

# 游戏编程模式

作者：[美] Robert Nystrom 尼斯卓姆

## 目录

- [版权信息](#)
- [版权声明](#)
- [内容提要](#)
- [作者简介](#)
- [译者简介](#)
- [前言](#)
- [市面上已有的书籍](#)
- [本书和设计模式有什么联系](#)
- [如何阅读本书](#)
- [关于示例代码](#)
- [何去何从](#)
- [致谢](#)
- [第1篇 概述](#)
- [第1章 架构、性能和游戏](#)
- [1.1 什么是软件架构](#)
- [1.1.1 什么是好的软件架构](#)
- [1.1.2 你如何做出改变](#)
- [1.1.3 我们如何从解耦中受益](#)
- [1.2 有什么代价](#)
- [1.3 性能和速度](#)
- [1.4 坏代码中的好代码](#)
- [1.5 寻求平衡](#)
- [1.6 简单性](#)
- [1.7 准备出发](#)
- [第2篇 再探设计模式](#)
- [第2章 命令模式](#)
- [2.1 配置输入](#)
- [2.2 关于角色的说明](#)
- [2.3 撤销和重做](#)
- [2.4 类风格化还是函数风格化](#)
- [2.5 参考](#)
- [第3章 享元模式](#)
- [3.1 森林之树](#)
- [3.2 一千个实例](#)
- [3.3 享元模式](#)
- [3.4 扎根之地](#)
- [3.5 性能表现如何](#)
- [3.6 参考](#)
- [第4章 观察者模式](#)
- [4.1 解锁成就](#)
- [4.2 这一切是怎么工作的](#)
- [4.2.1 观察者](#)
- [4.2.2 被观察者](#)
- [4.2.3 可被观察的物理模块](#)
- [4.3 它太慢了](#)
- [它太快了](#)
- [4.4 太多的动态内存分配](#)
- [4.4.1 链式观察者](#)
- [4.4.2 链表节点池](#)
- [4.5 余下的问题](#)
- [4.5.1 销毁被观察者和观察者](#)
- [4.5.2 不用担心，我们有GC](#)
- [4.5.3 接下来呢](#)
- [4.6 观察者模式的现状](#)
- [4.7 观察者模式的未来](#)
- [第5章 原型模式](#)
- [5.1 原型设计模式](#)

- [5.1.1 原型模式效果如何](#)
- [5.1.2 生成器函数](#)
- [5.1.3 模板](#)
- [5.1.4 头等公民类型 \(First-class types\)](#)
- [5.2 原型语言范式](#)
  - [5.2.1 Self语言](#)
  - [5.2.2 结果如何](#)
  - [5.2.3 JavaScript如何](#)
- [5.3 原型数据建模](#)
- [第6章 单例模式](#)
  - [6.1 单例模式](#)
    - [6.1.1 确保一个类只有一个实例](#)
    - [6.1.2 提供一个全局指针以访问唯一实例](#)
  - [6.2 使用情境](#)
  - [6.3 后悔使用单例的原因](#)
    - [6.3.1 它是一个全局变量](#)
    - [6.3.2 它是个画蛇添足的解决方案](#)
    - [6.3.3 延迟初始化剥离了你的控制](#)
  - [6.4 那么我们该怎么做](#)
    - [6.4.1 看你究竟是否需要类](#)
    - [6.4.2 将类限制为单一实例](#)
    - [6.4.3 为实例提供便捷的访问方式](#)
  - [6.5 剩下的问题](#)
- [第7章 状态模式](#)
  - [7.1 我们曾经相遇过](#)
  - [7.2 救星：有限状态机](#)
  - [7.3 枚举和分支](#)
  - [7.4 状态模式](#)
    - [7.4.1 一个状态接口](#)
    - [7.4.2 为每一个状态定义一个类](#)
    - [7.4.3 状态委托](#)
  - [7.5 状态对象应该放在哪里呢](#)
    - [7.5.1 静态状态](#)
    - [7.5.2 实例化状态](#)
  - [7.6 进入状态和退出状态的行为](#)
  - [7.7 有什么收获吗](#)
  - [7.8 并发状态机](#)
  - [7.9 层次状态机](#)
  - [7.10 下推自动机](#)
  - [7.11 现在知道它们有多有用了吧](#)
- [第3篇 序列型模式](#)
- [第8章 双缓冲](#)
  - [8.1 动机](#)
    - [8.1.1 计算机图形系统是如何工作的 \(概述\)](#)
    - [8.1.2 第一幕，第一场](#)
    - [8.1.3 回到图形上](#)
  - [8.2 模式](#)
  - [8.3 使用情境](#)
  - [8.4 注意事项](#)
    - [8.4.1 交换本身需要时间](#)
    - [8.4.2 我们必须有两份缓冲区](#)
  - [8.5 示例代码](#)
    - [8.5.1 并非只针对图形](#)
    - [8.5.2 人工非智能](#)
    - [8.5.3 缓存这些巴掌](#)
  - [8.6 设计决策](#)
    - [8.6.1 缓冲区如何交换](#)
    - [8.6.2 缓冲区的粒度如何](#)
  - [8.7 参考](#)
- [第9章 游戏循环](#)
  - [9.1 动机](#)
    - [9.1.1 CPU探秘](#)
    - [9.1.2 事件循环](#)
    - [9.1.3 时间之外的世界](#)

