1. 프로그래밍 공통

- OOP
- OOP의 5가지 설계 원칙
- 절차지향 프로그래밍 VS 객체지향 프로그래밍

[OOP란]

OOP는 현실 세계를 프로그래밍으로 옮겨와 현실 세계의 사물들을 객체로 보고, 그 객체로부터 개발하고자 하는 특징과 기능을 뽑아와 프로그래밍하는 기법입니다. OOP로 코드를 작성하면 재사용성과 변형가능성을 높일 수 있습니다.

[OOP의 5가지 설계 원칙]

1. SRP(Single Responsibility Principle, 단일 책임 원칙): 클래스는 단 하나의 목적을 가져야 하며, 클래스를 변경하는 이유는 단 하나의 이유여야 한다.

2. OCP(Open-Closed Principle, 개방 폐쇠 원칙):
   클래스는 확장에는 열려 있고, 변경에는 닫혀 있어야 한다.

3. LSP(Liskov Substitution Principle, 리스코프 치환 원칙):
   상위 타입의 객체를 하위 타입으로 바꾸어도 프로그램은 일관되게 동작해야 한다.

4. ISP(Interface Segregation Principle, 인터페이스 분리 원칙):
   클라이언트는 이용하지 않는 메소드에 의존하지 않도록 인터페이스를 분리해야 한다.

5. DIP(Dependency Inversion Principle, 의존 역전 법칙):
   클라이언트는 추상화(인터페이스)에 의존해야 하며, 구체화(구현된 클래스)에 의존해선 안된다.

[절차지향 프로그래밍 VS 객체지향 프로그래밍]

- 절차지향 프로그래밍 물이 위에서 아래로 흐르는 것처럼 순차적인 처리를 중요시하는 프로그래밍 기법이다. 가장 대표적인 언어로 C언어가 있다. 컴퓨터의 처리구조와 유사해 실행속도가 빠르다. 코드의 순서가 바뀌면 동일한 결과를 보장하기 어렵다.

- 객체지향 프로그래밍 실제 세계의 사물들을 객체로 모델링하여 개발을 진행하는 프로그래밍 기법 가장 대표적인 언어로 Java가 있다. 캡슐화, 상속, 다형성 등과 같은 기법을 이용할 수 있다. 다형성은 동일한

키보드의 키가 다른 역할을 하는 것처럼 하나의 메소드나 클래스가 다양한 방법으로 동작하는 것을 의미한다. 절치지향 언어보다 실행속도가 느리다.

[RESTful API]

REST(REpresentational State Transfer)ful API는 HTTP 통신에서 어떤 자원에 대한 CRUD 요청을 Resource와 Method로 표현하여 특정한 형태로 전달하는 방식입니다. RESTful API는 아래와 같은 것들로 구성됩니다.

- Resource(자원, URI)
- Method(요청 방식, GET or POST 등)
- Representation of Resource(자원의 형태, JSON or XML 등)

[함수형 프로그래밍]

함수평 프로그래밍의 가장 큰 특징은 immutable data와 first class citizen으로서의 함수입니다. 함수형 프로그래밍은 부수효과가 없는 순수 함수를 이용하여 프로그램을 만드는 것이다. 부수 효과가 없는 순수 함수란 데이터의 값을 변경시키지 않으며 객체의 필드를 설정하는 등의 작업을 하지 않는 함수를 의미합니다.

[메모리 구조]

- 코드 영역:
  실행할 프로그램의 코드가 저장되는 영역으로 텍스트 영역이라고도 부릅니다. 사용자가 프로그램 실행 명령을 내리면 OS가 HDD에서 메모리로 실행 코드를 올리게 되고, CPU는 코드 영역에 저장된 명령어를 하나씩 처리하게 됩니다.
- 데이터 영역:
  프로그램의 전역 변수(global)와 정적 변수(static)가 저장되는 영역입니다. 데이터 영역은 프로그램의 시작과 함께 할당되며, 프로그램이 종료되면 소멸합니다.
- 힙 영역:
  프로그래머가 직접 관리할 수 있는 메모리 영역으로 이 공간에 메모리를 할당하는 것을 동적 할당이라고 부릅니다. Java에서는 가비지 컬렉터가 자동으로 해제해줍니다. 힙 영역은 스택 영역과 달리 낮은 주소에서 높은 주소로 메모리가 할당됩니다.
- 스택 영역:
  함수의 호출과 함께 할당되며 지역 변수와 매개 변수가 저장되는 영역입니다. 스택 영역에 저장되는 함수의 호출 정보를 스택프레임이라고 합니다. 스택 영역은 함수의 호출이 완료되면 소멸합니다. 스택 영역은 높은 주소에서 낮은 주소로 메모리가 할당됩니다.

[Parameter와 Argument의 차이]

Parameter: 함수를 선언할 때 사용된 변수 Argument: 함수가 호출되었을 때 함수의 파라미터로 전달된 실제 값

[Call By Value와 Call By Reference 차이]

- Call By Value
  인자로 받은 값을 복사하여 처리하는 방식
  Call By Value에 의해 넘어온 값을 증가시켜도 원래의 값이 보존된다.
  값을 복사하여 넘기기 때문에 메모리 사용량이 늘어난다.
- Call By Reference
  인자로 받은 값의 주소를 참조하여 직접 값에 영향을 주는 방식

값을 복사하지 않고 직접 참조하기 때문에 속도가 빠르다.
원래의 값에 영향을 주는 리스크가 존재한다.

예를 들어 아래와 같은 코드가 있다고 할 때, a라는 새로운 변수가 생성되어 Call By Value로 전달되기 때문에 메모리를 많이 사용하지만 a를 변경하여도 원래 값인 f는 영향을 받지 않습니다.

```
public class Main {

	public static void main(String[] args) {
		Foo f = new Foo("f");
		changeReference(f); // It won't change the reference!
		modifyReference(f); // It will modify the object that the reference
variable "f" refers to!
	}

	public static void changeReference(Foo a) {
		Foo b = new Foo("b");
		a = b;
	}

	public static void modifyReference(Foo c) {
		c.setAttribute("c");
	}

}
```

[프레임워크와 라이브러리 차이]

라이브러리:
사용자가 흐름에 대한 제어를 하며 필요한 상황에 가져다가 쓸 수 있다.
프레임워크:
전체적인 흐름을 자체적으로 제어한다.
프레임워크와 라이브러리는 실행 흐름에 대한 제어 권한이 어디 있는지에 따라 달라집니다. 프레임워크를 사용하면 사용자가 관리해야 하는 부분을 프레임워크에 넘김으로써 신경써야 할 것을 줄이는 제어의 역전(IoC, Inversion Of Control)이 적용됩니다.

[병렬 처리 프레임워크의 종류와 특징]

- Hadoop
  HDFS(Hadoop Distributed File System)를 활용해 데이터를 주고 받는다. 데이터가 여러 노드에 분산되어 저장되기 때문에 손실의 우려가 없다는 장점이 있다. 하지만 File I/O를 기반으로 작동하기 때문에 처리 속도가 느리다.
- Spark
  In-Memory 상 에서 데이터를 주고받고 연산을 수행한다. 메모리를 사용해 데이터를 처리하기 때문에 Hadoop보다 속도가 약 100배 정도 빠르다. 하지만 메모리상에서 처리하기 때문에 장애가 발생한 경우 응용 프로그램을 처음부터 다시 시작해야 한다.

[동기와 비동기의 차이]

- 동기(Synchronous) 방식
  요청을 보내고 실행이 끝나면 다음 동작을 처리하는 방식
  순서에 맞추어 진행되기 때문에 제어하기 쉽다.
  여러가지 요청을 동시에 처리할 수 없어 효율이 떨어진다.
  동기 방식의 예시로는 콜센터 종업원이 일을 처리하는 방식이 될 수 있다. 콜센터의 직원은 한 손님의 전화 응대가 끝난 후에 다음 손님의 응대를 진행할 수 있다.
- 비동기(Asynchronous) 방식
  요청을 보내고 해당 동작의 처리 여부와 상관없이 다음 요청이 동작하는 방식 작업이 완료되는 시간을 기다릴 필요가 없기 때문에 자원을 효율적으로 사용할 수 있다. 작업이 완료된 결과를 제어하기 어렵다.
  비동기 방식의 예제로는 이메일이 있다. 우리는 한 사람에게 이메일을 보냈을 때 답변을 받지 않고도 이메일을 다시 보낼 수 있다.

[SQL Injection]

SQL Injection이란 공격자가 악의적인 의도를 갖는 구문을 삽입하여 공격자가 원하는 SQL을 실행하도록 하는 웹해킹기법입니다. 예를 들어 아래와 같은 간단한 SQL 문이 있을 때 INPUT1에 'OR 1=1--을 넣는 것입니다.

```
SELECT * FROM USER WHERE ID = 'INPUT1' AND PASSWORD = 'INPUT2'

SELECT * FROM USER WHERE ID = '' OR 1=1 --INPUT1' AND PASSWORD = 'INPUT2'
```

INPUT1으로 'OR 1=1--을 넣으면 보이는 것처럼 뒤의 내용은 주석처리가 되고 WHERE 문은 항상 참이 됩니다.

이러한 공격을 방지하기 위해 특수문자 및 SQL 예약어들을 필터링하거나 SQL 오류 메세지를 노출하지 않는 등의 방법을 취해야 합니다.

1-2. 프로그래밍 공통 - 고급

[메세지 큐(Message Queue)란?]

메세지 큐(Message Queue)란 Queue 자료구조를 이용하여 데이터(메세지)를 관리하는 시스템으로, 비동기 통신 프로토콜을 제공하여 메세지를 빠르게 주고 받을 수 있게 해준다. 메세지 큐에서는 Producer(생산자)가 Message를 Queue에 넣어두면, Consumer가 Message를 가져와 처리하게 된다. 메세지 큐에는 Kafka, Rabbit MQ, AMPQ 등이 있다.

[Docker(도커)와 Kubernates(쿠버네티스)]

Docker는 컨테이너 기반의 가상화 기술입니다. 기존에는 하드웨어를 가상화하였기 때문에 Host OS 위에 Guest OS를 설치해야 했습니다. 하지만 이러한 방식은 상당히 무겁고 느려 한계가 많이 있었습니다.

