中国科学: 信息科学

SCIENTIA SINICA Informationis

论文

Developing a Java Game from Scratch

许嘉禾[1]

1. 南京大学, 南京 210000
E-mail: 1816036457@qq.com

收稿日期: 2022-xx-xx; 接受日期: 2022-xx-xx; 网络出版日期: 2022-xx-xx

国家自然科学基金 (批准号: 0000000, 0000000, 00000000)

摘要 本文主要介绍自己在用 java 开发一个多人网络对战小游戏的过程和心得体会。开发框架采用的是 maven，图形化采用的是 java.swing，网络通信采用的是 NIO。

关键词 Java, 游戏开发, 图形化, 网络通信, 并发编程

1 开发目标

本游戏的总体设计思路借鉴了小时候玩的 4399 小游戏 “大鱼吃小鱼”，将其改造成了葫芦娃大战小妖怪的版本。葫芦娃和小妖怪各有红橙黄绿青蓝紫 7 种颜色，对应由低到高的 7 个等级，另外葫芦娃多增加一个七彩颜色的最终无敌形态。玩家通过方向键操控葫芦娃在画面中移动，碰到等级小于等于自己的小怪则吃掉，并加分升级；碰到等级大于自己的小怪则掉血。游戏可支持单人、双人或三人同时接入，最先达到 2000 分的玩家获胜。

2 设计理念

因为是实现多人网络对战，所以总体上分为了两个大模块，分别是 server 端和 client 端。server 端接受用户的接入请求，并且负责游戏的整体运行逻辑，对于用户的输入进行相应反馈，并且把整个游戏的画面传给用户。client 端负责接受用户的输入（在此游戏中就是键盘输入），并且把键盘输入信息传输给 server 端，接收 server 端传过来的游戏画面信息，并显示。

这样设计，server 端基本可以复用之前单机游戏的代码，因为整体的处理逻辑没有变，只是用户的输入不再是直接的键盘获取，而是变为从与用户连接的 channel 中获取（这点在第三部分技术问题中具体展开）。而 client 端要做的事情也很简单，就是接受用户的键盘输入，将按键信息传给 server 端，并读取 server 端的画面信息显示。

引用格式: 作者 1, 作者 2, 作者 3, 等. 中国科学 LaTeX 模版. 中国科学: 信息科学, 在审文章
Xing M, Xing M M, Xing M, et al. Title for citation (in Chinese). Sci Sin Inform, for review

整个项目均采用面向对象的设计方法，对游戏中的各种物体都做抽象，包括葫芦娃、小怪、地板格等，每个物体也都有自己的行为。采用面向对象的设计方法，代码能够很好地模拟现实，写代码的思路也很清晰。比如，我需要一个屏幕用于显示画面，那么我就定义一个 Screen 类，其中有方法 display () 用于显示。我需要不断的有小怪产生出来，那么我就定义一个 MonGenerator 类，内部定义一个不断产生小怪的方法。总之，采用面向对象的设计方法，我需要什么物件，我就为其定义一个类，并赋予相应行为。

另外，作为一个延续多阶段的大项目，代码的可复用性和可扩展性也是相当重要的。面向对象的设计方法可以较好的保证这一点。前面提到，我需要什么物体就为其定义一个类。类中已经定义好的属性和方法应该都是这个类的固有特征，是很明确的，并不会因为外界需求的变化而变化。外界需求变化，需要改变的很可能只是调用这个类的方式，或是增添一些新的方法，原有的代码完全可以保持并且复用。以及，语言本身的继承和多态机制也很好的支持了可复用性和可扩展性。

3 技术问题

3.1 多线程

当新开一个线程并启动以后，这个线程就会像脱缰的野马一样自己运行下去，可以看成是一个独立的个体（当然仍需考虑与其他线程的同步问题，之后再谈）。因此，对于我们这个包含很多活动物体（葫芦娃及小怪）的游戏，为每个活动物体创建一个线程是很合适的。一个葫芦娃或小怪创建出来后，它就应该按照自己的运行逻辑独立地运行下去，不应该受到主程序的束缚。具体来说，葫芦娃是在新玩家接入是被创建，而小怪则是自游戏开始之后就不断地被自动创建。我定义了一个“小怪发生器” MonGenerator 类，这个类的对象同样是一个独立于主线程之外的线程，用于每隔一段时间随机产生一个小怪，并投入运行。

