实验四报告

# 策略模式

1. 应用场景分析

*描述飞机大战游戏中哪个应用场景需要用到此模式，目前代码实现中存在的问题及使用该模式的优势。*

**应用场景：飞机发射不同类型的子弹，以及切换发射模式时。**

**目前问题：**

**飞机子弹的发射代码存在冗余，且对发射子弹的方式封装的不够好，增添新发射模式时不够灵活。**

**优势：**

**1.提高了代码的重用率，对于客户端避免了重写代码。**

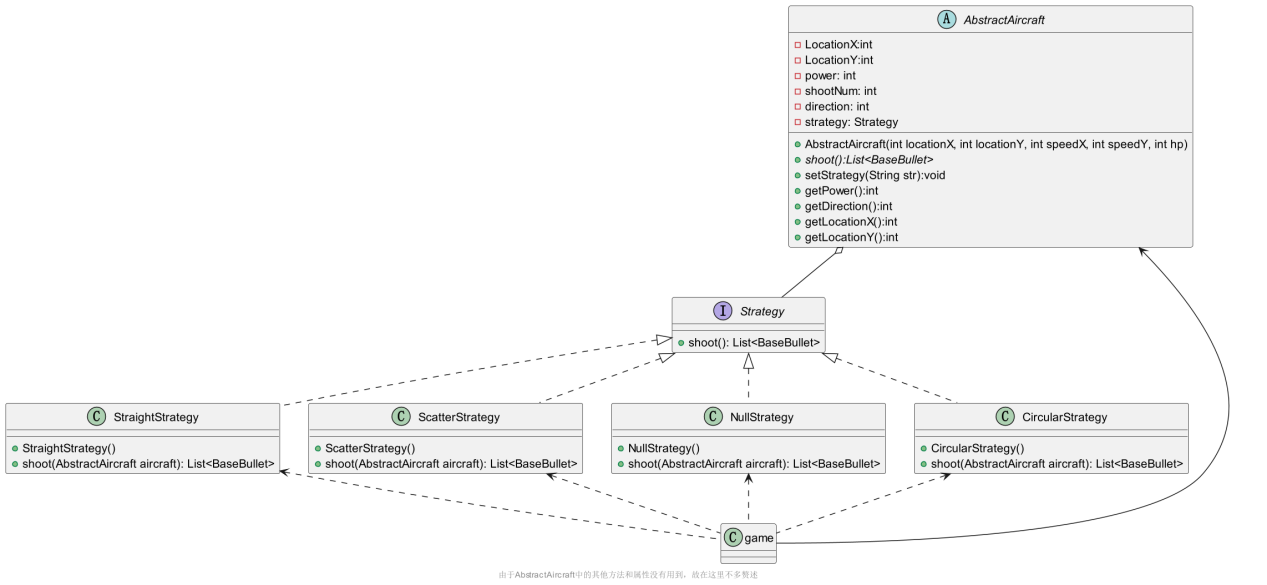
**2.封装性提高，避免重复使用if-else语句，使得代码结构更清晰，可读性更高。**

