

# Java程序员修炼之道 (图灵程序设计丛书 79)

作者: [英]Benjamin J. Evans [荷兰]Martijn Verburg

## 版权信息

书名: Java程序员修炼之道

作者: Benjamin J. Evans, Martijn Verburg

译者: 吴海星

ISBN: 978-7-115-32195-4

本书由北京图灵文化发展有限公司发行数字版。版权所有，侵权必究。

---

您购买的图灵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。

# 目录

[版权声明](#)

[序](#)

[前言](#)

[致谢](#)

[关于本书](#)

[阅读须知](#)

[读者对象](#)

[路线图](#)

[代码约定及下载](#)

[软件需求](#)

[作者在线](#)

[关于作者](#)

[关于封面图片](#)

[第一部分 用Java 7做开发](#)

[第1章 初识Java 7](#)

[1.1 语言与平台](#)

[1.2 Coin项目：浓缩的都是精华](#)

[1.3 Coin项目中的修改](#)

[1.3.1 switch语句中的String](#)

[1.3.2 更强的数值文本表示法](#)

[1.3.3 改善后的异常处理](#)

[1.3.4 Try-with-resources \(TWR\)](#)

[1.3.5 钻石语法](#)

[1.3.6 简化变参方法调用](#)

[1.4 小结](#)

[第2章 新I/O](#)

[2.1 Java I/O简史](#)

[2.1.1 Java 1.0到1.3](#)

[2.1.2 在Java 1.4中引入的NIO](#)

[2.1.3 下一代I/O-NIO.2](#)

[2.2 文件I/O的基石：Path](#)

[2.2.1 创建一个Path](#)

[2.2.2 从Path中获取信息](#)

[2.2.3 移除冗余项](#)

[2.2.4 转换Path](#)

[2.2.5 NIO.2 Path和Java已有的File类](#)

[2.3 处理目录和目录树](#)

[2.3.1 在目录中查找文件](#)

[2.3.2 遍历目录树](#)

[2.4 NIO.2的文件系统I/O](#)

[2.4.1 创建和删除文件](#)

[2.4.2 文件的复制和移动](#)

[2.4.3 文件的属性](#)

[2.4.4 快速读写数据](#)

[2.4.5 文件修改通知](#)

[2.4.6 SeekableByteChannel](#)

- [2.5 异步 IO 操作](#)
  - [2.5.1 将来式](#)
  - [2.5.2 回调式](#)
- [2.6 Socket和Channel的整合](#)
  - [2.6.1 NetworkChannel](#)
  - [2.6.2 MulticastChannel](#)
- [2.7 小结](#)
- [第二部分 关键技术](#)
- [第3章 依赖注入](#)
  - [3.1 知识注入：理解IoC和DI](#)
    - [3.1.1 控制反转](#)
    - [3.1.2 依赖注入](#)
    - [3.1.3 转成DI](#)
  - [3.2 Java中标准化的DI](#)
    - [3.2.1 @Inject注解](#)
    - [3.2.2 @Qualifier注解](#)
    - [3.2.3 @Named注解](#)
    - [3.2.4 @Scope注解](#)
    - [3.2.5 @Singleton注解](#)
    - [3.2.6 接口Provider<T>](#)
  - [3.3 Java中的DI参考实现：Guice 3](#)
    - [3.3.1 Guice新手指南](#)
    - [3.3.2 水手绳结：Guice的各种绑定](#)
    - [3.3.3 在Guice中限定注入对象的生命周期](#)
- [3.4 小结](#)
- [第4章 现代并发](#)
  - [4.1 并发理论简介](#)
    - [4.1.1 解释Java线程模型](#)
    - [4.1.2 设计理念](#)
    - [4.1.3 这些原则如何以及为何会相互冲突](#)
    - [4.1.4 系统开销之源](#)
    - [4.1.5 一个事务处理的例子](#)
  - [4.2 块结构并发（Java 5之前）](#)
    - [4.2.1 同步与锁](#)
    - [4.2.2 线程的状态模型](#)
    - [4.2.3 完全同步对象](#)
    - [4.2.4 死锁](#)
    - [4.2.5 为什么是synchronized](#)
    - [4.2.6 关键字volatile](#)
    - [4.2.7 不可变性](#)
  - [4.3 现代并发应用程序的构件](#)
    - [4.3.1 原子类：java.util.concurrent.atomic](#)
    - [4.3.2 线程锁：java.util.concurrent.locks](#)
    - [4.3.3 CountdownLatch](#)
    - [4.3.4 ConcurrentHashMap](#)
    - [4.3.5 CopyOnWriteArrayList](#)
    - [4.3.6 Queue](#)