3.2 并发控制

当有多个线程时，就不得不考虑线程之间的竞争与协作。最典型的就是当两个线程同时要访问同一个资源时，就产生了冲突。java 中，每个对象都关联一个管程，可以通过 synchronized 关键字使得线程在进入某个方法或某个代码块前必须申请该对象上的管程，占有该管程直到方法或代码块执行结束（当然也可以主动放弃，如 wait ()）。这样一来，就可以实现不同线程对共享资源的互斥访问。具体来说，如果是对象内部的方法访问自身属性时可能产生竞争，则在该方法上用 synchronized 关键字修饰；如果是外部的某个代码块访问共享资源可能产生竞争，则整个代码块用 synchronized (共享资源）修饰。

在本游戏中，比较明显的共享资源是葫芦娃和小怪脚下的格子。不同的活动物体不应该同时移动到同一个格子上，否则葫芦娃和小怪就会交错通过（正常逻辑是要么葫芦娃把小怪吃掉，要么小怪挡住葫芦娃且葫芦娃掉血）。因此，在每一步移动前都需要获得面前这块格子的管程，即把移动的代码放到 synchronized 代码块中。

此外，还有一些不是那么明显的共享资源。比如玩家的得分和生命值。葫芦娃主动撞向小怪会导致自身掉血或吃掉小怪加分；而小怪撞向葫芦娃也会导致其掉血或分数增加。也就是说，葫芦娃线程

和小怪线程都会访问到葫芦娃的生命值和分数值。因此，需要将葫芦娃的 addScore 方法和 getHurt 方法都用 synchronized 修饰。这个问题是我在写完后运行了很多次才偶然发现的，一开始写时完全没考虑到。所以，还是不够仔细，也是并发编程的经验还太欠缺。

综上所述，在有多线程时，对共享资源的互斥访问是极其重要的问题。道理不难懂，但要真正做好还得多实践。一是写代码时就要保持足够清醒，对有可能被多个线程同时访问的资源都加以处理，二就是多测试，多分析。很多时候会遇到各种奇奇怪怪的报错，原因就是对某个共享变量的访问冲突导致的，但看报错信息你可能会怀疑是不是自己别的什么地方写错了。不过，并发的错误还是有自己独特的特点，那就是偶发性。你连着运行几次，每次报错的时间点都不一样，或者有的时候出错有的时候正常，那多半就是并发带来的错误了。

3.3 保存与观看回放

大体思路是，在游戏过程中将画面信息保存到文件中，之后从文件中把画面信息读取出来并显示。游戏本身的屏幕显示，我是用一个 Refresh 线程，每隔 30ms 将屏幕 repaint 一下，于是保存画面信息的工作也可以放在这个 Refresh 线程中，即每次 repaint 之后顺便将 screen 对象写入文件，保证保存下来的屏幕信息与游戏过程中看到的完全一致。之后的回放，则同样新开一个线程，每隔 30ms 从文件中读取一个 screen 对象并显示。

在具体实现上，可以利用课上讲过的 ObjectOutputStream 和 ObjectInputStream。在保存画面时，利用 ObjectOutputStream 在文件末尾添加对象；回放时，利用 ObjectInputStream 不断从文件中读取对象。但仅仅这样做会遇到问题，报出 java.io.StreamCorruptedException: invalid type code 错误。原因是，每次追加一个对象信息到文件，都要 new 一个 ObjectOutputStream，而 ObjectOutputStream 的构造函数中会调用 writeStreamHeader() 方法，即写入头部信息。这样一来，每往文件末尾追加一个对象都会写入一个头部信息，而读取文件的 ObjectInputStream 只会过滤掉第一个头部信息，所以后面的读取就会出错。

