相关主题 RECOMMEND ARTICLE
- Introduction to 3d game engine design using directx-9 and c#(10)
- Introduction to 3d game engine design using directx-9 and c#(9)
- OGRE 3D 程序设计 (9)
- Introduction to 3d game engine design using directx-9 and c#(8)
- Introduction to 3d game engine design using directx-9 and c#(7)
- Introduction to 3d game engine design using directx-9 and c#(6)
- OGRE 3D 程序设计 (8)
- OGRE 3D 程序设计 (7)

MORE

推荐文章 RECOMMEND ARTICLE
- 游戏音乐制作案例之《QQ三国》
- 游戏音乐制作案例之《武林外传》
- 策划与程序和美工的沟通
- 龙神传说 原画
- 《LAST online》原画
- 《AION》新原画
- 数据广播方案的优化
- 网络游戏的位置同步

MORE

热门文章 HOT ARTICLE
- [电子书下载]游戏设计 — 原理与实践
- [电子书下载]网络游戏开发
- 游戏设计全过程
- [电子书下载]游戏设计技术
- [电子书下载]游戏设计理论
- CS游戏人物模型制作教程
- CG人物插画基本流程
- [转贴]MAX高级人头教程

MORE

文章标题	重构实践——利用配置文件实现设计的高度抽象		
来源：	[GroovOV]	浏览：	[677]

设计动机：

1、抽象程度低。引入任何一个新类型事物，都需要加入其对应的个体类或群体类及群体管理器。

2、不同个体之间的耦合程度高。具体体现在有着某种关系的个体之间，如追踪与被追踪关系。其中，一类个体A的状态受到另一类个体B的直接干涉（B通过设置A的一类标志，来迫使A的状态转变），这显然与自然规则和OOD原则相违背。按照面向对象的设计思想，一个对象应该尽量做到内聚，也就是说对象应该有自主能力，它掌控着自身所有的状态和变化，必要时提供自身的一些信息（比如，位置信息）。

3、灵活度低。如果需要添加/删除/更新功能，则需要对每个个体类型的代码进行相应修改。

设计思路：

1、对于抽象程度低的设计缺陷，采用提炼所有存在类中并集，加之配置文件运行时动态具类化（即，根据配置文件和创建参数动态确定创建对象的类型，如Tiger、Deer等）。之所有考虑提炼并集而不是交集，目的是让其以配置文件形式而非代码（即，继承）来具类。
2、通过提炼AI对象之间的相互作用关系，得出一种以角色（Role）为基础的关系抽象。在此设计中，每个对象都一个或者多个角色，同时又有其感兴趣的一个或者多个角色。角色管理器担当着管理角色－对象列表（如图1）。对象在创建时根据其角色将自身注册到全局的角色管理器中。一个对象能够在需要时在角色管理器中搜索其感兴趣的角色对象，获取其参数后，根据计算值更新自己的状态。如时序图2所示。

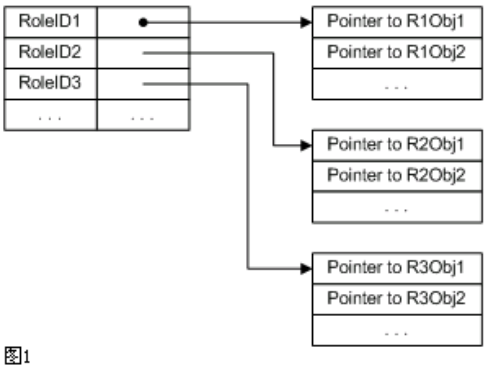
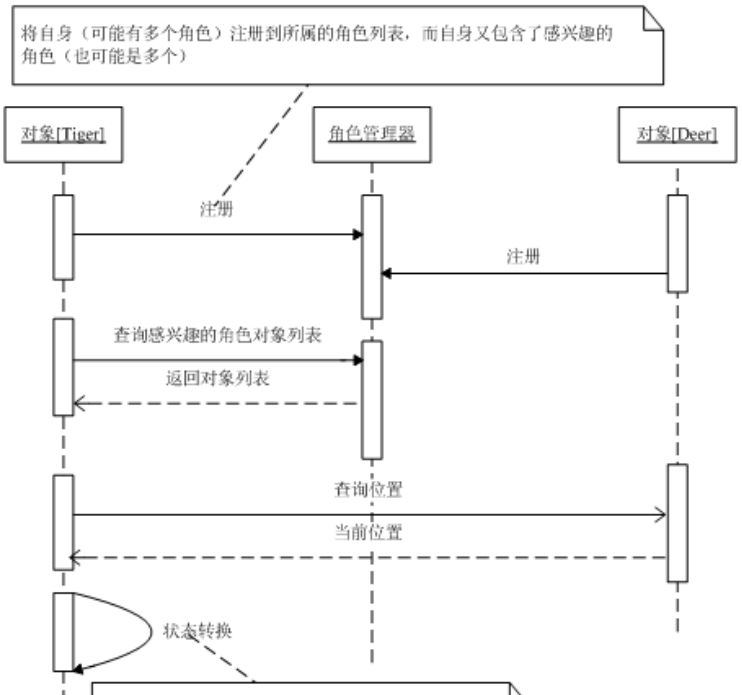