## [4.4 控制执行](#)

### [4.4.1 任务建模](#)

#### [4.4.2 ScheduledThreadPoolExecutor](#)

## [4.5 分支/合并框架](#)

### [4.5.1 一个简单的分支/合并例子](#)

#### [4.5.2 ForkJoinTask与工作窃取](#)

### [4.5.3 并行问题](#)

## [4.6 Java内存模型](#)

## [4.7 小结](#)

## [第5章 类文件与字节码](#)

### [5.1 类加载和类对象](#)

#### [5.1.1 加载和连接：概览](#)

#### [5.1.2 验证](#)

#### [5.1.3 Class对象](#)

#### [5.1.4 类加载器](#)

#### [5.1.5 示例：依赖注入中的类加载器](#)

## [5.2 使用方法句柄](#)

### [5.2.1 MethodHandle](#)

### [5.2.2 MethodType](#)

### [5.2.3 查找方法句柄](#)

#### [5.2.4 示例：反射、代理与方法句柄](#)

#### [5.2.5 为什么选择MethodHandle](#)

## [5.3 检查类文件](#)

### [5.3.1 介绍javap](#)

### [5.3.2 方法签名的内部形式](#)

### [5.3.3 常量池](#)

## [5.4 字节码](#)

### [5.4.1 示例：反编译类](#)

### [5.4.2 运行时环境](#)

### [5.4.3 操作码介绍](#)

#### [5.4.4 加载和储存操作码](#)

#### [5.4.5 数学运算操作码](#)

#### [5.4.6 执行控制操作码](#)

#### [5.4.7 调用操作码](#)

#### [5.4.8 平台操作操作码](#)

#### [5.4.9 操作码的快捷形式](#)

#### [5.4.10 示例：字符串拼接](#)

## [5.5 Invokedynamic](#)

### [5.5.1 invokedynamic如何工作](#)

#### [5.5.2 示例：反编译invokedynamic调用](#)

## [5.6 小结](#)

## [第6章 理解性能调优](#)

### [6.1 性能术语](#)

#### [6.1.1 等待时间](#)

#### [6.1.2 吞吐量](#)

#### [6.1.3 利用率](#)

#### [6.1.4 效率](#)

- [6.1.5 容量](#)
- [6.1.6 扩展性](#)
- [6.1.7 退化](#)
- [6.2 务实的性能分析法](#)
  - [6.2.1 知道你在测量什么](#)
  - [6.2.2 知道怎么测量](#)
  - [6.2.3 知道性能目标是什么](#)
  - [6.2.4 知道什么时候停止优化](#)
  - [6.2.5 知道高性能的成本](#)
  - [6.2.6 知道过早优化的危险](#)
- [6.3 哪里出错了？我们担心的原因](#)
  - [6.3.1 过去和未来的性能趋势：摩尔定律](#)
  - [6.3.2 理解内存延迟层级](#)
  - [6.3.3 为什么Java性能调优存在困难](#)
- [6.4 一个来自于硬件的时间问题](#)
  - [6.4.1 硬件时钟](#)
  - [6.4.2 麻烦的nanoTime\(\)](#)
  - [6.4.3 时间在性能调优中的作用](#)
  - [6.4.4 案例研究：理解缓存未命中](#)
- [6.5 垃圾收集](#)
  - [6.5.1 基本算法](#)
  - [6.5.2 标记和清除](#)
  - [6.5.3 jmap](#)
  - [6.5.4 与GC相关的JVM参数](#)
  - [6.5.5 读懂GC日志](#)
  - [6.5.6 用VisualVM查看内存使用情况](#)
  - [6.5.7 逃逸分析](#)
  - [6.5.8 并发标记清除](#)
  - [6.5.9 新的收集器：G1](#)
- [6.6 HotSpot的JIT编译](#)
  - [6.6.1 介绍HotSpot](#)
  - [6.6.2 内联方法](#)
  - [6.6.3 动态编译和独占调用](#)
  - [6.6.4 读懂编译日志](#)
- [6.7 小结](#)

