面向对象源码阅读分析报告

姓名：陈璟

学号：2018K8009908046

一、报告主题：MaNGOS Zero Spell

1.1 什么是 MaNGOS Zero？

要谈论 mangos zero，也许我们要先从魔兽世界开始。《魔兽世界》（World of Warcraft）是由著名游戏公司暴雪娱乐所制作的第一款网络游戏，属于大型多人在线角色扮演游戏。游戏以该公司出品的即时战略游戏《魔兽争霸》的剧情为历史背景，依托魔兽争霸的历史事件和英雄人物，魔兽世界有着完整的历史背景时间线。玩家在魔兽世界中冒险、完成任务、新的历险、探索未知的世界、征服怪物等。

现在大多数魔兽世界 的“私服”实际上是服务端软件的模拟器。是在对客户端游戏软件和与服务端的进行通信的数据进行分析解密以后，开发出来的模拟原游戏服务器功能的软件，这样的模拟器软件大都也是开源软件。比如像现在大部分魔兽私服使用的 mangos 服务端软件，mangos 项目是一个开源的自由软件（如同 linux 或者 firefox），并且遵守其中最为严格的 GPL 协议，保证源代码绝对的自由。

MaNGOS 是(Massive Network Game Object Server）的缩写。由于暴雪公司对类似的开发小组采取过一些法律行动, 为了避免麻烦，如同它的名字，mangos 强调自己并非一个魔兽服务器模拟器，而是一个开源的多人在线游戏服务器的软件。说到底是个网游的游戏引擎。mangos 开发小组强调其软件是用 C++和 C#编程语言实现的一个支持大型多人在线角色扮演游戏服务器的程序框架，在这个框架下，它理论上应该支持任何客户端的网络游戏，由于现在很多人使用魔兽世界来对它进行测试，所以针对魔兽世界的脚本和数据库文件比较完善，很多人就利用这个开源项目来实现魔兽私服。而游戏的内容，例如故事情节，任务场景的脚本等都是由别的小组独立开发的。

开发小组一再强调，这是个研究教育性质的对怎样开发大型网游的服务器端有好处的项目，是一个技术细节毫无保留向公众开放的软件，是一件很有意义的事情，如果你使用它作为盈利目的，那你本身就违反了软件的协议。所以任何利用 mangos 项目进行私服活动的组织和个人都违反了 mangos 的宗旨，mangos 项目也不会对它们负责。

mangos 的技术细节上是这样的，核心部分是个和特定游戏没有关系的核心框架程序，主要是

进行进程调度，创造世界，建立心跳机制，处理网络接入等。数据库可以使用的开源数据库软件MySQL，编译器使用的是 GCC。至于游戏内容数据库，游戏人物，时间，世界脚本，都是由这个核心程序所支持的扩展脚本来实现，所以有一些独立出来的项目专门模拟魔兽世界来开发支持mangos 的核心程序。

1.2 什么是 Spell?

Spell(法术)是游戏中玩家角色和 NPC 所使用的技能。主要包括以下几种:

玩家角色的技能书中的技能

NPC, 怪物所使用的技能

药剂和食物以及任何物品带来的 Buff 效果

所有作用在玩家和 NPC 身上的 Buff 和 Debuff

在载具上所能使用的技能

好了，现在我们已经明白了源码阅读报告主题的两大组成部分，MaNGOS Zero 和 Spell 分别代表着什么，接下来将通过功能分析、核心流程、复杂设计意图这几个方面来分别解读这份源码。

二、功能分析

2.1 法术三要素：主体，客体，执行过程

在进一步讨论法术是如何运作之前，我们需要先明确它作用的游戏对象，也就是主体——法术的释放者，和客体——法术的接受者。要搞懂这部分内容，我们需要先明白 mangos 中游戏对象代表着什么。

狭义的游戏对象是指游戏世界中所能看到及可交互的对象，如玩家、怪物、物品等，我们这里也主要讨论这类对象在服务器上的组织及实现。在魔兽世界中，通过服务器发下来的 GUID 我们可以了解到，游戏中有 9 大类对象，包括物品(Item)，背包(Container)，生物(Unit)，玩家(Player)，游戏对象(GameObject)，动态对象(DynamicObject)，尸体(Corpse)等。

