ASP.NET과 Redis로 랭킹 서버 만들기 - 2

강좌는 총 세편으로 이루어져 있습니다.

[ASP.NET과 Redis로 랭킹 서버 만들기 - 1](http://lab.gamecodi.com/board/zboard.php?id=GAMECODILAB_Lecture_series&no=130&z=0)

[ASP.NET과 Redis로 랭킹 서버 만들기 - 2](http://lab.gamecodi.com/board/zboard.php?id=GAMECODILAB_Lecture_series&no=131&z=0)

[ASP.NET과 Redis로 랭킹 서버 만들기 - 3](http://lab.gamecodi.com/board/zboard.php?id=GAMECODILAB_Lecture_series&no=132&z=0)

강좌 순서 입니다.

1. Redis

1.1 윈도우용 Redis설치

1.2 클라이언트 테스트

2. 어플리케이션 구축

2.1 랭킹 구현을 위한 Redis 명령어 알아보기

2.2 ASP.NET프로젝트 구축

2.3 웹 서비스를 위한 IIS설정

2.4 API 테스트

2.5 유니티 클라이언트 테스트

3. 결론

3.1 조금 더 보완할 점

3.2 참고 자료

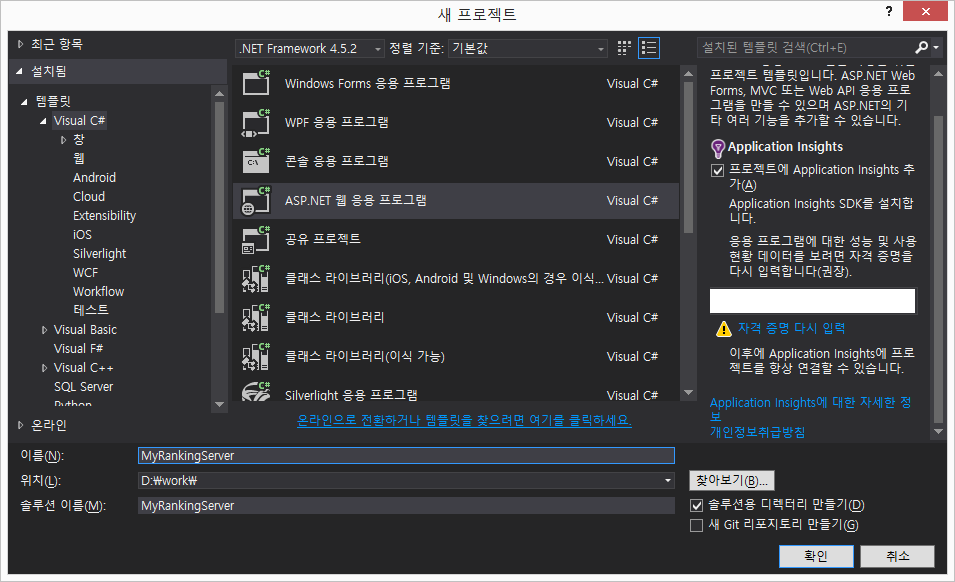
2.2 ASP.NET프로젝트 구축

지금부터 레디스와 통신할 서버 어플리케이션을 만들어 볼 것입니다.

ASP.NET을 활용하여 웹서버 방식으로 구축해보려 합니다.

visual studio 2015를 이용해서 기본적인 프로젝트의 틀을 생성해 보겠습니다.

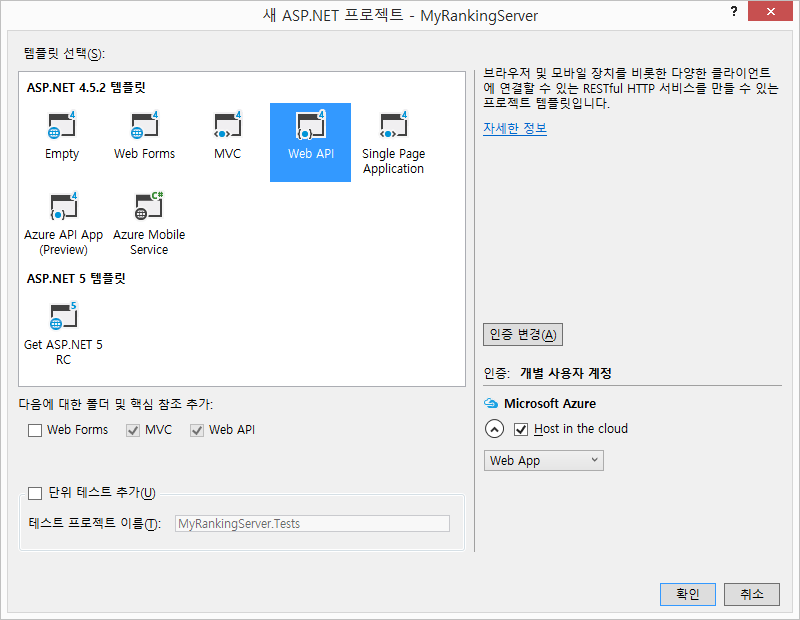
2.2.1 프로젝트 환경 구성



새 프로젝트 -> Visual C# -> ASP.NET 웹 응용 프로그램을 선택하고

MyRankingServer란 이름으로 프로젝트를 생성합니다.