解决方法是，自己定义一个 MyObjectOutputStream 类，继承自 ObjectOutputStream，重写 writeStreamHeader() 方法，使得它不会写入头部信息。于是，在写文件过程中，只要判断当前是不是第一次往文件里写，若是，则用 ObjectOutputStream，写入第一个头部信息；若不是，则用 MyObjectOutputStream，不写入头部信息。这样利用 ObjectInputStream 就能正常读取了。

从以上的报错和解决过程可以看出，学习一个东西，不能仅仅很浅表地知道它能怎么用，还应关注一下更底层的实现细节，这样运用起来才能更有把握。

3.4 网络通信

定义一个 nio selector，用于控制多个 channel。server 端的 serverSocketChannel 注册到这个 selector 上，注册操作为 ACCEPT，表示接收客户端的接入请求。若有用户请求接入，则为其 socketChannel 注册到这个 selector 上，注册操作为 READ | WRITE。

selector 可以在一个 while（true）循环中不断检查注册到其上的各个 channel 的状态，包括 acceptable, readable, writable, 然后根据其状态调用相应的 handle 方法对其进行处理。运用 selector 的好处是，可以使用一个线程实现对多个 channel 的管理，减少了开销，提升了效率。

server 和 client 都是通过 channel 进行读写，而 channel 又是与 buffer 进行交互。server 端写的是 screen 画面信息，读的是用户的按键信息；client 端写的是按键信息，读的是 screen 画面信息。具体实现上，server 端通过 ObjectOutputStream 包裹 ByteArrayOutputStream 的方式，将 screen 对象转换成 byte 数组，然后放入 buffer，再将 buffer 内容写入 channel；用 buffer 读取 channel 中的按键信息（按键信息只有上下左右 4 种情况，在接收到按键时就转换成 int 表示），然后用 ObjectInputStream 包裹 ByteArrayInputStream 的方式将 buffer 的 byte 数组转成 int 型数，即提取到了按键信息。client 端类似，不再赘述。

3.5 图形化显示

我的图形化显示主要是基于老师已经写好的 AsciiPanel 框架。这里 AsciiPanel 类继承自 javax.swing 包下的 JPanel 类，这个包就是 java 提供的图形化框架。在老师原有的代码下运行，可以看到在显示屏上每个格子显示该位置对象的 glyth 属性对应的 ascii 字符，但这里显示出 ascii 字符并不是像在终端 print 一个 char 型变量那样，而是在运行时读取了一张布满 ascii 字符的图片，从图片对应位置截取需要的部分予以显示。在 AsciiPanel 的构造方法中，有一个 loadGlyphs 方法，其中利用 ClassLoader 类的 getResource 方法找到想要的资源，再利用 ImageIO 类的 read 方法把资源图片读进内存的 BufferedImage 对象中。

理解了上述的原理后，将原来的 ascii 字符界面改造成自己想要的图形界面就简单了。首先将自己找到的或手绘的图片放进 resources 目录下，图片大小需要稍作处理。然后在 loadGlyph 方法中仿照老师的方式把自己的图片读取进来。接着修改 glyphs 数组，这个数组里元素的下标就对应 Thing 对象的 glyph 属性值。也就是说，假设在葫芦娃中的 glyth 属性赋值为 1，那么 glyph[1] 就是葫芦娃的图片。这样做也是为了与原有的设计尽量保持吻合。

关于显示，这是 client 端处理的事情。在从 channel 中读出 screen 对象后，调用 repaint。而显示的帧频率其实是由 server 端控制的，server 端每隔 30ms 往 channel 写 screen 对象，client 端是只要读到就立即显示。

4 工程问题

用了老师推荐的 maven 自动构建工具。个人体会，最大的好处是依赖资源的添加，只要在 pom.xml 文件里把所依赖的库添加进去，在打包的时候就会自动把这些资源（包括这些资源依赖的资源）都下载下来，而不用自己预先下载到本地。非常方便。

