네이버 면접준비

1. 자바 및 언어

1.1.1자바란? :

1.1.2자바의 장점

1.1.3JAVA 바이트 코드와 다른 언어와의 차이는 무엇인가 ?

자바는 가상버신 바이트 코드로 컴파일 하며 실행시키려면 가상머신이 필요 하지만 c 나 cpp의 경우 각머신에 맞는 기계어로 컴파일한다. 예를들어 C의 경우 Linux에서 컴파일 하면 Linux 기계어로 Windows에서 컴파일하면 Windows 기계어로 되지만 자바는 JVM이 중간다리 역할을 해주기 때문에 운영체제가 무엇인지 판단한 뒤 운영체제의 기계어로 다시 번역해준다.

1.1.4 JVM

자바 바이트 코드를 os에 맞게 해석해주는 가상 머신

1.1.5가비지컬렉션

불필요한 메모리를 알아서 정리해주는 것 자바에서는 명시적으로 불필요한 데이터를 표현하기위해 일반적으로 null을 선언

1.1.6 자바 thread : synchronized 키워드를 제공해 스레드간 동기화를 시켜 data의 thread-safe를 가능케 합니다.

기본적으로 자바에서 제공하는 collection들의 경우 thread safe하지 않지만 Syncronized 명령어를 사용해서 thread-safe하게 설계도 가능하다.

Syncronized 명령어에서 변수에서 사용할때와 메서드에 사용할때의 차이 ??

1.1.7 제네릭

제네릭은 클래스 내부에서 사용할 데이터 타입을 외부에서 지정하는 기법을 의미

Ex ) LinkedList<T> 로 되어있으면 클래스를 정의할 때 사용하지 않고 들어오는 객체로 판단을 할수 있게됨

1.1.8 자바 static

클래스 로더가 클래스를 로딩해서 메소드 메모리 영역에 적재할 때 클래스 별로 관리된다. 따라서 클래스의 로딩이 끝나는 즉시 사용할 수 있다.

Static 변수로 생성 된 정적 멤버들은 Heap 영역이 아닌 Static 영역으로 할당된다.

장점으로는 모든 객체가 공유하여 하나의 멤버를 어디서든지 참조 할 수 있는 장점을 가지지만 가비지 컬렉션의 대상이 되진 않음 즉 메모리가 할당된 채로 존재하여 프로그램의 종료시 까지 메모리가 할당된 채로 존재하게 됨

1.1.9 업캐스팅과 다운캐스팅

캐스팅은 타입을 변환하는 것을 의미 자바에서 상속관계에 있는 부모와 자식클래스간에는 서로간 형변환이 가능

업캐스팅 : 자바의 상속관계는 상위클래스의 모든 특성을 하위클래스가 받아서 사용할 수 있다. 즉 서브 클래스는 수퍼클래스 취급을 받을 수 있다 여기서 업캐스팅을 통해 객체가 슈퍼클래스 타입으로 형변환을 시킨다.

다운캐스팅은 업캐스팅과 반대의 의미로 생각하면 된다. 슈퍼클래스 객체가 서브 클래스 객체로 다운 형변화를 일이키는 것 이는 자신의 특성을 잃어버린 서브 클래스의 객체를 다시 복구시켜주는 것

업캐스팅의 장점은 상위 클래스를 상속받은 모든 하위 클래스의 객체를 수용할 수 있다.

1.1.10 자바 컬렉션

여러 원소들을 담을 수 있는 자료구조를 뜻함 배열이 가장 기본적인 자료구조이다.

1.2 C# 관련

1.2.1 C#의 장점

완벽한 객체지향 언어

개발자가 사용 편리한 인터페이스 환경

메모리에 대한 사용자 부담이 적다(가비지 컬렉션 적용)

타입이 엄격 및 문법이 엄격

많은 문법의 확장

닷넷 플랫폼의 대표적 언어 닷넷의 모든 장점을 가지고 있다.

1.2.2 c# 컴파일 과정

1.3 파이썬 관련

파이썬은 컴파일 과정없이 인터프리터가 소스 코드를 한줄 씩 읽어 들여 곧바로 실행하는 스크립트 언어 즉 컴파일 과정이 필요하지 않아 실행 결과를 바로 확인하고 수정하면서 손쉽게 코드를 작성할 수 있음

1. 자료구조

2.1 리스트

2.1.1 ArrayList와 LinkkedList 차이

2.1.1.1 ArrayList

내부적으로 데이터를 배열에서 관리하며 데이터의 추가 삭제를 위해 임시배열을 생성해 데이터를 복사하는 방법을 사용 대량의 자료를 추가 삭제하는 경우 그만큼 데이터의 복사가 많이 일어나게 되어 성능 저하를 일으킬 수 있다. 그리고 각각의 인덱스를 가지고 있어 한번에 참조가 가능하기엔 유리

Get을 통해 인덱스 기반이기 때문에 O(1) 을가진다

삽입과 삭제는 삭제이후 다른 데이터를 복사해야하기 때문에 최악의 경우 N의 성능

2.1.1.2 LinkedList

노드와 링크를 이용하여 선형으로 구성한 리스트

Get에 N 추가 삭제에는 (1)의 성능을 보임

2.2 스택 큐

2.2.1 스택 : 선입 후출의 자료구조 리스트로 구현하는 것보다 배열로 사용하는 것이 더 좋다

2.2.2 큐 : 선입 선출의 자료구조로 리스트로 구현한느것이 배열로 사용하는 것이 더 좋다. 그 이유는 배열로 구현할경우 배열을 한칸씩 당겨줘야하는 연산이 필요하기 때문에 비효율적이다.

