C#이란?

마이크로소프트에서 개발한 .Net 프레임워크 기반 범용 목적의 다중 패러다임 프로그래밍 언어

.Net Framework

* 웹 앱, 모바일 앱, 데스크톱 프로그램, 게임 프로그램, 사물인터넷 프로그램 등을 만들기 위한 오픈소스, 크로스 플랫폼 개발 환경
* 자바에서의 자바가상머신과 비슷하다

.Net Framework / .Net Core / Xamarin의 차이점

* 윈도우 환경에서 개발, 실행하는 것은 .Net Framework
* 리눅스, Mac에서 개발, 실행하는 것은 .Net Core
* 모바일 환경에서 개발, 실행하는 것은 Xamarin

클래스에는 main문이 존재해야한다

* Public static void Main(string[] args)

C#에서의 클래스 소멸자는 접근 지정자를 따로 표기하지 않는다

C#에서의 클래스 상속 또한 접근 지정자를 따로 표기하지 않는다, 클래스 앞에 sealed 키워드를 붙이면 해당 클래스는 상속이 불가능하다

C#에서의 배열선언 -> 데이터형[] 변수명 int[] arr

* C++과는 다르게 선언과 함께 메모리가 할당되지는 않는다

C#에서의 컬렉션

특정 타입의 데이터만을 넣는 것이 아니라 다양한 타입의 데이터를 넣을 수 있다(Object 형식의 개체로 저장하기 때문에)

1. ArrayList

* 동적으로 크기가 증가하는 배열
* MS에서는 ArrayList 대신 List 사용을 권장

1. HashTable

* Key – Value 형태로 데이터를 저장
* Dictionary 사용을 권장

1. Queue

* 선입선출 자료구조

1. Stack

* 선입후출 자료구조

웹 서버

* HTTP/HTTPS를 이용하여 통신하는 서버를 의미

API 서버

* XML., JSON등을 사용하여 데이터를 주고받는 서버를 의미

ASP.NET

* .NET과 C#프로그래밍 언어를 사용해 웹 응용 프로그램을 만들기 위한 프레임워크

.NET 기본 프로젝트 생성

* cmd창에 dotnet new webapp -o MyWebApp -–no-https 입력

Redis

1. 캐시로 사용하기

Look-Aside(Lazy Loading)

* + 읽을 때 캐시에 데이터가 있는지 확인하고 없다면 DB에서 가져온다
  + 처음에는 캐시에 아무 데이터가 없기 때문에 DB에 작업이 몰릴 수 있어, DB에서 캐시에 데이터를 넣어주는 Cache Warming 작업을 하기도 한다

Write-Around

* + 모든 데이터를 DB에 저장하고 캐시미스가 일어난 경우 캐시에서 데이터를 가져온다
  + DB와 캐시의 데이터가 다를 수 있다는 단점이 존재한다

Write-Through

* + 저장할 때 모든 데이터를 캐시와 데이터베이스 모두에 저장한다
  + 모든 데이터를 두단계를 거쳐 저장하기 때문에 성능 저하가 있다

1. Redis의 데이터 타입

* Strings : 단순 증감 연산
* Bitmaps: 데이터 저장공간 절약, 정수로 된 데이터만 카운팅 가능
* Lists(Queue): 메시지 큐로 사용
* Hashes, Sets, Sorted Sets(스코어 순으로 정렬, 스코어가 같다면 사전순으로 정렬)
* HyperLogLogs: 대용량의 데이터를 카운팅 할 때 적절, set과 비슷하지만 12KB로 고정되어 저장, 저장된 데이터는 다시 확인할 수 없다
* Streams: Log를 저장하기 좋다, append-only로 동작

1. Redis에서 데이터 영구 저장

* Redis는 서버 재시작 시 모든 데이터가 사라지며, 복제 기능을 사용하여도 항상 데이터 복원이 가능한 것이 아니기 때문에 캐시 이외의 용도로 사용한다면 적절한 데이터 백업이 필요하다
* AOF(Append Only File) : Redis프로토콜 형태로 저장, 데이터가 변경되는 명령이 들어온다면 해당 명령어를 모두 저장한다
* RDB: binary형태로 저장, 스냅샷을 찍어 해당 시점의 데이터를 파일로 저장한다

백업이 필요하지만 어느 정도 손실을 감안할 수 있을 때 -> RDB사용

장애 직전 상황까지 모든 데이터가 보장되어야 하는 경우 -> AOF사용

두 개의 방법을 모두 사용하는 것이 가장 안전

Redis 아키텍처의 종류

1. Replication

* 단순한 복제 연결
* 비동기식으로 복제가 이루어지기 때문에 마스터노드에서 장애가 발생할 시 수동으로 복구를 해주어야 한다

1. Sentinel

* sentinel노드가 다른 노드를 모니터링, 마스터 노드에서 장애 발생 시 자동으로 페일오버를 발생시켜 레프리카 노드가 마스터 노드가 된다

1. Cluster

* 데이터가 여러 마스터 노드에 분할되어 저장되는 샤딩기능을 제공
* 모든 노드가 서로를 감시하며, 마스터 노드 장애 발생 시 자동 페일오버

HTTP

처음 세션을 연결할 때, 데이터 통신에 암호화할 대칭키를 비대칭키를 통해 교환, 이후 대칭키를 이용해 통신 -> 보안과 속도를 동시에 챙기기 위함

HTTP는 데이터 통신을 할 때만 연결 유지

GET

* POST방식보다 전송속도가 빠르며, 브라우저 히스토리에 기록이 남아 캐싱가능
* 전송하는 데이터양의 한계 존재
* URL에 정보가 노출되기 때문에 보안에 취약

POST

* 데이터들을 Body에 담아서 전송, 데이터양의 제한이 없음
* GET방식보다 보안이 강함
* 브라우저 히스토리에 기록이 남지 않으며, 캐싱 불가
* 개발자도구 등을 통해 body의 내용을 확인할 수 있기 때문에 암호화를 해야한다

미들웨어

* HTTP요청에 대해 요청대리자가 존재하며, ASP.NET Core 파이프라인의 수행을 차례대로 수행
* 대리자가 다음 대리자에 요청을 전달하지 않을 때, 이를 요청 파이프라인을 단락(short-circuiting)한다고 한다. 단락은 불필요한 작업을 방지
* 다만, 클라이언트에 응답을 전송한 후 단락을 진행하면 안된다
* Use, Run, Map 세가지 메소드를 통해 등록
* Use는 미들웨어를 순서대로 등록하는 메소드, Run은 모든 미들웨어 등록 후 마지막으로 호출하는 엔드포인트 메소드, Map은 특정 URI에 대해서 분기를 나누는 메소드

Dapper

* 매개변수화 된 쿼리를 사용하여 SQL Injection을 방지한다

DI(Dependency Injection)

* 컨트롤러에서 객체를 생성하게 된다면, 객체의 클래스에 대한 의존도가 높아짐, 이러한 의존성을 낮추기 위해 중간에 인터페이스를 삽입하는 방법을 이용함
* ASP.NET은 DI구현체를 등록할 수 있는 방법을 수명주기에 따라 3가지로 구분
* Transient는 각각의 요청마다 새로운 객체를 생성한다, Scope는 클라이언트 요청당 한 번 생성, Singleton은 처음 요청 또는 configurations을 빌드할 때 생성

UseRouting()

* 라우팅 미들웨어를 사용하여 클라이언트의 HTTP 요청을 처리할 라우팅 경로를 찾는다
* 요청된 URL에 따라 컨트롤러를 결정하는데 사용