# 일반

* **“의존성을 줄이기”** 위해 상속보다는 interface를 주로 활용하자
  + 우선 interface 부터 사용하고 추후 상속 할지 고민

| public class PlayerPlayTrackService  {  ...  public IPlayTrack Create(PlayTrackType type)  {  if (type.IsAttain)  {  return new PlayTrackAttain(this, type);  }  else  {  return new PlayTrackAccrue(this, type);  }  }  // 공통으로 사용하는 것들은 여기에 선언 |
| --- |

| public interface IPlayTrack  {  void Apply(long amt);  long GetTotal(long playerUid);  void SaveIfChanged(long playerUid);  long GetVariable();  public long GetRecord(long playerUid);  } |
| --- |

* **작업을 시작하기 전에 unit test 항목들을 먼저 기획서 보고 확인 후 작업하자**
* 객체 생성 함수명은 of로 생성
  + make 종류는 of 를 사용하여 진행
  + 객체 생성 함수명은 of로 생성
  + of 는 bool을 리턴하지 않는다. **-> 객체로만 리턴**

| public static GroupOption Of(long creatorUid, TGroupOption tOption)  {  var option = OfTGroupOption(tOption);  option.CreatorUid = creatorUid;  return option;  }  public static GroupOption OfTGroupOption(TGroupOption option)  {  return new GroupOption()  {  Name = option.Name,  Password = option.Password,  Public = option.Public,  RequiredLevel = option.RequiredLevel,  Looting = GroupLootingType.OfValue(option.Looting)    };  }  ~~public static bool OfValidate(long creatorUid, TGroupOption tOption)~~  ~~{~~  ~~return true;~~  ~~}~~ |
| --- |

* 객체 변환은 **Parse**

| *// 아이템 보상 지급용 json을 얻어온다. - ex)1001:1,1002:1,1003:1*  public IList<Tuple<RItem, int>> ParseFromItemQty(string rewardCommand)  {  if (string.IsNullOrEmpty(rewardCommand))  {  return ImmutableList<Tuple<RItem, int>>.Empty;  }  *// 1234:1, 2123:1*  *// (\d+):(\d+) => 이걸로 전부 다 찾아서, 그룹 변수 뽑으면 되요*  var createdItemPattern = @"\s\*,\s\*";  var subPattern = @"\s\*:\s\*";  var createdItems = Regex.Split(rewardCommand, createdItemPattern);  ...  return goods;  } |
| --- |

* if 걸때 바로 return 값을 검사하는 방식이 아닌 변수 하나에 리턴 값 받아오는 방식으로 진행

| bool has = \_constructors.TryGetValue(resObj.Type, out var constructor);  if (!has)  return null;  return constructor.Invoke(resObj, sz); |
| --- |

* string형은 readonly 대신 const 사용
  + string 은 클래스 이므로 상수로 인식해야 한다.

| public const string DefaultItems = @"{}"; *// 빈 아이템값 수동으로 db에 업데이트 해주어야 한다.* |
| --- |

* class 에 맞는 역활을 하게 한다.

| public class CouponSystem : XSystem  {  private static readonly ILoggerEx Logger = LoggerFactoryEx.CreateLogger<CouponSystem>();  ~~public static readonly long UnitTestScratchCouponId = CouponType.Scratch.Value;~~  ~~public static readonly long UnitTestPublicCouponId = CouponType.Public.Value;~~ |
| --- |

| public class CouponMetaOption  {  public const string **DefaultItems** = @"{}"; *// 빈 아이템값 수동으로 db에 업데이트 해주어야 한다.*  public static readonly int DefaultCouponLength = 12;  public string Name { get; set; }  public string Desc { get; set; }  public CouponType Type { get; set; }  public string Seed { get; set; }  public DateTime StartAt { get; set; } = DateTime.MinValue;  public DateTime EndAt { get; set; } = DateTime.MinValue;  public int Length { get; set; }  public string Items { get; set; } = **DefaultItems**;  public static CouponMetaOption OfEmpty()  {  return new CouponMetaOption()  {  Name = string.Empty,  Desc = string.Empty,  Type = CouponType.Scratch,  Seed = string.Empty,  Length = 0,  };  }  } |
| --- |

