2023-08-03~ 2023-08-04

* 기본 개념

.Net Framework: 다양한 프로그램을 만들기 위한 오픈 소스 플랫폼 개발 환경

* 플랫폼 차이

.NET Framework VS .NET Core VS Xamarin

Framework: 윈도우 환경

Core: 윈도우 환경 + 리눅스 환경 + Mac 데스크톱 환경에도 구동 가능한 서비스 or 웹

Xamarin: 모바일 or Mac 데스크톱

* C#

C# 🡪 명령형 선언형 둘 다 지원하는 다중 패러다임 언어이다.

Task, await 관련 문법에 관한 공부가 많이 중요하다.

* API 서버

JSON, XML 프로토콜의 데이터를 주고받는 것

* ASP.NET

.net 과 c# 프로그래밍 언어를 사용해 웹 서비스를 구축하기 위한 프레임워크

Cmd 창에서 donet 명령어를 통해 Web서버를 구축할 수 있다.

* Redis
* 캐싱: 데이터를 더 빠르고 효율적으로 접속 가능하게 하는 시스템.

1. 원본 접근 < 캐시 접근 (속도 더 빠름)
2. 반복이 잦은 데이터를 캐시에 올려 놓으면 높은 효율을 가져올 수 있음.
3. Key – Value 방식으로 저장하기 때문에 직관적이고 빠르게 저장, 읽어오기 가능

* Cache Warming: 캐시 메모리와 DB를 한번 업데이트 해주는 것
* 캐싱 전략

1. Write – Aside: 새로운 데이터는 무조건 DB에 저장. 캐시 미스 시, Redis에서 데이터를 가져옴.

Redis(캐시) 내 데이터와 DB의 데이터가 다를 수 있다는 단점.

1. Write – Through: 캐시와 DB에 모두 데이터를 저장. 항상 Redis와 DB가 최신 상태를 유지하고 있다는 장점.

다만 데이터 추가 시, 매번 업데이트를 수행하기 때문에 속도가 비교적 느리다는 단점.

자주 사용하지 않는 데이터도 캐시에 올리기 때문에 리소스 관리 면에서 효율성이 떨어진다는 단점.

* 그래서 캐시에 저장 시, 몇 분, 몇 시간 동안만 사용하겠다는 Time out을 설정해야 한다.
* Data Type

1. Strings: 가장 기본적인 데이터 타입. Set command로 저장되는건 모두 이것
2. Bitmaps: String의 변형. bit단위 연산 가능.
3. Lists: 데이터를 순서대로 저장. Queue로 사용하기 적절.
4. Hash: 하나의 키 안에 여러가지 필드와 벨류로 저장
5. Set: 중복되지 않는 문자열의 집합.
6. Sorted\_Set: Set과 동일. 그러나 정렬 시, 스코어라는 숫자 고유 값을 통해 저장. 스코어가 동일 시, 사전순으로 저장.
7. HiperLogLogs: 많은 데이터를 다룰 때 사용. 중복되지 않은 데이터의 개수를 카운트할 때 사용.
8. Streams: log 저장이 가장 유용.

String: 단순 증감 연산할 때 좋음

Bit: 큰 양의 데이터를 관리할 때 유용하다. (Ex 접속자 수를 비트로 표현, 만명 넘어가도 1.2mb정도)

HiperLogLogs: 저장되는 데이터 개수에 상관없이 모든 값이 12kb고정. 대량의 데이터를 카운팅 하는 것에 유용. 한번 불러온 값을 다시 꺼내 올 수 없음. 데이터를 잠시 보호하는 용도로 쓴다. (방문 ip 개수, crawling 개수, 검색엔진에서 검색 개수)

Lists: 메시지 큐로 사용하기 적합. 블록킹 기능을 제공해, 불필요한 홀딩을 방지 가능.

\*\*\* 캐스 스탬프 \*\*\*

-Redis 프로그래밍 예제 분석 StackExchange.Redis

[참고자료 링크]

1. <https://jacking75.github.io/NET_lib_CloudStructures/>
2. <https://infodbbase.tistory.com/135>
3. <https://firstboos.tistory.com/entry/C-Redis-Client-StackExchangeRedisExtensions>

* Nuget으로 기본적으로 필요한 라이브러리들을 세팅한다.
* 분석한 예제는 2번 링크와 1번 링크에 있는 예제를 분석했다. 학습 순서는 2 번 링크 🡪 1번 링크 순으로 분석하는 것이 쉬움.

핵심 구조체

1. ConnectionMultiplexer: Redis 서버와 연결하기 위한 핵심 구조체로, 내가 실행시킨 Redis 서버의 IP주소와 포트주소를 세팅하여 연결이 가능하다.

Ex) ConnectionMultiplexer con;

con.Connection(“127.0.0.1:”+ 6739) // Resid 서버의 기본 포트 번호는 6739이다.

1. IDatabase: 연결된 Redis 서버로부터 데이터베이스를 가져온 후 담을 수 있다.

Ex) IDatabase db = c.GetDataBase();

이 db를 이용해 Set Get 메소드를 사용하여 여러가지 데이터 타입들을 저장 및 불러올 수 있다.