在 mangos 的实现中，对象使用类继承的方式，由 Object 基类定义游戏对象的公有接口及属性，包括 GUID 的生成及管理、构造及更新 UpdateData 数据的虚接口、设置及获取对象属性集的方法等。然后分出了两类派生对象，一是 Item，另一是 WorldObject。Item 即物品对象，WorldObject 顾名思义，为世界对象，即可添加到游戏世界场景中的对象，该对象类型定义了纯虚接口，也就是不可被实例化，主要是在 Object 对象的基础上又添加了坐标设置或获取的相关接口。

从 WorldObject 派生出的类型就有好几种了，Unit，GameObject，DynamicObject，Corpse。Unit 为所有生物类型的基类，同 WorldObject 一样，也不可被实例化。它定义了生物类型的公有属性，如种族、职业、性别、生命、魔法等，另外还提供了相关的一些操作接口。游戏中实际的生物对象类型为 Creature，从 Unit 派生，另外还有一类派生对象 Player 为玩家对象。Player 与 Creature 在实现上最大的区别是玩家的操作由客户端发来的消息驱动，而 Creature 的控制是由自己定义的 AI 对象来驱动，另外 Player 内部还包括了很多的逻辑系统实现。

另外还有两类特殊的 Creature，Pet 和 Totem，其对象类型仍然还是生物类，只是实现上与会有些特殊的东西需要处理，所以在 mangos 中将其作为独立的派生类。

早期的大部分游戏对象结构都是采用的类似这种方式。可能与早期对面向对象的理解有关，当面向对象的概念刚出来时，大家认为继承就是面向对象的全部，所以处处皆对象，处处皆继承。例如，所有的子类都从 CObject 派生。大多数情况下，直接派生的也是抽象类，其中带一些功能而另一些子类则不带这些功能，比如可控制/不可控制，可动画/不可动画等。mangos 的实现中基本就是这种情况，从 Object 直接派生的 Unit 和 WorldObject 都是不可直接实例化的类。

随着面向对象技术的深入，以及泛型等概念的相继提出，软件程序结构方面的趋势也有了很大改变，游戏对象的实现也有类似的转变，趋向于以组合的方式来实现游戏对象类型，也就是实现一个通用的 entity 类型，然后以脚本定义的方式组合出不同的实际游戏对象类型。

而使用类继续的方式实现游戏对象有两大问题，一是它要求系统中的所有对象都必须从一个起点衍生而成，也就是说所有对象类在编译的时候已经确定，这可能是一个不受欢迎的限制，如果开发者决定添加新的对象类，则必须要对基类有所了解，方能支持新类。另一个问题在于所有的对象类都必须实现同样的一些底层函数。

对于第二个问题，可以通过接口继承的方式来避免基类的方法太多。在 mangos 的实现中就采用了类似的方法，从 Object 虚基类派生的 Unit 和 WorldObject 仍然还是不可实例化的类，这两种对象定义了不同的属性和方法，分来实现不同类型的对象。在游戏内部可以根据对象的实际类型来 Object 指针向下转型为 Unit 或 WorldObject，以调用需要的接口。但是第一个问题是始终无法避免的。

所以我们便有了通用实体这么一个概念，其主要方法是将原来基类的接口进行分类，分到一个个不同的子类中，然后以对象组合的方式来生成我们所需要的实际游戏对象类型。这个组合的过程可以通过脚本定义的方式，这样便可以在运行时生成为同的对象类型，也就解决了上面提到

的第一个问题。

通用实体的实现方法在目前的游戏引擎及开源代码中也可以看到影子。一个是 BigWorld，从提供的资料来看，其引擎只提供了一个 entity 游戏对象，然后由游戏内容实现者通过 xml 和 python 脚本来自由定义不同类型的 entity 类型，每种类型可有不同的 property 和不同的方法。这样原来由基类定义的接口完全转移到脚本定义，具有非常强的灵活性。

另外还有一个是 CEL 中的 entity 实现。按照 CEL 的描述，entity 可以是游戏中的任意对象，包括玩家可交互的对象，如钥匙、武器等，也可以包括不能直接交互的对象，如游戏世界，甚至任务链中的一部分等。entity 本身并没有任何特性，具体的功能实现需要靠附加 property 来完成。简单来说，property 才定义了 entity 可以做什么，至于该怎么做，那又是依靠 behavior 来定义。所以，最终在 CEL 中一个游戏对象其实是由 entity 组合了多个 property 及多个 behavior 而生成的。