另外，目录结构也很清晰，源代码区 main 和测试区 test 并列，各个类及其测试代码所在的路径也是完全对应的（测试代码可通过 vscode 中的 Source action -> Generate tests 自动生成）。

关于测试，在写测试代码的时候，我发现其实不太好写。首先是类之间的耦合度比较高，一个类的成员变量里有另一个类。这就意味着，在对一个类进行测试的时候，往往还需要创建另一个类的对象。此外，有些方法的效果也很难通过测试代码的方式去检验。大多数方法的返回值都是 void，也就是说，这些方法的作用完全就是对象状态的改变。一些简单的状态改变还比较好测，但像葫芦娃的 run 方法，就是葫芦娃的行动逻辑，接收用户的输入，并根据当前的全局状态，改变自身的状

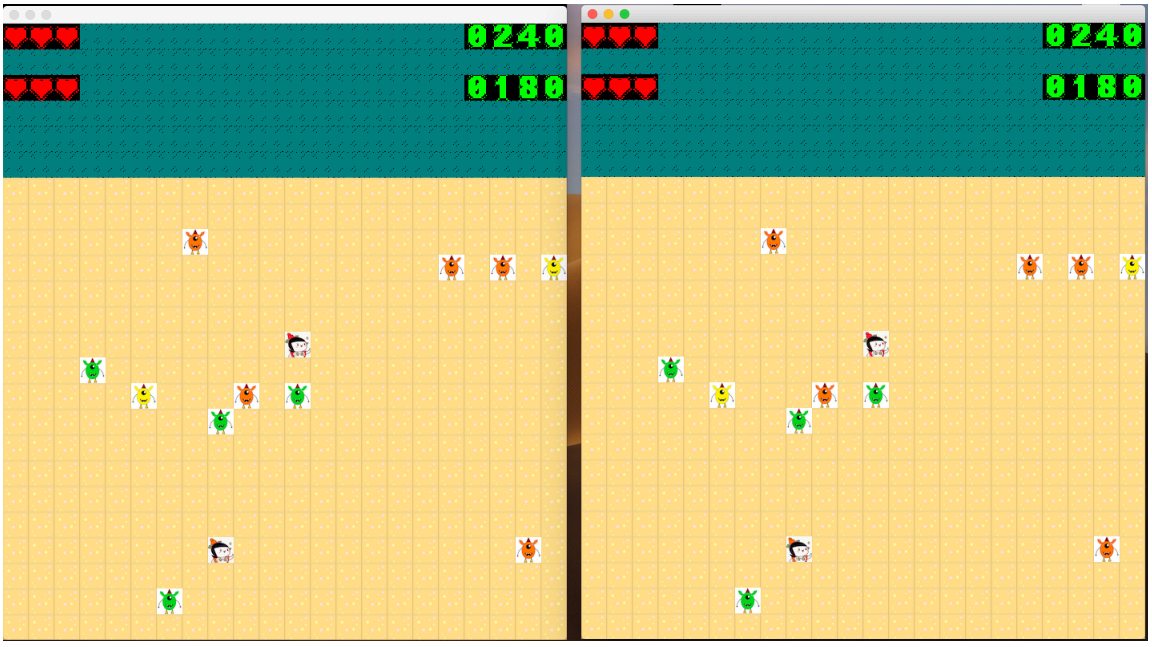

图 1　游戏运行

Figure 1　Running

态及屏幕状态。这个实在很难用单元测试的方法去验证。因此，主要还是通过运行起来之后观察运行情况来分析调试。

5　游戏运行

开发完之后，通过 mvn package 命令得到两个 jar 包，分别是 server 和 client。先启动 server，然后运行 client，可在命令行中加上一个参数表示 server 端的 hostname，若无参数的话默认服务器就在本机，即 hostname 为 localhost。

游戏运行效果如图1所示.