그래서 이를 극복하고자 프로세스를 격리시킨 컨테이너를 통해 가상화를 하는 Docker(도커)와 같은 기술들이 등장하게 되었고, 도커를 통해 구동되는 컨테이너를 관리하기 위한 Kubernates(쿠버네티스)가 등장하게 되었습니다.

[Docker(도커)의 장/단점]

장점
쉽고 빠른 실행 환경 구축
하드웨어 자원 절감
Docker Hub와 같은 공유 환경 제공
단점
개발 초기의 오버헤드
Linux 친화적

[TDD(Test-Driven Development)]

TDD(Test-Driven Development)는 매우 짧은 개발 사이클의 반복에 의존하는 개발 프로세스로, 개발자는 우선 요구되는 기능에 대한 테스트케이스를 작성하고, 그에 맞는 코드를 작성하여 테스트를 통과한 후에 상황에 맞게 리팩토링하는 테스트 주도 개발 방식을 의미합니다.

개발자는 테스트를 작성하기 위해 해당 기능의 요구사항을 확실히 이해해야 하기 때문에 개발 전에 요구사항에 집중할 수 있도록 도와주지만 테스트를 위한 진입 장벽과 작성해야 하는 코드의 증가는 단점으로 뽑힙니다.

[DDD(Domain-Driven Design)]

DDD(Domain-Driven Design)는 실세계에서 사건이 발생하는 집합인 Domain(도메인)을 중심으로 설계하는 방법입니다. 옷 쇼핑몰을 예로 들면 손님들이 주문하는 도메인, 점주들이 관리하는 도메인 등이 있을 수 있습니다. 이러한 도메인들이 서로 상호작용하며 설계하는 것이 도메인 주도 설계입니다. 도메인 주도 설계에서 도메인은 각각 분리되어 있는데, 이러한 관점에서 MSA(MicroService Architecture)를 적용하면 용이한 설계를 할 수 있다. DDD에서는 같은 객체들이 존재할 수 있는데, 예를 들어 옷 구매자의 입장에서는 (name, price)와 같은 객체 정보를 담지만, 판매자의 입장에서는(madeTie, size, madeCountry) 등이 있을 수 있습니다. 즉, 문맥에 따라 객체의 역할이 바뀔 수 있는 것이 DDD입니다.

[MSA란?]

MSA(Microservice Architecture)는 모든 시스템의 구성요소가 한 프로젝트에 통합되어 있는 Monolithic Architecture(모놀리식 아키텍쳐)의 한계점을 극복하고자 등장하게 되었습니다. MSA는 1개의 시스템을 독립접으로 배포가능한 각각의 서비스로 분할합니다. 각각의 서비스는 API를 통해 데이터를 주고받으며 1개의 큰 서비스를 구성합니다.

- 장점 일부 서비스에 장애가 발생하여도 전체 서비스에 장애가 발생하지 않는다. 각각의 서비스들은 서로 다른 언어와 프레임워크로 구성될 수 있다. 서비스의 확장이 용이하다.
- 단점 서비스가 분리되어 있어, 테스팅이나 트랜잭션 처리 등이 어렵다. 서비스 간에 API로 통신하기 때문에 그에 대한 비용이 발생한다. 서비스 간의 호출이 연속적이기 때문에 디버깅 및 에러 트레이싱이 어렵다.

2. 자료구조

[자료구조와 알고리즘]

자료구조는 데이터를 원하는 규칙 또는 목적에 맞게 저장하기 위한 구조이고, 알고리즘이란 자료구조에 쌓인 데이터를 활용해 어떠한 문제를 해결하기 위한 여러 동작들의 모임입니다.

[스택, 큐, 트리, 힙 구조 설명]

- 스택:
  세로로 된 바구니와 같은 구조로 먼저 넣게 되는 자료가 마지막으로 나오게 되는 First-In Last-Out(FILO) 구조이다.
- 큐:
  가로로 된 통과 같은 구조로 먼저 넣게 되는 자료가 가장 먼저 나오는 First-In First-Out(FIFO) 구조이다.
- 트리:
  정점과 간선을 이용해 사이클을 이루지 않도록 구성한 Graph의 특수한 형태로, 계층이 있는 데이터를 표현하기에 적합하다.
- 힙:
  최댓값 또는 최솟값을 찾아내는 연산을 쉽게 하기 위해 고안된 구조로, 각 노드의 키값이 자식의 키값보다 작지 않거나(최대힙) 그 자식의 키값보다 크지 않은(최소힙) 완전이진트리이다.

[우선순위 큐와 내부 구조 및 시간복잡도]

우선순위큐는 가장 우선순위가 높은 데이터를 먼저 꺼내기 위해 고안된 자료구조입니다. 우선순위 큐를 구현하기 위해서 일반적으로 힙을 사용합니다.
힙은 완전이진트리를 통해서 구현되었기 때문에 우선순위 큐의 시간복잡도는 O(logn)입니다.

[해시 테이블와 해시 테이블의 시간 복잡도]

해시 테이블은 (Key, Value)로 데이터를 저장하는 자료구조 중 하나로 빠른 데이터 검색이 필요할 때 유용합니다.
해시 테이블은 Key값에 해시함수를 적용해 고유한 index를 생성하여 그 index에 저장된 값을 꺼내오는 구조입니다.

해시 테이블은 고유한 index로 값을 조회하기 때문에 평균적으로 O(1)의 시간복잡도를 갖습니다. 하지만 해시의 index값이 충돌이 발생한 경우 충돌된 index값에 대해 연결된 데이터들을 조회하여 원하는 값을 조회하기 때문에 O(N)까지 증가할 수 있습니다.

[LinkedList와 ArrayList 차이]

ArrayList는 데이터들이 순서대로 늘어선 배열의 형식을 취하고 있지만, LinkedList는 자료의 주소값으로 서로 연결된 형식을 가지고 있습니다. 이러한 구조에 의해 둘은 각각의 장단점을 가지고 있습니다.

- ArrayList 원하는 데이터에 무작위로 접근할 수 있다. 리스트의 크기가 제한되어 있으며, 리스트의 크기를 재조정하는 것은 많은 연산이 필요하다. 데이터의 추가/삭제를 위해서는 임시 배열을 생성하여 복제하고 있어 시간이 오래 걸린다.
- LinkedList 리스트의 크기에 영향 없이 데이터를 추가할 수 있다. 데이터를 추가하기 위해 새로운 노드를 생성하여 연결하므로 추가/삭제 연산이 빠르다. 무작위 접근이 불가능하며, 순차 접근만이 가능하다.

[큐와 스택의 구현]

- 큐(Queue):
  Array로 구현하면 poll 연산 이후 객체를 앞당기는 작업이 필요하다. 하지만 List로 구현하면 객체 1개만 제거하면 되므로 삽입 및 삭제가 용이한 LinkedList로 구현하는 것이 좋다.
- 스택(Stack):
  List로 구현하면 객체를 제거하는 작업이 필요하다. 하지만 Array로 구현하면 삭제할 필요 없이 index를 줄이고 초기화만 하면 되므로, Array로 구현하는 것이 좋다.

[AVL 트리]

AVL 트리란 한 쪽으로 값이 치우치는 이진 균형 트리(Balanced Search Tree, BST)의 한계점을 보완하기 위해 만들어진 균형 잡힌 이진 트리입니다. AVL은 항상 좌/우로 데이터를 균형잡힌 상태로 유지하기 위해 추가적인 연산을 진행합니다.

[레드블랙 트리]

레드블랙 트리는 모든 노드를 빨간색 또는 검은색으로 색칠합니다. 그리고 연결된 노드들은 색이 중복되지 않도록 관리됩니다.

3. 알고리즘

버블소트

버블소트는 서로 인접한 두 원소를 비교하여 정렬하는 알고리즘입니다. 0번 인덱스부터 n-1번 인덱스까지 n번까지의 모든 인덱스를 비교하며 정렬합니다. 시간복잡도는 O(n2) 입니다.

힙소트

힙소트는 주어진 데이터를 힙 자료구조로 만들어 최대값 또는 최소값부터 하나씩 꺼내서 정렬하는 알고리즘입니다. 힙소트가 가장 유용한 경우는 전체를 정렬하는 것이 아니라 가장 큰 값 몇개만을 필요로 하는 경우입니다. 시간복잡도는 O(nlog2n) 입니다.

머지소트

머지소트는 주어진 배열을 크기가 1인 배열로 분할하고 합병하면서 정렬을 진행하는 분할/정복 알고리즘입니다. 시간복잡도는 O(nlog2n) 입니다.

퀵소트

퀵소트는 매우 빠른 정렬 속도를 자랑하는 분할 정복 알고리즘 중 하나로 합병정렬과 달리 리스트를 비균등하게 분할합니다. 피봇을 설정하고 피봇보다 큰값과 작은값으로 분할하여 정렬을 합니다. 시간복잡도는 O(nlog2n) 이며 리스트가 계속해서 불균등하게 나눠지는 경우 시간복잡도가 O(n2) 까지 나빠질 수 있습니다.

삽입소트

삽입정렬은 두 번째 값부터 시작하여 그 앞에 존재하는 원소들과 비교하여 삽입할 위치를 찾아 삽입하는 정렬 알고리즘입니다. 삽입 정렬의 평균 시간복잡도는 O(n2) 이며, 가장 빠른 경우 O(n) 까지 높아질 수 있습니다.

[정렬 알고리즘 시간복잡도 비교]

[동적 프로그래밍(Dynamic Programming)이란?]

동적 프로그래밍(Dynamic Programming) 이란 주어진 문제를 풀기 위해서, 문제를 여러 개의 하위 문제(subproblem)로 나누어 푼 다음, 그것을 결합하여 해결하는 방식입니다. 동적 프로그래밍에서는 어떤 부분 문제가 다른 문제들을 해결하는데 사용될 수 있어, 답을 여러 번 계산하는 대신 한 번만 계산하고 그 결과를 재활용하는 메모이제이션(Memoization) 기법으로 속도를 향상 시킬 수 있습니다.

[동적 프로그래밍(Dynamic Programming)의 두 가지 조건]

동적 프로그래밍(Dynamic Programming)으로 문제를 해결하기 위해서는 주어진 문제가 다음의 조건을 만족해야 한다.

- Overlapping Subproblem(중복되는 부분문제): 주어진 문제는 같은 부분 문제가 여러번 재사용된다.
- Optimal Substructure(최적 부분구조): 새로운 부분 문제의 정답을 다른 부분 문제의 정답으로부터 구할 수 있다.