2.2 法术中实现的类的封装

上面提到的类的封装都是游戏对象的，在这一模块中我们将讨论法术中实现的类。

主要包括以下几种：

SpellCastTargets：法术释放目标

Spell：法术释放过程

SpellCastResult：法术释放结果

CurrentSpellTypes：当前法术种类

SpellEvent：法术事件

SpellCastTargets 的主要功能包括设置目标（包括 setUnitTarget，setCorpseTarget 等各种对象），设置地点，更新，读写等。

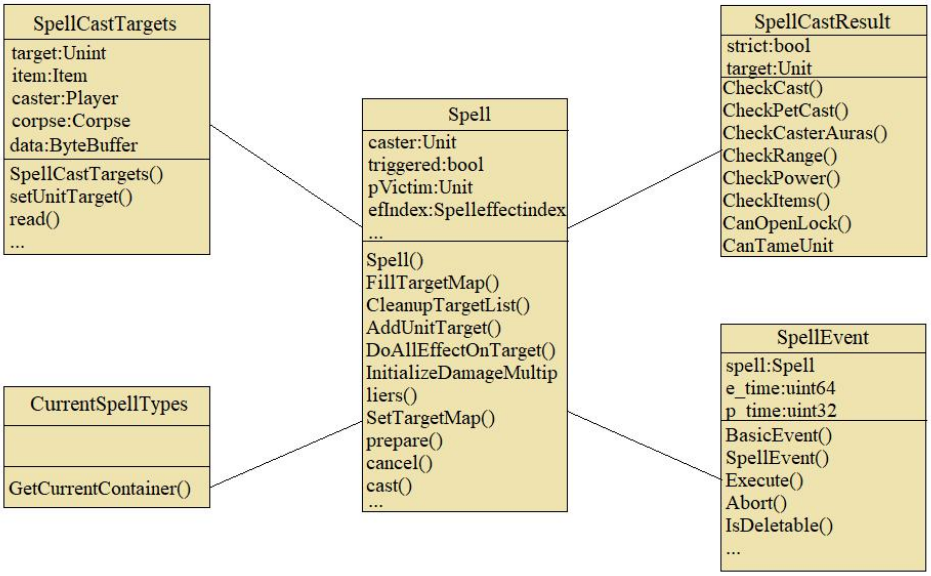
Spell 的主要功能包括目标定位，准备触发，清除目标列表，增加目标，初始化伤害系数，准备，取消，施法，冷却，自动施法，延迟，计算法力值消耗，更新初始法师坐标，检查目标生物种类等。

