中国科学 : 信息科学

SCIENTIA SINICA Informationis

实验报告

Developing a Java Game from Scratch

蒙芷露[1]

E-mail: mzl0830@sina.com

摘要 该实验报告描述了作者的Java项目RebelliousMonster的设计思路、技术与工程问题的解决方案和优化讨论。以及对Java课程的感想和收获小结。

关键词 Java, 项目报告, 课程小结, 面向对象

1 开发目标

开发的是一个支持多玩家互动的像素小游戏。在本游戏中，玩家作为小怪兽可以通过吞下最多一个石块作为武器随身携带，并发射石块攻击小妖怪。小怪兽也会被小妖怪攻击，但是小妖怪每次攻击的数值不确定。

游戏支持多玩家在线对战，目前上限是2个。位于服务端的玩家有保存游戏和重载存档的权限。

灵感来源于4399的神奇小妖怪游戏。时间有限做了一定的简化，如石块撞击小妖怪会使小妖怪直接死亡消失，没有晕厥效果。

2 设计理念

2.1 总体代码设计

代码的总体设计在继承作业框架代码的基础上采用了面向对象设计的思想。保留原有框架的UI架构、键盘事件处理架构，通过在WorldScreen 类内支持妖怪、怪兽、石块对象行动、交互来实现游戏逻辑并最大化利用原有代码。

游戏最初的架构集中在WorldScreen中，即最终版本中的服务端。其中各个对象都是独立的线程，通过函数接口交互，如攻击，避让。这样的设计不仅使得游戏主要角色之间功能明确，耦合度低，可以各自设计算法独立运行；还使得在重载存档时可以及时启动线程，使得画面不至于无法启动或者保存重载前的对象。

在拓展本地游戏到支持多玩家的过程中，游戏逻辑主要依赖于服务端，客户端仅保留了terminal用于显示，不具备world和screen。即在客户端上只需要实现屏幕定时刷新和向服务端发送接收的键盘输入。这样降低了代码的复杂度，使客户端轻量化，减小了开发的工作量。

2.2 核心类设计

2.2.1 Monster/Goblin/Stone

从面向对象的角度来设计，即尽可能的模拟现实世界。在原本的游戏中，Monster（即怪兽）受玩家控制，而Goblin（妖怪）是自动移动的，而石块有两种状态：保持静止，或被怪兽作为武器发射，发射时沿着发射方向移动直到遇到障碍物。这三类的设计应当保留以上特性。为了使得代码复用，作者还将这三类的共同方法或属性在其共同父类中予以实现，如移动方法。

三个类的移动方法有相似的部分，如在进入某个地块之前判断是否被占用，即并发问题。针对这一问题，作者采用了synchronized关键字将部分代码划为临界区。保证以上三类在移动到下一个预期位置时方法是原子化的，不会在中途被打断。但他们的方法也有不同之处，如怪兽需要根据键盘键入信息来决定下一步移动方向，妖怪和石块有各自的移动规则：妖怪向某一个方向前行，遇到障碍转换方向直到能继续直行；石块按照被发射的方向直行直到遇到障碍物停下。因此在实现中，把计算位置和实际移动到下一步进行了切分。

三个类之间有消息发送的过程。如妖怪会攻击怪兽，即向怪兽发送hp减少一定数值的消息。怪兽会发射吞下的石块，即发射后石块自动移动不再受怪兽控制。此处三个类都实现了Runnable接口，使得互相独立行动和交互成为可能。如怪兽发射石块，即给定石块方向后启动石块的run 方法，石块则开始移动直到遇到障碍物退出run 方法。该方法下一次依旧可以由怪兽启动，由此实现了石块的重复利用。

三个类有共同的属性，在其父类中已有实现，如显示样式。其中怪兽和妖怪还有hp值属性，也在父类（Creature）中实现。共同方法和属性在父类中实现减少了代码量，也便于在后期有新需求的时候进行添加和修改。