2.3 그래프

노드와 연결하는 간선을 하나로 모아 놓은 구조 그래프는 사이클이 존재할 수있고 방향과 무방향 그래프 모두 가능하다. 자체간선도 가능 루트 부모 자식의 관계가 없다.

2.4 트리

노드와 간선으로 으루어져 있고 사이클을 이루이 않는 구조 특징으로는 하나의 루트노드로 이루어져 있으며 루트노드는 0개 이상의 자식노드를 가지고 있다. 트리에는 사이클이 존재할 수 없다.

2.4.1 이진트리 : 각 노드가 최대 두개의 자식을 갖는 트리

리프노드들은 LinkedList로 연결되어 있고 데이토노드 크기는 인덱스 노드의 크기와 같이 않아도 된다.

AVL 트리 RedBlack Tree AVL 트리

2.4.1.1 Red-Black Tree : 자가 균형 이진트리로 특정한 규칙을 가진다

2.4.1.1.1 규칙

1. 모든 노드는 Red 이거나 Black 이다

2. 루트는 Black 이다

3. 모든 리프(NULL)은 Black 이다.

4.노드가 Red이면 그 노드의 자식은 Black 이다.

5. 각 노드로부터 그 노드의 자손인 리프로 가는 경로들은 모두 같은 수의 Black 노드를 포함한다.

2.4.1.1.2 삽입

Restructuring 부모의 형제가 Black 일때 수행

삽입될 노드의 부모와 부모의 부모 이를 오름차순으로 정렬하고 가운데 있는 값을 부모로 만들고 나머지 둘을 자식으로 만든다 부모로 올라간 것을 Black 으로 자식들을 Red로바꾼다

시간복잡도는 Restructuring 자체는 O(1)이지만 어떤 노드를 insertion 한 뒤 일어나므로 총수행시간은 logn 임

Recoloring : 부모의 형제가 Red일 때 수행

삽입은 기본 이진탐색 트리를 기본으로 하며 위 규칙을 위반할 경우 새로운 로직이 적용

Recoloring : 삽입된 노드의 부모의 형제 색깔이 Red인경우

Restructuring 삽입된 노드의 부모의 형제 색깔이 Black 인경우 NULL 인경우

현재 insrt된 노드와 부모 형제를 검정으로 만들어 준다. 이때 루트는 다시 검정으로 바뀜

But 큰트리의 서브트리 였다면 Double Red가 발생할 수 있습니다. 이로 인해 계속 Recolring 이 발생할 수 있음 한번의 Recoloring은 O(1) 수행되지만 최악의 경우 log(n)까지 수행이 될수 있음 즉 O(logn)이 소모되게됨

특징은 Root노드 까지 가면서 만나는 Black Node 개수는 같다

2.4.1.2 AVL 트리

2.4.2 B+Tree : 자식 노드가 2개 이상인 B-Tree를 개선시킨 자료구조 B+Tree는 모든 노드에 데이터를 저장했던 BTree와 다른 특성을 가지고 있으며 리프토드만 인덱스와 함께 데이터를 가지고 있고 나머지 노드 들은 데이터를 위한 인덱스 만을 갖는다.

2.4.3 Trie 구조 :

문자열을 저장하고 효율적으로 탐색하기 위한 트리 형태의 자료구조

사용목적은 문자열을 탐색하고자 할 때 빠르게 탐색하고 싶지만 각노드에서 자식들에 대한 포인터들을 배열을 모두 저장하고 있어야한다는 점에서 저장 공간의 크기가 크다는 단점도 있음 검색어 자동완성 사전에서 찾기 그리고 문자열 검사 같은 부분에서 사용할 수 있다.

시간복잡도는 제일 긴 문자열을 L 총 문자열들의 수를 M이라고 할 때 생성시 시간 복잡도는 M\*L 모든 문자열들을 넣어야하니 M개의 대하여 트라이 자료구조에 넣는건 가장 긴 문자열 L만큼 걸린다. 탐색시 시간복잡도는 문자열 길이 만큼 L이 걸림

2.5 셋

데이터를 비순차적으로 저장할 수 있는 순열 자료구조 비순차적이라 데이터 삽입 순서대로 저장되지 않아 일반적으로 순서를 기대할 순 없음 중복을 허용하지 않아 같은값을 삽입하면 마지막에 삽입한 값 하나만 저장됨

2.5.1 HashSet

HashCode를 사용해서 저장하기 전 먼저 hashCode를 얻어낸 다음 저장되어 있는 객체들의 해시 코드와 비교한 뒤 같은 해시 코드가 있다면 객체를 비교한뒤 동일한 객체가 있다면 중복 저장을 하지 않습니다.

시간 복잡도는 추가 contains next 의 O(1) O(1) O(h/n) 의 시간복잡도를 가짐

2.5.2 TreeSet

Tree 구조(Red-Black Tree)를 사용하여 Set을 사용하는 경우 장점은 자동 정렬을 해준다는 장점이 있습니다. 기본적으로 오름차순으로 정렬을 하고 추가 삭제에는 시간이 많이 걸림

시간 복잡도는 모드 Logn 의 시간복잡도를 가짐

2.5.3 LinkedHashSet

HashSet의 저장 순서를 보장해주는 Set 기본적인 구조는 LinkedHashMap 을 상속받아 사용하게 됨

시간복잡도는 모두 O(1) 을가짐

HashSet > TreeSet > LinkedHashSet 순서로 성능의 차이를 보임

기본적으로 Thread -safe 를 보장하지 않음 즉 동기화가 되지않음

2.6 맵

맵은 KEY와 Value로 이루어져 있는 자료구조 입니다.

각각의 키는 반드시 unique해야합니다.