## [9.1.4 秒的长短](#)

## [9.2 模式](#)

## [9.3 使用情境](#)

## [9.4 使用须知](#)

[你可能需要和操作系统的事件循环进行协调](#)

## [9.5 示例代码](#)

### [9.5.1 跑，能跑多快就跑多快](#)

### [9.5.2 小睡一会儿](#)

### [9.5.3 小改动，大进步](#)

### [9.5.4 把时间追回来](#)

### [9.5.5 留在两帧之间](#)

## [9.6 设计决策](#)

### [9.6.1 谁来控制游戏循环，你还是平台](#)

### [9.6.2 你如何解决能量损耗](#)

### [9.6.3 如何控制游戏速度](#)

## [9.7 参考](#)

## [第10章 更新方法](#)

### [10.1 动机](#)

### [10.2 模式](#)

### [10.3 使用情境](#)

### [10.4 使用须知](#)

#### [10.4.1 将代码划分至单帧之中使其变得更加复杂](#)

#### [10.4.2 你需要在每帧结束前存储游戏状态以便下一帧继续](#)

#### [10.4.3 所有对象都在每帧进行模拟，但并非真正同步](#)

#### [10.4.4 在更新期间修改对象列表时必须谨慎](#)

## [10.5 示例代码](#)

### [10.5.1 子类化实体](#)

### [10.5.2 定义实体](#)

### [10.5.3 逝去的时间](#)

## [10.6 设计决策](#)

### [10.6.1 update方法依存于何类中](#)

### [10.6.2 那些未被利用的对象该如何处理](#)

## [10.7 参考](#)

## [第4篇 行为型模式](#)

## [第11章 字节码](#)

### [11.1 动机](#)

#### [11.1.1 魔法大战](#)

#### [11.1.2 先数据后编码](#)

#### [11.1.3 解释器模式](#)

#### [11.1.4 虚拟机器码](#)

## [11.2 字节码模式](#)

### [11.3 使用情境](#)

### [11.4 使用须知](#)

#### [11.4.1 你需要个前端界面](#)

#### [11.4.2 你会想念调试器的](#)

## [11.5 示例](#)

### [11.5.1 法术API](#)

### [11.5.2 法术指令集](#)

### [11.5.3 栈机](#)

### [11.5.4 组合就能得到行为](#)

### [11.5.5 一个虚拟机](#)

### [11.5.6 语法转换工具](#)

## [11.6 设计决策](#)

### [11.6.1 指令如何访问堆栈](#)

### [11.6.2 应该有哪些指令](#)

### [11.6.3 值应当如何表示](#)

### [11.6.4 如何生成字节码](#)

## [11.7 参考](#)

## [第12章 子类沙盒](#)

### [12.1 动机](#)

### [12.2 沙盒模式](#)

### [12.3 使用情境](#)

### [12.4 使用须知](#)

## [12.5 示例](#)

- [12.6 设计决策](#)
- [12.6.1 需要提供什么操作](#)
- [12.6.2 是直接提供函数，还是由包含它们的对象提供](#)
- [12.6.3 基类如何获取其所需的状态](#)
- [12.7 参考](#)
- [第13章 类型对象](#)
- [13.1 动机](#)
- [13.1.1 经典的面向对象方案](#)
- [13.1.2 一种类型一个类](#)
- [13.2 类型对象模式](#)
- [13.3 使用情境](#)
- [13.4 使用须知](#)
- [13.4.1 类型对象必须手动跟踪](#)
- [13.4.2 为每个类型定义行为更困难](#)
- [13.5 示例](#)
- [13.5.1 构造函数：让类型对象更加像类型](#)
- [13.5.2 通过继承共享数据](#)
- [13.6 设计决策](#)
- [13.6.1 类型对象应该封装还是暴露](#)
- [13.6.2 持有类型对象如何创建](#)
- [13.6.3 类型能否改变](#)
- [13.6.4 支持何种类型的派生](#)
- [13.7 参考](#)
- [第5篇 解耦型模式](#)
- [第14章 组件模式](#)
- [14.1 动机](#)
- [14.1.1 难题](#)
- [14.1.2 解决难题](#)
- [14.1.3 宽松的末端](#)
- [14.1.4 捆绑在一起](#)
- [14.2 模式](#)
- [14.3 使用情境](#)
- [14.4 注意事项](#)
- [14.5 示例代码](#)