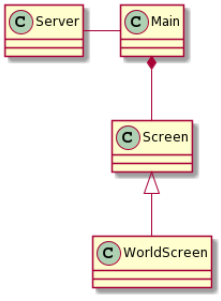

图 1　整体架构设计

Figure 1　Overall Structure

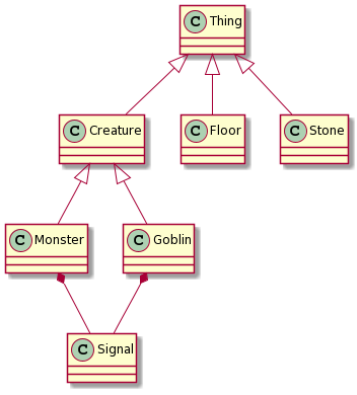

图 2　核心游戏类设计

Figure 2　Core Things Class

2.2.2 WorldScreen

游戏的主要逻辑都在此处实现。包括地图的初始化，怪兽、妖怪、石块状态的初始化和线程启动。jw06中要求的存档也在此处以WorldScreen 特有的方法实现。

由于WorldScreen继承了Screen的监听键盘事件的函数，需要设计一种方法，使得键盘输入能够传递给怪兽。一开始直接在WorldScreen中调用怪兽的移动方法传递信息，后来考虑到网络通信功能和降低类间耦合度的目标，决定使用BlockinQueue 进行数据传输。即WorldScreen调用put方法将键盘事件信息加入该结构，怪兽从该结构中读取进行处理，同样的逻辑可以应用在网络通信中。在实现中，发现只需要读取KeyEvent的keyCode，为了调试方便，将BlockingQueue的类型参数从KeyEvent 修改成Integer。尽管接口修改看似麻烦，但减少了需要传输的信息量，同时有利于后期构造单元测试用例。

如何使得WorldScreen中的怪兽、妖怪知道游戏结束而停止移动需要考虑到各个对象之间的同步。于是设计了Signal类用于传递一些全局共享同步的信息。由WorldScreen在构造时给怪兽、妖怪对象分配。这里利用了Java对象引用的特性。一开始直接使用Boolean这一基本类型，无法达到修改值能够全局同步的效果。因为基本类型是传值调用。用自定义类包装后，可以作为一个公共信号分享，一旦被修改，所有共享这一Signal 对象的对象都能得知修改。这一机制在存档中也发挥了重要的作用。

在实现存档读档功能的时候，自然的考虑将存档读档的接口在该类实现。因为该类保留了必要的信息。并且存档读档的时机取决于键盘键入。即在keyPressed中就可以调用对应函数处理。

在网络通信中，WorldScreen的功能进行了进一步的拓展。由于客户端需要得知服务端WorldScreen上所有对象的显示信息，包括坐标、颜色、显示字符。因此WorldScreen 需要能够按照一定格式对这些信息编码并进行输出，因此设计了一个对外接口displayInfo()来供外界获得这一显示信息。

2.2.3 Main/ClientMain

在服务端即游戏主要界面，Main内包括了游戏主要的逻辑所在类Screen（WorldScreen）和显示的架构，同时承担了监听键盘事件的任务。即Main内游戏所需要的基础设施（键盘交互、屏幕显示，以及后来添加的网络通信）。

Main承担了大部分和Server类交互的任务，主要是将必要的信息交给Server，如向客户端发送的屏幕信息。Main和Server相互独立，没有继承或者组合的关系。一开始作者认为游戏逻辑和网络通信并没有太多关联，但在实现中发现，网络通信频繁的需要请求游戏逻辑中的信息，因此是否需要将Server作为Main 的内部类实现还有待思考和实践。

