中国科学: 信息科学

SCIENTIA SINICA Informationis

论文

# Developing a Java Game from Scratch

彭建涛[1]

1. 南京大学计算机科学与技术系, 江苏南京 000000
E-mail: fiyqkrc@gmail.com

收稿日期: 2021-xx-xx; 接受日期: 2021-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** 该 java 项目是 JAVA 高级程序设计的课程作品。该程序在 jFrame 的基础上实现了一个简单的 UI 框架; 同时, 该程序实现了一个最多四人联机对抗魔物的 rogoulike rpg 游戏。

**关键词** rougelike, java, 游戏, 网络通信, 并发

## 1 开发目标

游戏预览1。

1. UI 框架: 游戏最初基于 jw04 的框架, 想利用 AsciiFont 提供的图形组合设计复杂的游戏角色, 做一个类似飞机大战的 FPS 游戏。为此在 jw04 的基础上设计了一个可以移动玩家观察视角和引入大地图的 scene-view 结构。jw05 发布后, 将已经做的部分改成 UI 框架, 截止目前, UI 框架可以完成鼠标、键盘事件在组件之间传递、提供监听键盘事件的接口、响应窗口调整大小、自动布局、背景等功能。

2. 随机地图: 为避免单一关卡带来的疲倦感和避免关卡设计耗费大量时间, 我决定做一个随机地图的生成程序。在参考网上相关思路后, 借助 jw04 的迷宫生成模块实现了一个随机地牢生成程序。

3. 网络通信: 多人游戏时, 游戏状态以房主为准。同步操作采用帧同步和状态同步, 游戏动作同步主要由帧同步。状态同步会定期由服务端或者发生网络波动的客户端发起。游戏不需求同时开始, 在房主处于多人游戏状态下, 其它玩家可以随时加入游戏。

4. 游戏设计:

   - 游戏角色本身不是一个线程, 但是每一个角色都由一个 controller 控制。

**引用格式:** 作者 1, 作者 2, 作者 3, 等. Developing a Java Game from Scratch. 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


**图 1　游戏截图**
**Figure 1**　game screenshot

- 在脱离战斗和锁定敌人的情况下，怪物行为具有随机性, 怪物技能 CD
- 多种怪物分工明确
- 对控制角色的控制器写成线程，解决键盘操作卡手的问题。
- 实现了所有生物通用的增益系统，玩家自机角色可以捡拾使用宝箱掉落的 Buff, 并在短时间内获得某一方面的属性强化
- 实现地图出口，可交互宝箱等可交互对象

5. 存档功能：所有需要存档的对象实现统一接口，实现从 Object 到 JSON 字符串的互相转换。将地图和游戏信息、creature 信息等打包为 json 字符串作为游戏存档，并提供管理存档的 UI 界面。