[재귀 알고리즘과 재귀의 시간 복잡도]

재귀 알고리즘이란 함수 내부에서 함수가 자기 자신을 또 다시 호출하여 문제를 해결하는 알고리즘입니다. 재귀 알고리즘은 자기가 계속해서 자신을 호출하므로 끝없이 반복되게 되므로 반복을 중단할 조건이 반드시 필요합니다.

팩토리얼을 계산하는 재귀 함수에서는 T(n) = T(n-1) + c (C는 n과 f(n-1)을 곱하는 비용)을 조회하고 점화식을 계산하면 아래와 같이 O(n)이 됨을 보일 수 있습니다.

T(n) = T(n-1) + c
= T(n-2) + 2c
= T(n-3) + 3c
= ……
= T(2) + (n-2)c
= T(1) + (n-1)c
≤ c + (n-1)c = c + cn - c = cn --> O(n)

[팩토리얼의 재귀/반복문 손코딩]

```
    private static int recursiveFactorial(int num) {
        if(num > 1) {
            return recursiveFactorial(num - 1) * num;
        }
        return 1;
    }

    private static int loopFactorial(int num) {
        int answer = 1;
        for (int i = 2; i <= num; i++) {
            answer *= i;
        }
        return answer;
    }
```

[피보나치 수열 재귀/반복문 손코딩]

```
    private static int recursiveFibonacci(int index) {
        if (index <= 2){
            return 1;
        }
        return recursiveFibonacci(index - 1) + recursiveFibonacci(index - 2);
    }
```

```
    private static int loopFibonacci(int index) {
        int answer = 1;
        int before = 1;
        int temp;
        for (int i = 2; i < index; i++) {
            temp = answer;
            answer += before;
            before = temp;
        }
        return answer;
    }
```

3. 알고리즘 - 고급

[n개의 배열에서 k(k<=n) 번째로 큰수를 찾는 알고리즘] 이러한 문제를 해결하기 위해 일반적으로 퀵정렬을 사용합니다.
하지만 퀵정렬을 사용하면 정렬이 불필요한 부분들을 정렬하면서 효율적이지 못하게 됩니다.
퀵선택 알고리즘은 퀵정렬을 한 후에 피봇과 K를 비교하여 아래와 같이 수행합니다.

pivot의 인덱스가 k와 같은 경우 : 그대로 그 인덱스의 값을 리턴하면 된다. pivot의 인덱스가 k보다 작은 경우 : pivot의 인덱스+1부터 마지막 인덱스까지 다시 Partition함수에 넘겨준다. pivot의 인덱스가 k보다 큰 경우 : 첫 번째 인덱스부터 pivot의 인덱스-1까지 다시 Partition함수에 넘겨준다. 퀵정렬 알고리즘과의 다른 점은 예를 들어 Pivot의 인덱스가 7이고 K가 5인 경우에, 피봇의 오른쪽 부분은 재귀 함수를 돌지 않아 한 쪽만으로 재귀를 진행하는 것입니다.

이러한 이유로 퀵선택 알고리즘의 시간복잡도는 n+n/2+4/n+....1=O(n) 입니다.

[허프만 코딩이란]

허프만 코딩은 문자의 빈도를 이용해 압축하는 방법으로 빈도가 높은 문자에 짧은 코드를 부여합니다. 허프만 코드는 접두부 코드와 최적 코드를 사용합니다.

접두부 코드: 문자에 부여된 코드가 다른 이진 코드의 접두부가 되지 않는 코드 최적코드: 인코딩된 메세지의 길이가 가장 짧은 코드

[특정 수 이하의 3과 5의 배수의 합 구하기 손코딩]

```
    private static int addMultipleOf3And5(int maxNum) {
        int div = maxNum / 3;
        int sum3 = (1 + div) * div * 3 / 2 ;
        div = maxNum / 5;
        int sum5 = (1 + div) * div * 5 / 2 ;
        div = maxNum / 15;
        int sum15 = (1 + div) * div * 15 / 2 ;
        return sum3 + sum5 - sum15;
    }
```

4. 네트워크

[웹 동작 방식]

사용자가 브라우저에 URL을 입력
브라우저는 DNS를 통해 서버의 진짜 주소를 찾음
HTTP 프로토콜을 사용하여 HTTP 요청 메세지를 생성함
TCP/IP 연결을 통해 HTTP요청이 서버로 전송됨
서버는 HTTP 프로토콜을 활용해 HTTP 응답 메세지를 생성함
TCP/IP 연결을 통해 요청한 컴퓨터로 전송
도착한 HTTP 응답 메세지는 웹페이지 데이터로 변환되고, 웹 브라우저에 의해 출력되어 사용자가 볼 수 있게 됨

[TCP와 UDP 차이]

TCP는 연결형 서비스로 3-way handshaking 과정을 통해 연결을 설정합니다.그렇기 때문에 높은 신뢰성을 보장하지만 속도가 비교적 느리다는 단점이 있습니다.
UDP는 비연결형 서비스로 3-way handshaking을 사용하지 않기 때문에 신뢰성이 떨어지는 단점이 있습니다. 하지만 수신 여부를 확인하지 않기 때문에 속도가 빠릅니다. TCP는 신뢰성이 중요한 파일 교환과 같은 경우에 쓰이고 UDP는 실시간성이 중요한 스트리밍에 자주 사용됩니다.

[GET과 POST 차이]

- GET은 데이터를 조회하기 위해 사용되는 방식으로 데이터를 헤더에 추가하여 전송하는 방식입니다. URL에 데이터가 노출되기 때문에 보안적으로 중요한 데이터를 포함해서는 안됩니다.
- POST는 데이터를 추가 또는 수정하기 위해 사용되는 방식으로 데이터를 바디에 추가하여 전송하는 방식입니다. 완전히 안전하다는 것은 아니지만 URL에 데이터가 노출되지 않아 GET보다는 안전합니다.

[공인 IP와 사설 IP 차이]

- 공인 IP
  전세계에서 유일한 IP로 ISP(인터넷 서비스 공급자)가 제공하는 IP주소
  외부에 공개되어 있기 때문에 인터넷에 연결된 다른 장비로부터 접근이 가능하다.
  그에 따라 방화벽 등과 같은 보안 설정을 해주어야 한다.
- 사설 IP
  어떤 네트워크 안에서 사용되는 IP주소
  IPV4의 부족으로 인해 모든 네트워크가 공인 IP를 사용하는 것이 불가능하기 때문에 네트워크 안에서 라우터를 통해 할당받는 가상의 주소이다.
  별도의 설정 없이는 외부에서 접근이 불가능하다.

[웹 접근성의 국제표준]

웹 접근성을 높이기 위해 고안된 웹 표준은 웹에서 표준적으로 사용되는 기술이나 규칙을 의미합니다. 웹 표준을 정하기 위하 W3C(World Wide Web Consortium)이 설립되었으며 웹 표준으로 구조 언어인 HTML, 표현 언어인 CSS, 동작 언어인 Script를 지정하였습니다.

[OSI 7계층]

- 7 계층(응용 계층): 사용자와 직접 상호작용하는 응용 프로그램들이 포함된 계층
- 6 계층(표현 계층): 데이터의 형식(Format)을 정의하는 계층
- 5 계층(세션 계층): 컴퓨터끼리 통신을 하기 위해 세션을 만드는 계층
- 4 계층(전송 계층): 최종 수신 프로세스로 데이터의 전송을 담당하는 계층
- 3 계층(네트워크 계층): 패킷을 목적지까지 가장 빠른 길로 전송하기 위한 계층
- 2 계층(데이터링크 계층): 데이터의 물리적인 전송과 에러 검출, 흐름 제어를 담당하는 계층
- 1 계층(물리 계층): 데이터를 전기 신호로 바꾸어주는 계층

[HTTP 프로토콜이란?]

HTTP(Hyper Text Transfer Protocal)이란 서버/클라이언트 모델을 따라 데이터를 주고 받기 위한 프로토콜입니다. HTTP는 애플리케이션 레벨의 프로토콜로 TCP/IP 위에서 작동합니다. HTTP는 상태를 가지고 있지 않는 Stateless 프로토콜이며 Method, Path, Version, Headers, Body 등으로 구성됩니다.

[HTTP vs HTTPS]

HTTP는 평문 데이터를 전송하는 프로토콜이기 때문에, HTTP로 비밀번호나 주민번호 등을 주고 받으면 제3자에 의해 조회될 수 있습니다. 이러한 문제를 해결하기 위해 HTTP에 암호화가 추가된 프로토콜이 HTTPS입니다. HTTPS에는 대칭키 암호화와 비대칭키 암호화가 모두 사용됩니다. 비대칭키 암/복호화는 비용이 매우 크기 때문에 서버와 클라이언트가 주고받는 모든 메세지를 비대칭키로 암호화하면 오버헤드가 발생할 수 있습니다. 그래서 서버와 클라이언트가 최초 1회로 서로 대칭키를 공유하기 위한 과정에서 비대칭키 암호화를 사용하고, 이후에 메세지를 주고 받을 때에는 대칭키 암호화를 사용합니다. 이러한 과정을 정리하면 다음과 같습니다.

클라이언트(브라우저)가 서버로 최초 연결 시도를 함
서버는 공개키(엄밀히는 인증서)를 브라우저에게 넘겨줌
브라우저는 인증서의 유효성을 검사하고 세션키를 발급함
브라우저는 세션키를 보관하며 추가로 서버의 공개키로 세션키를 암호화하여 서버로 전송함
서버는 개인키로 암호화된 세션키를 복호화하여 세션키를 얻음
클라이언트와 서버는 동일한 세션키를 공유하므로 데이터를 전달할 때 세션키로 암호화/복호화를 진행함

공개키로 암호화된 메세지는 개인키를 가지고 있어야만 복호화가 가능하기 때문에, 서버(기업)을 제외한 누구도 원본 데이터를 얻을 수 없습니다.

[3 Way-Handshake]

3 Way-Handshake란 TCP 네트워크에서 통신을 하는 장치가 서로 연결이 잘 되었는지 확인하는 방법입니다. 송신자와 수신자는 총 3번에 걸쳐 데이터를 주고 받으며 통신이 가능한 상태임을 확인합니다.

[HTTP 1 vs HTTP 2]

HTTP1은 기본적으로 연결당 하나의 요청/응답을 처리하여 다음과 같은 문제를 가지고 있었습니다.