图1



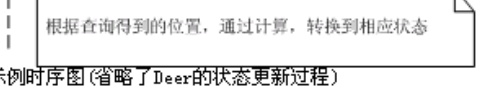


图2 示例时序图(省略了Deer的状态更新过程)

实现：

1、AI个体抽象类CIndividual，维护着个体的智能，是OpenSteer::SimpleVehicle和Ogre::UserDefinedObject的子类。
2、AI个体抽象包装类CIndividualWrapper，负责对象的视觉表现，对CIndividual起着包装作用，是CIndividual的友元。
3、角色管理器模板类CRolesManager，实现如下：

```
000001 class CIndividual;
000002
000003 /** 全局角色管理器
000004 @author GroovOV(groovOv.luo@gmail.com)
000005
000006 负责管理角色对应的对象列表
000007 */
000008
000009 template
000010 class CRolesManager
000011 {
000012 public:
000013 ~CRolesManager()
000014 {
000015 RoleRegistryMapIter iter;
000016 TRAV(m_roleRegistryMap , iter)
000017 {
000018 SAFE_DELETE( iter->second );
000019 }
000020 }
000021
000022 /** 将对象指针添加到角色ID对应的列表中
000023 */
000024 void Add(unsigned int roleID , T* pObj)
000025 {
000026 RoleRegistryMapIter iter = m_roleRegistryMap.find(roleID);
000027 if(iter != m_roleRegistryMap.end())
000028 iter->second->push_back(pObj);
000029 else
000030 {
000031 std::list* listPtr = new std::list;
000032 listPtr->push_back(pObj);
000033 m_roleRegistryMap.insert(make_pair(roleID , listPtr));
000034 }
000035 }
000036
000037 /** 从角色列表中移除对象
000038 */
000039 void Remove(unsigned int roleID , T* pObj)
000040 {
000041 RoleRegistryMapIter iter = m_roleRegistryMap.find(roleID);
000042 if(iter != m_roleRegistryMap.end())
000043 iter->second->erase(pObj);
000044 }
000045
000046 /** 返回指定角色的对象列表
000047 */
000048 std::list* GetObjList(unsigned int roleID)
000049 {
000050 RoleRegistryMapIter iter = m_roleRegistryMap.find(roleID);
000051 if(iter != m_roleRegistryMap.end())
000052 return iter->second;
000053 else
000054 return NULL;
000055 }
000056 private:
000057 /** 角色-对象注册表
000058 */
000059 stdext::hash_map*> m_roleRegistryMap;
000060 typedef typename stdext::hash_map*>::iterator RoleRegistryMapIter;
000061 };
000062
000063 #define g_rolesManager DMF::globe_instance< CRolesManager >::Instance()
000064
```

4、配置脚本(利用TinyXML实现读/存)：

```
000001
000002
000003
000004
000005
000006
000007
000008
000009
000010
000011
000012
000013
000014
000015
```

000016
000017
000018
000019
000020
000021
000022
000023
000024
000025
000026
000027
000028
000029
000030
000031
000032
000033
000034
000035
000036
000037
000038
000039
000040
000041
000042
000043
000044
000045
000046
000047
000048
000049
000050
000051
000052
000053
000054
000055
000056
000057
000058
000059
000060
000061
000062
000063
000064
000065
000066
000067
000068
000069
000070
000071
000072
000073
000074
000075
000076
000077
000078
000079 . . . . . .
000080
000081
000082 . . . . . .
000083
000084