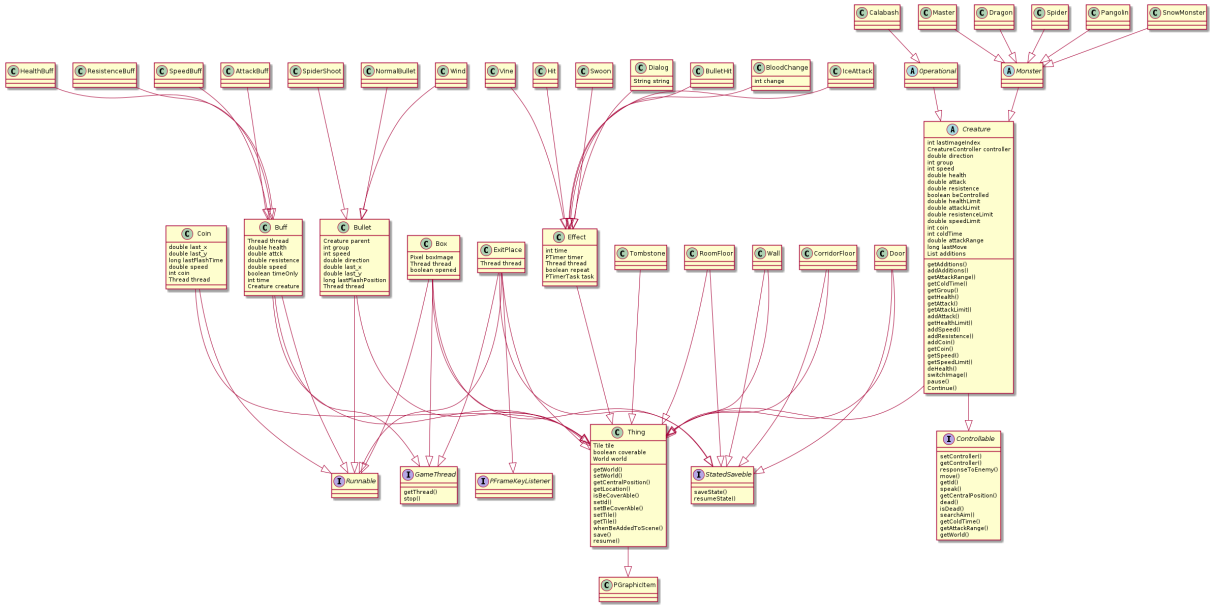


**图 2　类图**

**Figure 2**　class picture

6. 截图和录制：利用该项目已有的 UI 框架，自定义组件获取每一帧图像并转换格式输出。

## 2　设计理念

### 2.1　物体

图2是有关游戏图形组件的结构图

为了提高代码复用率，大部分逻辑代码实现在 Creature,Thing,Effect,Bullet 等几个父类里，它们的衍生子类大多只有资源和参数定制。其中，项目还设计了包括 controllable,PFrameKeyLisnter,GameThread 等接口用来表示不同的功能模块，某个类实现了某个模块即代表支持且需要进行对应功能的操作，这有利于系统其它部分从一堆 Thing 类型的变量识别需要的变量。例如，World 会在存档是将所有为 StatedSavable（状态可以被保存为 json 格式）实例的 Thing 类型变量保存至存档中。因为 Creatrue 实现了 StatedSavable 接口，意味着后来的所有 Creature 子类自创建就默认拥有写进存档的功能，不需要额外代码。

### 2.2　UI

因为 jw04 原来的框架局限太大，无法满足自定义多种形状和大小的图形需求，所以最终做成了 UI 框架。可惜由于底层渲染的方式是由一个一个像素计算和操作，后期也难以用上其它方式改进，加上需要不断刷新，UI 框架反而成为了性能的最大拖累。UI 部分模块可以用来快速创建一些布局限制条件简单的可缩放 UI 界面。因为所有 UI 组件都是 PWidget 子类，因此，各种控件之间

