后端同步广播算法优化

|  |  |  |
| --- | --- | --- |
| **修改日期** | **修改人** | **修改内容** |
| 2013-6-25 | 叶键晖 | 新增后端同步广播优化报告 |
| 2013-6-26 | 陈小锋 | 怪物AI机制变更说明  新旧版怪物AI压力测试报告 |
| 2013-6-26 | 陈安兴 | 补充具体设计实现思路 |

目录

[后端同步广播算法优化 1](#_Toc360023322)

怪物AI机制变更…………………………………………………………………………………

新旧版怪物AI压力测试报告……………………………………………………………………

后端场景格子逻辑[实现思路 1](#_Toc360023322)

1. **后端同步广播优化报告**

前言：

之前后端服务器在同屏广播与九宫格广播等广播逻辑上出现了比较明显的性能问题，相关的同事采用分线的策略将场景中的玩家分散到一个个独立子场景中并限定一个子场景只能容纳150人，这种做法在一定程度上可以舒缓服务器的压力，但不能从根本解决问题。本文档主要针对这种现状来设计一种解决方案，经测试，在单场景进程下，基于新方案的框架比旧方案的在性能上有接近3倍的提高，在多场景进程下有2倍左右的提高，而且在内存资源消耗上，新旧方案的差异不大。

正文：

这里首先描述下新旧两解决方案的实现方式：

**旧版--->**

服务器采用遍历场景中所有的玩家是否在同屏范围或者九宫格范围内，满足条件时将满足的玩家添加进列表中最后发送。这里瓶颈最主要体现在玩家的遍历操作，如果场景中有1000个玩家，那每次判断都要比较1000次，再有，旧版的判断逻辑相对复杂，每次都要比较5次，也就是说cpu要构造同屏或九宫格范围的话需要做5\*1000次运算。效率可想而知

**新版-->**

每个场景进程启动是都首先搭建起一个地图网格系统，并且将相关的地图信息以key-value形式保存在进程的资源里，其中key为场景中对应点的坐标，value为对应点上的玩家id列表。玩家每次移动时都将玩家id从旧的列表迁移到新列表中，每次同步时，通过运算获取对应的同屏或九宫格坐标点集(key)，从而获取对应的玩家id(value),最后发送数据。这样做就能本质地避免对所有玩家的遍历操作，只需要知道坐标就可以知道对应的玩家

测试报告：

多说无益，现在再是一下两个解决方案在不同情况下的表现：

**单进程下800并发**

cpu

（旧方案cpu消耗）

mem (旧方案mem消耗)

ditc_cpu

(新方案cpu消耗)

