

[테이블의 정규화]

- RDBMS 설계 시 성능향상을 위해 고려할만한 기법
  - Query off Loading
  - Sharding

2. Persistent Layer

2) RDBMS

(2) Query off Loading

• DB 트랜잭션 비율

Query off Loading 이란

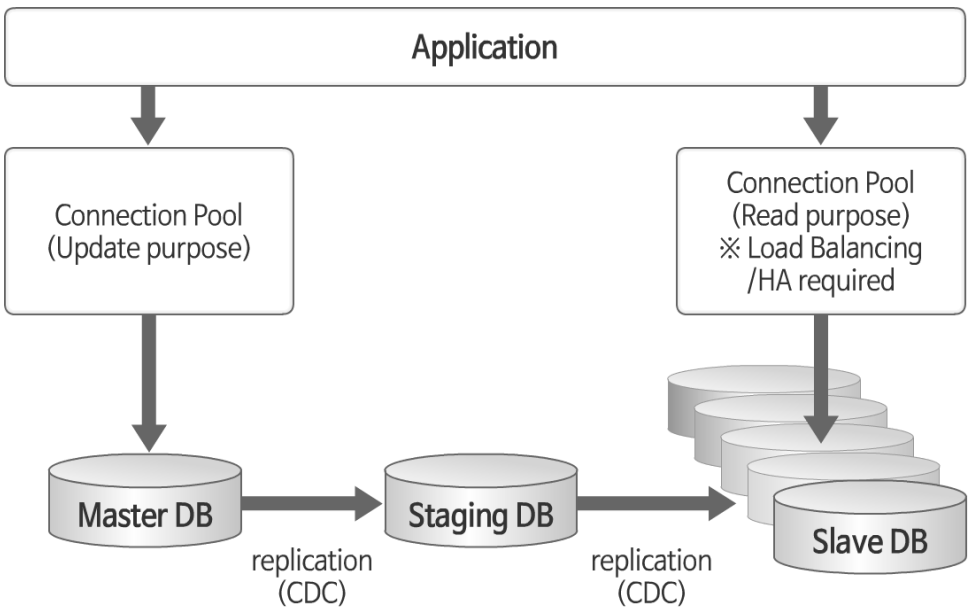
• **Read 와 다른 트랜잭션을 분리해 DB의 성능을 향상시키는 기법**

• 구성
- Master DB : 쓰기(Update)만 허용하며 해당 내용을 Staging DB로 복사
- Staging DB : Master DB와 Slave DB 사이에 존재하며 Master DB에서 복제된 내용을 N개의 Slave DB로 복제하기 위한 중간 저장소
- Slave DB : 읽기(Read)만 수행. N개 존재

2. Persistent Layer

2) RDBMS

(2) Query off Loading



[Query off Loading의 개념]

- Connection Pool을 통해 어플리케이션에서 DB로 쓰기 및 읽기 로직을 분리해 각 DB로 전송
- Slave DB는 여러 개이므로, 모든 Slave DB에 대한 로드 밸런싱 필요하며, 높은 가용성(HA, High Availability)을 가지도록 기능을 제공해야 함
- CDC(Change Data Capture) : 각 DB간 복제
- 복구 용도로 저장해 두는 백로그를 읽어 각 DB에 Replay 하는 형식으로 진행

2. Persistent Layer

2) RDBMS

(3) Sharding

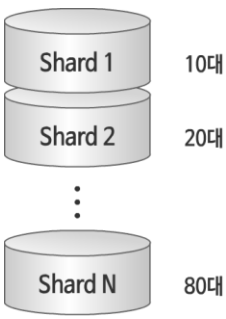
Sharding 이란

- **데이터베이스의 용량 한계를 극복하기 위한 기술로써 데이터를 여러 개의 데이터베이스에 분산 저장하는 방법**

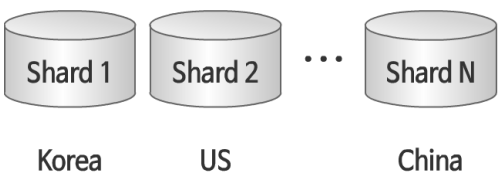
- Shard : 저장하는 각각의 데이터베이스

① 분류

- 수직적 샤딩 : 연속된 데이터에 대해 범위별로 데이터를 나누는 방법
  (10대, 20대, 30대, …., 90대)
- 수평적 샤딩 : Category에 따라 수평적으로 분리
  (Korea, US, Japan, China, …)



[수직적 샤딩]



[수평적 샤딩]

