**第五章：个体集群**

**虚假的分配与释放**

这篇文章主要对“个体集群”思想和对应于引擎的工具类进行介绍。

[个体集群思想]

个体集群：将大量的、具有相似性质的事物进行某种行为上的抽象和统一管理，例如射击游戏中的弹幕，或是一个粒子特效中使用的上千粒子。

[个体集群的必要性]

个体集群的概念，产生自Java堆分配的特点，了解Java的您一定明白，相比于可用free方法精确释放堆区内存的C语言不同，Java剥夺了程序员精确控制堆区释放的权利，由此使程序员从控制堆区的烦琐事务中解放出来，同时也就意味着，在要求性能的场合下，糟糕的设计结构会让程序寸步难行。

以粒子特效为例，譬如这里有一个由1000个粒子共同完成的特效，当收到开始特效的信息后，从堆区分配1000个粒子在时间上是来不及的，同理，在粒子特效完成后，将1000个粒子丢给GC也是不负责任的。

您可能会这样解决这个问题，将粒子抽象为一个类，在一个容器中，保存1000个这样的类实例，在特效开始后，依次遍历它们，并调用开始的方法。这的确是一个解决思路，而且可以解决的相当不错，但是，如果粒子的种类非常多，怎么办？抽象父类进行继承吗？要是这个游戏是一个射击游戏，包含有弹幕的元素，那么是要再抽象一个弹幕类呢，还是勉强地继承粒子类呢……

总之，以实现“提前分配”为目的，进行类的抽象是相当不明智的，我们需要的抽象，是对所有具备“分配”、“调用”、“释放”这一系列行为的对象进行概况，而且，介于我们使用的JavaSE平台，集群管理中对于“分配”与“释放”的支持显然是在逻辑上模拟的，并非真实地分配释放内存，这也是映射了本文的副标题——“虚假的分配与释放”。

[个体]

[interface engineextend.crowdcontroller.Individual]

接口Individual是对个体集群中的个体提供的抽象，我们来看一下这个接口的结构：

public interface Individual

{

public static final int SRC\_OUTER = 0;

public static final int SRC\_INNER = 1;

public static final int SRC\_HIDE\_ONLY = 2;

public static final int SRC\_EXTRA\_0 = 3;

public static final int SRC\_EXTRA\_1 = 4;

public static final int SRC\_EXTRA\_2 = 5;

public boolean isAvalible();

public void getUse(Object[] ARGS,float... FARGS);

public void doStp(int clock);

public void finish(int src);

public void destroy();

}

首先介绍其中的方法：

public boolean isAvalible();它告诉集群管理器（见下），我是否处在一个可被分配的状态。

public void getUse(Object[] ARGS,float... FARGS);getUse方法对一个可分配状态的个体进行初始化，并将这个个体分配出去。（也就是调整状态到不可再分配）

public void doStp(int clock);在绘制每一个帧的逻辑中，集群管理器会对其管理的个体进行遍历，对于已分配的个体调用其doStp方法，可以理解为是一个会被自动调用的Serial方法。

public void finish(int src);finish方法将一个处于已分配状态的个体再度设置到可分配状态。src可以区分该消息的来源，在一些场合下，根据src的不同进行不同的执行逻辑。

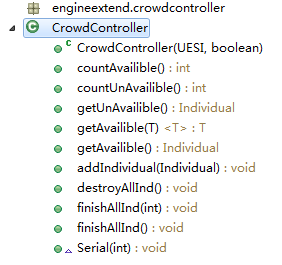
public void destroy();destroy方法用于永久销毁个体实例，调用此方法后，该类的实例允许被GC回收。

[集群管理器]

[class engineextend.crowdcontroller. CrowdController implements SerialTask]