2.6.1 HashMap

Key 값에 대한 Hashing을 통해 값을 저장하는 맵 방식

장점으로는 매우 많은 데이터라도 해싱함수를 사용하기 때문에 O(1)의 성능을 가진다 하지만 해시값이 중복될결우 최대 N까지 증가하는 문제점도 있음

단점으로는 정렬이 되지않고 입력 순서가 보장되지 않음

 HashMap은 Key에 대해서 Hashcode 함수를 이용하고 이 값을 Bucket의 크기로 나눈 나머지를 index로 이용하며 index가 겹치면 LinkedList

2.6.2 TreeMap

기본적으로 내부의 값들을 KEY값을 기준으로 정리하여 Tree구조로 만든 MAP

KEY의 순서를 유지할 수 있는방법은 Red Black Tree 의 자료구조를 이용함

시간복잡도는 logN을 기록

TreeMap 은 Key 의 순서가 오름차순으로 유지됩니다.

2.6.3 LinkedHashMap

2.7 힙

완전 이진트리의 일종으로 우선순위 큐를 위하여 만들어진 자료구조로 여러 개의 값들중 최댓값이나 최솟값을 빠르게 찾아내도록 만들어진 자료구조 힙ㅇㄴ 일종의 반정렬 상태를 유지한다.

2.8 우선순위 큐

데이터들이 우선순위를 가지고 우선순위가 높은 데이터가 먼저 나가게 되는 큐 기본적으우선순위 큐를 이용 사례로는 시뮬레이션 시스템이나 네트워크 트래픽 제어 등에 이용된다. 우선순위 큐는 배열 , 연결리스트 힙으로 구현이 가능하다 이중 힙으로 구현하는 것이 가장 효율적이다.

1. 알고리즘

모든 정렬은 오름차순 정렬을 기준으로….

3.1 정렬 알고리즘

선택정렬(Insertion Sort): 선택정렬은 앞에서부터 차례대로 정렬하는 방법으로 먼저 주어진 리스트 중에 최소값을 찾고 그 값을 맨앞에 위치한 값과 교체하는 방식

평균 최선 최악 n^2 공간복잡도 n

버블정렬(Bubble Sort) : 첫번재 원소부터 인접한 원소끼리 계속 자리를 교환하면서 맨끝부터 정렬하는 방식

평균 최선 최악 n^2 공간복잡도 n

삽입정렬 : 모든 요소를 앞에서부터 차례대로 이미 정렬된 배열 부분과 비교하여 자신의 위치를 찾아 삽입

평균 최악 n^2 최선의 경우 n 공간복잡도 n

합병정렬 : 작은 단위로 잘게 쪼개어 작은 단위부터 정렬해서 정렬된 단위들을 계속 병합해가면서 정렬해 나가는 방식 (매우중요)

**public** **int**[] mergeSort(**int**[] array,**int** start, **int** end, **int**[] temp) {

**if**(start<end) {

**int** mid=(start+end)/2;

mergeSort(array,start,mid,temp);

mergeSort(array,mid+1,end,temp);

**int** p=start;

**int** q=mid+1;

**int** index=p;

**while**(p<=mid||q<=end) {

**if**(q>end || (p<=mid && array[p]<array[q])) {

temp[index++]=array[p++];

}**else** {

temp[index++]=array[q++];

}

}

**for**(**int** i=start; i<=end; i++) {

array[i]=temp[i];

}

}

**return** array;

}

평규 최선 최악 nlogn 공간 복잡도 2n

퀵정렬 : 연속적인 분할에 의한 정렬 방식으로 처음 하나의 축을 먼저 정하여 이축의 값보다 작은값은 왼쪽에 큰값은 오른쪽으로 위치시킨뒤 왼쪽과 오른쪽 수 둘은 다시 각각의 축값이 1이 될 때 까지 정렬합니다.

평균 최선 nlogn 최악 n^2 공간복잡도 n

힙정렬 : 힙 정렬은 최소 힙 또는 최대 힙 트리를 구성해 정렬 해 나가는 방법

쉘정렬 : 삽입정렬의 개념을 확대하여 일반화한 정렬 방법

기수정렬 데이터의 비교를 통한 정렬이 아닌 분산정렬을 이용한 방법

* 1. 탐색알고리즘

DFS : 탐색을 할수 있을 곳까지 깊은곳 먼저 탐색하는방법 Ex) 트리의 Preorder와 같은느낌

BFS : 탐색을 넓혀 가며 탐색함 주로 queue를 이용하는 것이 특징 레벨을 이루고 있음

* 1. 최단경로

다익스트라

다이나믹 프로그래밍을 활용한 최단 경로 탐색알 고리즘 특정 한 하나의 정점에서 다른 모든 정점으로 가는 최단 경로를 알려줍니다. 단점은 음의 간선을 표현할 수없다

PriorityQueue<Node> queue=**new** PriorityQueue<>();

**boolean**[] check=**new** **boolean**[array.length];

queue.add(**new** Node(start,0));

array[start]=0;

**while**(!queue.isEmpty()) {

Node node=queue.poll();

**if**(check[node.end]) **continue**;

check[node.end]=**true**;

**for**(Node connect : list[node.end]) {

**if**(array[connect.end]>array[node.end]+connect.weight) {

array[connect.end]=array[node.end]+connect.weight;

queue.add(**new** Node(connect.end,array[connect.end]));

}

}

}

.

Floyd -Warshall

3.4 다이나믹 프로그래밍

DFS BFS 다익스트라 Floyd-WarShall DP 브르투포스 정렬 알고리즘 등에 대한것들

Stable한 솔트란?

