Broadview®
www.broadview.com.cn

我的世界：Minecraft模组开发指南

土球球 著

适合初学者学习模组开发
提升对 Java 编程语言的了解
帮助读者通过编程的方式
实现梦想中的游戏特性

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

我的世界：Minecraft模组开发指南

土球球 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

《我的世界》（Minecraft）是一款风靡全世界的沙盒游戏，是目前 PC 游戏中畅销的游戏之一。作为一款拥有很大自由度的游戏，在社区中也存在一些基于 Minecraft 本身的修改行为，并以一种被称为模组的方式广为传播。此外，通过编写 Java 程序的方式直接控制 Minecraft 的某些行为，在玩游戏中学习编程，能够大大提高青少年入门编程的兴趣。本书将聚焦于面向 Minecraft 模组的开发流程，读者在学习完本书后，将会拥有开发 Minecraft 模组的基本能力，如果读者对 Java 并不熟悉，那么读完本书后也将对 Java 有一个初步的认识。

本书可作为已经对 Minecraft 这款游戏有一定了解的玩家的模组开发入门教程，帮助玩家通过编程的方式实现自己梦想中的游戏特性。本书也可作为已经对模组开发有一定认识的开发者的参考用书，对于专注于旧版本模组开发的开发者，本书将介绍一些针对 Minecraft 新版本的全新特性。

图书在版编目（CIP）数据

我的世界：Minecraft模组开发指南 / 土球球著. —北京：电子工业出版社，2020.7

ISBN 978-7-121-35851-7

Ⅰ. ①我… Ⅱ. ①土… Ⅲ. ①游戏程序—程序设计 Ⅳ. ①TP317.6

中国版本图书馆CIP数据核字（2020）第084857号

责任编辑：孔祥飞　　特约编辑：田学清
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　　邮编：100036
开　　本：787×1092　1/16　印张：18.5　　字数：510千字
版　　次：2020 年 7 月第 1 版
印　　次：2020 年 7 月第 1 次印刷
定　　价：69.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前言

作为一款总销量过亿份、PC 版销量突破三千万份的沙盒游戏，Minecraft 已经成为国内青少年群体中广泛流行的游戏之一。Minecraft 的最初版本由瑞典公司 Mojang AB 开发并在 PC 上运行，其使用 Java 进行编写，是目前世界上畅销的电子游戏之一。

作为一款拥有极大自由度的沙盒游戏，从 Minecraft 测试版发布就开始了针对 Minecraft 游戏本身的修改。这类改动通常被称为模组（Mod，由 Modification 的前三个字母得名）。当时的模组和现在的模组的安装方式不同，是通过替换 Minecraft 游戏本体的方式完成的，这样不仅不方便，还容易引起不同 Mod 之间的冲突。而现在的 Mod 安装方式十分方便，我们只需要把若干个带 jar 后缀的文件放进 mods 目录下，然后启动游戏就可以了。

这归功于一类被称为 ModLoader 的特殊 Mod。ModLoader 本身也经历了若干代演化，目前十分流行的 ModLoader 由一个被称为 MinecraftForge 的组织提供。与此对应的 ModLoader 的名称为 ForgeModLoader，简称 FML。当然，目前还有一些知名度比较高的 ModLoader，如 LiteLoader 等。本书只针对 FML 来讲。

包括 FML 在内的这些 ModLoader 的主要作用只有一个——把模组从 mods 目录中取出，然后按照一套约定俗成的方式加载，并执行其中的部分代码。即使这样，Mod 之间的冲突仍然经常发生。因此，MinecraftForge 同时也提供了一套接口（Application Programming Interface，API），以供 Mod 作者调用，大大减少了 Mod 之间的相互冲突，这套接口被称为 ForgeAPI。在通常情况下，玩家不会刻意去区分 FML 和 ForgeAPI，因为在大多数情况下，这两者都是同时被提及的。

通过本书的学习，读者能掌握编写 Minecraft 模组的基本方法，从而为进一步的模组开发打下基础。同时，这本书也可以帮助开发者对 Java 面向对象的编程语言有更深层次的了解。

