MaNGos Zero——Pool 分析报告

赵睿 2018K8009909001

一．简介

本文是中国科学院大学面向对象课程的源码分析的报告，主要分析了 MaNGos Zero 的 PoolManager 模块，也就是怪兽和可交互物品的管理。

二．关键模块功能分析与建模

2.1 从游戏本身说起

进行源码分析之初，我认为通过阅读代码就可以对 PoolManager 模块有足够的了解和认识。但事实上，由于之前并没有接触过这款游戏，在阅读代码的过程中不同的类、结构体、变量与函数之间错综复杂的关系非常难以分析，于是我便先跳出了代码细节，对这款游戏，尤其是我要分析的部分，做了一个宏观的了解。

首先对代码中经常出现的几个名词进行一个解释：

Gameobject：游戏中的物品。

Creature：本意是生物，这里可以理解为游戏中的怪兽。

Pool：pool 可以理解为联合体，它将在代码中多次出现，不同的定义则表示了多个不同的联合体。(联合体中可能有怪兽、可能有物品)(祖孙联合体联系起了所有的东西)

Map：与它的本意相似，就是这个虚拟世界的地图。

Spawn：它的本意是生成，在游戏中意思与本意相似，就是让怪兽或物品进入到游戏世界。

Event：游戏中的事件。(这个事件可能关联着怪兽、物品、联合体)

至此，结合这些基础的了解，我们可以开始对源码进行分析了。这部分代码的功能比较琐碎，笼统而言，就是对怪兽、物品、联合体进行的一些操作，这些操作包括但不限于初始化、加载、检查、更新、出入游戏世界，但这部分代码本身并没有实现一个明确的、统一的、功能。因此，我想从其中一个具体的功能入手来进行分析。

2.2 从一个功能入手

我选择了初始化这个功能进行分析。这个功能看似简单，但是调用关系复杂，涉及到了代码中出现的全部的类，由它入手进行分析，有助于我们对代码有一个更清楚的认知。

对联合体的初始化方法定义在 PoolManager 类中，我们可以叫它联合体管理器。对 PoolManager 类进行如下建模。

类	联合体管理器（PoolManager）
方法	void Initialize(MapPersistentState* state)初始化联合体 void InitSpawnPool(MapPersistentState& mapState, uint16 pool_id)初始化-生成联合体 void SpawnPool(MapPersistentState& mapState, uint16 pool_id, bool instantly)生成联合体 void SpawnPoolGroup(MapPersistentState& mapState, uint16 pool_id, uint32 db_guid_or_pool_id, bool instantly)
属性	mPoolTemplat：PoolTemplateDataMap 联合体模板 mPoolCreatureGroups：PoolGroupCreatureMap 怪兽联合体 mPoolGameobjectGroups：PoolGroupGameObjectMap 物品联合体 mPoolPoolGroups：PoolGroupPoolMap 锚点池联合体

在联合体管理器中，有初始化联合体的方法，其定义代码如下。初始化联合体时的传递参数为地图的状态。遍历联合体模板 mPoolTemplat，如果生成开始于以 pool_entry 标记的这个联合体，即 AutoSpawn 返回 1，那么调用 InitSpawnPool 方法。

```
void PoolManager::Initialize(MapPersistentState* state)
{
    for (uint16 pool_entry = 0; pool_entry < mPoolTemplate.size(); ++po
ol_entry)
        if (mPoolTemplate[pool_entry].AutoSpawn)
        {
            InitSpawnPool(*state, pool_entry);
        }
}
```

InitSpawnPool 方法同样属于联合体管理器。它的定义代码如下。初始化-生成联合体（InitSpawnPool）时传递参数为地图状态和联合体的 id，并据此判断这个联合体是否可以进入地图，如果可以，调用 SpawnPool 方法。

```
void PoolManager::InitSpawnPool(MapPersistentState& mapState, uint16 po
ol_id)
{
    if (mPoolTemplate[pool_id].CanBeSpawnedAtMap(mapState.GetMapEntry()
))
    {
        SpawnPool(mapState, pool_id, true);
    }
}
```