SpellCastResult 的主要功能包括检查法力值、物品、宠物、范围等。

CurrentSpellTypes 的主要功能是确认当前释放的法术的种类。

SpellEvent 的主要功能包括确认当前的事件是基础事件还是施法事件。然后执行或中止。

类图大致如下图所示：



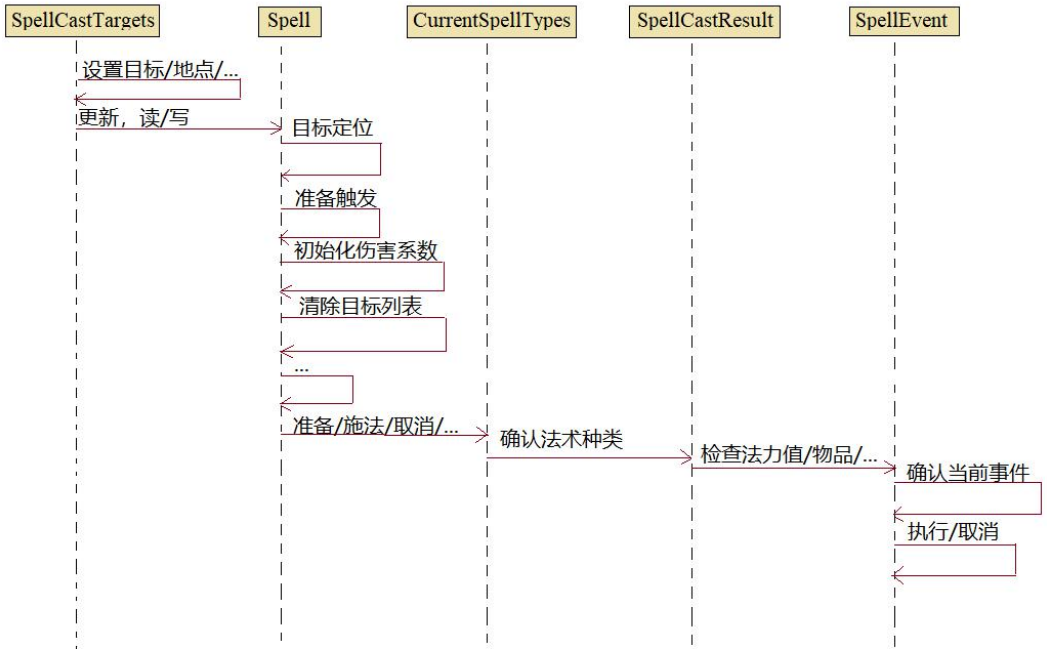
三、核心流程

3.1 法术如何运作？

人物都可以使用法术或特殊能力。当人物升级之后，他们可以购买、训练、研究或发现新的法术，并将它们抄入自己的法术技能书中。施法和使用特殊能力通常需要消耗法力、怒气、能量，有时还要消耗施法材料。

3.2 施法过程

在这里，我们用顺序图来大致表明施法过程，如下图所示。



四、复杂设计意图

在这个部分中，我们主要讨论 MaNGOS Zero 部分的设计模式。由于我目前了解的设计模式仅限于课堂上讲述的几种，而源码的量又非常大，我只是粗略地根据已有的知识做大致的推断，可能设计者的意图与我的推断并不符合，对可能出现的错误表示抱歉。

我在代码中发现了常用的几个模式的应用，包括 Singleton，Factory，State。

Singleton 单例模式：保证一个类仅有一个实例，并提供一个访问它的全局访问点。适用于当类只能有一个实例而且客户可以从一个众所周知的访问点访问它时，当这个唯一实例应该是通过子类化可扩展的，并且客户应该无需更改代码就能使用一个扩展的实例时。通过模版类将要遵循单一原则的类传入 Singleton 中。剥离类和实现。从而单一模式可以到处使用。例如

```
#define    sWorld MaNGOS::Singleton<World>::Instance()     //保证只有一个游戏世界
#define    sLog MaNGOS::Singleton<Log>::Instance()          //只有一个日志记录实例
```

Factory 工厂模式：通过传入指定的参数/或者不传入参数，通过 Factory 的某个方法(为了避免实例化 Factory 对象，一般方法为静态 static)，来获取一个对象。在 mangos 中用的是一个注册方法

```
template<class T, class Key = std::string>
class MANGOS_DLL_DECL FactoryHolder
{
```

```
public:
    typedef ObjectRegistry<FactoryHolder<T, Key >, Key > FactoryHolderRegistry;
    typedef MaNGOS::Singleton<FactoryHolderRegistry > FactoryHolderRepository;
```

将类型 T，和关键字传入注册类实例 ObjectRegistry，并定义了注册自己和解注册的函数

```
void RegisterSelf(void) { FactoryHolderRepository::Instance().InsertItem(this, i_key); }
void DeregisterSelf(void) { FactoryHolderRepository::Instance().RemoveItem(this, false); }
```

从而实现可以根据 Key 的类型，一般是字典方法，即用 String 代表不同的类实例，建立映射表，将对象注册进工厂中。

State 状态模式：允许一个对象在其内部状态改变时改变它的行为。适用于一个对象的行为取决于它的状态, 并且它必须在运行时刻根据状态改变它的行为。mangos 登录服在处理客户端发来的消息时用到了这样一个结构体

```
struct AuthHandler
{
    eAuthCmd cmd;
    uint32 status;
    bool (AuthSocket::*handler)(void);
};
```

该结构体定义了每个消息码的处理函数及需要的状态标识，只有当前状态满足要求时才会调用指定的处理函数，否则这个消息码的出现是不合法的。这个 status 状态标识的定义是一个宏，有两种有效的标识，STATUS_CONNECTED 和 STATUS_AUTHED，也就是未认证通过和已认证通过。而这个状态标识的改变是在运行时进行的，确切的说是在收到某个消息并正确处理完后改变的。

五、不足之处

经过我的思考，认为本次阅读的代码依然有以下可以改进的地方：

部分关键类设计得过于复杂，需要进一步细化和模块化，例如 Unit 和 Player 里面的战斗和技能很必要作为一个独立模块来设计和实现。

部分关键类还可以继续细分，例如 Creature 和 Player 等复杂的类，需要划分出状态，否

则太多状态变量和条件判断，很容易出错，而且不容易修改和扩充。