HOL(Head Of Line) Blocking (특정 응답 지연): 클라이언트의 요청과 서버의 응답이 동기화되어 지연 발생
RTT(Round Trip TIme) 증가 (양방향 지연): 패킷 왕복 시간의 지연 발생 헤더 크기의 비대: 쿠키 등과 같은 메타데이터에 의해 헤더가 비대해짐

그리고 HTTP2는 다음과 같은 기술을 사용하여 HTTP1의 성능 문제를 해결하였습니다.

- Multiplexed Streams: 하나의 커넥션으로 여러 개의 메세지를 동시에 주고 받을 수 있음

- Stream Prioritization: 요청온 리소스간의 의존관계를 설정하여 먼저 응답해야하는 리소스를 우선적으로 반환함
- Header Compression: 헤더 정보를 HPACK 압축 방식을 이용하여 압축 전송함
- Server Push: HTML문서 상에 필요한 리소스를 클라이언트 요청없이 보내줄 수 있음

5. 운영체제

[Byte Ordering이란]

Byte Ordering이란 데이터가 저장되는 순서를 의미합니다. Byte Ordering의 방식에는 빅엔디안(Big Endian)과 리틀엔디안(Little Endian)이 있습니다.

- Big Endian MSB가 가장 낮은 주소에 위치하는 저장 방식 네트워크에서 데이터를 전송할 때 주로 사용됨 가장 낮은 주소에 MSB가 저장되므로, offset=0인 Byte를 보면 양수/음수를 바로 파악할 수 있다.
- Little Endian MSB가 가장 높은 주소에 위치하는 방식 마이크로프로세서에서 주로 사용된다. 가장 낮은 주소에 부호값이 아닌 데이터가 먼저 오기 때문에, 바로 연산을 할 수 있다.

[메모리란]

메모리는 컴퓨터에서 작업을 수행하기 위해 처리 대상이나 결과 등을 저장하기 위한 공간입니다. 프로그램을 실행하기 위한 정보들은 메모리에 저장되어 처리됩니다.

[프로세스와 쓰레드의 차이]

- 프로세스
  메모리에 올라와 실행되고 있는 프로그램의 인스턴스
  특징
  1. 운영체제로부터 독립된 메모리 영역을 할당받는다. (다른 프로세스의 자원에 접근 X)
  2. 프로세스들은 독립적이기 때문에 통신하기 위해 IPC를 사용해야 한다.
  3. 프로세스는 최소 1개의 쓰레드(메인 쓰레드)를 가지고 있다.
- 쓰레드
  프로세스 내에서 할당받은 자원을 이용해 동작하는 실행 단위
  특징
  1. 쓰레드는 프로세스 내에서 Stack만 따로 할당 받고, Code, Data, Heap 영역은 공유한다. (Stack을 분리한 이유는 Stack에는 함수의 호출 정보가 저장되는데, Stack을 공유하면 LIFO 구조에 의해 실행 순서가 복잡해지기 때문에 실행 흐름을 원활하게 만들기 위함이다.)
  2. 쓰레드는 프로세스의 자원을 공유하기 때문에 다른 쓰레드에 의한 결과를 즉시 확인할 수 있다.
  3. 프로세스 내에 존재하며 프로세스가 할당받은 자원을 이용하여 실행된다.

[컨텍스트 스위칭(Context Switching)이란?]

Context Switching이란 인터럽트를 발생시켜 CPU에서 실행중인 프로세스를 중단하고, 다른 프로세스를 처리하기 위한 과정입니다. Context Switching는 현재 실행중인 프로세스의 상태(Context)를 먼저 저장하고, 다음 프로세스를 동작시켜 작업을 처리한 후에 이전에 저장된 프로세스의 상태를 다시 복구합니다. 여기서 인터럽트란 CPU가 프로세스를 실행하고 있을 때, 입출력 하드웨어 등의 장치나 예외상황이 발생하여 처리가 필요함을 CPU에게 알리는 것을 말합니다.

[멀티 프로세스 VS 멀티 쓰레드]

- 멀티 프로세스 하나의 프로그램을 여러 개의 프로세스로 구성하여 각 프로세스가 1개의 작업을 처리하도록 하는 것 특징
    1. 1개의 프로세스가 죽어도 자식 프로세스 이외의 다른 프로세스들은 계속 실행된다.
    2. Context Switching을 위한 오버헤드(캐시 초기화, 인터럽트 등)가 발생한다.
    3. 프로세스는 각각 독립적인 메모리를 할당받았기 때문에 통신하는 것이 어렵다.
- 멀티 쓰레드 하나의 프로그램을 여러 개의 쓰레드로 구성하여 각 쓰레드가 1개의 작업을 처리하도록 하는 것 특징
    1. 프로세스를 위해 자원을 할당하는 시스템콜이나 Context Switching의 오버헤드를 줄일 수 있다.
    2. 쓰레드는 메모리를 공유하기 때문에, 통신이 쉽고 자원을 효율적으로 사용할 수 있다.
    3. 하나의 쓰레드에 문제가 생기면 전체 프로세스가 영향을 받는다.
    4. 여러 쓰레드가 하나의 자원에 동시에 접근하는 경우 자원 공유(동기화)의 문제가 발생할 수 있다.

[데드락(DeadLock) 이란?]

데드락(DeadLock) 또는 교착상태란 한정된 자원을 여러 프로세스가 사용하고자 할 때 발생하는 상황으로, 프로레스가 자원을 얻기 위해 영구적으로 기다리는 상태입니다. 예를 들어 다음과 같은 상황에서 데드락이 발생할 수 있습니다.

자원 A를 가진 프로세스 P1과 자원 B를 가진 프로세스 P2가 있을 때, P1은 B를 필요로 하고 P2는 A를 필요로 한다면 두 프로세스 P1, P2는 서로 자원을 얻기위해 무한정 기다리게 됩니다.

[멀티 쓰레드 프로그래밍 작성 시 유의점]

멀티 쓰레드 프로그램을 개발한다면, 다수의 쓰레드가 공유 데이터에 동시에 접근하는 경우에 상호 배제를 제거해 교착 상태를 예방하고 동기화 기법을 통해 동시성 문제가 발생하지 않도록 발생하지 않도록 주의해야 합니다.

[세마포어(Semaphore) vs 뮤텍스(Mutex) 차이]

뮤텍스는 Locking 메커니즘으로 락을 걸은 쓰레드만이 임계 영역을 나갈때 락을 해제할 수 있습니다. 하지만 세마포어는 Signaling 메커니즘으로 락을 걸지 않은 쓰레드도 signal을 사용해 락을 해제할 수 있습니다. 세마포어의 카운트를 1로 설정하면 뮤텍스처럼 활용할 수 있습니다.

[CPU의 메모리 I/O 도중 생기는 병목 현상 해결 방법]

이러한 문제를 해결하기 위해 메모리를 계층화하여 병목현상을 해결하고 있습니다. 자주 접근하는 데이터의 경우에는 캐시에 저장하여 접근 속도를 향상 시킴으로써 부하를 줄이고 있습니다.

5. 운영체제 - 고급

[가상메모리와 페이지폴트]

가상메모리는 RAM의 부족한 용량을 보완하기 위해, 각 프로그램에 실제 메모리 주소가 아닌 가상의 메모리 주소를 할당하는 방식입니다. OS는 프로세스들의 내용(페이지) 중에서 덜 중요한 것들을 하드디스크에 옮겨 놓고, 관련 정보를 페이지 테이블에 기록합니다. CPU는 프로세스를 실행하면서 페이지 테이블을 통해 페이지를 조회하는데, 실제메모리에 원하는 페이지가 없는 상황이 발생할 수 있습니다(Valid bit를 통해 확인). 이것을 페이지 폴트라고 하는데 프로세스가 동작하면서 실제메모리에 필요한 데이터(페이지)가 없으면 가상메모리를 통

해서 해당 데이터를 가져오게 됩니다. 가상메모리는 하드디스크에 저장되어 있기 때문에, 페이지폴트가 발생하면 I/O에 의한 속도의 저하가 발생합니다.

[페이지 교체 알고리즘과 LRU(Least Recently Used)]

LRU(Least Recently Used)는 페이지를 교체하기 위한 알고리즘 중 하나입니다.

페이지를 교체하는 이유는 가상메모리를 통해 조회한 페이지는 다시 사용될 가능성이 높기 때문입니다. 페이지 교체를 위해서는 실제메모리에 존재하는 페이지를 가상메모리로 저장한 후에, 가상메모리에서 조회한 페이지를 실제메모리로 로드해야 됩니다. 그렇다면 어떤 실제메모리의 페이지를 가상메모리로 희생시킬 것이냐에 대한 문제가 발생하는데, 이때 사용하는 알고리즘 중 하나가 LRU(Least Recently Used) 알고리즘 입니다.

LRU 알고리즘은 실제메모리의 페이지들 중에서 가장 오랫동안 사용되지 않은 페이지를 선택하는 방식입니다. 그 외에도 먼저 적재된 페이지를 희생시키는 FIFO(First In First Out) 알고리즘이나 LRU 알고리즘을 응용하여 페이지에 Second-Change를 주는 LRU Approximation 등이 있습니다.

6. 데이터베이스

[인덱스(index)란?]

인덱스란 추가적인 쓰기 작업과 저장 공간을 활용하여 데이터베이스 테이블의 검색 속도를 향상시키기 위한 자료구조이다. 만약 우리가 책에서 원하는 내용을 찾는다고 하면, 책의 모든 페이지를 찾아 보는것은 오랜 시간이 걸린다. 그렇기 때문에 책의 저자들은 책의 맨 앞 또는 맨 뒤에 색인을 추가하는데, 데이터베이스의 index는 책의 색인과 같다.

데이터베이스에서도 테이블의 모든 데이터를 검색하면 시간이 오래 걸리기 때문에 데이터와 데이터의 위치를 포함한 자료구조를 생성하여 빠르게 조회할 수 있도록 돕고 있다.

만약 Index를 적용하지 않은 컬럼을 조회한다면, 전체를 탐색하는 Full Scan이 수행된다. Full Scan은 전체를 비교하여 탐색하기 때문에 처리 속도가 떨어진다.