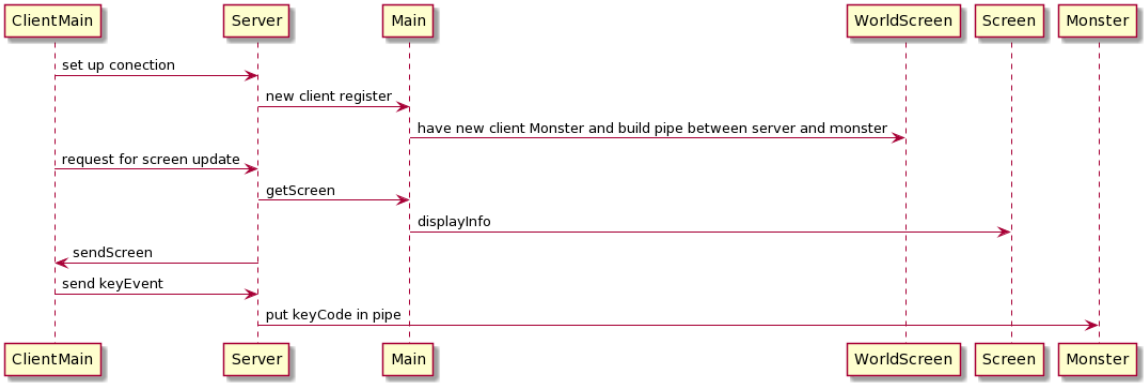

图 3　网络通信过程

Figure 3　Communication in NetWork

2.2.4 MainReactor/SubReactor/Handler/Server

网络通信主要由这四个类完成。一开始采用Server来完成基本功能，后来基于效率的考虑换用了Reactor模式来完成，在图3中为了方便仍然用Server来指代。

最开始的Server不考虑效率问题，基本完成了建立连接到刷新屏幕的功能。该版本确立了只向Client发送terminal信息的基本框架。但是在调试的过程中发现容易阻塞，于是参考老师提供的示例代码，在阅读理解后改用了Reactor模式且是multiple reactor的设计。

在multiple reactor模式中，MainReactor用来建立连接，SubReactor用于分配Accept 之外的任务，Handler才是任务处理函数的主体。因此主要逻辑应该在Handler中修改，MainReactor和SubReactor进行微调即可。

而游戏对服务器的要求是，首先在与Client建立连接的时候激活一个怪兽对象；其次是能够处理Client发送来的KeyEvent和刷新屏幕的请求。前者在MainReactor 中进行微调即可完成，后者则需要调整Handler内具体的方法。

阅读原有代码可以发现，Handler多次修改了SelectionKey上注册的事件，以完成先Accept再读取Client发送的内容再向Client发送回复的任务。其中涉及到读写状态的切换。而在游戏逻辑中，服务端接收KeyEvent是读，发送Screen信息是写，与原本的示例代码有逻辑上的相似之处。因此修改后的Handler在接收到Client发来的信息后会进行判断，如果是KeyEvent则直接读取，并保留SelectionKey监听的READ事件；如果是刷新屏幕的请求则修改SelectionKey的注册事件为WRITE，并在完成发送后将注册事件修改回READ，以便接收下一次Client 发来的请求。

在该架构下，Server端只是回应Client的请求，不需要设计复杂的逻辑对不同的Client进行控制。同时因为采用了Reactor，并且用线程池对任务进行管理，最终呈现的游戏效果比最开始的Server要顺畅丝滑许多。

3 技术问题

3.1 并发控制

游戏中，可移动的目标都被设计成了实现Runnable接口的类。在实现按照特定算法移动的同时，也带来了一系列问题，包括资源冲突、游戏存档读档时的重启等。

3.1.1 冲突管理

需要考虑的冲突管理集中在位置的访问上。一开始想到了两种思路：对每一个Floor对象加锁，或者对即将移动到某个位置的方法设置成synchronized。由于游戏的核心是移动的生物，而非Floor，且很多Floor多数情况下不会出现冲突的情况，于是考虑采用了第二种方法。

但是synchronized的粒度需要仔细思考。如果选定的方法执行时间太长，那么会影响游戏刷新，出现阻塞。而在Goblin和Monster中，计算出下一个需要移动的位置的耗时均远比移动所花费的时间长。因此把这两步拆分成两个方法，只对移动到下一个位置的方法设置synchronized。而在Stone的移动中也采用了同样的逻辑，判断下个位置能否进入的方法不属于被synchronized的范围，避免石块在移动的过程中游戏屏幕被迫阻塞。

