中国科学：信息科学

SCIENTIA SINICA Informationis

论文

Developing a Java Game from Scratch

欧阳欣[1]

E-mail: m15979634309@163.com

摘要 在一个学期的 JAVA 高级程序设计语言的学习中，随着我对 Java 语言了解的深入，对我将要完成的最终目标——Java Game 的整体形象也愈发清晰。Java 语言的许多特性帮助着我将游戏的构想编程实际的代码。本文章将讲述我是如何从零开始，逐步构建我的 Java Game 的。

关键词

1 前言

在课程的早期，Java 作业还主要是让我们来实际使用面向对象的方式来描述现实中的场景逻辑。由于自己早先学习过 C++ 语言，对面向对象编程有一定了解，因此，这些任务对我而言并不困难。而可能也是这个原因，导致我刚开始对 Java 这门语言的认识还停留在学习 C++ 那时的层面——这不过是 C++ 的一个简单版本罢了。

但是，自从要开始设计一个走迷宫系统时，我慢慢开始觉得 Java 的美妙之处。和 C++ 不同，Java 抛弃 C 的过程命令式语言的枷锁而转向对面向对象的现代设计模式使得它成为了解决从软件的构想到实际代码的这个 Gap 的优秀工具。

这里仅拿在 Game 设计上对我帮助比较大的一点来讲：Java 对设计模式的支持。在 C++ 的学习过程中，我就接触过设计模式的概念。简单的说，设计模式就是编写良好代码的一种范式，而对于 C++ 而言，要实现这种范式，唯有自己手工地一次次搭建起来。而对于 Java，它天生就能支持很多常用的设计模式。首先是工厂模式，而在 Java 中，很多系统类和很多第三方类都大量采用了工厂模式来创建对象。这种创建方式很大程度上方便了使用者。当我需要获取一个类对象的实例时，如果这个类有很多种构造方式，而对使用者来说，仅仅靠构造函数的注释和参数说明可能还是很难理解，那么用能产生这个类的实例的工厂方法获取需要的实例会是个很好的解决办法。可以想象，如果没有这种范式，一个依赖构造参数的不同而存在不同性质的类实例将很难用一种优雅的方式创建出来。在 Game 的设计中，我使用了几种对我的代码设计起了很大作用的设计模式，这将在后文讲到。

刚开始构思 Game 的设计的时候，出现了许多想法，而在写代码验证 demo、debug、项目持续开发的过程中，这些想法有些因为被证实不正确、过于复杂或者超出了能力范围而被舍弃，而最终落实到 Game 的最终版本的，都是经过了很多次优化和验证，被确认为可行的。

在刚开始构建 Game 时，我的初步构想是这样的：System 负责创建和管理资源，而 Game 中的玩家和怪物等 Element 都将通过 System 实现的 API 来访问资源以及和其他 Element。

这样的初步构想被写成代码放在项目的 dev-basic 分支中。从后来一步步阶段性开发的分支来看，这个初步的构想虽然在细节上有所不同，但整个框架并没用太大的变动。而这个开发过程是如何进行的以及这个 Game 的最终版本的各个方面是如何工作的，正是本文将要讲述的内容。

2 构想

小游戏的开发伴随着我的整个程序设计的学习生涯。在我看来，这些类型的游戏有很多共通之处。我理解这些游戏系统和一个计算机系统很是相似：玩家操纵的对象、怪物对象以及各个游戏中的可动甚至不可动的元素，都是一个进程，它们之所以能够进行各种变化以及被显示出来，都是那看不见的底层系统的工作；游戏的底层系统是一个简单的操作系统，它首先管理着游戏中的资源，对于大多数游戏而言，最普遍的资源就是 Position 资源，也就是游戏中的各个元素需要占用一个位置，这个位置属于谁、能给谁、有多少、取值范围是怎样的，都由底层系统实现，这就像是操作系统管理着内存资源。游戏的系统也提供了各个基础功能的实现，例如获取输入，显示图像，这就是游戏系统的 IO 设备。对于游戏中的各个对象元素而言，它们就类似操作系统中的一个个进程，通过系统提供的 API，完成各自的丰富行为，仅仅在需要的时候，通过系统提供的 API 与其他元素通信，而在大多数时候，它们可以自行运行，而不需要考虑其他元素的干扰。

我觉得，大多数简单的游戏都可以通过实现这样一个架构的游戏系统而实现，区别只是在于 API 的不同以及元素的不同行为方式。而我实现的 Game 正是在这种构想的基础上实现的。