1. DB

4.1 DB란?

여러 사람이 공유하여 사용할 목적으로 체계화해 통합 관리하는 데이터의 집합

4.1.1 Entity : 실제 객체 정보를 저장하고 관리하긴 위한 집합적인 것

4.1.2 Table : 데이터베이스에 저장되는 가장 큰단위 기본적으로 쿼리문을 수행하고 인덱스를 설정할 수 있는 단위

4.1.3 Column 세로줄 하나 하나를 컬럼이라고 부름

4.1.4 Row : 행

4.1.5 Schema 데이터베이스를 설계 생성하는 과정에서 각각의 테이블에 필요한 컬럼의 타입과 네이밍을 결정하는 것

* 1. Index

추가적인 쓰기 작업과 저장 공간을 활용하여 데이터베이스 테이블의 검색 속도를 향상시키기 위한 자료구조 데이터베이스에서 모든 데이터를 검색하면 오래 걸리기 때문에 데이터와 데이터의 위치를 포함한 자료구조를 생성하여 빠르게 조회할 수 있도록 돕고 있는 것

* + 1. 장점 : 데이터 테이블 full scan 대신 더 빠르게 데이터에 접근할 수 있게 해줌 즉 select 절의 속도가 빨라짐
    2. 단점 : insert update 의 속도가 저하됨 정렬된 상태를 유지해야하고 데이블 외에 인덱스 테이블에도 insert,update 해줘야 하기 때문에 insert와 update가 빈번히 일어나는 테이블에 대해서는 index 설정이 성능에 부정적 영향을 줄 수도 있음

인덱스를 만들면 파일의 크기는 늘어남

* + 1. 유의 사항 : 두가지 이상의 컬럼을 인덱스로 설정하는 결함 인덱스의 경우 설정 순서가 중요하다. 분별력이 높은 칼럼이 앞으로 오는 것이 성능 향상에 유리함. 전체 데이터 중 10~15% 이내의 데이터를 검색하는 경우 사용하거나 두개 이상의 컬럼이 where 절이나 join 조건 으로 자주 사용하는 경우 좋다고 합니다.
  1. DB 메모리 구조

오라클 데이터베이스의 경우 초기화 파라미터를 사용하여 메모리 구조를 생성 및 관리한다.

메모리 구조는 SGA 와 PGA 로 나뉜다.

* + 1. PGA
    2. SGA

공유 메모리 구조 그룹으로 하나의 오라클 데이터베이스 인스턴스의 데이터 및 제어정보를 포함하고 있고 모든 서버 백 그라운드 프로세스에 공유된다.

동적 SGA 구조를 사용하면 인스턴스를 종료하지 않고 Share Pool ,Database Buffer Cache , Large Pool Java Pool Streas Pool 크기를 변경할 수 있다.

* + - 1. SharedPool

유저 간의 공유할 수 있는 다양한 구성요수를 캐시에 저장

* + - 1. Database Buffer Cache

데이터베이스에서 검색된 데이터 블록을 캐시에 저장

* + - 1. Keep Buffer Pool

메모리에 데이터를 오랫동안 보존하도록 튜닝된 데이터베이스 버퍼 캐시

* + - 1. Recycle Buffer Pool

메모리에서 블록을 신속하게 재사용 또는 제거하도록 튜닝된 특별한 유형의 데이터베이스 버퍼 캐시

* + - 1. nK Buffer cache

기본 데이터 베이스 블록 크기와 다른 블록 크기를 저장하도록 설계

* + - 1. Redo Log Buffer

Instance 복구에 사용되는 리두 정보가 디스크에 저장된 물리적 리두 로그 파일에 기록될 때 까지 해당 정보를 저장

* + - 1. Large Pool

특정 대용량 프로세스 및 IO 서버 프로세스에 대한 대용량 메모리 할당을 제공하는 선택적 영역

* + - 1. Java Pool

JVM 내의 모든 세션별 Java 코드 및 데이터에 사용

* + - 1. Streams Pool

오라클 스트림에서 컵채 및 적용에 필요한 정보 저장

Transaction Entity 나중을 위한 column 등과 같은 기본 개념 및 추후 가능하다면 디비 튜닝까지??

* 1. 정규화

이상 문제를 해결하기 위해 속성들 끼리의 종속 관계를 분석하여 여러 개의 릴레이션으로 분해하는 과정 일반적으로 테이블을 여러 개로 분해하면 속도는 상대적으로 느려질 수 있지만 분해하지 않으면 이상 문제들이 발생

이상문제 : 삽입이상 : 데이터를 저장할 때 원하지 않는 정보가 함께 삽입되는 경우

삭제 이상 : 튜플을 삭제함으로서 유지되어야 하는 정보 까지도 연쇄적으로 삭제되는경우

갱신이상 : 중복된 튜플 중 일부의 속성만 갱신 시킴으로서 정보의 모순성이 발생하는 경우

함수적 종속이란

어떠한 릴레이션에서 속성들의 부분집합을 X Y라 할 때 특정 튜플에서 X의 값이 Y의 값을 함수적으로 결정 한다면 Y가 X에 함수적으로 종속 되었다고 함

재귀 규칙 Y가 X의 부분 집하이면 X-> Y 이다

증가 규칙 X -> Y이면 WX -> WY 이고 Wx -> Y이다

이행규칙 X->Y 이고 Y->Z 이면 X ->Z이다

유니온 규칙 X->Y 이고 X -> Z이면 X->YZ이다

분해규칙 X -> YZ 이면 X-> Y 와 X->Z 이다.

가이행 규칙 만일 W-> X이고 XY->Z 이면 WY - > Z이다.

정규화 과정

제1 정규형

도메인은 모두 원자값이어야 한다.

제 2 정규형

제 2 정규형은 부분 함수 종속 제거

제 3 정규형

제3 정규형 이행 함수 종속을 제거

BCNF 결정자 중 후보키가 아닌 것들을 제거

제4 정규형 다치 종속제거

제 5 정규형 조인 종속성 제거

* 1. 트랜잭션

데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산들을 의미한다.