这本书是做什么的

顾名思义，这本书是帮助 Minecraft 玩家入门模组开发的。本书不针对 Minecraft 的其他版本，如 Bedrock Edition 等。此外，对于不同的 Minecraft 版本，编写模组的方式也各不相同，为了不落后于游戏本身的发展，本书内容基于目前 Mod 社区十分流行的 Minecraft 版本——1.12.2 之上。

我不会 Java，这本书适合我吗

你完全可以阅读这本书。Minecraft 游戏是由 Java 编写的，因此，这本书讲解的编程知识

将只会考虑 Java。考虑到这一点，这本书会在讲解过程中，穿插一些编程的基础知识，可以使对 Java 还不熟悉的开发者快速了解 Java 的基础框架。当然，编写模组有时会用到一些更高级的 Java 知识，在通常情况下，对 Java 不熟悉的开发者很难遇到这些知识。作者将会在本书第 10 章提醒读者阅读相关的资料。

我可以使用我熟悉的 C/C++/C#/Python/JavaScript 等语言吗

非常遗憾，不可以。使用 Java 编写模组是接触并修改 Minecraft 的内部逻辑很好的方式。其他语言或许可以编写模组，但在实际应用中会非常困难。当然，Java 作为一门语法相对简单的编程语言，我相信有一定编程基础的读者会很快学会它的。另外，有一些编程语言和 Java 有着千丝万缕的联系，实际上也可以用于 Minecraft 的模组编写，不过在这本书里不讨论这类语言。

这本书可以让我成为一名熟练使用 Java 的开发者吗

非常遗憾，也不可以。这本书虽然会介绍一些关于 Java 的知识，但这些知识只是为完成本书涉及的编写模组任务的，还没有完全覆盖编写 Java 代码所需要的所有知识。如果读者想要写出更好的模组，则需要阅读专门讲解 Java 的参考书。事实上，作者也不建议开发者在对 Java 还不熟悉的时候就试图规划编写一个成体系的模组。

在硬件方面，有什么需要准备的吗

实际上，作为一款游戏，编写 Minecraft 模组的确对你使用的计算机硬件有一定的要求。一个基本的要求是内存空间至少为 4GB，在这里建议读者使用 64 位的操作系统，并拥有至少 6GB 的物理内存。另外一个要求是你需要一个比较好的网络环境。大量与开发模组有关的资源都需要从网络上下载，这不可避免地会涉及一些资源，考虑到网络大环境，读者可能需要在编写模组之前，自行准备一些必要的网络工具。

反侵权盗版声明

目录

第 I 部分　整装待发

第 II 部分　小试牛刀

第III部分 登堂入室

第 I 部分

整装待发

好的开头是成功的一半。作为本书的第 I 部分，作者深知这一部分对读者的重要程度。

作者在第 1 章概括性地引入了 Mod 开发和普通的代码开发的不同之处，以帮助读者在了解 Mod 开发前，对这一领域产生一定的感觉。如果读者在 Mod 开发的过程中有不清楚如何实现的地方，回头看看第 1 章，可能会带来不一样的想法。

作者在第 2 章尽可能详细地介绍了配置一个 Mod 开发环境的基本过程，以帮助读者在搭建 Mod 开发环境的过程中尽可能扫清障碍。虽然考虑到内部和外部的因素，配置的过程仍可能困难重重，但配置一旦完成，Mod 开发的大门就可以说是正式打开了。