* interface상속해서 사용하는 부분

| interface ICouponRedeemable  {  *// 쿠폰 사용*  int Redeem(DatabaseSystem db, string couponCode, long playerId, long couponId);  } |
| --- |

| public class CouponPublicRedeemable : ICouponRedeemable  {  *// 쿠폰 사용*  public int Redeem(DatabaseSystem db, string couponCode, long playerId, long couponId)  { |
| --- |

| public class CouponScratchRedeemable : ICouponRedeemable  {  *// 쿠폰 사용*  public int Redeem(DatabaseSystem db, string couponCode, long playerId, long couponId)  {  } |
| --- |

* null 오류는 항상 다음과 같은 오류로 return

| public int WriteToMail(long playerId, string senderName, string text, IList<TItem> items)  {  if (items == null)  {  Logger.Error($"mail reward itmes is null - playerUid({playerId}) senderName({senderName})");  return StatusCodeEx.**InvalidArgument**;  } |
| --- |

* 빈 객체 생성

| public class MailOption  {  public IList<TItem> AttachedItem { get; set; } = ImmutableArray<TItem>.Empty;  return new MailOption  {  SenderId = senderId,  SenderName = senderName,  ReceiverId = receiverId,  Text = text,  AttachedItem = items?? ImmutableArray<TItem>.Empty,  }; |
| --- |

* Linq에서 row => 라는 표현 되신 x => 라는 표현으로 진행
* 동사는 뒤에 붙이는 식으로

~~UseItem~~ => ItemUsed

* 아이템 사용에 대한 근거 Type

| public static ItemOriginFrom None { get; } = new ItemOriginFrom(0, "모름");  public static ItemOriginFrom Cheat { get; } = new ItemOriginFrom(1, "치트");  public static ItemOriginFrom Hunting { get; } = new ItemOriginFrom(2, "사냥");  public static ItemOriginFrom Reward { get; } = new ItemOriginFrom(3, "보상");  public static ItemOriginFrom ShopBuy { get; } = new ItemOriginFrom(4, "상점구매");  public static ItemOriginFrom ShopSell { get; } = new ItemOriginFrom(5, "상점판매");  public static ItemOriginFrom ShopTest { get; } = new ItemOriginFrom(6, "상점테스트");  public static ItemOriginFrom ItemEnhance { get; } = new ItemOriginFrom(7, "아이템 강화");  public static ItemOriginFrom ChangeJob { get; } = new ItemOriginFrom(8, "직업 변경");  public static ItemOriginFrom ItemEnhanceRelocate { get; } = new ItemOriginFrom(9, "아이템 강화 계승");  public static ItemOriginFrom Mail { get; } = new ItemOriginFrom(10, "메일");  public static ItemOriginFrom ItemUsed { get; } = new ItemOriginFrom(11, "아이템 사용"); |
| --- |

* 하나에서 여러개의 역활을 하는 방향으로 진행

| *// 골드, 붉은 보석 소모*  temp.Add(ObjectiveType.UseCurrency,  (res, progress, rSys) => new UseCurrencyObjectiveProgress(res, progress?.UseGold, rSys)); |
| --- |

* 아래와 같이 작업함으로써 많은 것을 쉽게 조절할 수 있다.

| private static Dictionary<ItemSubType, ImmutableArray<PlayTrackType>> ItemUsageToPlayTrack { get; } = new Dictionary<ItemSubType, ImmutableArray<PlayTrackType>>()  {  { ItemSubType.MpPotion, ImmutableArray.Create(PlayTrackType.ItemUseMpPotion) },  { ItemSubType.HpPotion, ImmutableArray.Create(PlayTrackType.ItemUseHpPotion) },  };  public static ImmutableArray<PlayTrackType> ToItemUsageTracks(ItemSubType type)  {  return ItemUsageToPlayTrack.TryGetValue(type, out var tracks)  ? tracks  : ImmutableArray<PlayTrackType>.Empty;  } |
| --- |

* 이벤트 보내는 이는 받는 이를 염두해 두지 말고 하자

| var args = new FxPlayerWalletSpent  {  ~~SubTYpe = OfSubType(walletType)~~  Type = walletType,  Amt = amt,  };  \_player.Flux.Emit(args); |
| --- |

* flux, objectiveType 은 최소한으로 줄이자
* 추상화를 이용하여 코드 길이를 줄이고 단순화 시키자.

| public interface IAchievementStorage  {  public int Insert(DatabaseSystem db, long accountId, long playerId, int achievementId, string progress, AchievementState state);  } |
| --- |