[인덱스의 자료구조]

- 해시 테이블 컬럼의 값으로 생성된 해시를 기반으로 인덱스를 구현한다. 시간복잡도가 O(1)이라 검색이 매우 빠르다. 부등호(<, >)와 같은 연속적인 데이터를 위한 순차 검색이 불가능하다.
- B+Tree 자식 노드가 2개 이상인 B-Tree를 개선시킨 자료구조이다. BTree의 리프노드들을 LinkedList로 연결하여 순차 검색을 용이하게 하였다. 해시 테이블보다 나쁜 O(log2n) 의 시간복잡도를 갖지만 해시테이블보다 흔하게 사용된다.

[DB 정규화]

제1정규형:
모든 속성 값이 원자 값을 갖도록 분해한다.

제2정규형:
제1정규형을 만족하고, 기본키가 아닌 속성이 기본키에 완전 함수 종속이도록 분해한다.
(여기서 완전 함수 종속이란 기본키의 부분집합이 다른 값을 결정하지 않는 것을 의미한다.)

제3정규형:
제2정규형을 만족하고, 기본키가 아닌 속성이 기본키에 직접 종속(비이행적 종속)하도록 분해한다.

(여기서 이행적 종속이란 A->B->C가 성립하는 것으로, 이를 A,B와 B,C로 분해하는 것이 제3정규형이다.)

BCNF 정규형:
제3정규형을 만족하고, 함수 종속성 X->Y가 성립할 때 모든 결정자 X가 후보키가 되도록 분해한다.

[트랜잭션(Transaction)이란?]

트랜잭션이란 데이터베이스 작업의 단위로써 하나 이상의 쿼리를 처리할 때 동일한 Connection 객체를 공유하여 에러가 발생한 경우 모든 과정을 되돌리기 위한 방법입니다.

[트랜잭션의 ACID란?]

- 원자성(Atomicity):
  - 트랜잭션에 포함된 작업은 전부 수행되거나 전부 수행되지 않아야 한다.
- 일관성(Consistency):
  - 트랜잭션을 수행하기 전이나 후나 데이터베이스는 항상 일관된 상태를 유지해야 한다.
- 고립성(Isolation):
  - 수행 중인 트랜잭션에 다른 트랜잭션이 끼어들어 변경중인 데이터 값을 훼손하지 않아야한다.
- 지속성(Durability):
  - 수행을 성공적으로 완료한 트랜잭션은 변경한 데이터를 영구히 저장해야 한다.

[이상 현상의 종류]

삭제 이상: 튜플 삭제 시 같이 저장된 다른 정보까지 연쇄적으로 삭제되는 현상
삽입 이상: 튜플 삽입 시 특정 속성에 해당하는 값이 없어 NULL을 입력해야 하는 현상
수정 이상: 튜플 수정 시 중복된 데이터의 일부만 수정되어 일어나는 데이터 불일치 현상

[DB 락의 종류]

DB 락은 여러 개의 트랜잭션들이 하나의 데이터로 동시에 접근하려고 할 때 이를 제어해주는 도구이다.

공유락(LS, Shared Lock): 트랜잭션이 읽기를 할 때 사용하는 락, 데이터를 읽을 수 있지만 쓸 수 없음
베타락(LX, Exclusive Lock): 트랜잭션이 읽고 쓰기를 할 때 사용하는 락, 데이터를 읽고 쓸 수 있음

[RDBMS와 NoSQL 차이]

- RDBMS 2차원의 행과 열로 데이터의 관계를 관리하는 데이터베이스
  장점: 스키마에 맞추어 데이터를 관리하기 때문에 데이터의 정합성을 보장할 수 있다.
  단점: 시스템이 커질 수록 쿼리가 복잡해지고 성능이 저하되며, 수평적 확장이 어렵다.
- NoSQL RDBMS가 비대해짐에 따라 관계가 복잡해져, 이를 극복하기 위해 등장하게 된 데이터베이스
  장점: NOSQL은 스키마 없이 Key-Value 형태로 데이터를 관리하여 좀 더 자유롭게 데이터를 관리할 수 있다.
  단점: 중복된 데이터가 추가 가능하여, 이에 대한 관리가 필요하다.

6. 데이터베이스 - 고급

[힌트(Hint)란?]

힌트란 SQL을 튜닝하기 위한 지시구문입니다. 옵티마이저가 최적의 계획으로 SQL문을 처리하지 못하는 경우에 개발자가 직접 최적의 실행 계획을 제공하는 것입니다. 힌트는 아래와 같이 SELECT 다음에 작성할 수 있으며, INDEX, PARALLEL 등 다양한 힌트절이 있습니다.

사용가능한 힌트절: PARALLE, INDEX, FULL ...

SELECT /*+ [힌트절] */

[클러스터링 vs 리플리케이션]

- 리플리케이션
  여러 개의 DB를 권한에 따라 수직적인 구조(Master-Slave)로 구축하는 방식이다.
  비동기 방식으로 노드들 간의 데이터를 동기화한다.
  장점: 비동기 방식으로 데이터가 동기화되어 지연 시간이 거의 없다.
  단점: 노드들 간의 데이터가 동기화되지 않아 일관성있는 데이터를 얻지 못할 수 있다.
- 클러스터링
  여러 개의 DB를 수평적인 구조로 구축하여 Fail Over한 시스템을 구축하는 방식이다.
  동기 방식으로 노드들 간의 데이터를 동기화한다.
  장점: 1개의 노드가 죽어도 다른 노드가 살아 있어 시스템을 장애없이 운영할 수 있다.
  단점: 여러 노드들 간의 데이터를 동기화하는 시간이 필요하므로 Replciation에 비해 쓰기 성능이 떨어진다.

[데이터베이스 튜닝과 방법]

DB 튜닝은 테이터베이스의 구조나 데이터베이스 자체, 운영체제 등을 조정하여 데이터베이스 시스템의 성능을 향상시키는 작업을 의미합니다. 튜닝은 DB 설계 튜닝 -> DBMS 튜닝 > SQL 튜닝의 단계로 진행할 수 있습니다.

7. 개발 언어(Java 위주)

[Java의 장점과 단점]

- 장점
  JVM 위에서 동작하기 때문에 운영체제에 독립적이다.
  가비지컬렉터가 메모리를 관리해주기 때문에 편리하다.
- 단점
  JVM 위에서 동작하기 때문에 실행 속도가 상대적으로 느리다.
  다중 상속이나 타입에 업격하는 등 제약이 있는 것이 많다.

[Java가 다중 상속을 지원하지 않는 이유]

다중 상속을 지원하면 다이아몬드 문제가 발생할 수 있기 때문입니다. 예를 들어 Human 클래스에 있는 walk() 메소드를 Female 클래스와 Male 클래스가 모두 구현하였다고 할 때, Female과 Male 클래스를 다중 상속 받은 Person 클래스의 입장에서는 코드의 충돌이 생기기 때문입니다.

[오버라이딩(Overriding)과 오버로딩(Overloading)]

- 오버라이딩(Overriding): 상위 클래스가 가지고 있는 메소드를 하위 클래스에서 재정의하여 사용하는 기술
- 오버로딩(Overloading): 매개변수의 타입과 개수를 변경하면서 같은 이름의 메소드를 여러 개 사용하는 기술

```
public class Person {
    public void print() {
        System.out.println("나는 사람입니다");
    }
}

public class Student extends Person {
    // Overriding
    public void print() {
        System.out.println("나는 학생입니다.");
    }

    // Overloading
    public void print(String name) {
        System.out.println(name + "는 학생입니다");
    }
}
```

[클래스(Class), 객체(Object), 인스턴스(Instance)의 개념]

클래스(Class): 객체를 만들어내기 위한 설계도 혹은 틀
객체(Object): 설계도(클래스)를 기반으로 선언된 대상, 클래스의 인스턴스라고도 부름
인스턴스(Instance): 객체에 메모리가 할당되어 실제로 활용되는 실체

```
// 클래스
public class Person {
    private String name;
}

public class Main {

    public static void main(String[] args) {
        // 객체 = 클래스의 인스턴스
        Person person;

        // 인스턴스
        person = new Person();
    }

}
```

[싱글톤 패턴(Singleton Pattern) 구현 및 사용 이유]

```
public class Person {

    private static Person instance;

    public static Person getInstance() {
        if(instance == null){
            instance = new Person();
        }
        return instance;
    }
}
```

싱글톤 패턴은 단 하나의 인스턴스를 생성하여 사용하는 디자인패턴입니다.
싱글톤패턴은 아래의 경우에 사용합니다.

해당 인스턴스가 절대적으로 1개만 존재한다는 것을 보증하고 싶은 경우
동일한 인스턴스를 자주 생성해주어야 하는 경우(메모리 낭비의 방지)
하지만 이러한 싱글톤 패턴은 객체 지향 설계의 원칙에 적합하지 않으며, LifeCycle 제어가 힘들고, 멀티스레드 환경에서 여러 개의 객체가 생성되는 문제가 발생할 수 있습니다. 그러한 이유로 멀티스레드 환경이라면 static 앞에 synchronized 키워드를 붙여 동기화 작업을 추가해주어야 합니다.(당연히 성능이 저하됩니다).

[추상클래스와 인터페이스의 차이]

- 추상클래스
  단일 상속만이 가능하다.
  모든 접근 제어자를 사용할 수 있다.
  변수와 상수를 선언할 수 있다.
  추상 메소드와 일반 메소드를 선언할 수 있다.
- 인터페이스
  다중 구현이 가능하다.
  public 접근 제어자만 사용할 수 있다.
  상수만 선언할 수 있다.
  추상메소드만 선언할 수 있다.

[Java의 List, Set, Map 차이]

- List
  데이터를 순차적으로 저장한다.
  데이터의 중복을 허용한다. 데이터로 null을 허용한다.
- Set
  순서없이 Key로만 데이터를 저장한다.
  Key의 중복을 허용하지 않는다.
  Key로 null을 허용하지 않는다.
- Map
  순서없이 Key, Value로 데이터를 저장한다.
  Value는 중복을 허용하지만 Key의 중복을 허용하지 않는다.
  Key로 null을 허용하지 않는다.