以下是第 I 部分讲到的所有知识点。

Java 基础：

知道存储源代码的 `.java` 文件和存储字节码的 `.class` 文件在 Java 中的地位，以及两者之间的关系。

知道 JVM、JRE 和 JDK 等概念，以及它们在 Java 的生态系统中的地位。

知道如何安装 OracleJDK 及如何配置 `JAVA_HOME`、`Path` 和 `CLASSPATH` 等环境变量。

知道 IntelliJ IDEA 等集成开发环境在 Java 开发中的地位，以及如何下载安装一个集成开发环境。

知道如何设置诸如 IntelliJ IDEA 等集成开发环境等处的编码为 UTF-8 编码。

知道 Java 代码的基本组成单元是包，包在 Java 代码中的组织结构，以及包和子包的关系。

Minecraft Mod 开发：

知道 Minecraft 本身存在游戏主循环及游戏刻的概念。

知道 Mod 及 Mod 框架的本质是通过在游戏代码中添加钩子的方式实现的。

知道 Mod 框架会通过使用事件系统和注册系统等方式简化 Mod 开发的操作。

知道针对 Forge 的 Mod 开发中 MDK 的存在及其下载位置。

知道如何使用 `gradlew.bat` 或 `./gradlew` 等文件在命令行完成开发环境的配置。

知道如何在 IntelliJ IDEA 等集成开发环境中启动 Minecraft，同时知道如何使用命令行的方式启动。

知道一个 Mod 项目的组织结构及其与项目中的 Java 代码的组织结构的关系。

知道如何构建一个 Mod，以及如何清除构建的相关文件。

第 1 章

电子游戏与 Mod 开发

1.1 电子游戏的运行机制

在讲解 Mod 开发前，先讲讲 Mod 开发所特有的对象。

Minecraft 是一款十分成功的电子游戏。不过，既然和市面上的电子游戏一样，Minecraft 是由计算机程序组织而成的，那么它就逃不过计算机程序本身，换言之，摆脱不了 CPU、内存、显示器及硬盘等的限制。和主流的电子游戏一样，Minecraft 试图为玩家提供一种沉浸式的游戏体验，也就是让玩家在游戏中操控一个虚拟角色带来的体验，要尽可能和现实世界中的体验贴合。

模拟现实世界，一个无论如何都无法逃过的概念就是时间，计算机程序需要在特定的时间为玩家铺设特定的游戏场景，并为玩家设定特定的游戏目标。这听起来十分自然，但实现起来却极其困难，因为时间的概念是连续的，而计算机程序只能处理离散的数据。因此，计算机程序在模拟游戏场景时，需要将连续的时间**离散化**（Discretize），并为每个离散的时刻模拟游戏场景。基于这一理念，引入游戏主循环的概念。

1.1.1 游戏主循环

刻（Tick）是计算机程序模拟游戏场景的基本单位。当游戏加载并运行时，将开始模拟的场景所处的时刻称为第 1 刻，下一个场景称为第 2 刻，以此类推。假设游戏在第 N 刻时停止，我们把计算机程序分解为以下若干步骤。

第一步：初始化游戏

第二步：加载游戏存档

第三步：读取用户输入

第四步：模拟第 1 刻场景

第五步：读取用户输入

第六步：模拟第 2 刻场景

第七步：读取用户输入

……

倒数第三步：读取用户输入

倒数第二步：模拟第 N 刻场景

倒数第一步：保存游戏存档

在上面的步骤中，加粗的部分是核心的模拟过程。我们注意到其中有大段重复的步骤，可以引入一个计数器，将这些步骤合并起来。

第一步：初始化游戏

第二步：加载游戏存档

第三步：引入计数器 tick，赋初值为 0

第四步：将 tick 的值设置为旧值加 1

第五步：读取用户输入

第六步：模拟第 tick 刻场景

第七步：tick 大于或等于 N 吗？如果小于 N 则跳到第四步，如果大于或等于 N 则跳到下一步

第八步：保存游戏存档

可以注意到绝大多数游戏都处于第四步和第七步之间，这是一个循环，我们称之为**游戏主循环**（Game Loop）。几乎所有游戏，其对应的计算机程序内部都至少有一个游戏主循环的实现。

1.1.2　更新频率

保证游戏内两个相邻时刻之间的时间间距相等，对于计算机程序的实现有着极大的便利。这里举一个简单的例子：可以使用相差多少刻这样的方式，来实现事件延时功能。例如，如果希望玩家按下按钮后，约 1 秒后按钮回弹，而相邻两刻之间总是相差固定的 50 毫秒，那么可以在玩家按下按钮后，指定计算机程序在 20tick 后处理回弹。

相邻两刻之差的倒数，就是游戏的更新频率。在通常情况下，Minecraft 这款游戏相邻两刻之间正是相差 50 毫秒，因此更新频率就是 20Hz。这一数值在社区中对应一个更流行的概念：TPS（Ticks Per Second），Minecraft 这款游戏的 TPS 通常为 20。