- [14.5.1 一个庞大的类](#)
- [14.5.2 分割域](#)
- [14.5.3 分割其余部分](#)
- [14.5.4 重构Bjom](#)
- [14.5.5 删掉Bjom](#)
- [14.6 设计决策](#)
- [14.6.1 对象如何获得组件](#)
- [14.6.2 组件之间如何传递信息](#)
- [14.7 参考](#)
- [第15章 事件队列](#)
- [15.1 动机](#)
- [15.1.1 用户图形界面的事件循环](#)
- [15.1.2 中心事件总线](#)
- [15.1.3 说些什么好呢](#)
- [15.2 事件队列模式](#)
- [15.3 使用情境](#)
- [15.4 使用须知](#)
- [15.4.1 中心事件队列是个全局变量](#)
- [15.4.2 游戏世界的状态任你掌控](#)
- [15.4.3 你会在反馈系统循环中绕圈子](#)
- [15.5 示例代码](#)
- [15.5.1 环状缓冲区](#)
- [15.5.2 汇总请求](#)
- [15.5.3 跨越线程](#)
- [15.6 设计决策](#)
- [15.6.1 入队的是什么](#)
- [15.6.2 谁能从队列中读取](#)
- [15.6.3 谁可以写入队列](#)
- [15.6.4 队列中对象的生命周期是什么](#)
- [15.7 参考](#)

## [第16章 服务定位器](#)

### [16.1 动机](#)

### [16.2 服务定位器模式](#)

### [16.3 使用情境](#)

### [16.4 使用须知](#)

#### [16.4.1 服务必须被定位](#)

#### [16.4.2 服务不知道被谁定位](#)

### [16.5 示例代码](#)

#### [16.5.1 服务](#)

#### [16.5.2 服务提供者](#)

#### [16.5.3 简单的定位器](#)

#### [16.5.4 空服务](#)

#### [16.5.5 日志装饰器](#)

### [16.6 设计决策](#)

#### [16.6.1 服务是如何被定位的](#)

#### [16.6.2 当服务不能被定位时发生了什么](#)

#### [16.6.3 服务的作用域多大](#)

### [16.7 其他参考](#)

## [第6篇 优化型模式](#)

## [第17章 数据局部性](#)

### [17.1 动机](#)

#### [17.1.1 数据仓库](#)

#### [17.1.2 CPU的托盘](#)

#### [17.1.3 等下，数据即性能](#)

### [17.2 数据局部性模式](#)

### [17.3 使用情境](#)

### [17.4 使用须知](#)

### [17.5 示例代码](#)

#### [17.5.1 连续的数组](#)

#### [17.5.2 包装数据](#)

#### [17.5.3 热/冷分解](#)

### [17.6 设计决策](#)

#### [17.6.1 你如何处理多态](#)

#### [17.6.2 游戏实体是如何定义的](#)

### [17.7 参考](#)

## [第18章 脏标记模式](#)

### [18.1 动机](#)

#### [18.1.1 局部变换和世界变换](#)

#### [18.1.2 缓存世界变换](#)

#### [18.1.3 延时重算](#)

### [18.2 脏标记模式](#)

### [18.3 使用情境](#)

### [18.4 使用须知](#)

#### [18.4.1 延时太长会有代价](#)

#### [18.4.2 必须保证每次状态改动时都设置脏标记](#)

#### [18.4.3 必须在内存中保存上次的衍生数据](#)

### [18.5 示例代码](#)

#### [18.5.1 未优化的遍历](#)

#### [18.5.2 让我们“脏”起来](#)

### [18.6 设计抉择](#)

#### [18.6.1 何时清除脏标记](#)

#### [18.6.2 脏标记追踪的粒度多大](#)

### [18.7 参考](#)

## [第19章 对象池](#)

### [19.1 动机](#)

#### [19.1.1 碎片化的害处](#)

#### [19.1.2 二者兼顾](#)

### [19.2 对象池模式](#)

### [19.3 使用情境](#)

### [19.4 使用须知](#)

#### [19.4.1 对象池可能在闲置的对象上浪费内存](#)

#### [19.4.2 任意时刻处于存活状态的对象数目恒定](#)

#### [19.4.3 每个对象的内存大小是固定的](#)

#### [19.4.4 重用对象不会被自动清理](#)

[19.4.5 未使用的对象将占用内存](#)