3 设计目标

Game 的玩法设计参考的是小时候很喜欢的一个 flash 游戏 Box Head3。这个游戏由玩家操作一个小人，通过杀死关卡中出现的僵尸而获得装备的升级以及新装备的获取。而僵尸也会随着关卡的提升越来越多，越来越强力。而玩家可以使用的武器也有许多，不仅有枪械，也有地雷、炸弹、也有油桶和墙等，这些道具各自有各自的有趣玩法。

Game 的设计目标是实现 Box Head3 的大体玩法，其中比较核心的设计有如下几种：

1. 玩家能在各种武器间切换
2. 玩家可在地图上创建放置物
3. 可多人对战

4 设计理念

基于前文的游戏系统的构想，我提出了 Game 的如下设计理念，如图1所示。这是 Game 的整个系统的架构图，可以看到，Element 是运行在 GSystem 上的各个游戏元素的基类。Element 实例通

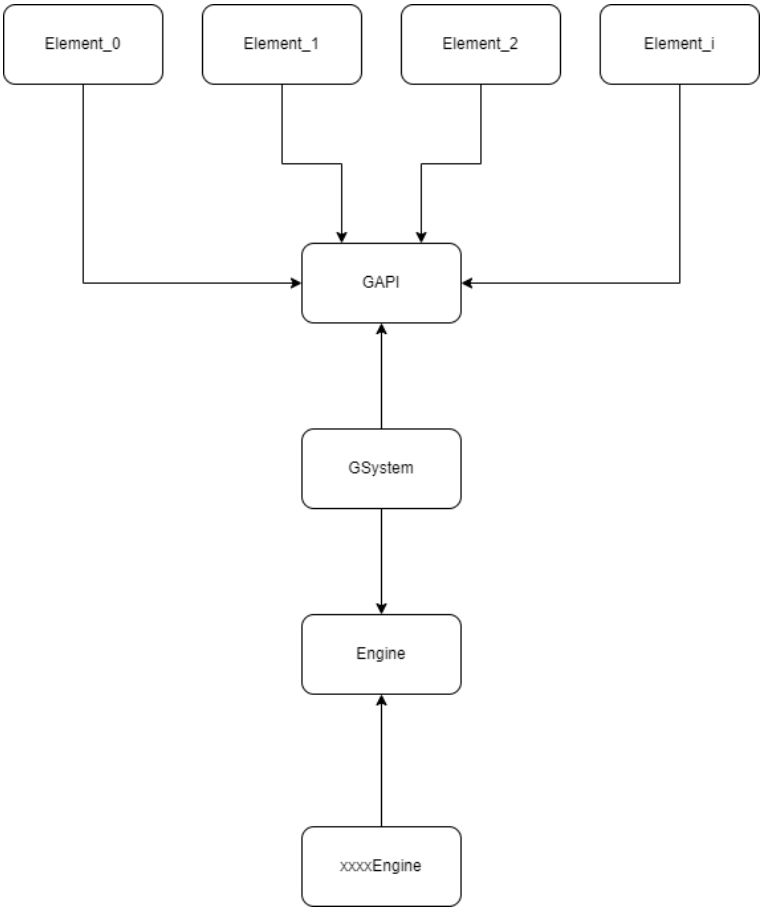

图 1　系统设计
Figure 1

过 GAPI 调用 GSystem 的功能，而 GSystem 自身在管理系统中的各个资源外，还通过 Engine 接口完成获取输入和显示画面的 IO 功能。

以下将详述 Game 系统中的各个部分的结构

4.1　GAPI 及 GSystem

Game 所使用的 GAPI 有如下方法:

1. display（显示 Element）
2. tryOccupy（请求占用一个位置）
3. exist（查询该位置是否存在其他 Element 占用）
4. release（释放对一个位置的占用）
5. getInput（获取输入）

6. getElement（获取对目标元素的引用）

7. register（注册一个 Element）

8. unregister（取消一个 Element 的注册）

9. logOut（显示文本信息）

GSystem 实现了 GAPI 接口。GSystem 管理的主要资源有两个:

1. EPosition 表

2. Element 注册表

其中，EPosition 表示的是 Game 中的位置资源，它实际上是一个锁，同一时间只能有一个 Element 拥有一个 EPosition 对象。GSystem 在初始化时会根据地图大小创建相应个数的 EPosition 对象，每个 EPosition 记录一个唯一的二维坐标。通过持有 EPosition，元素可以告知 GSystem 自己在地图上的位置，同时 GSystem 可以获知 EPosition 的持有者是哪个 Element 实例，并由此进行各种操作，包括 Element 之间的通信。