考虑到相邻时刻之间的时间间距相等这一需求，需要在计算机程序中引入延时的概念，同时，还要引入一个计时器。

第一步：初始化游戏

第二步：加载游戏存档

第三步：引入计数器 tick，赋初值为 0

第四步：启动计时器 timer

第五步：将 tick 的值设置为旧值加 1

第六步：读取用户输入

第七步：模拟第 tick 刻场景

第八步：终止 timer 并重置，得到时间相差 t 毫秒

第九步：延时 (50 - t) 毫秒

第十步：tick 大于或等于 N 吗？如果小于 N 则跳到第四步，如果大于或等于 N 则跳到下一步

第十一步：保存游戏存档

需要注意上面的第九步。第九步基于一个假设：t 比 50 要小，换言之，**在 Minecraft 中，每次读取用户输入和模拟场景的过程，需要在短短的 50 毫秒内做完**。这并不是一个很容易达到的要求，尤其在游戏中添加的 Mod 非常多的时候。如果该要求无法达到，则游戏主循环执行一次的时间就会超过 50 毫秒，游戏的 TPS 就会低于 20，Minecraft 的后台日志就会出现这样一行“臭名昭著”的文字：

Can't keep up! Did the system time change, or is the server overloaded?

因此，在设计 Mod 时，我们需要格外小心位于游戏主循环内执行的代码，并尽量使执行效率达到最高。只有这样，才能让玩家将我们设计的 Mod 和其他数十甚至数百个 Mod 一起使用时，仍然保证执行一次游戏主循环的时间在 50 毫秒内。

1.1.3　游戏状态

从游戏存档本身推知其后任何一个 tick 的场景是不现实的，但是从某个 tick 的场景推知下一个 tick 的场景是很容易做到的。因此，我们会为游戏设置一个**状态**（State），它需要做以下几件事：

- 从存档读入（记为 `state.load()`）
- 写入存档（记为 `state.save()`）
- 处理用户输入（记为 `state.handleInput()`）
- 从上一 tick 更新到下一 tick（记为 `state.tick()`）

我们把计数器 tick 也整合到游戏状态中，同时将 tick 是否大于或等于 N 的比较过程内化进 `state.tick()`，使用一个标志来标记它。这样一来，这个游戏状态还多出了以下两个对象：

- 计数器（记为 `state.currentTick`）
- 是否接着运行游戏的标记（记为 `state.isRunning`）

重新分解游戏的运行步骤：

第一步：初始化游戏，得到游戏状态 state

第二步：加载游戏存档，也就是 state.load()

第三步：启动计时器 timer

第四步：将 state.currentTick 的值设置为旧值加 1

第五步：读取用户输入，也就是 state.handleInput()

第六步：模拟第 tick 刻场景，并更新 state，也就是 state.tick()

第七步：终止 timer 并重置，得到时间相差 t 毫秒

第八步：延时 (50 - t) 毫秒

第九步：state.isRunning 标记为真吗？如果为真则跳到第三步，否则跳到下一步

第十步：保存游戏存档，也就是 state.save()

这其实已经非常接近 Minecraft 游戏的运作机制了。当然，Minecraft 游戏本身的执行逻辑，还是有细小的差别的，比如第五步已经内化进了第六步，游戏也不是只有到游戏主循环结束后才保存存档。对 Java 非常熟悉的读者，可以通过检查 net.minecraft.server.MinecraftServer 类的 run 方法验证上面的所有步骤。

1.1.4　游戏状态的组织结构

游戏状态需能从存档读入，同时需能写入存档。这两点要求游戏状态本身应该包含能够导出成世界存档的全部信息。我们很容易想到游戏状态的内部：若干分立的维度，每个维度都有若干区块，区块里存储着方块状态及方块实体（Tile Entity）。当然，每个维度都会有大量实体，其中有一部分实体是特殊的——它们被称作玩家。对这些游戏元素来说，Java 等编程语言提供了一些非常有效的方式将它们组织起来，包括但不限于类、接口、列表、集合和映射等。

本节大致介绍了 Minecraft 游戏运行的主要框架。接下来，本书将从 Mod 开发者的基本需求出发，介绍对 Minecraft 而言，Mod 在其中扮演了什么样的角色。