CrowdController类是与Individual接口共同完成Geiv下的集群管理功能，有必要对CrowdController进行一些讲解：

框架图：



构造器：UESI是引擎的句柄，后面的布尔值表示是否使用系统默认的帧逻辑。（有时候，我们使用Serial构造Serial级联，这样可以有选择地暂停或终止一部分Serial，这在实现游戏暂停时很有用。）

countAvailible:返回当前管理的集群中可分配的个数。

countUnAvailible:返回当前管理的集群中已分配个数。

getUnAvailible:获得一个已分配的个体。

getAvailible:获得一个可分配的个体。

addIndividual:向集群中增加一个个体。

destroyAllInd:销毁集群中所有个体。

finishAllInd:以给定的来源，将集群中的所有个体释放（finish）

finishAllInd:以Individual.SRC\_OUTER为来源，将集群中的所有个体释放。

Serial:该类实现了Serial接口，在每一次绘制结束后，会对群体中所有出于已分配状态的个体，执行其doStp方法。

[例子：发射子弹]

编写程序，使用方向键控制小方块，使用A键发射小矩形子弹。

Main.java:这是我们的程序入口

public class Main{

public static void main(String[] args) {

UESI UES = new R();

new ShootRect(UES);

}

}

ShootRect.java:这个类是我们控制的方块，他可以根据键盘移动和发射子弹。见注释详解。

public class ShootRect implements SerialTask{//它实现了Serial，以主动扫描的输入方式响应键盘。

UESI UES;

Obj rect;//这个图元表示被控制的方形。

CrowdController cc;//集群管理器用来管理它的子弹群体。

public ShootRect(UESI UES){

this.UES = UES;

rect = UES.creatObj(UESI.BGIndex);

rect.addGLRect("FFFFFF",0,0,40,40);

rect.setPosition(CANExPos.POS\_CENTER);

cc = new CrowdController(UES,true);//构造集群管理器

for(int i = 0;i < 32;i++){

cc.addIndividual(new Bullet(UES));//将32枚子弹装入集群管理器。装入的数量取决于子弹在屏幕上出现的最大数量。

}

UES.addSerialTask(this);//将Serial实现加入绘制任务队列的末尾。

rect.show();//显示rect

}

@Override

public void Serial(int clock) {

if(UES.getKeyStatus(KeyEvent.VK\_LEFT)){

rect.setDx(rect.getDx() - 3.0f);//左键按下时，想做移位3.0个像素。下同

}

if(UES.getKeyStatus(KeyEvent.VK\_RIGHT)){

rect.setDx(rect.getDx() + 3.0f);

}

if(UES.getKeyStatus(KeyEvent.VK\_UP)){

rect.setDy(rect.getDy() - 3.0f);

}

if(UES.getKeyStatus(KeyEvent.VK\_DOWN)){

rect.setDy(rect.getDy() + 3.0f);

}

if(UES.getKeyStatus(KeyEvent.VK\_A)){//这里响应发射键

if(clock%10 == 0){//使用时序，即一秒发射6发

cc.getAvailible().getUse(null, rect.getCentralX(),rect.getCentralY());//将一个待分配的子弹分配除去，使用参数为发射体方块的中心点。（这里请参考下面Bullet的实现）

}

}

}

}

Bullet.java://子弹类，它实现Individual接口

public class Bullet implements Individual {

Obj bullet;//这里是子弹的图元

public Bullet(UESI UES) {

bullet = UES.creatObj(UESI.BGIndex);

bullet.addGLRect("FFFFFF", 0, 0, 5, 12);

bullet.setGLFill(true);

}

@Override

public boolean isAvalible() {

return !bullet.isPrintable();//如果子弹已经投射在屏幕上，意味着其已经分配

}

@Override

public void getUse(Object[] ARGS, float... FARGS) {

bullet.setCentralX(FARGS[0]);//设置子弹的中心位为float参数的第一个。

bullet.setCentralY(FARGS[1]);

bullet.show();//这里，子弹显示在屏幕上，也就意味着，isPrintable方法将返回true，上面的isAvalible返回false。

}

@Override

public void doStp(int clock) {

if (bullet.getDy() > 0) {//如果纵坐标大于0

bullet.setDy(bullet.getDy() - 5.5f);//则向上移动

} else {

finish(0);//否则，结束分配状态。

}

}

@Override

public void finish(int src) {

bullet.hide();//将子弹的图元隐藏，因此isAvalible将返回true。

}

@Override

public void destroy() {

bullet.destroy();//销毁图元

}

}

演示效果：