Element 注册表是一个 String->Element 的哈希表结构，每个 Element 要使用 GSystem，就要向 GSystem 进行注册操作，将自己的字符索引信息加入到这个哈希表中。因而，除了通过获取目标 EPosition 的持有者的方式查找 Element 实例，还能基于字符串名称进行 Element 实例的寻找。

4.2 Element 对象继承体系

继承体系如图2所示。

Element 继承了 Thread 类，因此，可以使用 Element 的 start 方法启动一个线程运行 ELement 实例的 run 方法。Element 是一个抽象类，它定义了其子类的基本运行方法：一、创建后使用者需要调用 init(GAPI) 方法，这样其子类能够进行 GAPI 的调用并进行注册操作；二、可以对子类调用 submit(Operation) 方法，Operation 的不同子类代表对该实例进行不同的影响（这个影响也同时由该方法的被调用者的实现决定），这是 Element 子类之间实现信息传递的方式。三、Element 提供了一个默认的 process(Operation) 方法供子类调用，这个方法的功能是解析自己被传入的信息，完成 Element 互动。

PassiveElement 继承了 Element 类，它也是一个抽象类。这个抽象类重写了 run() 方法，其逻辑为：当自身有被传入的 Operation 实例时，就调用 process(Operation) 方法进行处理。

Barrier 继承了 PassiveELement 类，这个类代表游戏中的不可摧毁也不可通过的元素。它实现的 process(Operation) 方法为空，因此不会对其他 Element 的 submit 产生任何响应。

ActiveElement 继承了 PassiveElement 类，它是一个抽象类，它重写了 run() 方法，他会首先创建一个线程运行其声明的一个 activeProcessor() 方法，然后再调用其父类 PassiveElement 的 run 方法。因而，ActiveElement 的子类将会在主线程上处理 Element 互动信息，在一个子线程上运行 activeProcessor() 方法代表的“主动操作”。所谓“主动操作”即其行为逻辑包含对其他 Element 实例的互动。同时，ActiveElement 为其子类实现了一系列的和移动、显示方法有关的方法。

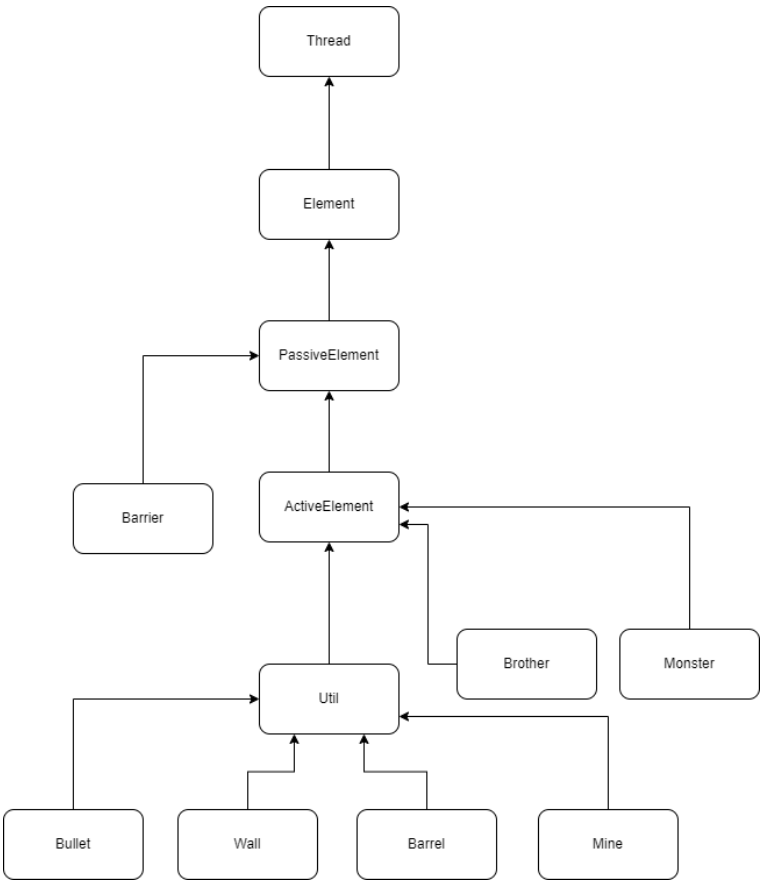

图 2 Element 继承体系

Figure 2

Brother 类继承了 ActiveElement（还有许多其他类继承并实现了 ActiveElement 这个抽象类，这里只介绍 Brother）。Brother 的 activeProcessor() 方法逻辑是：调用 GAPI 进行角色属性的文本显示，然后获取按键输入信息，调用 GAPI 的申请移动操作，如果申请失败则此次移动输入无效；或者是针对攻击命令，创建并启动一个 Util 对象线程；或者是针对切换武器命令，更改当前持有武器属性。