이름은 아무거나 입력하셔도 됩니다.



다음 화면에서 Web API를 선택합니다.

그림에 나온 설명대로 RESTful API를 만들고 서비스할 수 있는 프로젝트입니다.

일반적으로 모바일 게임에서 웹서버와 연동을 한다고 하면 클라이언트에서 uri를 호출하는 방식을 많이 사용하는데

WebAPI가 바로 그런 기능에 딱 맞는 프로젝트 템플릿 이라고 할 수 있습니다.

이제 ASP.NET을 사용하는 기본 웹 프레임워크가 만들어 졌습니다.

여기에 클라이언트의 요청을 받는 부분과 레디스 명령어를 호출하는 로직을 넣을 것입니다.

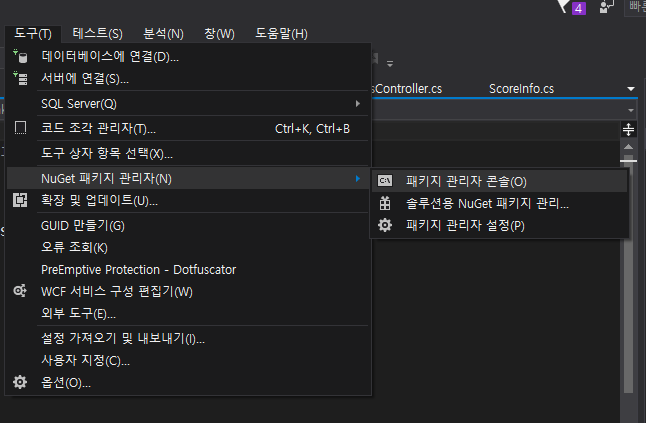
먼저 레디스 명령어를 호출하고 결과값을 받는 RedisManager를 생성해 보겠습니다.

첫 번째로 할 일은 레디스 클라이언트 라이브러리를 프로젝트에 추가하는 것입니다.

우리는 닷넷프레임워크를 사용하고 있기 때문에 닷넷용 레디스 클라이언트를 추가해 보겠습니다.

여러 가지 라이브러리들이 있지만 제일 많이 사용하는 stackexchange.redis로 결정했습니다.

도구 – NuGET 패키지 관리자 -> 패키지 관리자 콘솔을 눌러 아래 명령어를 입력합니다.



Install-Package StackExchange.Redis

잠시 후 라이브러리가 다운로드 되고 프로젝트에 자동으로 추가 되는 것을 확인할 수 있습니다.

아래는 stackexchange.redis의 github 페이지 주소입니다.

<https://github.com/StackExchange/StackExchange.Redis>

인터넷에 공개된 레디스 클라이언트 라이브러리를 사용하지 않고 자체적인 클라이언트를 만들어서 레디스와 통신하는것도 가능합니다.

레디스 공식 사이트에 프로토콜 규칙도 나와 있으므로 TCP소켓 통신을 이용해서 직접 구현해 보는것도 좋겠네요.

2.2.2 레디스 매니저 작성

레디스와 연결할 클라이언트 라이브러리까지 준비가 완료되었습니다.

이제 레디스 명령어들을 사용하여 랭킹에 필요한 기능들을 구현할 차례입니다.

총 세 가지 기능을 구현할 계획입니다.

첫째. 유저의 점수를 등록한다.

둘째. 랭킹 리스트를 가져온다.

셋째. 유저의 순위를 가져온다.

레디스와 관련된 기능들은 RedisManager라는 클래스를 만들어서 구현할 것입니다.

아래 순서대로 RedisManager.cs파일을 생성 합니다.

프로젝트에서 마우스 우클릭 - 추가 – 폴더 – Redis 폴더 생성

Redis폴더에서 마우스 우클릭 – 추가 – 클래스 – RedisManager.cs 생성

● 꼭 Redis폴더 안에 넣을 필요는 없습니다.

먼저 레디스 커넥션을 생성합니다.

·미리보기 | 소스복사·

1. using StackExchange.Redis;
2. namespace MyRankingServer.Redis
3. {
4. using Models;
6. public class RedisManager
7. {
8. static ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost");
9. }
10. }

유저의 점수를 등록하는 매소드는 zincrby 라는 명렁어를 통해서 구현합니다.