[19.5 示例代码](#)

[空闲表](#)

[19.6 设计决策](#)

[19.6.1 对象是否被加入对象池](#)

[19.6.2 谁来初始化那些被重用的对象](#)

[19.7 参考](#)

[第20章 空间分区](#)

[20.1 动机](#)

[20.1.1 战场上的部队](#)

[20.1.2 绘制战线](#)

[20.2 空间分区模式](#)

[20.3 使用情境](#)

[20.4 使用须知](#)

[20.5 示例代码](#)

[20.5.1 一张方格纸](#)

[20.5.2 相连单位的网格](#)

[20.5.3 进入战场](#)

[20.5.4 刀光剑影的战斗](#)

[20.5.5 冲锋陷阵](#)

[20.5.6 近在咫尺，短兵相接](#)

[20.6 设计决策](#)

[20.6.1 分区是层级的还是扁平的](#)

[20.6.2 分区依赖于对象集合吗](#)

[20.6.3 对象只存储在分区中吗](#)

[20.7 参考](#)

[欢迎来到异步社区！](#)

# 版权信息

书名：游戏编程模式

ISBN：978-7-115-42688-8

本书由人民邮电出版社发行数字版。版权所有，侵权必究。

---

您购买的人民邮电出版社电子书仅供您个人使用，未经授权，不得以任何方式复制和传播本书内容。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

---

• 著 [美] Robert Nystrom

译 GPP翻译组

责任编辑 陈冀康

• 人民邮电出版社出版发行 北京市丰台区成寿寺路11号

邮编 100164 电子邮件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

• 读者服务热线：(010)81055410

反盗版热线：(010)81055315

# 版权声明

Simplified Chinese translation copyright © 2016 by Posts and Telecommunications Press

ALL RIGHTS RESERVED

Game Programming Patterns by Robert Nystrom

Copyright © 2014 by Robert Nystrom

本书中文简体版由作者**Robert Nystrom**授权人民邮电出版社出版。未经出版者书面许可，对本书的任何部分不得以任何方式或任何手段复制和传播。

版权所有，侵权必究。



# 内容提要

游戏开发一直是热门的领域，掌握良好的游戏编程模式将是开发人员的必备技能。本书细致地讲解了游戏开发需要用到的各种编程模式，并提供了丰富的示例。

全书共6篇20章。第1篇概述了架构、性能和游戏的关系，第2篇回顾了GoF经典的6种模式。第3篇到第6篇，按照序列型模式、行为型模式、解耦型模式和优化型模式的分类，详细讲解了游戏编程中常用的13种有效的模式。

本书提供了丰富的代码示例，通过理论和代码示例相结合的方式帮助读者更好地学习。无论是游戏领域的设计人员、开发人员，还是想要进入游戏开发领域的学生和普通程序员，都可以阅读本书。

## 作者简介

Robert Nystrom是一位具备超过20年职业编程经验的开发者，而其中大概一半时间用于从事游戏开发。在艺电（Electronic Arts）的8年时间里，他曾参与劲爆美式足球（Madden）系列这样庞大的项目，也曾投身于亨利·海茨沃斯大冒险（Henry Hatsworth in the Puzzling Adventure）这样稍小规模的游戏开发之中。他所开发的游戏遍及PC、GameCube、PS2、XBox、X360以及DS平台。但最傲人之处在于，他为开发者们提供了开发工具和共享库。他热衷于寻求易用的、漂亮的代码来延伸和增强开发者们的创造力。

Robert与他的妻子和两个女儿定居于西雅图，在那里你很有可能会见到他正在为朋友们下厨，或者在为他们上啤酒。

欢迎访问：电子书学习和下载网站 (<https://www.shgis.cn>)

文档名称：《游戏编程模式》[美] Robert Nystrom 尼斯卓姆 著.epub

请登录 <https://shgis.cn/post/313.html> 下载完整文档。

手机端请扫码查看：