2. Persistent Layer

2) RDBMS

(3) Sharding

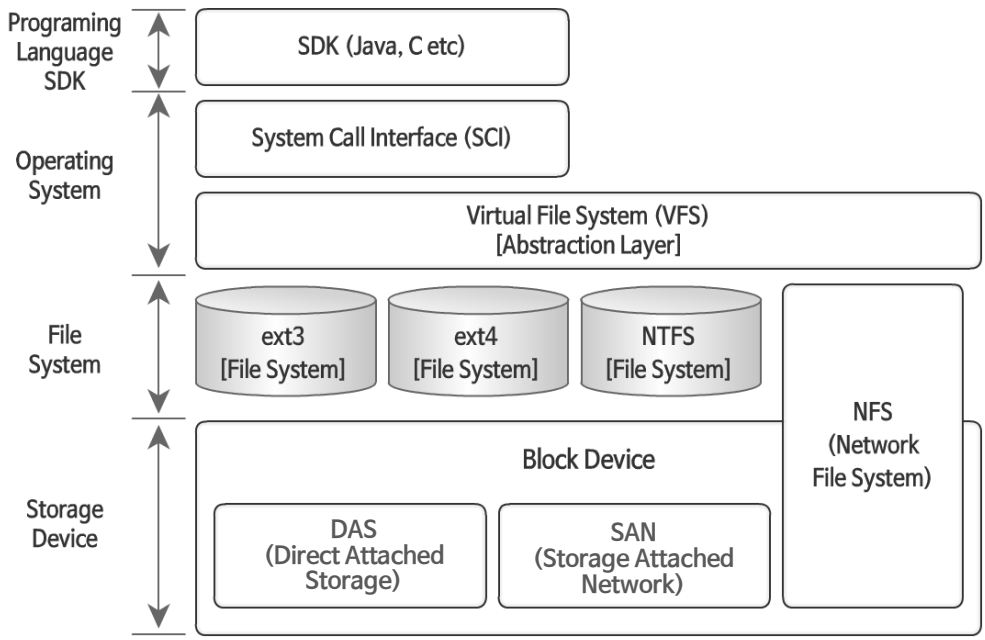
- 데이터 편중 현상에 대해 각 샤드 서버의 성능을 비대칭적으로 설계
- 데이터 편중 현상을 예측할 수 없는 경우 해시 방식(데이터를 모든 샤드에 걸쳐 분산 배치)을 사용
- 데이터의 분산저장으로 시스템의 전체 용량을 늘릴 수 있지만 애플리케이션의 복잡도가 올라가고 데이터가 편정되는 것을 방지하는 등 여러가지 요소를 고려한 후에 시스템 설계 반영

2. Persistent Layer

3) 파일 시스템

(1) 파일 시스템 스택

① SDK
② 운영체제 계층
③ 파일시스템
④ 스토리지 하드웨어 : DAS, SAN, NFS



[파일 시스템의 스택 구조]

2. Persistent Layer

3) 파일 시스템

SDK란?

- 프로그래밍 언어에서 지원하는 파일 시스템 접근용 API
- C의 fopen, JAVA의 java.io.* 등 라이브러리

운영체제 계층

- 다양한 파일 시스템을 같은 형태의 스토리지로 추상화
- 리눅스의 VFS(Virtual File System)