[Java의 Vector와 ArrayList 차이]

- Vector
  동기화를 지원한다.
  속도가 느리지만 병렬 상황에서 안전하다.
  크기가 증가하는 경우, 2배 증가함(10 -> 20)
- ArrayList
  동기화를 지원하지 않는다.
  속도는 빠르지만 병렬 상황에서 안전하지 않다.
  크기가 증가하는 경우, 1.5배 증가함(10 -> 15)

[Java의 StringBuffer와 StringBuilder 차이]

- StringBuffer
  동기화를 지원한다.
  속도가 느리지만 병렬 상황에서 안전하다.
- StringBuilder
  동기화를 지원하지 않는다.
  속도는 빠르지만 병렬 상황에서 안전하지 않다.

[synchronized란?]

Java에서 지원하는 synchronized 키워드는 여러 쓰레드가 하나의 자원을 이용하고자 할 때, 한 스레드가 해당 자원을 사용중인 경우, 데이터에 접근할 수 없도록 막는 키워드입니다. synchronized 키워드를 이용하면 병렬 상황에서 자원의 접근을 안전하게 하지만, 자원을 이용하지 않는 쓰레드는 락에 의한 병목현상이 발생하게 됩니다.

메소드 synchronized: 한 시점에 하나의 쓰레드만이 해당 메소드를 실행할 수 있다.
변수 synchronized: 한시점에 하나의 쓰레드만이 해당 변수를 참조할 수 있다.

[Java8]

Java8에서는 함수형 프로그래밍을 위해 함수형 인터페이스와 람다와 함께 stream API가 추가되었고, Null-safe한 작업을 위한 Optional API, Date와 Time을 함께 처리하기 위한 LocalDateTime API 등이 추가되었습니다.

[try-with-resources]

try-with-resources란 Java7 버전에 추가된 기능으로, 리소스를 다 사용한 객체를 자동으로 반납(close)해줍니다. try-with-resources를 사용하면 코드가 try-finally보다 유연해지며, try-finally에서와 달리 누락없이 모든 자원을 반납할 수 있습니다.

try-with-resources를 통해 객체가 자동으로 반납되기 위해서는 AutoCloseable 인터페이스를 구현하고 있어야 합니다.

[Stream API의 장점과 단점]

- 장점
  코드를 간결하게 작성하여 가독성을 높일 수 있다.
  병렬스트림과 같은 기술을 이용하면 처리 속도를 많이 높일 수 있다.

- 단점
  잘못 사용하면 기존의 Java 방식보다 오히려 성능이 떨어질 수 있다.
  코드들이 추상화되어 있어 실수가 발생할 수 있다.

[람다(Lambda)와 람다(Lambda)의 사용법]

람다는 불필요한 코드를 줄이고, 가독성을 높이기 위한 익명 함수로써, 함수의 이름과 반환타입 없이 손쉽게 함수를 선언할 수 있습니다. 람다는 아래와 같이 괄호와 화살표로 표현할 수 있으며, 람다의 반환값은 함수형 인터페이스이므로, 이를 이용해주어야 합니다.

```
@FunctionalInterface
interface MyFunctionalInterface {
    String test();
}

public class Lambda {

    public static void main(String[] args) throws Throwable {
        String str = "This is My String";

        // Labmda Expression
        MyFunctionalInterface fi = () -> str.replaceAll("\\s+", "");
        System.out.println(fi.test());
    }

}
```

Java8의 Stream API를 이용하면서 람다를 사용해 본 경험이 있습니다.

```
public static String quiz2() {
    String result = Stream.of("TONY", "a", "hULK", "B", "america", "X", "nebula",
"Korea")
            .filter(w -> w.length() > 1)
            .map(String::toUpperCase)
            .map(w -> w.substring(0, 1))
            .collect(Collectors.joining(" "));

    return result;
}
```

[Java의 동작 과정]

Java 소스 파일을 javac로 컴파일하여 class파일(Java 바이트 코드)을 생성함
클래스로더가 컴파일된 Java 바이트 코드를 런타임 데이터 영역(Runtime Data Areas)로 로드함
실행 엔진(Execution Engine)이 자바 바이트코드를 실행함

[JVM의 구조]

JVM의 구조 중 메모리 구조는 다음과 같이 구성됩니다.

- Method Area(메소드 영역): 클래스 변수의 이름, 타입, 접근 제어자 등과 같은 클래스와 관련된 정보를 저장한다. 그 외에도 static 변수, 인터페이스 등이 저장된다.
- Heap Area(힙 영역): new를 통해 생성된 객체와 배열의 인스턴스를 저장하는 곳이다. 가비지 컬렉터는 힙 영역을 청소하며 메모리를 확보한다.
- Stack Area(스택 영역): 메소드가 실행되면 스택 영역에 메소드에 대한 영역이 1개 생긴다. 이 영역에 지역변수, 매개변수, 리턴값 등이 저장된다.
- PC register(PC 레지스터): 현재 쓰레드가 실행되는 부분의 주소와 명령을 저장한다.(CPU의 레지스터와 다르다.)
- Native Method Stack(네이티브 메소드 스택): 자바 외의 언어(C, C++ 등)로 작성된 코드를 위한 메모리 영역이다. JNI를 통해 사용된다.

[가비지 컬렉터(Garbage Collector)란?]

'더이상 참조되지 않는 메모리'인 가비지를 청소해주는 JVM의 실행 엔진의 한 요소입니다. JVM은 new와 같은 연산에 의해 새롭게 생성된 객체들 중에서 더이상 참조되지 않는 객체를 정리해줍니다. 가비지 컬렉터는 Heap 영역을 위주로 탐색하며 메모리를 정리해줍니다.

[가비지 컬렉션(Garbage Collection)의 과정]

가비지 컬렉션(GC)은 메모리를 정리하는 과정입니다. 그렇기 때문에 일반적으로 메모리의 사용을 중단한 채로 진행이 되어야 합니다. JVM은 GC를 실행하기 위해 애플리케이션의 실행을 멈추는 stop-the-world를 먼저 실행하게 됩니다. stop-the-world를 실행하면 GC를 실행하는 쓰레드를 제외한 모든 쓰레드가 작업을 멈춥니다. 그리고 GC가 끝나면 다시 작업을 재개합니다. GC의 작업은 Young 영역에 대한 Minor GC와 Old 영역에 대한 Major GC로 구분됩니다.

- Young 영역: 새롭게 생성한 객체들이 위치한다. 대부분의 객체는 금방 접근 불가능한 상태가 되기 때문에, 많은 객체가 Young 영역에 생성되었다가 사라진다.
  Old 영역: Young 영역에서 계속 사용되어 살아남은 객체가 복사되는 영역이다. Young 영역보다 크게 할당되며, 더 적은 GC가 발생한다.

Young 영역은 또 1개의 Eden 영역과 2개의 Survivor 영역으로 구성되는데, Young 영역에 대한 GC는 다음과 같이 작동한다.

새로운 객체가 Eden 영역에 생성됨
Eden 영역에 GC가 동작하고, 그 중에서 살아남은 객체가 Survivor0으로 이동함
2번의 동작이 반복되어 Survivor0이 꽉차게 됨
Survivor0 영역에 GC가 동작하고, 살아남은 객체들은 Survivor1으로 이동하고 Survivor0을 비우게 됨
(2개의 Survivor 영역 중 1개는 반드시 비어있어야 됨)
위의 동작들이 반복되어 특정 횟수만큼 살아남은 객체는 Old 영역으로 이동함

그리고 Old 영역이 가득차서 Survivor 영역에서 Old 영역으로 Promotion이 불가능할 때 Old 영역에 대한 GC(Major GC)가 실행됩니다.

[가비지 컬렉션(Garbage Collection) 알고리즘의 종류]

- Serial GC:
  mark-sweep-compact 알고리즘을 사용한다. Old영역에서 살아있는 객체를 식별(Mark)하고, 살아있는 객

체만을 남긴다.(Sweep) 그리고 난 후에 객체들을 앞부분부터 채워 객체가 존재하는 부분과 존재하지 않는 부분으로 나눈다.(Compaction)

- Parallel GC:
  기본적인 알고리즘은 Serial GC와 같지만 여러 쓰레드를 이용하여 GC를 처리한다.
- Parallel Old GC(Parallel Compacting GC):
  Serial GC의 Sweep 알고리즘 대신 Summary를 사용한다. Summary 단계는 앞서 GC를 수행한 영역에 대해서 별도로 살아있는 객체를 식별하며, Sweep보다 조금 더 복잡하다.
- Concurrent Mark & Sweep GC(이하 CMS):
  Initial Mark 단계에서는 살아 있는 객체를 찾는 것으로 끝낸다.(Stop-the-World 시간이 짧음) 그리고 찾은 객체에서 참조하는 객체를 Concurrent하게(여러 쓰레드가 동시에) 따라가는 Concurrent Mark 단계가 수행된다. 그 이후에 Stop-the-World가 실행되고 Concurrent하게 Remark)가 동작한다. 애플리케이션의 응답속도가 매우 중요할 때 사용한다.
- G1(Garbage First) GC:
  바둑판의 각 영역에 객체를 할당하고 GC를 실행한다. 위에서 설명한 Young영역과 Old영역에 대한 개념을 사용하지 않고, 객체를 할당한다.

[가비지 컬렉터(Garbage Collector) 작동의 문제를 진단하는 방법과 해결 하는 방법은?]

위의 가비지 컬렉션의 동작 과정을 보면 알겠지만 Survivor 영역 중 하나는 반드시 비어 있는 상태로 남아 있어야 합니다. 만약 두 Survivor 영역에 모두 데이터가 존재하거나, 두 영역 모두 사용량이 0이라면 시스템이 정상적인 상황이 아니라고 생각하면 됩니다.

추가적으로, GC에 대한 로그를 확인하여 옵션을 수정할 지 코드를 수정할 지 정해야 합니다.

[가비지 컬렉션(Garbage Collection)에 의한 시스템 중단 시간을 줄이는 방법]

옵션을 변경하여 GC의 성능을 높이기
young 영역과 old 영역의 힙 크기를 높여 GC의 빈도를 줄이는 것
객체의 할당과 promotion을 줄이는 것
위의 설명 중에서 힙 크기를 높여 GC의 빈도를 줄이는 해결책이 있습니다. 사실 논리적으로만 생각하면 힙의 크기를 높이면, GC 영역이 넓어져 실행시간이 길어지므로 무의미해진다고 생각이들 수 있습니다. 하지만 Minor GC의 실행시간은 힙의 크기보다는 살아남은 객체의 수에 의해 더욱 지연됩니다. 그렇기 때문에 short-lived 객체를 위한 young 영역의 크기를 높인다면 GC의 실행 시간과 호출 빈도를 모두 줄일 수 있습니다.(하지만 만약 애플리케이션에서 long-lived 객체를 많이 사용한다면, survivor영역으로 복제되는 객체가 많아져 GC에 의한 멈추는 시간이 증가할 수 있습니다.)