* + 1. 특징

트랜잭션은 데이터베이스 시스템에서 병행 제어 및 회복 작업시 처리되는 작업의 논리적 단위이다.

사용자가 시스템에 대한 서비스 요구 시 시스템이 응답하기 위한 상태 변환 과정의 작업 단위

하나의 트랙잭션은 commit 되거나 rollback 된다.

* + 1. 성질
       1. 원자성

트랜잭션의 연산은 데이터베이스에 모두 반영 되던지 아니면 전혀 반영 되지 않아야 한다.

트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행중에 다른 트랜잭션의 연산이 끼어 들 수 없다.

* + - 1. 일관성

트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터 베이스 상태로 변환한다. 시스템이 가지고 있는 고정 요소는 트랜잭션 수행전과 수행 완료 후의 상태가 같아야 한다.

* + - 1. 독립성

둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행중에 다른 트랜잭션의 연산이 끼어 들 수 없다.

수행중인 트랜잭션은 완전히 완료될 때 까지 다른 트랜잭션에서 수행 결과를 참조할 수 없다.

* + - 1. 영속성(지속성)

성공적으로 완료된 트랜잭션의 결과는 시스템이 고장 나더라도 영구적으로 반영되어야한다.

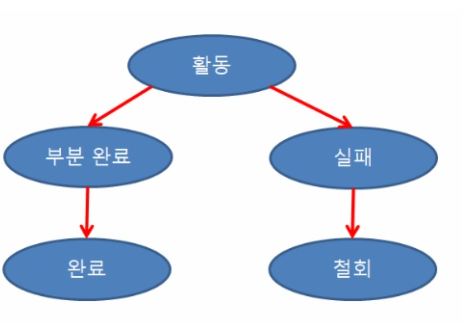
* + 1. 연산 및 상태
       1. Commit 연산

한 개의 논리적 단위에 대한 작업이 성공적으로 끝나고 데이터베이스가 자시 일관된 상태에 있을 때 트랜잭션이 행한 갱신 연산이 완료된 것을 트랜잭션 관리자에게 알려주는 연산

* + - 1. Rollback 연산

하나의 트랜잭션 처리가 비정상적으로 종료되어 데이터베이스의 일관성을 깨뜨렸을 때 이 트랜잭션의 일부가 정상적으로 처리되었더라도 트랜잭션의 원자성을 구현하기 위해 이 트랜잭션이 행한 모든 연산을 취소 하는 연산

* + - 1. 상태



* 1. SQL 과 NOSQL
  2. 클러스터

1. 네트워크

TCP/IP는 ?

패킷 통신 방식의 인터넷 프로토콜인 IP와 전송 조절 프로토콜인 TCP로 이루어져 있다. IP는 패킷 전달 여부를 보증하지 않고 패킷을 보낸 순서와 받는 순서가 다를수 있고 TCP는 IP위에서 동작하는 프로토콜로 데이터의 전달을 보증하고 보낸 순서대로 받게해준다.

HTTP FTP SMTP 등이 TCP를 기반으로 한 애플리케이션 프로토콜이다.

TCP/IP UDP 차이 3-way-handshaking 등

TCP : 연결형 서비스로 가상 회선 방식을 제공

3-way handshaking 과정을 통해 연결을 설정 4-way handshaking을 통해 해제한다.

흐름제어 및 혼잡제어가 가능

높은 신뢰성을 보장하고

UDP보다 속도가 느리며

Full-Duplex , Point to Point 방식

서버 소켓은 연결만을 담당하고 연결 과정에서 반환된 클라이언트 소켓은 데이터의 송수신에 사용된다 서버와 클라이언트는 1:1로 연결되며 스트림 전송으로 데이터의 크기가 무제한이다.

패킷에 대한 응답을 해야하므로 성능이 낮은 편이다. Streaming 서비스에 불리하다 그이유는 손실된 경우 재전송 요청을 하기 때문이다.

UDP

비연결형 서비스로 데이터그램 방식을 제공한다.

정보를 주보 받을 때 정보를 보내거나 받는다는 신호절차를 거치지 않는다.

UDP 헤더의 CheckSum 필드를 통해 최소한의 오류만 검출한다. 그렇기 때문에 신뢰성ㅇ이 낮은대신 TCP에 비해 속도가 빠르다.

서버와 클라이언트는 1:1 1:N N:M 으로 연결될 수 있고 데이터그램단위로 전송되며 65535바이트로 크기가 초과되면 잘라서 전송한다.

흐름제어가 없어서 패킷이 제대로 전송됬는지 오류가 없는지 확인할 수 없다. 파일 전송과 같은 신뢰성이 필요한 서비스 보단 성능이 중요시 되는 경우 사용

흐름제어와 혼잡제어

흐름제어는 데이터를 송신하는 곳과 수신하는 곳의 데이터 처리 속도를 조절하여 수신자의 버퍼오버플로우를 방지하는 것이다.혼잡제어는 네트워크 내의 패킷수가 넘치게 증가하지 않도록 방지하는 것

1. 프레임워크

6.1Spring을 주로 다룰 예정 DI/DL JPA 등등 BEAN

6.2. net

6.3 django

1. 웹

7.1 Http란?

인터넷에서 데이터를 주고 받을 수 있는 프로토콜

특징은 비연결성을 가지고 있다 비연결성은 클라이언트가 서버와 한번 연결을 맺은 후 클라이언트 요청에 대해 서버가 응답을 마치면 연결을 끊어버림

HTTP는 인터넷 상에서 불특정 다수의 통신환경을 기반으로 설계됨 만약 서버에서 다수의 클라이언트와 연결을 계속 유지해야한다면 많은 리소스가 필요하지만 연결을 유지하지않기 때문에 리소스가 줄어들어 더 많은 연결을 할 수 있음