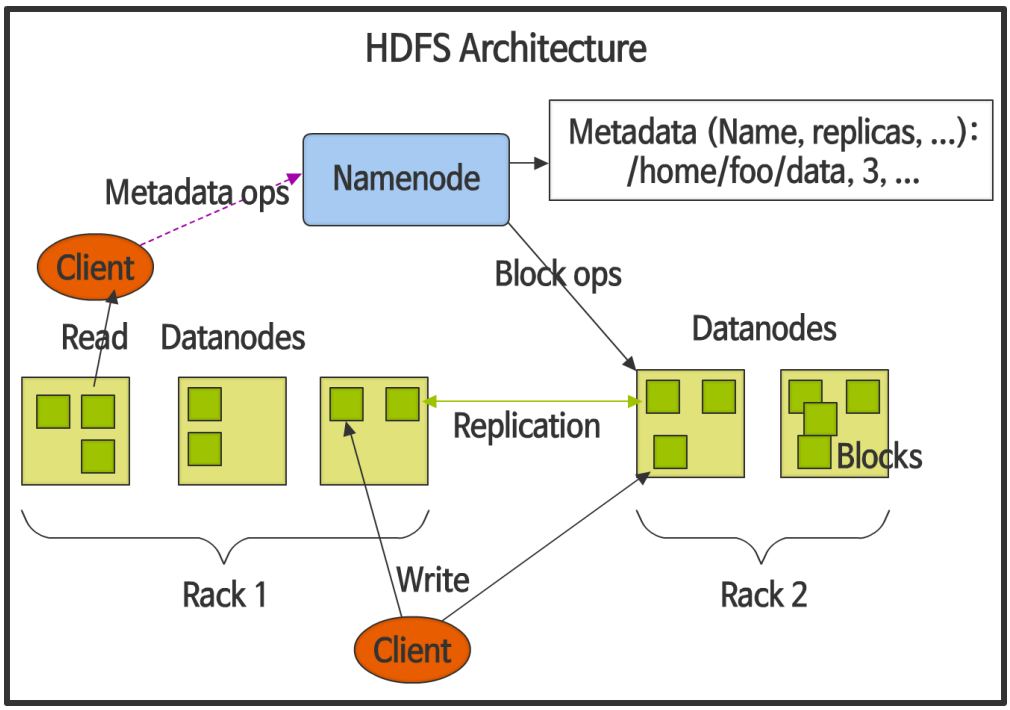
파일시스템

- 스토리지에 파일을 저장하고 관리하는 시스템

- 네이티브 파일 시스템 : OS에서 제공하는 로컬 파일 시스템 FAT32, NTFS, ext3 등
- 확장 파일 시스템 : NFS, GFS(Gluster File System), HDFS(Hadoop Distributed File System) 등으로 네이티브 파일 시스템을 이용해서 구성한 파일 시스템

2. Persistent Layer

3) 파일 시스템

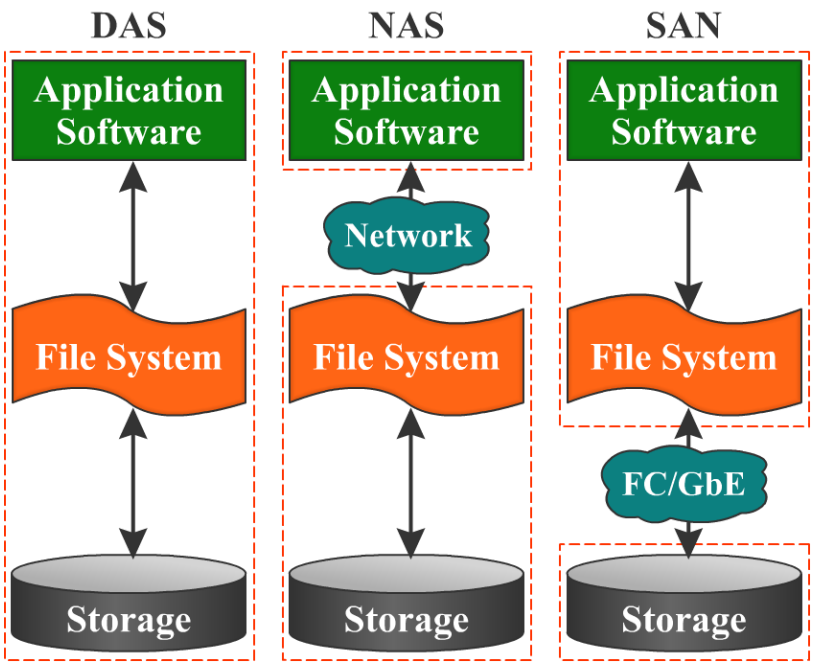


2. Persistent Layer

3) 파일 시스템

스토리지 하드웨어

- DAS(Direct Attached Storage) : 외장형 디스크처럼 서버에 직접 부착해서 사용하는 장치
- SAN(Storage Area Network) : 하나의 스토리지를 여러 개의 서버가 공유해서 사용하고, 각 서버는 SAN을 통해 접근
- NFS(Network File System) : 서버의 일정 영역을 다수의 클라이언트가 공유해서 사용하는 파일 시스템



2. Persistent Layer

3) 파일 시스템

Object storage

- 클라우드에서 제공하는 스토리지 (Amazon S3, OpenStack SWIFT)

- 기존 파일 I/O API대신 HTTP/REST를 사용하는 인터페이스 제공
- 파일 시스템 접근 시 OS 제약을 받지 않으며, 대용량 파일 서비스 구조에 적합