在实现吞吐石块的逻辑的时候，考虑到不同玩家可能同时想要吞下同一个石块，但是参照上述逻辑，对一定代码区间加synchronized以避免被中途打断即可。

3.1.2 线程管理

Monster/Stone/Goblin的主要逻辑都在run内实现，run方法结束有几种情况：Monster/Goblin死亡；Stone遇到障碍物停下；游戏存档需要暂时结束屏幕刷新而终止以上对象移动。

通过在while中检查Monster/Goblin的hp，检查Stone是否遇到障碍物，以及signal对象是否传递了停止信息，可以终止线程运行。

而对于需要重启线程的情况，则需要新建线程重新启动run方法。在读档时，由于使用了ArrayList来追踪Monster和Goblin对象，可以在重启游戏的函数中调用get方法进行获取和重启。而对于石块，由Monster中的pushStone方法进行启动。

3.2 通信效率

3.2.1 网络通信效率

网络通信的效率提升集中在避免SockerServer阻塞和数据读取接收上。前者用Reactor模式进行了优化；后者采用NIO进行了优化。即使用channel/buffer来进行优化。并且通过调用buffer的特定方法，可以获得格式整齐的数据。

3.2.2 对象通信效率

Server和Main之间由于没有设计成有聚合关系的对象，在传递screen信息的时候需要层层调用函数返回，略有不便，可以考虑改进。

但服务端信息传给Monster继承了WorldScreen中用BlockingQueue作为管道的思路，避免了层层函数调用。即MainReactor在建立连接的时候将一个BlockingQueue绑定在激活的Monster和处理后续网络连接任务的Handler之间。这样Handler在得到Client的键盘输入的时候直接将数据放入BlockingQueue，Monster直接读取即可。

3.3 输入输出

除了用NIO优化读写速率，以何种模式的数据格式保存信息也是一个值得思考的，会影响编码难度和效率的问题。此处存档和网络通信有相似之处。如果数据保存格式设计的好，那么代码逻辑是大体相似的。

一开始想对Screen进行序列化并保存，但是这样做数据量太大，也比较复杂。于是采用记录关键信息的方式，将移动对象的显示字符、位置坐标、颜色红绿蓝对应的INT值记录下来。一来数据简单精炼，二来在Java的NIO中也有对应的函数可以处理。

同样的思路用在了网络通信中。对WorldScreen的每一个位置，记录下当前位置的显示字符和颜色，每个单元的信息作为一行直接向服务端发送。与序列化整个Screen相比，这样发送数据使得Client可以根据接受到的数据立刻进行屏幕的写入，使得刷新卡顿减少。

4 工程问题

4.1 面向对象

在游戏设计和逐步添加功能的过程中，面向对象的设计思路一方面帮助作者分解问题，把复杂问题分解成小问题解决，另一方面方便进行类的设计和功能拓展。例如石块的移动，如果石块依赖

于WorldScreen，那么移动和停止的逻辑就会过于复杂。但是在使用面向对象的设计和实现之后，可以相对独立的进行设计、改进。

4.2 Done is better than perfect

在实现网络通信的时候，采用了分步走的思路。使得开发的难度曲线尽量平滑。即先实现基本功能，再根据性能缺点，在保留基础思路的基础上进行改进。

4.3 单元测试

面对复杂的系统，对每一个类尽量实现单元测试可以排除类内方法自身的错误。同时在编写单元测试的过程中，可能会意识到之前类设计不当之处从而改进。如KeyEvent的传递改成Int值传递。修改后更便于测试Monster/Goblin/Stone类，同时提高了对象间信息传输的效率。

4.4 自动构建工具

利用Maven可以更好的管理资源，同时利用测试等插件进行统一测试和测试覆盖率查看。最后利用自动构建工具进行打包。

5 课程感言

学习计算机本身是一个需要不断输入并且不断输出的过程。Java课的作业不仅鼓励我们去动手，而且给的任务非常有趣，做起来很有成就感。课后补充材料也帮助我更进一步的理解了课堂上所学的内容。尽管由于时间问题项目还有诸多不足，但是在总结回顾的过程中思考了进一步改进的可能。总体来说Java是我心目中让我体会到何为”大学“的一门非常值得推荐的课程。

Title

Zhilu MENG[1]

E-mail: mzl0830@sina.com