

婚礼

三、功能实现

1. 工程结构

1. 首先

拆分装配接口，一个是装配接口，一个是调度接口。这样可以保持接口的单一职责，避免使用方调用装配操作。装配是活动在创建或者审核的时候初始化的装配动作。

2. 本节

会在实体对象中添加属于实体本身的功能，这样会让代码更加干净，也会体现出充血的作用。

- 等等

记得记得

2. 装配实现

2.1 大家

装配由原本的策略全部装配，扩展为支持权重装配。

2.2 等价

源码: cn.bugstack.domain.strategy.service.armory.StrategyArmoryDispatch
<<http://cn.bugstack.domain.strategy.service.armory.strategyarmorydispatch/>>

源码: cn.bugstack.domain.strategy.service.armory.StrategyArmoryDispatch
<<http://cn.bugstack.domain.strategy.service.armory.strategyarmorydispatch/>>

```

1  @Override
2  public boolean assembleLotteryStrategy(Long strategyId) {
3      // 1. 查询策略配置
4      List<StrategyAwardEntity> strategyAwardEntities = repository.queryStrategyAwardList(strategyId);
5      assembleLotteryStrategy(String.valueOf(strategyId), strategyAwardEntities);
6      // 2. 权重策略配置 - 适用于 rule_weight 权重规则配置
7      StrategyEntity strategyEntity = repository.queryStrategyEntityByStrategyId(strategyId);
8      String ruleWeight = strategyEntity.getRuleWeight();
9      if (null == ruleWeight) return true;
10     StrategyRuleEntity strategyRuleEntity = repository.queryStrategyRule(strategyId, ruleWeight);
11     if (null == strategyRuleEntity) {
12         throw new AppException(ResponseCode.STRATEGY_RULE_WEIGHT_IS_NULL.getCode(), ResponseCode.STRATEGY_RULE_WEIGHT_IS_NULL.getMessage());
13     }
14     Map<String, List<Integer>> ruleWeightValueMap = strategyRuleEntity.getRuleWeightValueMap();
15     Set<String> keys = ruleWeightValueMap.keySet();
16     for (String key : keys) {
17         List<Integer> ruleWeightValues = ruleWeightValueMap.get(key);
18         ArrayList<StrategyAwardEntity> strategyAwardEntitiesClone = new ArrayList<>(strategyAwardEntities);
19         strategyAwardEntitiesClone.removeIf(entity -> !ruleWeightValues.contains(entity.getAwardId()));
20         assembleLotteryStrategy(String.valueOf(strategyId).concat("_").concat(key), strategyAwardEntitiesClone);
21     }
22     return true;
}

```

```

}

```

```

@Override
public boolean assembleLotteryStrategy(Long strategyId) {
    // 1. 查询策略配置
    List<StrategyAwardEntity> strategyAwardEntities = repository.queryStrategyAwardList(strategyId);

    // 2. 获取最小中奖率
    BigDecimal minAwardRate = strategyAwardEntities.stream()
        .map(StrategyAwardEntity::getAwardRate)
        .min(BigDecimal::compareTo)
        .orElse(BigDecimal.ZERO);

    // 3. 获取中奖率总和
    BigDecimal totalAwardRate = strategyAwardEntities.stream()
        .map(StrategyAwardEntity::getAwardRate)
        .reduce(BigDecimal.ZERO, BigDecimal::add);

    // 4. 用 1 % 0.0001 获得概率范围，百分位、千分位
    BigDecimal rateRange = totalAwardRate.divide(minAwardRate, 0, RoundingMode.CEILING);

    // 5. 生成策略中奖率搜索表（这里需要把List集合中，存放上对应的奖品占比即可，占比越多等于中奖率越高）
    List<Integer> strategyAwardSearchRateTables = new ArrayList<>(rateRange.intValue());
    for (StrategyAwardEntity strategyAward : strategyAwardEntities) {
        Integer awardId = strategyAward.getAwardId();
        BigDecimal awardRate = strategyAward.getAwardRate();
        // 计算出每个概率值需要存放多少个数据，循环填充
        for (int i = 0; i < rateRange.multiply(awardRate).setScale(0, RoundingMode.CEILING).intValue(); i++) {
            strategyAwardSearchRateTables.add(awardId);
        }
    }

    // 6. 对存储的奖品进行乱序操作
    Collections.shuffle(strategyAwardSearchRateTables);

    // 7. 生成Map集合，key值，对应的就是后端的概率值，通过概率来获得对应的奖品ID
    Map<Integer, Integer> shuffleStrategyAwardSearchRateTable = new LinkedHashMap<>();
    for (int i = 0; i < strategyAwardSearchRateTables.size(); i++) {
        shuffleStrategyAwardSearchRateTable.put(i, strategyAwardSearchRateTables.get(i));
    }
}

```

重构

```

1 usage  ± 小哥哥 *
@Override
public boolean assembleLotteryStrategy(Long strategyId) {
    // 1. 查询策略配置
    List<StrategyAwardEntity> strategyAwardEntities = repository.queryStrategyAwardList(strategyId);
    assembleLotteryStrategy(String.valueOf(strategyId), strategyAwardEntities);

    // 2. 权重策略配置 - 适用于 rule_weight 权重规则配置
    StrategyEntity strategyEntity = repository.queryStrategyEntityByStrategyId(strategyId);
    String ruleWeight = strategyEntity.getRuleWeight();
    if (null == ruleWeight) return true;

    StrategyRuleEntity strategyRuleEntity = repository.queryStrategyRule(strategyId, ruleWeight);
    if (null == strategyRuleEntity) {
        throw new AppException(ResponseCode.STRATEGY_RULE_WEIGHT_IS_NULL.getCode(), ResponseCode.STRATEGY_RULE_WEIGHT_IS_NULL.getMessage());
    }
    Map<String, List<Integer>> ruleWeightValueMap = strategyRuleEntity.getRuleWeightValueMap();
    Set<String> keys = ruleWeightValueMap.keySet();
    for (String key : keys) {
        List<Integer> ruleWeightValues = ruleWeightValueMap.get(key);
        ArrayList<StrategyAwardEntity> strategyAwardEntitiesClone = new ArrayList<>(strategyAwardEntities);
        strategyAwardEntitiesClone.removeIf(entity -> !ruleWeightValues.contains(entity.getAwardId()));
        assembleLotteryStrategy(String.valueOf(strategyId).concat("_").concat(key), strategyAwardEntitiesClone);
    }
    return true;
}

```

1. 如图对于 [第3节：策略概率装配处理 <https://t.zsxq.com/153HMmsal>](https://t.zsxq.com/153HMmsal) 本节对 StrategyArmoryDispatch#assembleLotteryStrategy 代码进行重构，支持权重的策略装配。