SpawnPool 方法同样定义在联合体管理器中，代码如下。生成联合体时传递的参数增加了一个布尔变量，表示是否立即生成，在本例中为 1。这个函数的主要作用是要生成的联合体分类（怪兽联合体、物品联合体、锚点池联合体），并根据分类调用不同的函数。

```
void PoolManager::SpawnPool(MapPersistentState& mapState, uint16 pool_i
d, bool instantly)
{
    SpawnPoolGroup<Pool>(mapState, pool_id, 0, instantly);
    SpawnPoolGroup<GameObject>(mapState, pool_id, 0, instantly);
    SpawnPoolGroup<Creature>(mapState, pool_id, 0, instantly);
}
```

以生成怪兽联合体为例继续向下分析。生成怪兽联合体需要用 SpawnPoolGroup <Creature>方法。这一方法同样定义在联合体管理器中，代码如下。生成怪兽联合体时传递的参数在之前的基础上增加了怪兽的 id（db_guid）。检查 mPoolCreatureGroups 容器中 pool_id 位置的怪兽联合体，然后调用 SpawnObject 方法。

```
void PoolManager::SpawnPoolGroup<Creature>(MapPersistentState& mapState
, uint16 pool_id, uint32 db_guid, bool instantly)
{
    if (!mPoolCreatureGroups[pool_id].isEmpty())
    {
```

```
        mPoolCreatureGroups[pool_id].SpawnObject(mapState, mPoolTemplat
e[pool_id].MaxLimit, db_guid, instantly);
    }
}
```

在经过一系列条件判断、分类后，我们终于进入到了核心的生成联合体的流程。SpawnObject 方法虽然从字面意思上看是生成一个怪兽（也可以是物品，此处接上文以怪兽为例），但事实上是生成一个联合体中的所有怪兽。而 Spawn1Object 则是生成一个怪兽。Spawn1Object 与 SpawnObject 方法都定义在 PoolGroup 结构体中。对其建模如下。

类	PoolGroup
方法	void SpawnObject(MapPersistentState& mapState, uint32 limit, uint32 triggerFrom, bool instantly)生成怪兽联合体 void Spawn1Object(MapPersistentState& mapState, PoolObject* obj, bool instantly)生成怪兽
属性	EqualChanced: PoolObjectList ExplicitlyChanced: PoolObjectList poolId: uint32

取 SpawnObject 主要功能部分进行分析，代码如下。从图的状态中获得已经生成的联合体的信息，赋值给一个类指针 spawns。用联合体中能容纳的怪兽数减去已生成的怪兽数，得到联合体还能生成的怪兽数。随机取出一个怪兽，把这个怪兽的信息加到联合体的信息里，把这个怪兽放到地图里。这里需要用到 Spawn1Object 方法。Spawn1Object 实现根据怪兽的信息把怪兽从数据库中加载出来，并放到地图上。

```
void PoolGroup<T>::SpawnObject(MapPersistentState& mapState, uint32 lim
it, uint32 triggerFrom, bool instantly)
{
    SpawnedPoolData& spawns = mapState.GetSpawnedPoolData();
    uint32 lastDespawned = 0;
    int count = limit - spawns.GetSpawnedObjects(poolId);
    if (triggerFrom)
       ……
    for (int i = 0; i < count; ++i)
    {
        PoolObject* obj = RollOne(spawns, triggerFrom);
        if (!obj)
        {continue;}
       ……
        spawns.AddSpawn<T>(obj->guid, poolId);
        Spawn1Object(mapState, obj, instantly);
        if (triggerFrom)
        ……
    }
}
```

至此，我们已经基本了解了初始化联合体这一功能是如何实现的。这里要插一句题外话，在上面关于 SpawnObject 方法的分析中，我们用到了“信息”一次，这在代码中表现