Util 类继承了 ActiveElement。它是一个抽象类，它的子类代表 Game 中的可放置元素，包括“子弹”、“地雷”、“障碍物”等。类似于 Brother，它可以通过重写 activeProcessor() 和 process(Operation) 两个方法完成自己的行为动作。它的子类和 Brother 这些类不同的地方在于，Brother 应该在被创建的时候被注册到游戏系统中，而 Util 子类则通常不需要，因为这些对象不是游戏中的关键元素。

4.3 Engine

Engine 是游戏系统 GSystem 的 IO 接口，承载着显示输出和获取输入的任务。这个接口的不同实现将给游戏带来不同的运行环境，如果这个接口的实现使用本地 IO，那么该游戏就能在运行的机器上读取键盘输入和进行显示器的窗口显示。如果使用的是网络 IO，那么运行这个游戏系统的机器

就变成了服务器，接受连接了这个服务器端口的客户端程序的输入，以及将显示信息传输给客户端。

5 并发控制

Game 的实现并没有实现一个集中的线程管理，这是一个潜在的缺陷。但仅在当前效果看，当前的线程管理还是足够可靠的。

前文已经讲到，每个 Element 实例都在各自的线程上运行，其共享的资源都要通过 GAPI 访问。这种代码结构上的隔离确保了一定的并发安全性。而最重要的设计在于，EPosition 被设计成是一个锁，任何 Eement 实例的线程对 EPosition 的访问都由于锁的存在保证了不会存在对某一个位置的竞争。其次，Element 之间的通信动作被 Operation 封装，也就是说，各个 Element 线程在通信时，仅需要将一个 Operation 对象插入到目标 Element 的 Operation 队列中，而对这个动作的获取和响应则交由目标 Element 自己的线程完成。只要这个插入、取出动作被实现成线程安全的，那就不会有问题，而这只需要使用线程安全的存储 Operation 的队列数据结构即可。因而，对象间的通信并发控制问题被转化成了一个并发数据结构的实现的问题，而后者已经由 Java 实现了。

Element 继承体系是面向对象设计的一个比较好的应用，它将一个复杂对象的属性功能分解成各个层级，不仅仅减少了实现游戏元素类的复杂度，也由于父类对并发的可靠控制，使得具体类的实现不需要去考虑并发问题，这也是面向对象设计方法带来的优势所在。

6 网络通信

Game 开发的前期设计了 Engine 接口，而这个接口的设计在之后将游戏从本地运行到分解为服务器端和客户端这个新的开发过程中起到了巨大的作用，面向对象设计的优势也再一次体现：通过接口解耦各个模块，因而能够实现灵活的模块替换。这部分的整体设计结构如图3所示。

6.1 ServerEngine

ServerEngine 实现了 Engine 接口。ServerEngine 持有一个 ServerSocket 实例，在主线程中，监听是否有新的来自客户端的连接。每当接收到一个新的连接，ServerEngine 就会创建一个 Player 类的子线程。Player 线程会维护一个与客户端的 Socket，并且不断接受来自客户端的输入信息。ServerEngine 对 Engine 接口的如 display 输出方法的实现方法是：调用维护目标客户端 Socket 的 Player 对象的 send() 方法传输相应数据

6.2 传输优化

传输中遇到的最大问题是信息的编码与解析。ServerEngine 和 Clinet 的通信使用 xml 格式字符串作为信息载体，采用如下的编码方式：ServerEngine 对 Client 的信息使用格式 (function:<function>),(arg0:<arg0>),(arg1:<arg1>)..... 编码一个函数调用。而 Client 的回复则较为简单为 (input:<String>)。