* 서버에서 가지고 있는 list양을 줄여 가지고 있는 db조회를 자주 하더라도 List 메모리 양을 줄이자
  + db 의존성을 높여서 작업하자
  + 서버는 최대한 단순하게 작업하자
* RActor 에서 HashSet을 사용하지 말고 ImmurableHaset을 이용하자.
* TryGet은 실패도 존재해야 한다. 단순히 값만 가져오려면 Get이 맞다.
* 아이템 강화 같이 아이템 관련 정보 변경은 바로 수정하면 안 됩니다. Clone으로 복제 후 작업 해주세요
* 변수 초기화 시, readonly 걸고 생성자에서 주로 초기화
* List Id 형식으로 보내지 말고 아래와 같이 변환 후 리스트로 전달

| var trackTypes = req.TrackTypes  .Select(PlayTrackType.OfValue)  .Where(x => x != PlayTrackType.None)  .ToList(); |
| --- |

* in 은 IList로 out은 List로

| var tracks = Player.Track.ToTPlayTracks(trackTypes);  public List<TPlayTrack> ToTPlayTracks(IList<PlayTrackType> values)  {  var tracks = new List<TPlayTrack>();  foreach (var trackType in values.Where(x => x != PlayTrackType.None))  {  var amt = GetTotal(trackType);  tracks.Add(new TPlayTrack  {  TrackType = trackType.Value,  Amt = amt  });  }  return tracks;  } |
| --- |

* int list를 바로 넘기지 말고 type으로변환 후에 넘기자

| var trackTypes = req.TrackTypes  .Select(PlayTrackType.OfValue)  .Where(x => x != PlayTrackType.None)  .ToList();  var tracks = Player.Track.ToTPlayTracks(trackTypes); |
| --- |

* 항상 early return 방식으로 처리
* 빈 list는 다음 함수를 이용하자

| return ImmutableArray<int>.Empty; |
| --- |

## contant 정보들 저장

| public sealed class CommonConstants  public static class ItemRule |
| --- |

## 

# 많이 쓰는 오류 코드(StatusCodeEx)

| public const int **Success** = 0;  public const int Error = 1;  public const int InvalidArgument = 1142; *// 아귀먼트 에러 (일반적으로 null오류시)*  *public const int DbAffectedError = 5; // db관련 실패* |
| --- |

## InvalidArgument

| if (option == null)  {  Logger.Error($"failed to insert coupon meta data - option(null)");  return StatusCodeEx.**InvalidArgument**;  } |
| --- |

## DbAffectedError

| int affected = db.Game.PlayerQuest  .UpdateStateAsync(\_player.Psn.Uid, questId, QuestState.Progress.Value, QuestState.Completed.Value, null)  .Result;  if (0 >= affected)  return StatusCodeEx.DbAffectedError; |
| --- |

# UnitTest

* 진행한 unit test

CouponMetaOptionTests

CouponSystemTests

PlayerQuestComponentTests

* UnitTest는 실패 체크를 한후, 마지막에 성공 체크를 한다.
  + 함수를 일부러 실패 시키고 그 실패를 성공과 비교하는 로직으로 작업

| [Test]  public void TestCreateCoupon()  {  *// 비정상 테스트*  var status = TestWorld.Coupon.Create(null, out var \_);  Assert.That(status, Is.EqualTo(StatusCodeEx.**FailedToAddCouponOfNull**));    *// 정상 테스트*  *// public 쿠폰*  var seed = Guid.NewGuid().ToString();  var publicOptoin = new CouponMetaOption  {  Name = "public coupon",  Desc = "from unit test",  Type = CouponType.Public,  Seed = seed,  StartAt = new DateTime(2023, 1, 1),  EndAt = new DateTime(2023, 12, 31),  Length = CouponMetaOption.DefaultCouponLength  };  status = TestWorld.Coupon.Create(publicOptoin, out var \_);  Assert.That(status, Is.EqualTo(StatusCodeEx.**Success**));  } |
| --- |