第三部分 JVM上的多语言编程

第7章 备选JVM语言

- [7.1 Java 太笨？纯粹诽谤](#)
  - [7.1.1 整合系统](#)
  - [7.1.2 函数式编程的基本原理](#)
  - [7.1.3 映射与过滤器](#)
- [7.2 语言生态学](#)
  - [7.2.1 解释型与编译型语言](#)
  - [7.2.2 动态与静态类型](#)
  - [7.2.3 命令式与函数式语言](#)
  - [7.2.4 重新实现的语言与原生语言](#)
- [7.3 JVM上的多语言编程](#)

- [7.3.1 为什么要用非Java语言](#)
- [7.3.2 崭露头角的语言新星](#)
- [7.4 如何挑选称心的非Java语言](#)
  - [7.4.1 低风险](#)
  - [7.4.2 与Java的交互操作](#)
  - [7.4.3 良好的工具和测试支持](#)
  - [7.4.4 备选语言学习难度](#)
  - [7.4.5 使用备选语言的开发者](#)
- [7.5 JVM对备选语言的支持](#)
  - [7.5.1 非Java语言的运行时环境](#)
  - [7.5.2 编译器小说](#)
- [7.6 小结](#)

第8章 Groovy: Java的动态伴侣

- [8.1 Groovy入门](#)
  - [8.1.1 编译和运行](#)
  - [8.1.2 Groovy控制台](#)
- [8.2 Groovy 101: 语法和语义](#)
  - [8.2.1 默认导入](#)
  - [8.2.2 数字处理](#)
  - [8.2.3 变量、动态与静态类型、作用域](#)
  - [8.2.4 列表和映射语法](#)
- [8.3 与Java的差异——新手陷阱](#)
  - [8.3.1 可选的分号和返回语句](#)
  - [8.3.2 可选的参数括号](#)
  - [8.3.3 访问限定符](#)
  - [8.3.4 异常处理](#)
  - [8.3.5 Groovy中的相等](#)
  - [8.3.6 内部类](#)
- [8.4 Java不具备的Groovy特性](#)
  - [8.4.1 GroovyBean](#)
  - [8.4.2 安全解引用操作符](#)
  - [8.4.3 猫王操作符](#)
  - [8.4.4 增强型字符串](#)
  - [8.4.5 函数字面值](#)
  - [8.4.6 内置的集合操作](#)
  - [8.4.7 对正则表达式的内置支持](#)
  - [8.4.8 简单的XML处理](#)
- [8.5 Groovy与Java的合作](#)
  - [8.5.1 从Groovy调用Java](#)
  - [8.5.2 从Java调用Groovy](#)
- [8.6 小结](#)

第9章 Scala: 简约而不简单

- [9.1 走马观花Scala](#)
  - [9.1.1 简约的Scala](#)
  - [9.1.2 match表达式](#)
  - [9.1.3 case类](#)
  - [9.1.4 actor](#)