2. Persistent Layer

4) NoSQL

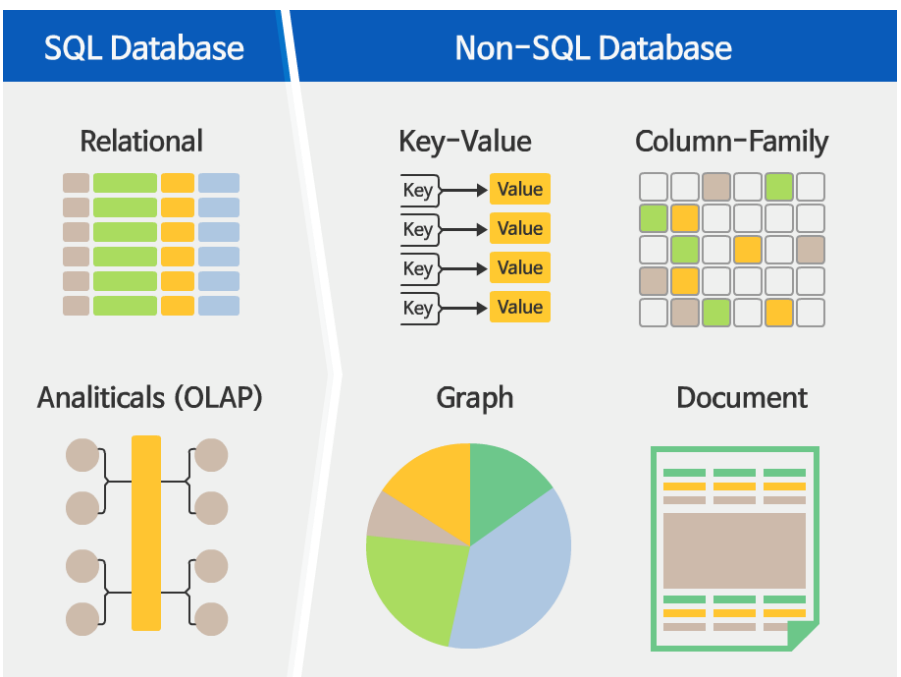
- 근래에 들어 블로그, 페이스북, 트위터 등 다양한 인터넷 서비스가 등장하면서 데이터베이스에 대한 요구 사항 변화
  - 빠른 응답 시간
  - 대용량

NoSQL이란?

- **단순한 데이터를 대용량으로 저장할 수 있으며 일반 사용자에게 빠른 성능을 제공하는 데이터베이스**

  - Key-Value 형태의 단순 저장 구조에서 출발
  - Key-value를 확장하거나, 그래프표현을 이용하거나, object를 문서형태로 정리한 모델들도 존재

- 일반적인 시스템 개발에서는 RDBMS로 충분하고, 대용량의 서비스를 빠른 성능으로 제공하는 경우에 고려해볼 필요가 있음



핵심정리

1. Business Layer

- 개요
  - 클라이언트로부터 요청을 받아 데이터베이스나 파일에서 데이터를 쓰거나 읽어 비즈니스 로직에 따라 처리하여 응답을 내보내는 계층
- 동기 요청 처리
  - 일반적인 요청(Request), 응답(Response) 형식의 처리로 응답이 올 때까지 클라이언트가 기다리고 있는 호출 형태
  - Shared Noting 아키텍처 : 분산 처리 시스템을 구성하는 여러 개의 노드가 서로 종속성을 가지지 않고 독립적으로 작동하는 아키텍처
- 상태 정보 저장소 (메모리 캐쉬)
  - 여러 서버의 메모리를 연결해 수십 기가의 메모리 저장소 만들어놓고 애플리케이션 서버에서 접근해 사용 (메모리 클러스터)
  - 특징 : 고가용성(High Availability), 확장성, 휘발성
- 비동기 요청 처리 (메시지 큐)
  - 요청을 보낸 후 비즈니스 로직이 처리가 완료되지 않은 상태에서 다음 로직 진행
  - 종류 : RabbitMQ, ZeroMQ, ActiveMQ
  - 패턴 : Fire & Forget 패턴, Publish & Subscribe 패턴, Routing 패턴, Call back 패턴
- 메시징 프로토콜
  - 단순한 텍스트 기반 메시지 포맷을 사용하는 HTTP JSON 기반의 REST 방식을 많이 사용

핵심정리

2. Persistent Layer

- 개요
  - 처리할 데이터를 저장하는 공간
- RDBMS
  - 2차원 테이블 형태로 데이터 저장
  - 성능 향상 기법 : Query off Loading, Sharding
  - Query off Loading : Read 와 다른 트랜잭션을 분리해 DB의 성능을 향상시키는 기법
  - Sharding : 데이터를 여러 개의 데이터베이스에 분산 저장하는 방법
- NoSQL
  - 단순한 데이터를 대용량으로 저장할 수 있으며 일반 사용자에게 빠른 성능을 제공하는 데이터베이스
- 파일 시스템