* new 로 객체를 만들때 매번 만들지 말고 함수로 하나 만들자

| ~~var publicOptoin = new CouponMetaOption~~  ~~{~~  ~~Name = "public coupon",~~  ~~Desc = "from unit test",~~  ~~Type = CouponType.Public,~~  ~~Seed = seed,~~  ~~StartAt = new DateTime(2023, 1, 1),~~  ~~EndAt = new DateTime(2023, 12, 31),~~  ~~Length = CouponMetaOption.DefaultCouponLength,~~  ~~Items = rewardItems~~  ~~};~~  var name = $"cheat coupon";  var desc = $"create as cheat";  var couponMetaOption = CouponMetaOption.OfDefault(type, name, desc, seed, rewardJson); |
| --- |

| public static CouponMetaOption OfDefault(CouponType type, string name = "Empty", string desc = "Empty Desc", string seed = "EmptySeed", string rewardItem = @"{}")  {  var startAt = DateTime.UtcNow.Date; *// 00시 00분 00초*  var endAt = startAt + TimeSpan.FromDays(7); *// 무조건 1주일 이벤트*    return new CouponMetaOption  {  Name = name,  Desc = desc,  Type = type,  StartAt = startAt,  EndAt = endAt,  Seed = seed,  Length = **DefaultCouponLength**,  Items = rewardItem  };  } |
| --- |

* flux 테스트 하기 위해서는 update(0)을 사용

| *// player.flux에 emit된 액션 실행*  ~~Task.Delay(100).Wait();~~  player.Update(0.1f); |
| --- |

* UnitTest에서 사용하는 Resource들은 ResourceDummySystem에 추가 하자.
* 각 테스트는 배타성을 가지고 있어야 합니다.
  + 하나의 테스트에서 너무 많은 일을 하면 안됩니다.

# 로그

* 로그 남길 때, 상수영역과 변수영역으로 나뉘어야 하고, 예제는 다음과 같습니다
  + 상수부는 소문자로 구성
  + 변수부는 카멜케이스로 $"({변수})" 형태로로 남깁니다.
  + 상수부와 변수부는 - 로 구분합니다.

| Logger.Error($"skill coll time is over - playerUid({\_actor.Uid}) skillId({skillId})"); Logger.Info($"coupon code generating prepared - couponUid({couponId}) count({begin + count}) added({count})"); |
| --- |

* 유저 행위를 남길 때는 반드시 playerUid 를 남길 것

| Logger.Error($"skill coll time is over - playerUid({\_actor.Uid}) skillId({skillId})"); |
| --- |

* out으로 errorMessage 방식은 사용하지 말고 바로 Logger.Error 로 찍자

| ~~error = $"skill coll time is over - playerUid({\_actor.Uid}) skillId({skillId})";~~ Logger.Error($"skill coll time is over - playerUid({\_actor.Uid}) skillId({skillId})"); return false; |
| --- |

# SQL 관련

* 바로 업데이트 하면 안되고 plus 방식이나 아래 같이 안전코드를 추가하자

| var sql = @"UPDATE `player\_play\_tracks`  SET `amt` = `amt` + @amt, `updated\_at` = @updated\_at  WHERE `player\_id` = @player\_id AND `track\_type` = @track\_type"; |
| --- |

* 최대 수만 저장하려면 다음과 같은 쿼리를 활용하자

| var sql = @"UPDATE `player\_play\_tracks`  SET `amt` = GREATEST(@amt, `amt`), `updated\_at` = @updated\_at  WHERE `player\_id` = @player\_id AND `track\_type` = @track\_type AND `amt` < @comparand"; |
| --- |

* 페이지로 체크 전달

| var sql = @"SELECT \* FROM `friend\_follows`  WHERE `follower\_account\_id` = @follower\_account\_id  ORDER BY friendship DESC  LIMIT @offset, @limit"; |
| --- |

* foreach 방식 select

| public Task<IEnumerable<PlayerPreservedActorRow>> GetByIdsAsync(long world\_id, IEnumerable<long> player\_ids)  {  if (!player\_ids.Any())  {  return Task.FromResult(Enumerable.Empty<PlayerPreservedActorRow>());  }  var sql = @"SELECT \*  FROM `player\_preserved\_actors`  WHERE `player\_id` IN @player\_ids";  var param = new  {  player\_ids,  };  return \_nodes  .AsWorld(world\_id)  .AsEnumerableAsync(conn => conn.QueryAsync<PlayerPreservedActorRow>(sql, param));  } |
| --- |

| public Task<IEnumerable<PlayerPlayTrackRow>> GetTracks(long player\_id, IEnumerable<int> track\_types, int offset, int count)  {  var sql = @"SELECT \* FROM `player\_play\_tracks` WHERE `player\_id` = @player\_id AND `track\_type` IN @track\_types  ORDER BY `id` DESC  LIMIT @count OFFSET @offset";  var param = new  {  player\_id,  track\_types,  count,  offset,  };  return \_nodes  .Node(player\_id)  .DoAsync(conn => conn.QueryAsync<PlayerPlayTrackRow>(sql, param));  } |
| --- |