ditc_mem

（新方案mem消耗）

总的来说,单进程下新方案比旧方案在cpu性能上提高将近3被,而且内存消耗并没有明显的增加，一下在看下多进程下的表现

**多进程下5000并发,启动smp,8场景进程，每个进程620并发：**

loop_cpu_500

（旧方案cpu消耗）

loop_mem_5000

（旧方案mem消耗）

ditc_cpu_5000

（新方案cpu消耗）

ditc_mem_5000

（新方案mem消耗）

**多进程下8000并发,启动smp,,8场景进程，每个进程1000并发：**

loop_cpu_8000

（旧方案cpu消耗）

ditc_cpu_8000

（新方案cpu消耗）

ditc_mem_8000

（新方案mem消耗）

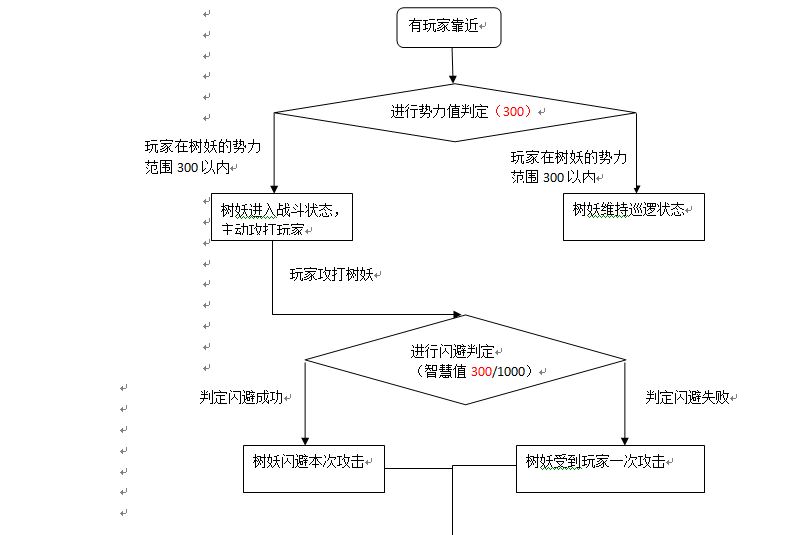
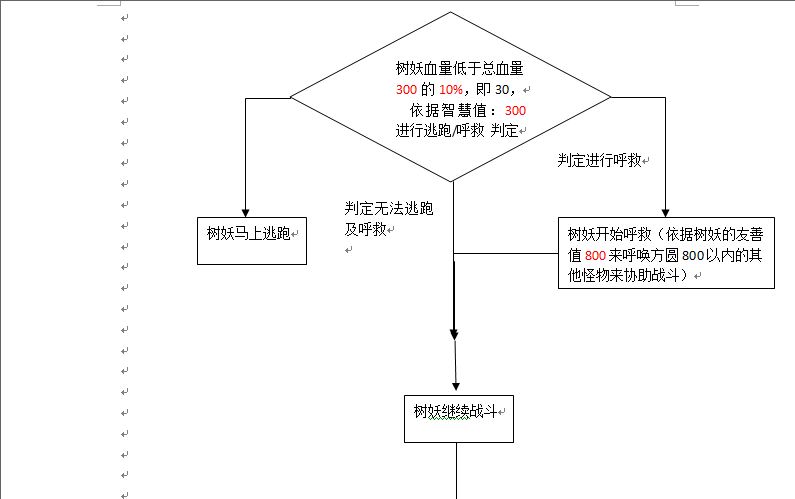
多进程下，新方案仍然表现出相对较好的性能，同时新方案走定为同屏坐标以及九宫格坐标逻辑时调用的是旧接口逻辑，有于旧接口本身存在错误，会将整个坐标范围放大，如果后续能对旧接口进行修正，效率会有更大的提高。同时由于只是额外存储id本身，所以新方案在内存消耗方面也没有太明显的缺陷。

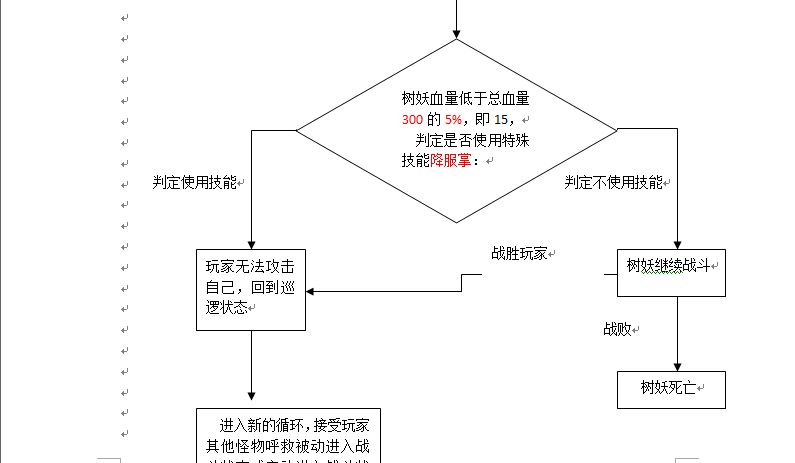
总结：

经测试,当场景进程稳定在500左右的并发下新方案的性能升最为明显，当并发数大于800时，由于一方面由于就接口返回错误的同屏坐标以及九宫格坐标，导致运算量加大，另一方面由于频繁操作列表与进程字典，新方案在性能表现上体现的优势没有之前明显。总的来所，新方案能比起就方案在性能上有较明显的提升，采用新方案的场景能同时容纳更多的玩家。

测试耗时对比：待补充

1. **怪物AI机制变更：**
   * 1. 怪物AI基本示意图

****

****

修改说明：首先，我们的怪物信息不再存储在进程字典表中，而是每一个怪物都开启一个怪物进程，怪物和怪物之间，怪物和玩家之间直接通过进程通信机制进行。这样做一方面把原本抽象的通信模式改为更为形象且方便的进程通信模式，另一方面也会更好的使用了erlang的gen\_fsm机制，使得程序更好的维护和拓展。

可行性分析：目前我们的场景数目有10个，每个场景假设平均有60个怪物，这样就是增加600个怪物进程，在erlang中，增加600个进程，对于程序的开销非常的小。

1. **新旧版怪物AI压力测试报告：**

测试环境:

* 1. 操作系统：Window 64位
  2. 系统内存：4G
  3. 处理器：AMD 2.70Gz处理器

测试用例:

在程序运行开始时，创建N个怪物进程，同时也创建6w个玩家进程**（由于目前测试环境下为空逻辑测试，实际情况的逻辑判断会非常多，所以此处设置一个非常大的玩家基数来模拟）**，然后让所有玩家进程同时对N个怪物进程进行交互。