## [9.2 Scala能用在我的项目中吗](#)

### [9.2.1 Scala和Java的比较](#)

### [9.2.2 何时以及如何开始使用Scala](#)

### [9.2.3 Scala可能不适合当前项目的迹象](#)

## [9.3 让代码因Scala重新绽放](#)

### [9.3.1 使用编译器和REPL](#)

### [9.3.2 类型推断](#)

### [9.3.3 方法](#)

### [9.3.4 导入](#)

### [9.3.5 循环和控制结构](#)

### [9.3.6 Scala的函数式编程](#)

## [9.4 Scala对象模型：相似但不同](#)

### [9.4.1 一切皆对象](#)

### [9.4.2 构造方法](#)

### [9.4.3 特质](#)

### [9.4.4 单例和伴生对象](#)

### [9.4.5 case类和match表达式](#)

### [9.4.6 警世寓言](#)

## [9.5 数据结构和集合](#)

### [9.5.1 List](#)

### [9.5.2 Map](#)

### [9.5.3 泛型](#)

## [9.6 actor介绍](#)

### [9.6.1 代码大舞台](#)

### [9.6.2 用mailbox跟actor通信](#)

## [9.7 小结](#)

## [第10章 Clojure：更安全地编程](#)

### [10.1 Clojure介绍](#)

#### [10.1.1 Clojure的Hello World](#)

#### [10.1.2 REPL入门](#)

#### [10.1.3 犯了错误](#)

#### [10.1.4 学着去爱括号](#)

### [10.2 寻找Clojure：语法和语义](#)

#### [10.2.1 特殊形式新手营](#)

#### [10.2.2 列表、向量、映射和集](#)

#### [10.2.3 数学运算、相等和其他操作](#)

### [10.3 使用函数和循环](#)

#### [10.3.1 一些简单的Clojure函数](#)

#### [10.3.2 Clojure中的循环](#)

#### [10.3.3 读取器宏和派发器](#)

#### [10.3.4 函数式编程和闭包](#)

### [10.4 Clojure序列](#)

#### [10.4.1 懒序列](#)

#### [10.4.2 序列和变参函数](#)

### [10.5 Clojure与Java的互操作](#)

#### [10.5.1 从Clojure中调用Java](#)

#### [10.5.2 Clojure值的Java类型](#)

- [10.5.3 使用Clojure代理](#)
- [10.5.4 用REPL做探索式编程](#)
- [10.5.5 在Java中使用Clojure](#)
- [10.6 Clojure并发](#)
  - [10.6.1 未来式与并行调用](#)
  - [10.6.2 ref形式](#)
  - [10.6.3 代理](#)
- [10.7 小结](#)
- [第四部分 多语种项目开发](#)
- [第11章 测试驱动开发](#)
  - [11.1 TDD概览](#)
    - [11.1.1 一个测试用例](#)
    - [11.1.2 多个测试用例](#)
    - [11.1.3 深入思考红—绿—重构循环](#)
    - [11.1.4 JUnit](#)
  - [11.2 测试替身](#)
    - [11.2.1 虚设对象](#)
    - [11.2.2 存根对象](#)
    - [11.2.3 伪装替身](#)
    - [11.2.4 模拟对象](#)
  - [11.3 ScakTest](#)
  - [11.4 小结](#)
- [第12章 构建和持续集成](#)
  - [12.1 与Maven 3相遇](#)
  - [12.2 Maven 3入门项目](#)
  - [12.3 用Maven 3构建Java7developer项目](#)
    - [12.3.1 POM](#)
    - [12.3.2 运行示例](#)
  - [12.4 Jenkins: 满足CI需求](#)
    - [12.4.1 基础配置](#)
    - [12.4.2 设置任务](#)
    - [12.4.3 执行任务](#)
  - [12.5 Maven和Jenkins代码指标](#)
    - [12.5.1 安装Jenkins插件](#)
    - [12.5.2 用Checkstyle保持代码一致性](#)
    - [12.5.3 用FindBugs设定质量标杆](#)
  - [12.6 Leiningen](#)
    - [12.6.1 Leiningen入门](#)
    - [12.6.2 Leiningen的架构](#)
    - [12.6.3 Hello Lein](#)
    - [12.6.4 用Leiningen做面向REPL的TDD](#)
    - [12.6.5 用Leiningen打包和部署](#)
  - [12.7 小结](#)
- [第13章 快速Web开发](#)
  - [13.1 Java Web框架的问题](#)
    - [13.1.1 Java编译为什么不好](#)
    - [13.1.2 静态类型为什么不好](#)