클라이언트에서는 SortedSetIncrement라는 매소드가 zincrby명령어와 매칭되는 매소드입니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 점수를 등록한다.
3. /// </summary>
4. /// <param name="nickname"></param>
5. /// <param name="score"></param>
6. public static void add\_score(string nickname, int score)
7. {
8. IDatabase db = redis.GetDatabase();
10. // 레디스 명령어 zincrby 에 해당하는 매소드를 호출한다.
11. // rank-bestscore 랭킹에 유저의 점수를 등록한다.
12. // 이미 등록되어 있다면 기존 점수에 추가한다.
13. db.SortedSetIncrement("rank-bestscore", nickname, score);
14. }

두 번째 구현은 랭킹 리스트를 가져오는 부분입니다.

zrevrange에 해당하는 매소드로 SortedSetRangeByRankWithScores를 호출합니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 상위 랭킹 리스트를 가져온다.
3. /// </summary>
4. /// <returns></returns>
5. public static List<ScoreInfo> get\_top\_rankings()
6. {
7. IDatabase db = redis.GetDatabase();
9. // 레디스 명령어 zrevrange에 해당하는 매소드를 호출한다.
10. // Order.Descending 파라미터를 통해서 높은점수-낮은점수 순으로 가져온다.
11. SortedSetEntry[] list = db.SortedSetRangeByRankWithScores("rank-bestscore", 0, 9, Order.Descending);
13. // 가져온 데이터들을 ScoreInfo형식으로 변환하여 리턴한다.
14. List<ScoreInfo> ranks = new List<ScoreInfo>();
15. for (int i = 0; i < list.Length; ++i)
16. {
17. ranks.Add(new ScoreInfo { nickname = list[i].Element, score = (int)list[i].Score });
18. }
19. return ranks;
20. }

마지막으로 유저의 순위를 가져오는 기능입니다.

레디스 명령어 zrank에 해당하는 매소드는 SortedSetRank입니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 유저의 순위를 가져온다.
3. /// 순위에 없을 경우 0을 리턴한다.
4. /// </summary>
5. /// <param name="nickname"></param>
6. /// <returns></returns>
7. public static long get\_user\_rank(string nickname)
8. {
9. IDatabase db = redis.GetDatabase();
11. // 레디스 명령어 zrank에 해당하는 매소드를 호출한다.
12. // Order.Descending 파라미터를 통해서 가장 높은 점수가 1등으로 처리되도록 한다.
13. long? rank = db.SortedSetRank("rank-bestscore", nickname, Order.Descending);
15. // 정상적인 값이 들어있다면 순위를 리턴한다.
16. if (rank.HasValue)
17. {
18. return rank.Value;
19. }
21. // 만약 아직 등록되지 않은 유저의 점수를 요청한 경우라면 -1을 리턴한다.
22. return -1;
23. }

랭킹 정보를 담을 ScoreInfo클래스는 Models 폴더에 ScoreInfo.cs라는 파일명으로 생성해 줍니다.

·미리보기 | 소스복사·

1. namespace MyRankingServer.Models
2. {
3. public class ScoreInfo
4. {
5. public string nickname { get; set; }
6. public int score { get; set; }
7. }
8. }

이 클래스는 랭킹 리스트를 가져올 때 뿐만 아니라 클라이언트로부터 점수를 등록을 요청받을 때도 사용할 것입니다.

2.2.3 클라이언트의 요청 처리하기 - RankingsController생성

레디스 매니저를 통해서 레디스와 통신할 준비는 모두 끝났습니다.

이제 클라이언트로부터 적절한 API를 요청받아 처리하는 부분을 작성해 보겠습니다.

ASP.NET MVC프로젝트 환경에서는 클라이언트의 요청을 받아들이는 부분을 컨트롤러 라고 부릅니다.

MVC에서 M = 모델,  V = 뷰,  C = 컨트롤러를 의미합니다.