Ex) db.StringSet(key,value); db.StringGet(key);

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<예시 코드 사진>

스크린샷, 텍스트, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<실행 화면: StringSet을 사용해 Redis에 직접 데이터를 추가해보았다.>

여기 까지가 2번 링크의 예제로 분석할 수 있는 구조체 내용 및 기능

1. RedisConfig : 실장님께서 올려주신 예제(Microsoft 공식 예제)에서부터 등장하는 구조체이다. F12를 통해 해당 정의로 이동해보면 다음과 같다.

텍스트, 멀티미디어 소프트웨어, 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

불러올 Redis 서버의 명칭, 연결 옵션, 데이터베이스 등을 설정할 수 있는 구조체이며, RedisConnection 구조체와 함께 사용된다.

1. RedisConnection: Redis 서버와 연결을 제공하기 위한 구조체로 1)에서 언급된 ConnectionMultiplexer를 멤버로 가지고 있다. 예제에서 호출한 생성자를 보면 다음과 같다.

텍스트, 멀티미디어 소프트웨어, 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Config: 3)의 Config이다.

Converter,Handler: 아직 잘 모르겠음.

Logger: 디버깅을 위한 객체 인 것 같다. 우리는 ZLogger를 사용할 예정이므로 해당 인자에 ZLogger 라이브러리의 인자를 넣을 가능성이 높다.

* Net.Core API 프로젝트 작업 정리 (2023-08-04 ~ 2023-08-06)

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

해당 프로젝트의 경우 Program.cs가 메인에 해당한다. 클라이언트가 보내는 요청에 관해 Controller에서 작업을 수행하고, 이에 관한 반응을 보내준다.

2023-08-06 기준 이메일과 페스워드를 요청하면 DB에 새로운 계정을 등록하는 기능까지 제작이 완료되었다. (AccountDB, AccountController)

우리가 해당 프로젝트를 진행하기 위해서 먼저 정의해야 하는 사항은 다음과 같다.

* Model: 우리가 해당 프로젝트를 구현하면서 필요한 구조체들을 정의하는 것이다. Res, Req의 규격 및 패킷 멤버 변수들의 규칙, 혹은 우리가 여러 DB를 통해 다루는 인 게임 데이터들의 구조체 (유저, 스킬, 몬스터 등).
* Service: 실질적인 기능들을 수행하는 객체들로, Controller에게 해당 객체에 관한 Interface가 주어져 Controller를 통해 제어된다. 그렇기에 구현 시, Interface와 함께 짝을 이루며 구현되는 것이 일반적이다. Ex) AccountDB, IAccountDB(인터페이스)

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* Controller: 실질적인 Service를 제공하는 친구들이다. POST와 같은 REST API를 통해 https 통신을 수행한다. 기본적으로 ControllerBase로부터 파생된다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

-질문 내용 정리

1. 예제를 확인해보면 컨트롤러가 작은 단위의 역할로 분리되어 있음을 확인할 수 있었다. 그런데 LoginController나 AccountController의 경우는 하나의 컨트롤러에서 모두 처리할 수 있도록 구현해도 괜찮을 것 같은데, 이렇게 따로 분류해서 처리하는 이유가 무엇일까?
2. Redis 예제 중 RedisConfig에 들어가는 Name의 구체적인 역할이 무엇인가?
3. NHN에서 제공하는 Redis 영상을 시청했을 때, 캐싱 전략에 관한 내용이 있었는데,

읽기 전략 중 Look-Aside과 Look\_Through 전략 두 가지가 있었다. Look-Aside에서 Redis가 다운되는 경우, 캐시를 새로 투입하거나 DB에만 새로운 데이터를 저장한 경우, 부하가 발생한다고 했다. 실제 이런 상황이 발생했을 때 곤란했던 경우가 무엇인가? Ex) r각 데이터베이스에 들어간 정보에 따라서 상황이 다를 것 같다.

1. Redis 쓰기 전략에서 Write Around 전략이, 캐시와 db의 내용이 다르다는 단점을 고려하더라도 훨씬 안정성이 높아 보인다. 여기서 또 다른 단점으로는 캐시 워밍이 자주 일어날 수 있다는 점인데, 그렇다면 캐시 워밍의 비용은 정확히 얼마나 되는 것인가?
2. Program.cs의 이해도 부족으로 인한 어려움이 있는데, 혹시 다른 참고 자료가 더 있을지 여러가지 예제를 보는데, 다들 Program.cs에 세팅된 설정이 다르더라, 그런데 나 같은 경우 빌드를 할 때 SwaggerUI 라는 OpenApi를 사용해서 사이트의 UI를 설정했는데 이렇게 할 경우 Controller의 충돌이 발생하는 경험을 하게 되었다. 그렇다고 SwaggerUi를 사용하지 않자니, 웹사이트가 뜨지 않는 문제점이 있었는데 이 이유가 무엇인지
3. 우리가 현재 작업하고 있는 ASP.NET.Core 프로젝트의 경우는 클라이언트와 서버가 함께 존재하고 있는 구조라고 보는 것인가?
4. Trasient 보다 Scope가 훨씬 좋은 것 같은데, 예제에서는 Transient로 선언한 이유가 무엇인가?