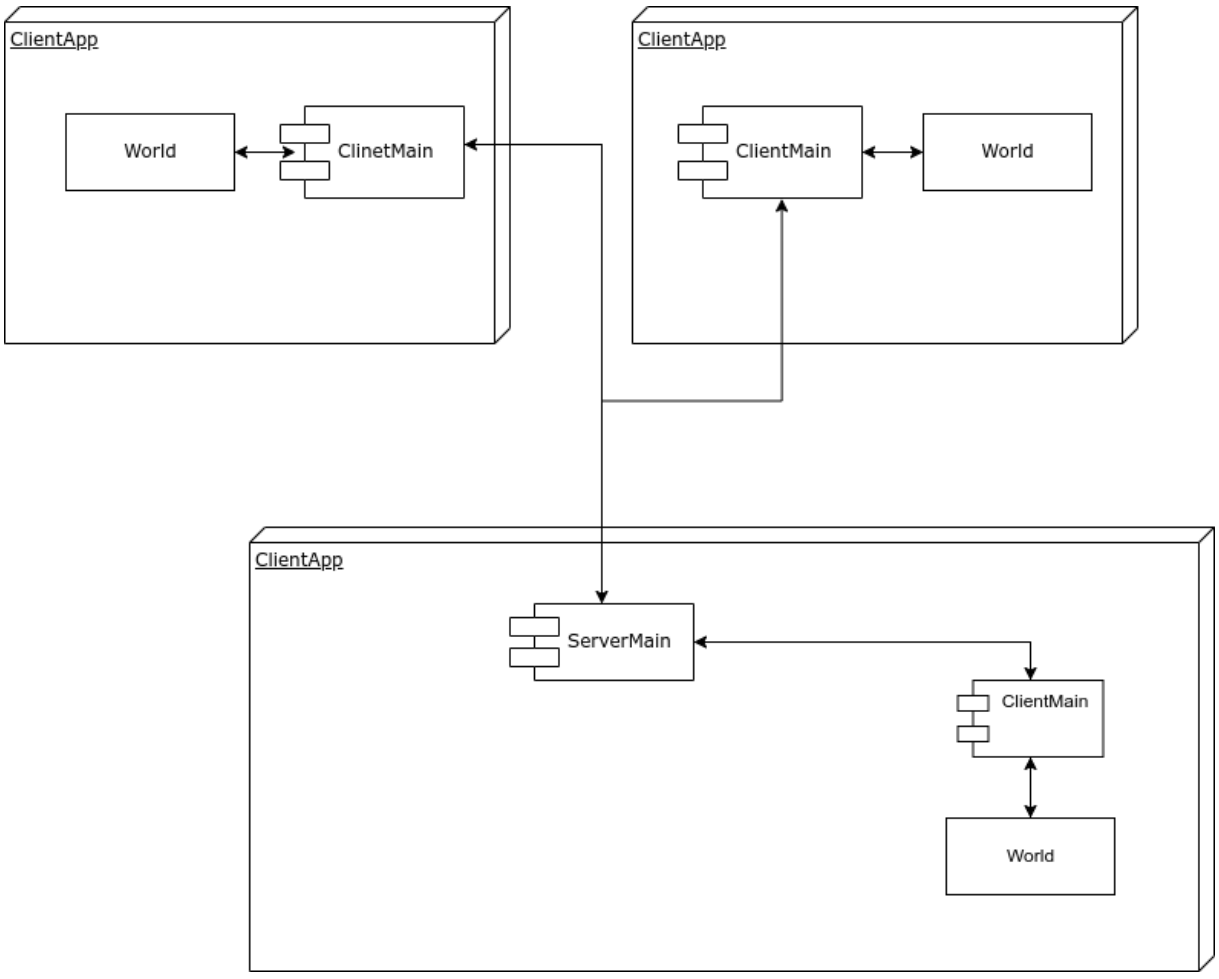


**图 3　联机模式结构图**
**Figure 3**　multi mode model

组合自由度很高；在继承 PWidget（最原始的 UI 组件）并通过重写部分方法，也可以自行编写定制的组件。

### 2.3　NetWork

联机模式结构图3

联机模式的同步策略同时采用帧同步和状态同步的方式。帧同步：系统每 20ms 收集一次来自所有客户端的输入信息，并打包广播给所有客户端，进行这一帧的运算；状态同步：当某个客户端发生丢帧事件后，会向服务端发送状态同步请求，同时服务端每 1 秒也会自动产生一次请求，状态同步时，服务端先向主客户端发送请求获取主客户端的状态信息，再向其它客户端进行广播。

网络部分被设计为多人模式下，启动一个多人游戏时，系统会创建一个客户端和一个服务端 (默认监听 9000 号端口) 并进行连接。服务端会将这个首先连接的客户端作为主客户端，并生成游戏存

档并发送给主客户端。游戏中的一切实际状态以主客户端为主，其它客户端的任何不同步会在状态同步中缩小或消除。

客户端可以通过输入 ip 地址连接服务端。收到请求后，如果连接人数没有超过限制，服务端会生成一个新的角色连同游戏存档发送给客户端进行游戏初始化，同时通知主客户端有新的玩家加入世界。在经过一次状态同步后，新加入的客户端会与其它客户端同步，并进入正常运行。

网络故障和客户端关闭处理。客户端和服务端在遇到网络障碍或者关闭时会自动退出游戏或者发送玩家退出游戏的命令。

### 2.4 存档

游戏提供了专门用于对象状态和 json 对象之间转换的接口:

```
public interface StatedSavable {
        public JSONObject saveState();
        public void resumeState(JSONObject jsonObject);
}
```