- [13.2 选择Web框架的标准](#)
- [13.3 Grails入门](#)
- [13.4 Grails快速启动项目](#)
  - [13.4.1 创建域对象](#)
  - [13.4.2 测试驱动开发](#)
  - [13.4.3 域对象持久化](#)
  - [13.4.4 创建测试数据](#)
  - [13.4.5 控制器](#)
  - [13.4.6 GSP/JSP页面](#)
  - [13.4.7 脚手架和UI的自动化创建](#)
  - [13.4.8 快速周转的开发](#)
- [13.5 深入Grails](#)
  - [13.5.1 日志](#)
  - [13.5.2 GORM: 对象关系映射](#)
  - [13.5.3 Grails插件](#)
- [13.6 Compojure入门](#)
  - [13.6.1 Hello Compojure](#)
  - [13.6.2 Ring和路由](#)
  - [13.6.3 Hiccup](#)
- [13.7 我是不是一只水獭](#)
  - [13.7.1 项目设置](#)
  - [13.7.2 核心函数](#)
- [13.8 小结](#)
- [第14章 保持优秀](#)
  - [14.1 对Java 8的期待](#)
    - [14.1.1 lambda表达式\(闭包\)](#)
    - [14.1.2 模块化\(拼图Jigsaw\)](#)
  - [14.2 多语言编程](#)
    - [14.2.1 语言的互操作性及元对象协议](#)
    - [14.2.2 多语言模块化](#)
  - [14.3 未来的并发趋势](#)
    - [14.3.1 多核的世界](#)
    - [14.3.2 运行时管理的并发](#)
  - [14.4 JVM的新方向](#)
    - [14.4.1 VM的合并](#)
    - [14.4.2 协同程序](#)
    - [14.4.3 元组](#)
  - [14.5 小结](#)
- [附录A java7developer: 源码安装](#)
  - [A.1 java7developer的源码结构](#)
  - [A.2 下载并安装Maven](#)
  - [A.3 构建java7developer](#)
    - [A.3.1 一次性的构建准备工作](#)
    - [A.3.2 clean](#)
    - [A.3.3 compile](#)
    - [A.3.4 test](#)
  - [A.4 小结](#)

[附录B glob模式语法及示例](#)

[B.1 glob模式语法](#)

[B.2 glob模式示例](#)

[附录C 安装备选JVM语言](#)

[C.1 Groovy](#)

[C.1.1 下载Groovy](#)

[C.1.2 安装Groovy](#)

[C.2 Scala](#)

[C.3 Clojure](#)

[C.4 Grails](#)

[C.4.1 下载Grails](#)

[C.4.2 安装Grails](#)

[附录D Jenkins的下载和安装](#)

[D.1 下载Jenkins](#)

[D.2 安装Jenkins](#)

[D.2.1 运行WAR文件](#)

[D.2.2 安装WAR文件](#)

[D.2.3 安装独立安装包](#)

[D.2.4 Jenkins的首次运行](#)

[附录E java7developer. Maven POM](#)

[E.1 构建配置](#)

[E.2 依赖项管理](#)

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

文档名称：《Java程序员修炼之道》 [英]Benjamin J. Evans [荷兰]Martijn Verburg. epub

请登录 <https://shgis.cn/post/1840.html> 下载完整文档。

手机端请扫码查看：