# Linq 관련

| *// 일단 있는지 확인*  bool has = \_achievements.TryGetValue(id, out var achievement);  if (!has)  {  *// 메모리에 없으면 DB에서 읽어오자.*  var state = storage.GetState(id);  if (state == AchievementState.None)  {  *// db 에 없으면 진행 상태로*  state = AchievementState.Progress;  }  achievement = \_achievements.GetOrAdd(id, new AchievementBase(state.Value));  } |
| --- |

# 구조

* type별 역활에 대해 정하고 너무 많은 걸로 지원하지 말자.

| public class PlayTrackType  {  public static PlayTrackType None { get; } = new PlayTrackType(0, nameof(None), false, ApplyUnit.Character);  public static PlayTrackType EnhanceLvAll { get; } = new PlayTrackType(101, nameof(EnhanceLvAll), true, ApplyUnit.Character);  public static PlayTrackType EnhanceLvWeapon { get; } = new PlayTrackType(102, nameof(EnhanceLvWeapon), true, ApplyUnit.Character);  public static PlayTrackType EnhanceLvAccessory { get; } = new PlayTrackType(103, nameof(EnhanceLvAccessory), true, ApplyUnit.Character); |
| --- |

* xxxService 는 player 가 가지고 있지 말자

# Linq 관련 예제

| var itemTypes = new Dictionary<ItemMainType, ItemSubType>()  {  {ItemMainType.Accessory, ItemSubType.Ring}  }; |
| --- |

| private static readonly ImmutableDictionary<AccumulatedObjectiveType, Func<string, int>> \_targetToValue  = ImmutableDictionary.CreateRange  (  new Dictionary<AccumulatedObjectiveType, Func<string, int>>()  {  { AccumulatedObjectiveType.EnhanceLv, x => ItemMainType.OfName(x).Value},  { AccumulatedObjectiveType.ItemUsage, x => ItemSubType.OfName(x).Value},  { AccumulatedObjectiveType.CharacterLevel, x => JobType.OfName(x).Value},  { AccumulatedObjectiveType.AcquireGold, x => WalletType.OfName(x).Value},  { AccumulatedObjectiveType.MonsterKill, x => MonsterGradeType.OfName(x).Value},  { AccumulatedObjectiveType.QuestClear, x => QuestStoryLine.OfName(x).Value},  { AccumulatedObjectiveType.AttributeAttain, x => AttrType.OfFixedName(x).Idx},  { AccumulatedObjectiveType.PoseAttempt, x => PoseState.OfName(x).Value}  }  ); |
| --- |

## groupBy 예제

| *// 업적 리소스 준비*  var groups = UnitWorld.R.Achievement  .Achievements  .GroupBy(x => x.Spec.GroupId)  .ToDictionary(  x => x.Key,  x => x.OrderBy(y => y.Spec.Step).ToImmutableArray()  );  foreach (var (groupId, rAchievements)in groups) |
| --- |

* 업적 그룹 평가

| *// 모든 그룹 단위로 목록 얻기*  public Dictionary<int, ImmutableArray<RAchievement>> ToAchievementsByOverThanGroupId(int pivotId = 0)  {  var rSys = GetSystem<ResourceSystem>();  return rSys.Achievement.Achievements  .Where(x => pivotId <= x.Spec.GroupId)  .GroupBy(x => x.Spec.GroupId)  .ToDictionary(  x => x.Key,  x => x.OrderBy(y => y.Spec.Step).ToImmutableArray()  );  } |
| --- |

# 변수명 규칙

* comparand
  + "comparand"는 일반적으로 Compare 메서드와 함께 사용되는 매개 변수입니다. Compare 메서드는 두 개의 값을 비교하여 결과를 반환합니다.
  + comparand 매개 변수는 CompareTo 메서드에서 비교 대상이 되는 값입니다.
  + CompareTo 메서드는 기본적으로 이전 값과 비교하여 결과를 반환하지만, comparand 매개 변수를 사용하여 비교 대상을 변경할 수 있습니다.
* apply
  + 밖에서는 어떻게 적용되는지 알 필요가 없다. 안에서 적용이라고 하자.
* conv
  + convertor 역활을 하는 약어
* pivotId
  + 기준이 되는값

| .Where(x => pivotId <= x.Spec.GroupId) |
| --- |