一个典型的 Creature 对象生成的 json 字符串:

```
{
        "beCoverAble": false,
        "attack": 20.0,
        "health": 400.0,
        "position": "{1980,620}",
        "id": 10493,
        "class": "game.graphic.creature.monster.SnowMonster",
        "resistance": 0.1,
        "speed": 2,
        "group": 1,
        "coin": 1
}
```

在游戏保存时，程序对于在 world 里的 StatedSaveable 实例获取 JSONObject；在游戏记载时，获取 json 字符串中的 class 名称创建 Object 并读取 jsonObject 恢复状态。将对象的保存和恢复交给具体的对象解决，而大多数的保存工作这些对象的基类就已经完成，这样避免了在加载和保存模块里大量复杂的逻辑代码，同时这些对象 State 恢复和保存的特性也可以为网络中的状态同步提供便利。

# 3 技术问题

## 3.1 并发控制

为了综合考虑程序正确性和并发效率的问题。UI 部分的刷新操作是不会对对象进行上锁操作；对于 World 中对象的移动等操作都是由不同的线程中完成，可能因为多线程同步问题产生冲突，会首先试探目标资源的可用性。如果可用，则先上锁，再试探一次，如果可行就进行下一步操作。

## 3.2 资源节省

因为游戏生成大地图的需求，需要同时生成的对象数量非常之多，不得以采取多种方法节省内存开销：

- 对于已经加载过的图片资源进行缓存，以减少磁盘 IO 次数，加快运行速度
- 对于大部分 Thing 的子类，如果可以共用图片资源，都不会再生成一份图形资源，而是所有类共用资源
- 在单机模式下，地图上的怪物不会被一次性加载出来，特制的维护型 thread 会保证只有角色周围一定范围内的怪物被加载。
- 为了提高 world 的渲染速度，地图被切成一块块方格。单次渲染中，只有在当前视野内的方格内的对象会被渲染。

## 3.3 通信效率

为了提高通信效率，我在主流的帧同步和状态同步中选择了实时性更强、视觉效果更好但更难实现的帧同步。因为如果采用状态同步，同时需要同步的还有大量的子弹和创造物，对网络压力更大；而采用帧同步可以只同步角色和怪物的动作请求，可以减少通信量。同时更重要的是：状态同步下游戏人物的帧数本质取决于一秒内同步的次数，而帧同步传递操作，在本地演算动作，在程序性能好的情况下，帧同步的帧数和观感要远强于状态同步。但纯粹的帧同步往往会因为网络延迟等原因导致不同客户端之间的数据产生差异，最终在某些边缘情况下导致差异被急剧放大。所以，在通信方面，程序采用了帧同步和状态同步的方式。

在通信方面，为了防止因为服务端因为挨个发包阻塞和网络波动导致不同客户端收到包的时间差异过大或者由于单个客户端网络波动导致全部客户端卡顿。单次的发送请求被作为单个线程提交给线程池执行，确保不会阻塞；

# 4 工程问题

## 4.1 运用的设计模式

- 单例模式：多人模式下的 Client 端采用单例模式，保证单个程序只有一个 Client 端
- 工厂方法：程序中 Position,Pixel,Message 都采用了静态工厂方法，它的好处在于，方法内部可能会使用 new 创建一个新的实例，但也可能直接返回一个缓存的实例。对于调用方来说，没必要知道实例创建的细节。

- 享元模式：大部分的 Thing 子类内部通过静态变量共享同一份图形资源，避免不必要的资源加载。
- 命令模式：Client 和 Server 调用发送 Message 只是委托其它线程进行发送行为；
- 观察者、状态模式：大部分 UI 组件在 size 变化后会调用特定的函数处理变化，重新渲染缓存的图像等工作。同时，将 size 变化传递给它的孩子组件，引起其它组件的变化。
- 备忘录模式：可保存状态的对象在游戏存档和加载时与 json 对象之间的互相转换。

### 4.2 工程方法

- 采用单元测试，确保单个子功能功能正常
- 程序模块划分清晰，在完成不同的模块合并后进行大量集成测试模块是否正常工作；
- 采用回归测试，确保程序在不断迭代的情况下功能的正常；