하지만 단점으로 서버는 클라이언트를 기억하고 있지 않아 동일한 클라이언트 모든 요청에 대해 매번 새로운 연결을 시도/해제의 과정을 거쳐야 하므로 연결 / 해제에 많은 오버헤드가 발생한다는 단점 이에 대한 해겨랙으로 keepalive 속성을 사용할 수 있지만 이것을 사용하면 상대방의 안부를 묻기위해 패킷을 주기적으로 보내는 것이지만 프로세스 수가 기하급수적으로 늘어나 메모리를 많이 사용하게됨

무상태성의 특징 때문에 서버는 클라이언트를 식별할 수가 없어 이를 해결하기 위해 쿠키와 세션의 개념이 도입됨

쿠키 : 클라이언트 로컬에 저장되는 키와 값이 들어있는 작은 데이터파일

Response Header 에 Set – Cookie 속성을 사용하면 클라이언트에 쿠키를 만들 수 있음

쿠키는 사용자가 따로 요청하지 않아도 브라우저가 Request 시에 Request Header에 넣어 서 서버에 전송합니다.

쿠키의 동작 방식

1. 클라이언트가 페이지에 요청 - > 서버에서 쿠키 생성 - > HTTP 헤더에 쿠키를 포함시켜 응답 - > 브라우저가 종료되어도 쿠키 만료기간이 있다면 클라이언트에서 보관 - > 같은 요청을 할 경우 HTTP 헤더에 쿠키를 함께 보냄 -> 서버에서 쿠키를 읽어 이전 상태 정보 변경 할 필요가 있을 때 쿠키를 업데이트 하여 변경된 쿠키를 HTTP 헤더에 포함시켜 응답.

세션 : 서버측에서 관리하는 것으로 세션 ID를 부여하고 웹 브라우저가 서버에 접속해서 브라우저를 종료할 때 까지 인증상태를 유지하는 기술

접속시간에 제한을 두어 일정 시간 응답이 없다면 정보가 유지되지 않게 설정이 가능하 정보를 서버에 두기 때문에 쿠키보다 보안에 좋지만 사용자가 많이지면 서버 메모리를 많이 차지하게되어 과부화를 주기 때문에 성능 저하의 요인이 된다. 클라이언트가 Request를 보내면 서버가 클라이언트에게 유일한 ID를 부여하는데 이걸 세션 ID라고함

동작방식 :

클라이언트가 서버에 접속 시 세션 ID를 발급 받는다. -> 클라이언트는 세션 ID에 대해 쿠키를 사용해서 저장하고 있음 -> 클라이언트는 서버에 요청할 때 이 쿠키의 세션 ID를 서버에 전달 -> 서버는 세션 ID를 전달 받아서 별다른 작업없이 세션 ID로 세션에 있는 클라이언트 정보를 가져옴 -> 클라이언트 정보를 가지고 서버 요청을 처리

세션으로만 사용해도 되지만 쿠키를 사용하는 이유는??

세션은 일정기간 동안 사용자로부터 들어오는 Request를 서버에 저장하는데 서버의 메모리가 감당할 수 없어질 수가 있고 속도가 느려질 수 있기 때문에 사용한다.

* 1. URL 과 URI 의 차이는 ?

URI : 인터넷 자원을 나타내는 고유 식별자

URL

RESTFul 하다라는건 ?

Rest의 원리를 잘 지키는 시스템을 RestFul 하다 라고 합니다.

RestFul 한 API란 각 구성요소들의 역할이 완벽하게 분리되어 있는 것

URI는 각 리소스를 명확하게 인식할 수 있는 용이한 표현방식을 가져야 합니다.

각 리소스에 대한 행위를 HTTP 메소드를 이용하여 Uniform 하게 정의 할 수 있어야합니다. Unifom 한 형식에 의해 제한 받는 세부 기능의 경우 JSON XML 등을 이용하여 별도로 정의함으로서 분리 할 수 있어야한다.

URI를 설계 해야하는 방법

슬래시를 이용해서 계층관계를 표현한다.

여러 리소스들의 집합을 의미하는 컬렉션의 경우 복수형태 사용을 권장하고 있으며 각 리소스들은 단수를 사용한다.

언더스코어 사용자제

또한 URI는 호스트를 제외한 나머지 모든 요소에서 소문자를 사용하는 것을 권장

파일 확장자는 URI에 미포함

명사 위주의 URI 를 설계하는 것이 좋다.

REST 아키텍처에 적용되는 6가지 제한 조건

인터페이스 일관성 : 일관적인 인터페이스로 분리되어야 한다

무상태 : 각 요청 간 클라이언트의 콘텍스트가 서버에 저장되어서는 안된다.

캐시 처리 가능 : WWW에서와 같이 클라이언트는 응답을 캐싱할 수 있어야한다. 잘 관리되는 캐싱은 클라이언트 – 서버간 상호작용을 부분적으로 혹은 완전하게 제거하여 scalability와 성능을 향상시킨다.

계층화 : 클라이언트는 보통 대상 서버에 직접 연결되었는지 또는 중간 서버를 통해 연결되었는지를 알 수 없다. 중간 서버는 로드 밸런싱 기능이나 공유 캐시 기능을 제공함으로서 시스템 규모 확장성을 향상시키는데 유용하다.

Code on Demand : 자바 애플릿이나 자바 스크립트의 제공을 통해 서버가 클라이언트가 실행 시킬 수 있는 로직을 전송하여 기능을 확장 시킬 수 있다.

클라이언트 / 서버 구조 : 아키텍처를 단순화 시키고 작은 단위로 분리함으로서 클라이언트 서버의 각 파트가 독립적으로 개선될 수 있도록 해준다.