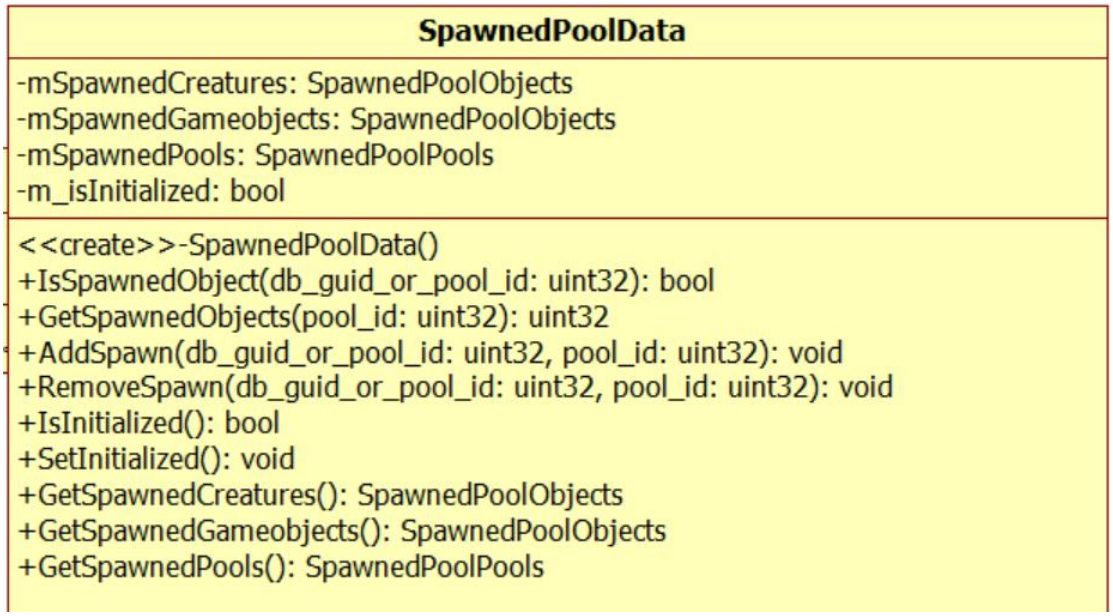
为“id”，引出了 SpawnedPoolData 类，可以叫它已生成的联合体的信息。这个类中的方法主要改变联合体、物品、怪兽的 id，而不改变物品、怪兽、联合体本身，因此这些方法往往要与另外两个类中的方法一同使用。

2.3 类图与模块主要功能

由于这部分代码没有一个统一的功能，因此我将按类分别具体阐述这个模块的功能。

2.3.1SpawnedPoolData 类

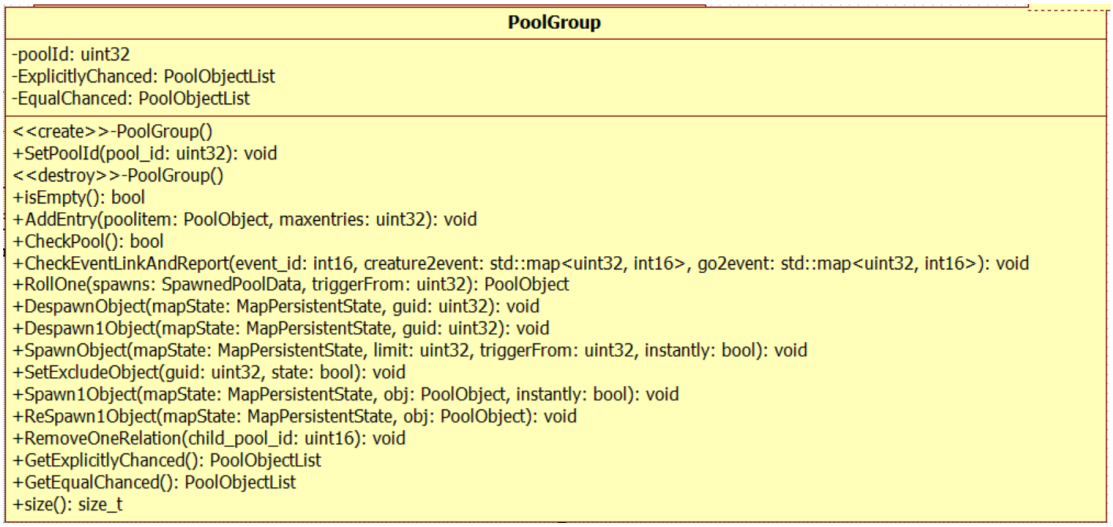
正如上文所述，SpawnedPoolData 这个类中主要提供的就是针对“id”的操作，具体见下面的 UML 类图。