6　课程感言

6.1　关于课程内容

课程内容上，可以说是干货满满，而且处处体现着“高级”二字。语言基础仅花一节课过完（其实还不到）。然后是面向对象，主要是传递“用代码模拟现实”的思想。当然，这部分主要还是自己多写多练，在实践中体会。

接下来就是一个很“高级”的点，即类加载机制。这部分介绍了类是如何被加载到内存中的，还介绍了字节码。应该说，写一般的程序是不需要的了解这些的，但理解底层的原理能帮助我们更好的分析程序和调试。这一部分的作业也很有意思，是利用一个自定义的 ClassLoader 来实现“隐写术”，既帮助我们更好地理解类加载机制，也让我们知道了它能干什么。

接着是泛型，又是一个不太容易理解的知识点，尤其是通配符、协变逆变这些。但老师讲解的还是很清楚的，多看几遍视频并自己琢磨琢磨还是能理解的。之后比较“高级”的点就是并发编程、输入输出和网络通信了，这些都是开发一个大项目所必须用到的。老师讲的都很清楚。尤其是并发编程这一块，在课堂上讲完之后，老师还录了高质量的视频，方便我们反复观看学习，可以说是极其负责了。当然，这部分内容还是得在实践中才能更好的理解，尤其是遇到各式各样的 bug 之后 debug 的过程。

总的来说，课程质量真的非常的高，可以感受到老曹的用心。而且，老曹上课时散发出的自信和从容也让我印象深刻，上课体验非常良好，这样的自信和从容显然是建立在自身过硬的专业水平之上的。

6.2 关于大作业

大作业还是颇具难度和挑战性的（对我来说）。应该说，大作业每个阶段的任务都是和课程内容紧密关联的，这点老师考虑的很周到。但光用课上讲到的内容显然还是远远不够的，有很多需要自学的和自己思考体会的东西。比如，网络通信这一块，上课只是简单地讲了一个 echoserver 的例子，但我们作业是要实现多人网络对战，这其中存在着巨大的 gap。我就在网络对战这个地方卡了很久，因为没有很好的思路，虽然我也知道拖延不好，但想半天想不出来的时候就丧失动力了。所以，我的一点小小的建议是，既然作业要实现网络对战这么一个很高的要求，那课上讲网络通信的时候是否可以更深入更具体一些，或是稍微给点网络对战的提示或思路，也就是减少一些上课与作业之间的 gap。

然后再谈谈我自己在整个过程中的一些感受吧。本身自己玩过的游戏不多，创造力和想象力也比较匮乏，在最开始设计游戏的时候就想到“大鱼吃小鱼”这个逻辑挺简单的游戏，觉得可玩性和可实现性都挺高。然后实现的时候，接收键盘输入的逻辑和画面的显示都是在迷宫部分老师已经写好的，只要稍加改动就可以大体复用，在这点上老师减轻了我们不少的负担。但之后对游戏逻辑设计、并发编程、保存回放、网络对战等部分还是相当的困难，在这个过程中遇到了很多的 bug，由于线程竞争问题出现的不一致性属于是小问题了，经常出现动不了或者画面不显示的情况，也是经常被深深的无力感所笼罩，感觉自己能力真的不行。当然最后还是都一一解决了，或是通过在程序中各处加 print 帮助调试分析，或是和同学讨论，或是上网查找解决方案。

总的说来，整个大作业的过程真的是很艰难，即使最后做出了一个可以流畅运行的游戏，但跟群里各位大佬展示的相比还是只能说非常平庸。这个过程也是让我认识到了自己和别人的差距，以及自己能力上的局限性。未来还有很长的路要走啊，还是要多写多练。

6.3 关于考试

整张考卷给我的感受是，自己好像学了不少东西，但学的还远远不到位，或者说，没学到的比学到的更多，就像苏格拉底的圆圈理论，未知的远比已知的要辽阔的多。要非常感谢老曹很用心的出了这样一张试卷，让我意识到自己学习的局限性，也 push 我在今后的学习中以更加严谨、更加踏实的态度去学习，对知识始终保持敬畏心。

6.4

最后的最后，再次感谢老曹。上了您的课，我受益匪浅。

Developing a Java Game from Scratch

Jiahe Xu[1]

1. *Nanjing University, Nanjing* 210000*, China*
E-mail: 1816036457@qq.com