7.3 http 동작 과정

1. 사용자가 웹 브라우저에 URL 주소 입력

2. DNS 서버에 웹 서버의 호스트 이름을 IP 주소로 변경 요청

3. 웹 서버와 TCP 연결 시도

3 -way HandShake : 클라이언트 – 서버간 신뢰성있는 연결을 하기 위해 3번의 패킷 교환 과정

SYN 클라이언트가 서버로 임의로 생성한 시퀀스 번호를 전달

SYN ACK 서버는 클라이언트에서 전달한 시퀀스를 +1 시켜서 전달한다.

ACK : 클라이언트가 서버에서 전달해준 시퀀스를 +1시켜서 다시 전달함(서버 클라이언트간 패킷 교환이 정상적으로 이루어졌다는 신호)

4. 서버에게 GET POST PUT 등과 같은 명령 전송

5. 서버가 클라이언트에게 데이터(웹 문서) 회신

6.서버 – 클라이언트간 연결 해제

웹브라우저 동작과정

REST한 방식이뭔지

GET PUT POST DELETE TRACE OPTIONS 상세 분류

Http 상태코드

Https란 ?

1. 프로젝트

8.1 졸업작품

8.2 회사에서 했던것들

1. 컴퓨터 CS

9.1 프레임워크와 라이브러리 차이는?

9.1.1 프레임워크 : 소프트웨어의 구체적인 부분에 해당하는 설계와 구현을 재사용이 가능하게끔 일련의 협업화된 형태의 클래스들을 제공하는 것으로 애플리케이션의 틀과 구조를 결정할 뿐 아니라 개발된 개발제의 코드를 제어하여 구체적이고 확장 가능한 기반코드를 가지고 있고 설계자가 의도하는 여러 디자인 패턴의 집합으로 구성되어 있다.

9.1.2 라이브러리 : 자주 쓰일 만한 기능들을 모아 놓은 유틸(클래스)들의 모음집

프레임워크에는 라이브러리에 뼈대가 되는 클래스들과 그 클래스들의 관계로 만들어진 일종의 설계의 기본 틀이 추가된다 프레임워크는 제작자가 기초로 해서 만들라는 기본 틀이라는 것을 제공하고 이를 사용해 확장해 가며 입맛대로 개발하면 되는 것

* 1. 프로세스와 스레드의 차이
     1. 프로세스 : 메모리에 올라와 실행되고 있는 프로그램의 인스턴스
        1. 프로세스 메모리 구조 :

Code 영역 : 프로그램을 실행시키는 실행 파일 내의 명령어들이 올라가는공간

Data 영역 : 전역변수 static 변수의 할당

Heap 영역 : 동적할당을 위한 메모리 영역

Stack 영역 : 지역변수 , 함수 호출시 전달되는 인자(파라미터)를 위한 메모리 영역

* + - 1. 프로세스 상태변화

프로세스 상태에는 ready , blocked running

9.2.1.3프로세스 스케줄링

CPU 할당 순서 및 방법을 결정하는 일 우선순위 알고리즘과 라운드 로빈 알고리즘을 혼합해서 스케줄링하는 것이 보편정

선점 스케줄링:

비선점 스케줄링

* + 1. 스레드 : 프로세스 내에서 할당받은 자원을 이용해 동작하는 실행단위

스레드는 프로세스 내에서 Stack마 SEK로 할당 받고 Code Data Stack영역을 공유한다. Stack을 공유하면 LIFO 구조에 의해 실행 순서가 복잡해지기 때문에 실행흐름을 원할히 하기 위해 그렇다.

스레드는 프로세스의 자원을 공유하기 때문에 다른 스레드의 의한 결과를 즉시 확인할 수 있다.

* + - 1. 멀티스레드

하나의 프로그램을 여러 개의 스레드로 구성하여 각 스레드가 1개의 작업을 처리하도록 하는 것

프레스를 위해 자원을 할당하는 시스템콜이나 Context Swtiching 오버헤드를 줄일 수 있다. 스레드는 메모리를 공유하기 때문에 통신이 쉽고 자원을 효율적으로 사용할 수 있다 하지만 단점으로는 하나의 스레드에 문제가 생기면 전체 프로세스가 영향을 받는다. 그리고 여러 스레드가 하나의 자원에 동시에 접근하는 경우 자원 공유(동기화)의 문제가 발생할 수 있다.

* + - 1. DeadLock

한정된 자원을 여러 프로세스가 사용하고자 할 때 발생하는 상황으로 프로세스가 자원을 얻기위해 영구적으로 기다리는 상태입니다.

DeadLock 조건 4가지

상호배제

환형대기

비선점

점유와 대기

해결방안 :

9.2.2.3

9.3. OOP 개념

현실 세계를 프로그래밍으로 옮겨와 현실 세계의 사물들을 객체로 보고 그 객체로부터 개발하고자 하는 특징과 기능을 뽑아와 프로그래밍 하는 기법을 의미 OOP로 코드를 작성하면 재사용성과 변형 가능성을 높일 수 있음

9.3.1 객체지향언어의 특징

캡슐화 : 데이터와 데이터를 처리하는 함수를 하나로 묶는 것 캡슐화된 객체의 세부 내용이 외부의 은폐되어 변경이 발생할 때 오류의 파급효과가 적다. 캡슐화된 객체들은 재사용이 용이하다.

정보은닉 : 캡슐화에서 가장 중요한 개념으로 다른 객체에게 자신의 정보를 숨기고 자신의 연산만을 통하여 접근을 허용하는 것

추상화 : 불필요한 부분을 생략하고 객체의 속성중 가장 중요한 것에만 중점을 두어 개략화하는 것 즉 모델화

상속성 : 이미 정의된 상위 클래스의 모든 속성과 연산을 하위클래스가 물려받는 것

다형성 메시지에 의해 개체 가 연산을 수행하게 될 때 하나의 메시지에 대해 각 객체가 가지고 있는 고유한 방법으로 응답할 수 있는 능력을 의미 객체들은 동일한 메소드명을 사용해서 같은 의미의 응답을 함.