具体而言，其功能有获得联合体的信息，包括已生成对象（物品、怪兽、瞄准池）的数量、是否正在生成、添加生成对象的信息、删除生成对象的信息等。

2.3.2PoolGroup 类

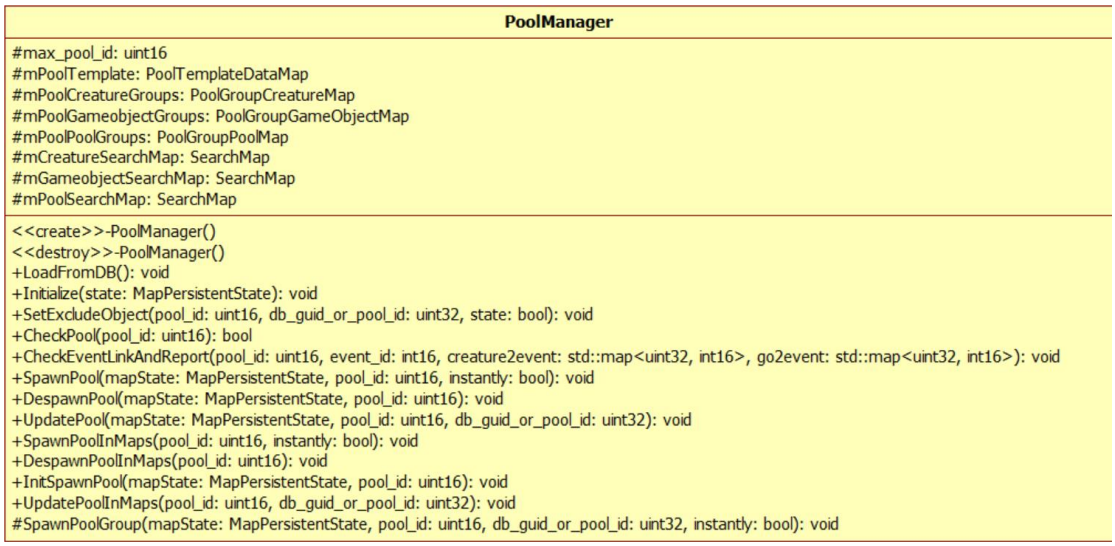
PoolGroup 这个类的主要功能是对联合体中的对象（怪兽、物品、瞄准池）进行一些检查与操作。



具体而言，其功能有：将对象添加到对应的队列中、检查联合体中取出对象的概率是否正确、从联合体中随机取出一个对象、检查事件与对象是否对应、对联合体中的对象进行操作（包括生成、丢弃、重生）等。

2.3.3 PoolManager 类

PoolManager 这个类的功能可以分为三方面，一方面，它建立起联合体与地图的关系，另一方面，它调用 PoolGroup 中的方法实现对联合体的操作。



具体而言，其功能有：从数据库中加载对象与联合体、初始化联合体、生成联合体、检查联合体概率、检查时间链接、更新联合体；把联合体放进地图、从地图上删除、更新地图上的联合体等。