모델은 데이터가 정의되어 있는 영역이며

뷰는 클라이언트 화면에 보여지는 부분

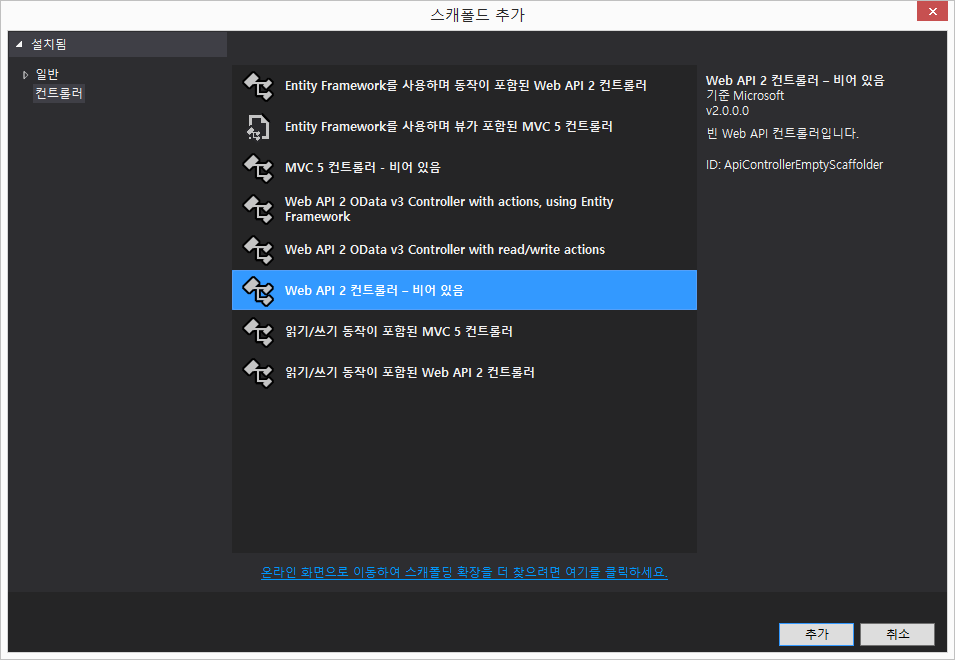
컨트롤러는 요청을 받아서 처리하는 부분을 뜻합니다.

이 프로젝트에서는 API요청을 받아 결과값을 문자열로 응답해주는 로직만 있으면 되기 때문에 뷰에 관한 코딩은 구현하지 않을 것입니다.

Controllers폴더에서 마우스 우클릭 – 추가 – 컨트롤러를 선택합니다.

Web API 2 컨트롤러 – 비어있음을 선택하고

컨트롤러 이름은 RankingsController 라고 입력합니다.



유저의 요청도 세 가지 기능으로 정의할 수 있습니다.

첫째. 랭킹 리스트를 가져온다.

둘째. 유저의 순위를 가져온다.

셋째. 유저의 점수를 등록한다.

각각의 기능들은 앞서 구현했던 레디스 매니저의 기능과 1:1로 매칭 됩니다.

랭킹 리스트를 요청하는 부분입니다.

GET 방식으로 요청합니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 상위 랭킹을 가져온다.
3. /// </summary>
4. /// <returns></returns>
5. public IEnumerable<ScoreInfo> GetTopRankings()
6. {
7. return RedisManager.get\_top\_rankings();
8. }

유저의 순위를 가져오는 부분입니다.

마찬가지로 GET방식으로 요청하는데 뒤에 nickname이라는 파라미터를 붙여서 호출합니다.

매소드 파라미터로 string nickname을 넣었는데 클라이언트에서 호출할 때 nickname을 키로 하는 파라미터를 붙이면 자동으로 이 매소드가 호출됩니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 유저의 순위를 가져온다.
3. /// 호출방식 : /api/rankings?nickname={nickname}
4. /// </summary>
5. /// <param name="nickname"></param>
6. /// <returns></returns>
7. public long GetUserRank(string nickname)
8. {
9. return RedisManager.get\_user\_rank(nickname);
10. }

점수 등록 요청 부분입니다.

점수 등록은 post방식으로 form데이터를 받아 처리하도록 하였습니다.

form데이터 항목들은 ScoreInfo클래스에 들어있는 프로퍼티들과 매칭되어 작동합니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 점수를 등록한다.
3. /// </summary>
4. /// <param name="scoreinfo"></param>
5. /// <returns></returns>
6. [HttpPost]
7. public IHttpActionResult Post(ScoreInfo scoreinfo)
8. {
9. RedisManager.add\_score(scoreinfo.nickname, scoreinfo.score);
10. return Ok();
11. }

여기까지 유저의 요청을 처리하는 컨트롤러를 구현해 봤습니다.

온라인 게임 서버를 개발하시다가 웹서버 로직을 처음 접해본 분이라면 소켓을 생성하고 리슨하는 부분이 어디인지 궁금해 하실텐데

그런 부분들은 IIS라고 불리우는 웹서버에 이미 다 구현되어 있습니다.

따라서 우리는 클라이언트의 요청이 어플리케이션으로 넘어온 이후부터의 로직만 작성하면 됩니다.

코드에 오류가 없는지 확인하기 위해 빌드 – 솔루션 빌드를 클릭하여 프로젝트를 빌드해 봅시다.