설정을 변경하여 GC의 성능을 높이기
애플리케이션을 중단시킨 후에 GC를 병렬로 동시에 진행시키는 것
애플리케이션과 GC작업을 동시에(concurrent) 진행시키는 것

개발자의 코드를 변경하여 GC의 성능을 높이기
Collection 등을 활용할 때 사용할 객체의 크기를 명시해주기
스트림을 바로 사용하기
변경 전: byte[] fileData = readFileToByteArray(new File("myfile.txt"));
변경 후: FileInputStream fis = new FileInputStream(fileName);
불변(Immutable) 객체 사용하기
위의 설명 중에서 불변의 객체를 사용하자는 해결책에 대해 설명하도록 하겠습니다.

```
public class MutableHolder {
  private Object value;
  public Object getValue() { return value; }
  public void setValue(Object o) { value = o; }
}

public class ImmutableHolder {
  private final Object value;
  public ImmutableHolder(Object o) { value = o; }
  public Object getValue() { return value; }
}
```

만약 위와 같은 MutableHolder가 있다라고 하면, MutableHolder는 계속 다른 값을 참조하여 생존하며 Old 영역으로 이동하게 됩니다. 하지만 Old 영역으로 가서도 참조하는 객체가 바뀌기 때문에, Minor GC를 수행할 때 Old 영역으로 와서 MutableHolder까지 검사하여 Young 영역을 정리해주어야 합니다. 즉, 검사해야 하는 범위가 늘어나게 되는 것입니다. 하지만, 불변의 객체 ImmutableHolder를 사용하게 된다면 상황이 달라집니다. 당연히 ImmutableHolder가 참조하는 값이 먼저 존재해야 ImmutableHolder가 존재할 수 있습니다. 그렇기 때문에 ImmutableHolder가 Old 영역으로 이동하게 되면 MutableHolder Minor GC에 대한 검사를 하지 않아도 되므로, 스캔의 범위를 줄여 성능을 높일 수 있는 것입니다.

8. 백엔드(Spring 위주)

[WAS와 WS의 차이]

WAS(Web Application Server)
비지니스 로직을 넣을 수 있음
Tomcat, PHP, ASP, .Net 등
WS(Web Server)
비지니스 로직을 넣을 수 없음
Nginx, Apache 등

[많은 트래픽이 발생한 경우 대처하는 방법]

스케일 업(Scale Up): 서버에 CPU나 RAM 등을 추가하여 서버의 하드웨어 스펙을 향상시키는 방법이다.
스케일 아웃(Scale Out): 서버를 여러 대 추가하여 시스템을 증가시키는 방법이다.

[CORS 란?]

CORS(Cross-Origin-Resource-Sharing)란 도메인이 다른 2개의 사이트가 데이터를 주고 받을 때 발생하는 문제입니다. 예를 들어 mangkyu.com에서 mang.com으로 데이터를 요청한다고 하면, 따로 설정을 해주지 않는 한 CORS 에러를 만나게 됩니다.

CORS가 생기게 된 이유는 서버 내에서 요청이 허락된 도메인에만 데이터를 주기 위해서인데, 요청을 허락하기 위해서는 Access-Control-Alow-Origin: {도메인} 과 같은 내용을 Response의 헤더에 추가해주어야 합니다다. 만약 도메인을 *으로 설정하면 모든 도메인에 대해 요청을 허락할 수 있습니다. 그 외에도 Access-Control-Allow-Methods, Access-Control-Max-Age 등을 설정해줄 수 있습니다.

Access-Control-Allow-Orgin : 요청을 보내는 페이지의 출처 [*, 도메인]
Access-Control-Allow-Methods : 요청을 허용하는 메소드. Default : GET, POST
Access-Control-Max-Age : 클라이언트에서 preflight 요청 (서버의 응답 가능여부에 대한 확인) 결과를 저장할 시간
Access-Control-Allow-Headers : 요청을 허용하는 헤더

[아파치는 멀티 프로세스인가 멀티 쓰레드인가?]

아파치는 기본적으로 멀티 프로세스로 구현되어 있다. 하지만 설정에 따라 멀티 쓰레드를 같이 운용할 수 있다.

[톰캣은 멀티 프로세스인가 멀티 쓰레드인가?]

톰캣은 요청을 처리하기 위한 쓰레드 풀을 관리하고 있다. 그리고 요청이 오면 해당 쓰레드 풀에서 쓰레드를 꺼내 요청을 처리하도록 한다.

[디자인 패턴]

- 생성 패턴
  팩토리 패턴: 객체를 생성하기 위한 디자인 패턴
  추상 팩토리 패턴: 객체를 생성하는 팩토리를 생성하기 위한 디자인 패턴
  빌더 패턴: 상황에 따라 동적인 인자를 필요로 하는 객체를 생성하기 위한 디자인 패턴
  싱글톤 패턴: 객체를 1개만 생성하여 항상 참조가능하도록 고안된 디자인 패턴
- 구조 패턴
  어댑터 패턴: 호환성이 맞지 않는 두 클래스를 연결하여 사용하기 위한 디자인 패턴
  프록시 패턴: 어떤 객체에 접근 제어를 위해 대리인을 사용하는 디자인 패턴
  데코레이터 패턴: 어떤 객체에 새로운 기능 추가를 위해 대리인을 사용하는 디자인 패턴
  퍼사드 패턴: 어떤 복합적인 기능에 대해 간략화된 인터페이스를 제공하는 디자인 패턴
- 행위 패턴
  전략 패턴: 상황에 따라 다른 전략을 사용하기 위한 디자인 패턴
  옵저버 패턴: 값을 관찰하여 빠르게 반영하기 위한 디자인 패턴
  커맨드 패턴: 실행될 기능을 캡슐화하여 재사용성이 높은 클래스를 설계하기 위한 디자인 패턴
  DatabaseController=> SingletonPattern을 사용하여 데이터베이스를 제어하는 하나의 인스턴스만을 생성

DatabasePool => ObjectPool Pattern을 사용하여 데이터베이스 객체를 미리 생성하여 Performance 향상

UnitFactory => FactoryPattern을 사용하여 객체 생성을 최적화 + Singleton Pattern을 사용하여 하나의 공장을 사용

BaseFrame => ObserverPattern을 사용하여 사용자의 정보가 생신되면 View의 값들도 갱신되게 함

PlayerInfo => StrategyPattern을 사용하여 상황에 따라 다른 스킬을 사용

[Servlet(서블릿)이란?]

서블릿이란 클라이언트의 요청을 처리하고, 그 결과를 반환하는 Servlet 클래스의 구현 규칙을 지킨 자바 웹 프로그래밍 기술입니다. Spring MVC에서 Controller로 이용되며, 사용자의 요청을 받아 처리한 후에 결과를 반환합니다.

[Spring 기초지식(DI, DL, IoC, AOP)]

DI(Dependency Injection): 한 객체에서 다른 객체를 필요로 하여 의존성을 갖게 하는 기술
DL(Dependency Look-up): 한 객체에서 필요로 하는 다른 객체를 찾아서 사용하는 기술
IoC(Inversion of Control): 직접 제어야하는 부분에 대한 권한을 프레임워크 등에 넘기는 기술
AOP(Aspect Oriented Programming): 공통의 관심 사항을 추출하여 원하는 곳에 적용하는 기술

[VO와 DTO, BO, DAO란?]

DAO(Data Access Object): DB에 접근하여 실제 데이터를 조회 또는 조작하는 클래스, Repository 또는 Mapper에 해당함
BO(Business Object): 여러 DAO를 활용해 비지니스 로직을 처리하는 클래스, Service에 해당함
DTO(Data Transfer Object): 데이터를 주고 받기 위해 사용하는 클래스
VO(Value Object): 실제 데이터만을 저장하는 클래스
아래의 예시에서는 완벽하지는 않지만 이해하기에는 충분한 3가지 개념의 코드를 보여주고 있습니다.

```
@Getter
// 데이터만 저장하는 VO
public class Board {

    private String title;
    private String contents;

}

// DB에 접근하여 데이터를 조회하는 DAO(여기서는 DB쪽 X)
public class BoardRepository {

    private List<Board> boardList = new ArrayList<>();

    public void add(Board board) {
        boardList.add(board);
    }

    public List<Board> findByTitleContaining(String title) {
        return boardList.stream()
                .filter(b -> b.getTitle().contains(title))
                .collect(Collectors.toList());
    }

}

// DAO를 가지고 비지니스 로직을 처리하는 BO
public class BoardService {

    private BoardRepository boardRepository;

    public void addBoard(Board board) {
        boardRepository.add(board);
    }

    public List<Board> searchBoard(String title) {
        return boardRepository.findByTitleContaining(title);
```

```
    }

}
```

[디스패처 서블릿(Dispatcher Servlet)이란?]

디스패처 서블릿이란 톰캣과 같은 서블릿 컨테이너를 통해 들어오는 모든 요청을 제일 앞에서 받는 프론트 컨트롤러입니다. 디스패처 서블릿은 공통된 작업을 처리한 후에, 적절한 세부 컨트롤러로 작업을 위임해줍니다. 그리고 각각의 세부 컨트롤러는 처리할 부분을 처리하고 반환할 view를 Dispatcher Servlet에 넘기게 됩니다.

[Spring에서의 싱글톤 패턴]

Java로 기본적인 싱글톤 패턴을 구현하고자 하면 다음과 같은 단점들이 발생한다.

private 생성자를 갖고 있어 상속이 불가능하다.
테스트하기 힘들다.
서버 환경에서는 싱글톤이 1개만 생성됨을 보장하지 못한다.
전역 상태를 만들 수 있기 때문에 바람직하지 못하다.

그래서 스프링은 컨테이너를 통해 직접 싱글톤 객체를 생성하고 관리하는데, 이를 통해 다음과 같은 장점을 얻을 수 있다.

static 메소드나 private 생성자 등을 사용하지 않아 객체지향적 개발을 할 수 있다.
테스트를 하기 편리하다.

[MVC 패턴이란?]