**3.增加可扩展性，当实现新的发射逻辑时，创建新的策略类即可。**

1. 解决方案

*借鉴策略模式的解题思路，设计解决该场景问题的方案。*

1. *将PlantUML插件绘制的类图截图到此处*
2. *描述你设计的UML类图中的每个角色（类、接口），并对它的关键属性、方法和作用进行简要说明。*



**接口：Strategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)，即要求实现它的类必须实现发射子弹的函数**

**作用：作为每个策略的接口，对发射子弹的策略参数及函数名称进行规范**

**类：NullStrategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)**

**作用：实现不射子弹的逻辑，返回空的子弹列表。**

**类：StraightStrategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)**

**作用：实现直射子弹的逻辑，根据传入飞机的参数，对子弹列表进行初始化并返回**

**类：ScatterStrategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)**

**作用：实现散射子弹的逻辑，根据传入飞机的参数，对子弹列表进行初始化并返回**

**类：CircularStrategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)**

**作用：实现环射子弹的逻辑，根据传入飞机的参数，对子弹列表进行初始化并返回**

**抽象类：AbstractAircraft**

**属性：**

**LocationX:X坐标**

**LocationY:Y坐标**

**Power:子弹伤害值**

**shootNum:子弹数量**

**Direction：相对发射位置**

**Strategy:发射策略**

**方法：**

**AbstractAircraft(int locationX, int locationY, int speedX, int speedY, int hp)构造函数**

**一系列对私有属性的getter和setter方法**

**+ setStrategy(String str):void**

**根据传入的字符串，对策略进行选择。**

**+getPower():int**

**+getDirection():int**

**+getLocationX():int**

**+getLocationY():int**

**发射子弹：**

**shoot():List<BaseBullet>**

**类：game**

**方法和属性不多做说明。**

**作用：作为主方法，实现对各个飞机策略的初始化，以及英雄机各个策略模式之间的切换。**

**game中通过各个飞机对各个具体策略进行使用，与各种具体策略为关联关系。**

# 数据访问对象模式

1. 应用场景分析

*描述飞机大战游戏中哪个应用场景需要用到此模式，分析使用该模式的优势。*

**应用场景：在对游戏的排行榜进行存储以及读取**

**优势：**

**1. 分离了数据访问逻辑和业务逻辑，将数据访问逻辑封装在DAO对象中，使得业务逻辑可以独立于具体的数据访问细节。这样可以降低代码的耦合性，提高代码的可维护性和可测试性。**

**2. 提供了统一的数据访问接口，通过这些接口，可以统一管理数据访问操作，这些一致的数据访问方式降低了代码的复杂性。**

**3. 隐藏了底层数据操作细节，DAO模式将底层的数据访问细节封装在DAO对象中，对上层代码隐藏了数据库的具体实现细节，有助于提高代码的安全性。**

1. 解决方案

*借鉴数据访问对象模式的解题思路，设计解决该场景问题的方案。*

1. *将PlantUML插件绘制的类图截图到此处*
2. *描述你设计的UML类图中的每个角色（类、接口），并对它的关键属性、方法和作用进行简要说明。*



**接口：DAO**

**声明了以下5个方法**

**Record createRecord(String userid, int score);**

**List<Record> getRecordsById(String userid);**

**void updateRecord(Record record);**

**void deleteRecord(String userid);**

**void printRecord();**

**作用：为对数据访问和操作提供一组规范的接口和方式。**

**类：Record**

**共有四个属性：rank,userid,score,time,分别表示：排名，用户名称，分数，记录时间。**

**方法：**

**构造函数:public Record(int rank,String userid,int score,LocalDateTime time)**

**作用：对自身进行初始化。**

**一系列setter和getter方法：**

**getRank()**

**setRank()**

**getId()**

**getScore()**

**setScore()**

**getTime()**

**对上述私有变量分别进行赋值和取值**

**作用：作为一项记录的结构体，对记录的数据进行封装。**

**类:RankList**

**属性：List<Record> ranklist**

**作用：用于暂时存储数据库中全部记录便于增删查改**

**属性：Comparator<Record> comparatorbyscore**

**作用：传入sort函数中的比较器，用于对各项记录进行排序操作。**

**属性： String DataPath**

**作用：记录数据库的存储位置**

**方法：**

**构造函数：**

**RankList()：给数据库路径初始化**

**void loadRecord()：读取csv格式数据文件，并将其解析后存入ranklist变量中**

**void writeRecord()：打开csv格式数据文件，并将当前的ranklist以csv格式写入文件**

**Record parseRecord(String str)：将从csv中读取出的每行字符串格式的记录解析成Record类型的，并返回。**

**Record createRecord(String userid,int score)：调用Record的构造函数，返回Record**

**List<Record> getRecordsById(String id) ：首先调用loadRecord读取数据文件，存入ranklist，遍历ranklist，当id相同时则将其添加到暂存数组，并将其全部返回。**

**void updateRecord(Record record)：首先调用loadRecord读取数据文件，存入ranklist；**

**接着将遍历ranklist中每一项record1，当record1.score>record.score时，将record.rank++，即排名+1;当record1.score<record.score时，将record1的rank++。**

**最终使用writeRecord()将其存回文件。**

**void deleteRecord(String id)：先调用loadRecord()将数据读入ranklist，再调用getRecordsById得到所有给定id的记录。接着遍历ranklist将这些记录删除，并调整其他记录的名次，最后调用writeRecord()将其写入文件。**

**void printRecord() ：调用loadRecord()将数据读入ranklist，再按一定格式将其打印至控制台。**

**void sort(String by)： 按照给定的“by”方法选择不同的比较器，使用collection内置比较函数对ranklist进行排序。**

**类：DAOpatternDemo**

**作用：对DAO的一系列操作进行测试。**