9.3.2 SOLID 원칙

Single Respongsiblity Principle( 단일책임 원칙) : 소프트웨어의 설계 부품(클래스 함수)는 단 하나의 책임만을 가져야 한다.

설계가 잘된 프로그램은 기본적으로 새로운 요구사항과 프로그램 변경에 영향을 받는 부분이 적다 즉 응집도는 높고 결합도는 낮은 프로그램이다. 만약 한 클래스가 수행 기능이 많다면 클래스 내부의 함수 끼리 강한 결합을 발생할 가능성이 높아진다 이에 따라 유지보수 비용이 증가하게 되므로 책임을 분리시켜야한다.

Open-Closed Principle (개방 폐쇄 원칙) : 기존의 코드를 변경하지않고(close) 기능을 수정하거나 추가 할 수 있도록 설계 해야한다.

즉 인터페이스의 설계가 필요하다는 것을 의미

Liscov Substitution Principle (리스코프 치환원칙 ) : 자식클래스는 부모클래스에서 가능한 행위를 수행 할 수 있어야한다. 부모클래스와 자식 클래스 사이의 행위에는 일관성이 있어야한다는 원칙이며 이는 객체 지향 프로그래밍에서 부모 클래스의 인스턴스를 사용해도 문제가 없어야한다는 것을 의미

상속 관계에서 일반화 관계 IS A 관계가 성립되어야 한다.

Dependecncy Inversion Principle(의존 역전 법칙) : 의존 관계를 맺을 때 변화하기 쉬운 것 보단 변화하기 어려운 것에 의존해야 한다는 원칙이다. 여기서 변화하기 어려운 것이란 즉 추상적인 것을 의미 변화하기 쉬운 것은 구체화된 클래스를 의미하고 어려운 것은 추상클래스나 인터페이스를 의미한다.

Interface Segregation Principle 인터페이스 분리원칙 한 클래스는 자신이 사용하지 않는 인터페이스는 구현하지 말아야 한다. 하나의 일반적인 인터페이스 보다는 여러 개의 구체적인 인터페이스가 낫다. 즉 사용하지 않는 기능에는 영향을 받지 말아야한다는 것을 의미

9.3.3 다형성 상속

9.4 디자인패턴

9.5 Parameter 와 Argument 차이

Parameter = 함수를 선언할 때 사용된 변수

Argument : 함수가 호출 되었을 때 함수의 파라미터로 전달된 실제 값

9.5 Call By Value Call By reference 차이

Call by value = 인자로 받은 값을 복사하여 처리하는 방식 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 존재

9.6 동기 비동기

동기 :

비동기 :

9.7 Message Queue 란?

10 . 인공지능

11. 빅데이터

12.. 블록체인

13그동안 나왔던 질문들 복기

13.1컴파일 언어와 인터프리터언어(스크립트 언어)의 차이는 ?

13.1.1 컴파일 언어 :

컴파일 이라는 과정을 통해 프로그래머가 작성한 코드를 기계어로 번역해 실행하는 언어임

소스 코드를 컴파일 하는 과정을 거쳐야 하므로 실행 및 수정에 비교적 많은 시간을 소요하지만 한번 기계어로 번역되면 빠른 실행 속도를 보여주는 것이 장점

13.1.2 인터프리터 언어

별도의 컴파일 과정 없이 인터프리터가 소스 코드를 한 줄씩 읽어가며 바로 실행하는 언어를 의미 . 컴파일 없이 곧바로 실행하므로 결과를 바로 확인하고 빠르게 수정 가능하지만 번역과 실행이 동시에 이뤄져 컴파일 언어보다 느린 실행 속도를 보임

13.2GIT 과 GITHUB의 차이는 ?

13.2.1 Git

버전 관리시스템중 유명한 것으로 형상 관리 프로그램

로컬에서 관리되는 버전 관리 시스템으로 소스코드 수정에 따른 버전을 관리해주는 시스템

13.2.2 GtiHub

클라우드 방식으로 관리되는 저장소

13.3자료구조에서 map이란 무엇일까

13.4메서드 오버로딩과 메서드 오버라이드 차이

13.5웹 서버와 웹 애플리케이션 서버의 차이 좀 딥하게

13.5.1 웹 서버

웹 브라우저와 같은 클라이언트로부터 HTTP 요청을 받아들이고 HTML 문서와 같은 웹페이즈를 정적으로 처리해 반환하는 프로그램

13.5.2 웹 애플리케이션 서버

13.6 MSA 란

작고 독립적으로 배포 가능한 각각의 기능을 수행하는 서비스로 구성된 것 완전히 독립 배포가 가능하고 다른 기술 스택이 사용 가능한 단일 사업 영역에 초점을 둔다.

각각의 배포를 하기 때문에 일부의 서비스가 문제가 생겨도 전체에 영향을 주지 않는다. 하지만 단점으로는 테스트가 어렵고 부하가 많을 경우 트랜잭션을 어떻게 유지해야할지 결정하는데 어려움을 가지고 있다.

13.7 TDD란

13,8 SQL Injection 과 그에 대한 공격방법 대안방안

HTTP를 통해 컴퓨터나 장치에 애플리케이션을 수행해주는 미들웨어

13.9 자바에서의 String StringBuilder StringBuffer 차이

13.10 다형성에 대한 설계?

13.11 자바 8과 자바 11 자바 14에서 크게 바뀐점?

14.추가적으로 궁금한것들 아래 달아주세염 :

15. 안드로이드 관련

16. 프론트 관련