MVC(Model-View-Controller)패턴은 아키텍쳐를 설계하기 위한 디자인 패턴입니다.

MVC 패턴은 애플리케이션을 개발할 때 그 구성요소를 3가지로 나눕니다.

Model: 데이터를 저장하는 컴포넌트
View: 사용자 인터페이스(UI) 컴포넌트
Controller: 사용자의 요청을 처리하고 Model과 View를 중개하는 컴포넌트

[Spring MVC란?]

Spring MVC란 웹 애플리케이션 개발을 위한 MVC 패턴 기반의 웹 프레임워크입니다. Spring MVC는 애플리케이션의 구성요소를 Model, View, Controller로 분리합니다. 또한 Spring MVC는 아래와 같은 컴포넌트들로 구성됩니다.

Dispatcher Servlet: 클라이언트의 요청을 먼저 받아들이는 서블릿으로, 요청에 맞는 컨트롤러에게 요청을 전달함
Handler Mapping: 해당 요청이 어떤 컨트롤러에게 온 요청인지 검사함
Controller: 클라이언트의 요청을 받아 처리하여 결과를 디스패처 서블릿에게 전달함
ViewResolver: View의 이름을 통해 알맞은 View를 찾음
View: 사용자에게 보여질 UI 화면

[Spring MVC 작동 원리]

클라이언트는 URL을 통해 요청을 전송한다.
디스패처 서블릿은 핸들러 매핑을 통해 해당 요청이 어느 컨트롤러에게 온 요청인지 찾는다.
디스패처 서블릿은 핸들러 어댑터에게 요청의 전달을 맡긴다.
핸들러 어댑터는 해당 컨트롤러에 요청을 전달한다.
컨트롤러는 비지니스 로직을 처리한 후에 반환할 뷰의 이름을 반환한다.
디스패처 서블릿은 뷰 리졸버를 통해 반환할 뷰를 찾는다.
디스패처 서블릿은 컨트롤러에서 뷰에 전달할 데이터를 추가한다.
데이터가 추가된 뷰를 반환한다.

[Spring MVC의 장점과 단점 그리고 SpringBoot]

- 장점
  의존성 주입을 통해 컴포넌트 간의 결합도를 낮출 수 있어 단위테스트가 용이함
  제어의 역전을 통해 빈(객체)의 라이프싸이클에 관여하지 않고 개발에 집중할 수 있음
- 단점
  XML을 기반으로 하는 프로젝트 설정은 너무 많은 시간을 필요로 함
  톰캣과 같은 WAS를 별도로 설치해주어야 함
  해결책(Spring Boot)
  자동설정(AutoConfiguration)을 도입하여 Dispatcher Servlet 등과 같은 설정 시간을 줄여줌
  프로젝트의 의존성을 독립적으로 선택하지 않고 spring-boot-starter로 모아두어 외부 도구들을 사용하기 편리함
  내장 톰캣을 제공하여 별도의 WAS를 필요로 하지 않음

[Spring @Bean, @Configuration, @Component 어노테이션]

@Bean: 개발자가 직접 제어가 불가능한 외부 라이브러리 또는 설정을 위한 클래스를 Bean으로 등록할 때 사용
@Configuration: 1개 이상의 @Bean 메소드를 갖는 클래스의 경우에 반드시 명시해 주어야 함
@Component: 개발자가 직접 개발한 클래스를 Bean으로 등록하고자 하는 경우에 사용

[Spring Filter와 Interceptor의 차이]

8. 백엔드(Spring 위주) - 고급

[Spring Security 작동 원리]

사용자가 아이디 비밀번호로 로그인을 요청함 AuthenticationFilter에서
UsernamePasswordAuthenticationToken을 생성하여 AuthenticaionManager에게 전달
AuthenticaionManager는 등록된 AuthenticaionProvider(들)을 조회하여 인증을 요구함
AuthenticaionProvider는 UserDetailsService를 통해 입력받은 아이디에 대한 사용자 정보를 DB에서 조회함
입력받은 비밀번호를 암호화하여 DB의 비밀번호화 매칭되는 경우 인증이 성공된
UsernameAuthenticationToken을 생성하여 AuthenticaionManager로 반환함
AuthenticaionManager는 UsernameAuthenticaionToken을 AuthenticaionFilter로 전달함
AuthenticationFilter는 전달받은 UsernameAuthenticationToken을 LoginSuccessHandler로 전송하고, 토큰을 response의 헤더에 추가하여 반환함

[Spring AOP의 작동 원리(JDK 동적 프록시와 CGLib 프록시)]

다이내믹 프록시 객체의 생성 요청
포인트컷을 통해 부가 기능 대상 여부 확인
어드바이스로 부가 기능 적용
실제 기능 처리

Spring AOP는 기본적으로 JDK 동적 프록시를 이용하며, 자바의 인터페이스와 스프링 컨테이너 외에는 특별한 기술이나 환경을 필요로 하지 않습니다. 하지만 이러한 프록시 방법은 다음과 같은 2가지 단점을 가지고 있습니다.

프록시 적용을 위해 반드시 인터페이스를 생성해야 함
구체 클래스로는 빈을 주입받을 수 없고, 반드시 인터페이스로만 주입받아야 함

그래서 이러한 문제를 해결하고자 클래스 상속을 기반으로 프록시를 구현하는 CGLib 프록시가 등장하게 되었습니다. CGLib 프록시는 상속을 기반으로 구현되므로 final 클래스나 final 메소드면 프록시 생성이 불가능한 제약이 있습니다.

[AspectJ AOP의 작동 원리]

Spring AOP가 아닌 또 다른 강력한 AOP 프레임워크 중 하나인 AspectJ는 프록시를 이용하지 않았습니다. 대신 AspectJ는 타깃 클래스 파일의 바이트 코드를 조작하여 부가기능을 직접 넣어주는 방법(위빙)을 사용합니다. 그래서 우리가 만든 코드에서는 부가 기능이 분리되어 있지만 바이트 코드에서는 핵심 기능과 부가 기능이 섞여 있는 구조입니다. AspectJ가 프록시를 사용하지 않고 어려운 복잡한 바이트 조작 방법을 사용하는 이유는 크게 2가지가 있습니다.

바이트 코드를 조작하면 Spring과 같은 컨테이너의 도움이 필요 없기 때문이다.
프록시 방식보다 훨씬 강력하고 유연한 AOP를 제공할 수 있다.

바이트 코드를 끼워넣는 시점은 컴파일 시점과 클래스 파일이 JVM 메모리로 올라가는 시점 총 2가지가 있습니다.

[Spring WebFlux란?]

Spring WebFlux란 Blocking+동기 방식으로 동작하는 Spring MVC의 한계점을 극복하기 위해 Spring5에 처음 등장하게 되었습니다. 기존의 Spring MVC에서는 HTTP 요청들을 큐에 넣어두고, 멀티쓰레드를 기반으로 동작하고 있습니다. 하지만 이러한 방식은 응답성이 상대적으로 떨어지기 때문에 비동기적으로 요청을 처리하기 위한 방법이 필요하게 되었고, 리액티브 프로그래밍을 통해 비동기 데이터 스트림으로 Non-Blocking 애플리케이션을 개발하기 위한 Spring WebFlux 프레임워크가 등장하게 되었습니다.

[CDN(Content Delivery Network)란?]

CDN(Content Delivery Network)는 물리적으로 떨어져 있는 사용자에게 컨텐츠를 더 빠르게 제공하기 위해 고안된 기술입니다. 만약 우리나라에 있는 사람이 미국에 있는 서버로부터 이미지나 파일 등을 다운받으려고 하면 시간이 오래 걸립니다. 따라서 느린 응답속도와 다운로드 시간을 극복하기 위해 서버를 분산시켜 캐싱해두고, 빠르게 다운받을 수 있게 합니다.

CDN은 콘텐츠에 대한 요청이 발생하면 사용자와 가장 가까운 위치에 존재하는 서버로 매핑시켜, 요청된 파일의 캐싱된(사전 저장된) 버전으로 요청을 처리합니다. 서버가 파일을 찾는 데 실패하는 경우 CDN 플랫폼의 다른 서버에서 콘텐츠를 찾은 다음 엔드유저에게 응답을 전송합니다.

1. 기술 외 공통 면접 질문

[자기소개 및 지원동기]
[주요 기술스택 및 역할]

[자기소개서 기반의 프로젝트 질문]
어떤 역할을 담당하였는가?
서버는 어떻게 구성하였는가?
어떤 언어와 프레임워크, 라이브러리?
몇명의 사용자, 어느 정도의 트래픽
해당 기술 스택을 사용한 이유
구조에 대한 설명 및 설계 관련 내용
어떠한 알고리즘, 자료구조 혹은 디자인패턴을 사용하였는가?
개선점은 무엇이였나?
어떻게 프로젝트를 관리했고(스케줄, 소통 등) 어떻게 기여했는가?(문서화, 구조 설계, 개발 등)
자신의 프로젝트에서 다양한 상황을 가정해보자. (이런 경우 혹은 저런 경우엔 어떻게 하실 건가요? 같은 질문을 자주 함)

[프로젝트에 있어서 가장 중요한 것은 무엇이라 생각하나요?]
[테스트 코드의 장점 및 어떤 생각을 하면서 작성하는가?]
[코드 리뷰는 어떻게 진행하는가?]
[이미 진행했던 코드를 개선한 경험]
[새로운 정보의 습득 방법]
[학부 시절 기억나는 과목]
[인상 깊었던 개발자 세미나]
[해당 개발자가 되려는 이유(웹 or 앱 등)]
[어떤 개발자가 되고 싶은가?]
[개발을 좋아하는가?]
블로그 or 깃 허브로 보여주면 좋습니다. [협업 스타일]
[협업 할 때 일을 나누는 방법]
[협업 시 생긴 갈등의 해결 방법]
[AWS VPC 설정 여부]
[최근에 읽은 개발 서적은?]
[최근에 관심을 갖게 된 기술은?]
[개발자가 되기 위해 어떻게 공부하는가?]
[성격의 장/단점과 어떻게 평가받는지(단점을 고치려고 노력한 부분도)]
[주로 이용하는 페이지 or 동영상]
[실천하고 있는 열정 프로젝트가 있는가?]
[IT 비전공자에게 효과적인 소통을 하고 있는가?]
[업무 스트레스를 해결하는 방법]
[가장 힘들었던 일과 해결한 경험]