1.2　Mod 在游戏程序中的地位

既然 Mod 修改的是游戏原版意料之外的游戏行为，那么就一定会在代码的某些地方发生改变。作为示例，我们考虑玩家进入世界的过程。

游戏主循环：

……

第一步：创建一个代表玩家的游戏元素

第二步：设置玩家的游戏进度、位置和面朝方向

第三步：在世界上生成玩家

……

如果想要在世界上生成玩家前执行一些 Mod 自己的代码，我们会这样做。

游戏主循环：

……

第一步：创建一个代表玩家的游戏元素

第二步：设置玩家的游戏进度、位置和面朝方向

第三步：执行我们 Mod 自己的代码

第四步：在世界上生成玩家

……

在代码中上述行为被称作添加**钩子**（Hook）。这种方法很容易想到，但是就 Minecraft 而言，

一个需要面对的问题是：如果很多个 Mod 希望同时在同一个地方下钩子呢？

游戏主循环：

……

第一步：创建一个代表玩家的游戏元素

第二步：设置玩家的游戏进度、位置和面朝方向

第三步：执行我们 Mod 自己的代码

第四步：执行 ModAlpha 的代码

第五步：执行 ModBeta 的代码

第六步：执行 ModGamma 的代码

……

第 *N* 步：在世界上生成玩家

……

这样的代码太过复杂，而突出的问题在于——它允许一个 Mod 直接修改 Minecraft 本体代码。如果每个 Mod 都私下直接修改 Minecraft 代码，那么每个 Mod 运行时所面对的代码，都将和编写 Mod 时期望的代码有一定差异。为了解决这个问题，世界上出现了各种各样的 Mod 框架，在通常情况下，Mod 框架为 ModLoader 本身，相对应的，在本书中是 FML。每个框架都会维护一个 Mod 的列表，在合适的时机加载 Mod，在 Minecraft 本体代码的合适位置添加钩子，并在钩子触发时告知 Mod 应该做什么。由于添加钩子的工作由 Mod 框架统一完成，因此 Mod 框架的出现，在避免 Mod 直接修改 Minecraft 本体代码的同时，允许 Mod 为 Minecraft 添加各种丰富的功能。为具体解释 Mod 框架的一部分实现细节，在此引入事件系统的概念。

1.2.1　事件系统

就**事件系统**（Event System）而言，Mod 框架会维护一个**事件监听器**（Event Listener）的列表，并在合适的时机**发布事件**（Post Event）。

首先定义事件监听器。如果 `listener` 是一个事件监听器，那么它被要求实现以下方法。

- 接收事件 `event`：`listener.accept(event)`

然后基于事件和事件监听器模拟事件系统的全过程。

初始化：

……

第一步：定义事件监听器的列表 `listenerList`

第二步：向列表添加事件监听器 `listener`：`listenerList.add(0, listener)`

……

游戏主循环：

……

第一步：创建一个代表玩家的游戏元素

第二步：设置玩家的游戏进度、位置和面朝方向

第三步：引入事件 event，代表玩家进入世界的事件

第四步：引入索引 index，赋初值为 0

第五步：index 大于或等于 listenerList.size() 的返回值吗？如果大于或等于则跳到第十步，否则跳到第六步

第六步：引入 listenerList 的第 index 的元素 listener：listener = listenerList.get(index)

第七步：向 listener 发布事件 event：listener.accept(event)

第八步：将 index 赋值为旧值加 1：index = index + 1

第九步：跳到第五步

第十步：在世界上生成玩家

……

在上面的步骤中，只有初始化的第二步，也就是向 listenerList 添加 listener 这一步，是和特定的 Mod 相关联的，剩下的步骤全部可以由 Mod 框架完成。由于这一步是在初始化阶段完成的，因此 Mod 框架完全可以在加载 Mod 时完成这件事，事实上大部分 Mod 框架就是这么做的。我们注意到，对于任何一个钩子，事件监听和触发的机制都是类似的。将事件监听器的列表包装起来，在此引入事件总线的概念。