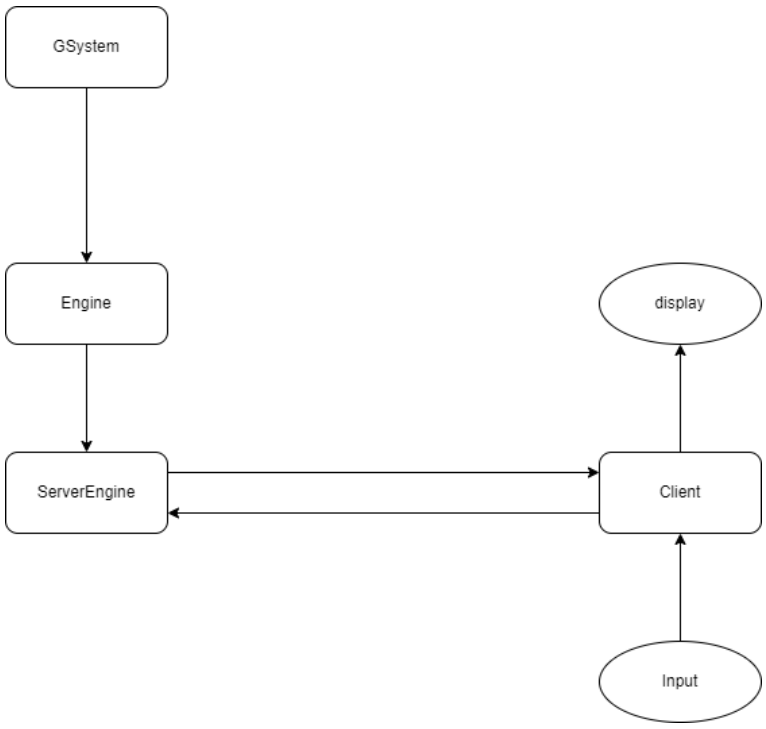

图 3 Element 继承体系
Figure 3

7 开发过程

Game 的开发是一个长期过程，在这个过程中，一下的几个方法的采用很大程度上提高了开发效率。

7.1 设计模式

7.1.1 工厂模式

Game 中存在许多类需要被频繁创建。通过采用工厂方法来创建例如 Brother、Monster、Barrier 等初始对象，省去了大量的构造初始化参数的代码，提高了开发效率和代码美观度。同时，工厂方法的应用使得这些对象可以通过反序列化的方式从配置中创建，这使得 Game 的初始化工作可以完全交由工厂类，通过读取配置文件信息，完全可以动态创建不同的地图、以及实现存档等功能。

7.1.2 命令模式

Element 对象之间的通信采用的是 Operation 封装一个命令，例如攻击命令。这种方式解决了 Element 之间的方法调用依赖，进而实现了前文所述的线程安全控制。

7.1.3 观察者模式

在 ServerEngine 实现 GSystem 的 Engine 的 API 调用时，就会通知其内部创建的所有 Player 对象，这些对象通过 API 调用时传入的 playerID 来识别是否要对自己维护的连接发送信息。通过这种模式，高效地实现了多个连接的信息传输。

7.2 版本控制系统

长期的开发过程中，每一个功能的完成并成功运行后，都会在 git 中提交一个 commit，记录项目的当前版本信息。在后续的继续开发中，如果发现开发过程出现问题，可以回退到一个可运行的项目版本进行再次开发，这避免了在开发过程中意外引入 bug 而导致的开发效率降低。

同时，每当需要修改项目设计，进行一些模块的大的修改或者配置、环境的更改时，可以创建一个新的分支。这样，在开发过程中不仅可以确保随时可以回退到一个早起可运行版本，也能同时在两个分支上进行开发，在确定分支是否有效后，随意选取一个进行后续开发。

在本项目中，进行了数十个 commit 并创建了 8 个分支。其中每个分支都是一个可运行的阶段性版本。

7.3 测试驱动开发

在开发过程中，先写测试，再进行项目开发，是一个提高开发效率的做法。首先，不成熟的代码可以通过实际运行来验证其可靠性，同时，写测试和完善代码的过程也能提示我将代码写的更加精炼。

例如，在写网络传输相关的代码时，就能先写测试代码，将不能熟练掌握的代码功能在单元测试中运行，检查存在的问题，在测试成功后，将这些代码应用到项目中，这样的开发过程效率很高。

8 总结

上完了这门 Java 高级程序课程后，我的感觉是非常充实的。虽然作业的任务量还是比较大的，但是这也强迫我去学习了很多新的东西，通过自己探索、写代码，锻炼自己的学习能力和代码能力。实际上，对于 java，课上讲到的东西只是其一小部分，而对于这门课讲到的内容，我在作业中所实践的也只是一小部分。也就是说，这门课同样让我认识到了自身能力的不足和需要去学习的世界是多么广阔。

如果要问我我对着门课有什么意见的话，我的看法是：ddl 间隔还是比较长。事实上，大部分的任务都是在 ddl 的前几天完成的。所以，如果作业能设置更多阶段性的 ddl 而保持任务量变化不大的话，那对作业的完成质量或许会有更大的提升吧。