* 原有方案：（轮询，怪物保存在场景进程字典中）

1. 测试情况:
   1. **N= 100：**
      1. 开始时间: 1372078857667 结束时间: 1372078863298,即耗时间为**6秒**
      2. 使用内存为9,672k
      3. Cpu使用50%
   2. **N=800**
      1. 开始时间: 1372078978769 结束时间: 1372079023822,即耗时为**45秒**
      2. 使用内存为10,032k
      3. Cpu使用50%
   3. 其他数值测试(修改玩家个数为6000，怪物个数N=1000)
      1. 开始时间: 1372080745676 结束时间: 1372080751604,即耗时为**6秒**
2. 结果分析:

在假定60000玩家同时在线的情况下，当怪物总数达到100的情况下，程序就耗时6秒，而正常情况下，每次刷新时间仅仅是限定为0.25秒（为了保证怪物在前端能正常表现各种特性），而6秒已远远大于了0.25秒，很明显可以看出，0.25的刷新时间给系统带来了很大的压力瓶颈。

* 新方案：（事件驱动，怪物自身是一个进程）

**（说明：由于新方案采用事件驱动方式，就没有了“刷新时间”瓶颈问题，因此在此需要进行压力测试的新方案对服务器的内存和cup消耗情况）**

1. 测试情况:
   1. **N= 1039000：**
      1. 使用内存为2,103,116 KB 内存。
      2. 开始时候为80%，运行过程中平稳保持在70%左右

结果：程序运行正常

* 1. **N= 1040000：**
     1. 使用内存为2,103,116 KB 内存。
     2. Cup的使用率：开始时候为80%，运行过程中平稳保持在70%左右
  2. 其他数值测试(修改玩家个数为6000，怪物个数N=1000)
     1. 使用内存为18.872KB 内存。
     2. Cup的使用率：开始时候为3%，运行过程中平稳保持在1%左右

结果：程序挂停，无法正常运行

（补充说明：在测试过程中，window系统有运行其他小程序，会占用一定的资源。）

1. 结果分析:
   1. 从以上的测试过程可以知道，当怪物进程数达到1040000时候，程序无法正常运行，可以认为程序允许最大的怪物个数为1039000。
   2. 在怪物个数为1039000，玩家进程为6w时，程序一样能快速正确的执行，说明该程序的压测结果令人满意，能轻松应付实际情况的运行。
2. **后端场景格子逻辑实现思路：**

主要解决三个问题：

1保存地图信息配置表。读取地图配置MapTile.xml文件。

2搜索九宫格玩家算法优化。通过实时更新场景中对象列表实现，优点在于九宫格搜索避免全场景对象遍历，但是同时会带来的是需要实时更新场景对象列表的压力。

3提供怪物触发的格子信息。

数据结构：

Put({idX,idY,type},Type)

Type是格子类型（可走，障碍）

Put({idX,idY,objList},ObjList)

ObjList是当前格子保存的对象列表，对象列表保存实例id

Put({idX,idY,monsterAlert}, MonsterAlertList)

MonsterAlertList是当前格子保存的怪物对象列表，该格子处于某怪物的警戒范围

idX，idY是地图格子坐标

场景对象信息表的维护处理：

1格子类型进程字典Put({idX,idY,type},Type)

场景初始化读地图配置表xml文件更新记录中的type字段

2对象列表进程字典Put({idX,idY,objList},ObjList)

玩家进入场景包12001和同步包12011,通过场景进程mod\_scene更新格子信息表{idx，idy}对应的MapInfo,同时移除上一位置格子的对象信息。同时怪物加载和怪物移动通过场景进程更新格子信息表

3怪物警戒列表进程字典 Put({idX,idY,monsterAlertList}, MonsterAlertList)

怪物出生和怪物移动更新怪物警戒范围的格子信息，出生时更新警戒范围全部格子，移动时更新边缘格子。玩家触发怪物AI,人物同步包12011同步到当前格子时，遍历当前格子的所属怪物列表，给怪物进程发消息，由对应的怪物进程来处理触发怪物AI的逻辑.

4供接口获取该进程字典对应格子坐标的对象列表

接口：

1get\_curpos\_type返回x,y对应地图格子类型（可走，障碍）

2 get\_cango\_pos返回x,y最近的可走格子

3get\_matrix\_objList(x,y).返回当前x,y下九宫格的对象列表

4get\_slice\_objList(x,y).返回当前x,y下小格子的对象列表

5get\_alert\_monsterList(x,y)返回处于怪物警戒的对象列表