如果 eventBus 是一个**事件总线**（Event Bus），那么它被要求实现以下方法。

- 注册事件监听器 listener：eventBus.register(listener)
- 发布事件 event：eventBus.post(event)

我们把和玩家登录事件有关的全过程，使用事件总线简化如下。

初始化：

……

第一步：定义事件总线 eventBus

第二步：向事件总线添加事件监听器 listener：eventBus.register(listener)

……

游戏主循环：

……

第一步：创建一个代表玩家的游戏元素

第二步：设置玩家的游戏进度、位置和面朝方向

第三步：引入事件 event，代表玩家进入世界的事件

第四步：向事件总线发布事件 event：eventBus.post(event)

第五步：在世界上生成玩家

……

实际的事件总线实现和上述实现略有差别，但本质上是一样的。很多 Mod 框架都会在 Minecraft 代码中塞入大量的钩子，并提供大量的事件，其中，我们很容易想到的是游戏运行时，在游戏主循环中触发的事件，那么是否有必要在游戏初始化时触发事件呢？答案是肯定的。

1.2.2　注册系统

现在考虑 Minecraft 中的游戏元素，比如物品种类。在之前的部分我们得知，Minecraft 使用一个字符串到物品类型的映射存储所有物品种类。如果想要添加新的物品种类，就要向该映射里添加新的键值对。

整理以下初始化过程。

初始化：

第一步：注册 Minecraft 的游戏元素

……

注册方块

注册状态效果

注册附魔

注册物品

- 定义字符串到物品类型对象的映射 map
- 向 map 添加键为 "minecraft:air"，值为空物品的键值对
- 向 map 添加键为 "minecraft:stone"，值为石头的键值对
- ……

……

第二步：进行 Minecraft 引擎的初始化工作

现在考虑 Mod 框架存在的情况。可能会这样修改 Minecraft 的初始化流程。

初始化：

第一步：初始化 Mod 框架

……

定义事件总线 eventBus

将所有 Mod 的事件监听器添加到 eventBus

……

第二步：注册 Minecraft 的游戏元素

……

注册物品

- 定义字符串到物品类型对象的映射 map
- 向 map 添加键为 "minecraft:air"，值为空物品的键值对
- 向 map 添加键为 "minecraft:stone"，值为石头的键值对
- ……

……

第三步：触发 Mod 游戏元素的注册事件

……

注册 Mod 物品

- **定义 Mod 物品注册事件 event**
- **向事件总线发布事件 event：eventBus.post(event)**

……

第四步：进行 Minecraft 引擎的初始化工作

Mod 框架的做法是在 Minecraft 的游戏元素注册完成后，Minecraft 引擎初始化开始前加入钩子，并触发不同游戏元素的注册事件，从而让 Mod 在事件监听器中注册物品等 Mod 提供的第三方游戏元素。

就 Minecraft 1.12.2 而言，FML 为很多不同种类的游戏元素都提供了注册事件，包括但不限于方块类型、物品类型、状态效果、药水类型、附魔类型和村民类型等。不过，Minecraft 体系庞杂，游戏元素种类繁多，FML 不可能面面俱到，那么怎么办？

FML 除注册事件外还提供了**生命周期**（Life Cycle）事件。生命周期事件有很多种，不过对 Mod 开发来说，常用的生命周期事件只有三种，分别被称为 Pre-Initialization 事件、Initialization 事件和 Post-Initialization 事件。这三种生命周期事件都会在游戏初始化时触发，其中，Pre-Initialization 事件安插在 Minecraft 引擎初始化前，而 Initialization 事件和 Post-Initialization 事件安插在 Minecraft 引擎初始化后。就 Mod 开发而言，何时使用什么生命周期事件往往有一些不成文的惯例，读者将会在本书的后续章节慢慢了解到一些这样的惯例。

现在再将添加了生命周期事件的初始化流程整理如下。

初始化：

第一步：初始化 Mod 框架

……

定义事件总线 eventBus

将所有 Mod 的事件监听器添加到 eventBus

定义生命周期事件总线 lifeCycleEventBus

将所有 Mod 的生命周期事件监听器添加到 lifeCycleEventBus

……

第二步：注册 Minecraft 的游戏元素