

<<松本行弘的程序世界>>

图书基本信息

书名：<<松本行弘的程序世界>>

13位ISBN编号：9787115255075

10位ISBN编号：7115255075

出版时间：2011-8

出版时间：人民邮电出版社

作者：松本行弘

页数：389

译者：柳德燕,李黎明,夏倩,张文旭

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

<<松本行弘的程序世界>>

内容概要

本书是探索程序设计思想和方法的经典之作。作者从全局的角度，利用大量的程序示例及图表，深刻阐述了Ruby编程语言的设计理念，并以独特的视角对与编程相关的各种技术进行了考察。阅读本书不仅可以深入了解编程世界各个要素之间的关系，而且能够学到大师级的程序思考方法。

本书面向各层次程序设计人员和编程爱好者，也可以供相关技术人员参考。

<<松本行弘的程序世界>>

作者简介

松本行弘

Ruby语言的发明人，在1993年发布了Ruby语言的第一个版本，以后一直从事Ruby的设计与开发。2011年加入著名SaaS厂商Salesforce旗下PaaS公司Heroku，任首席Ruby架构师，致力于加快Ruby Core的开发。

他还是NaCl及乐天技术研究所的研究员。

著有Ruby in a Nutshell和The Ruby Programming Language

等书。

他的博客地址为<http://www.rubyist.net/~matz/>。

<<松本行弘的程序世界>>

书籍目录

第1章 我为什么开发Ruby

- 1.1 我为什么开发Ruby
 - 1.1.1 编程语言的重要性
 - 1.1.2 Ruby的原则
 - 1.1.3 简洁性
 - 1.1.4 扩展性
 - 1.1.5 稳定性
 - 1.1.6 一切皆因兴趣

第2章 面向对象

- 2.1 编程和面向对象的关系
 - 2.1.1 颠倒的构造
 - 2.1.2 主宰计算机的武器
 - 2.1.3 怎样写程序
 - 2.1.4 面向对象的编程方法
 - 2.1.5 面向对象的难点
 - 2.1.6 多态性
 - 2.1.7 具体的程序
 - 2.1.8 多态性的优点
- 2.2 数据抽象和继承
 - 2.2.1 面向对象的历史
 - 2.2.2 复杂性是面向对象的敌人
 - 2.2.3 结构化编程
 - 2.2.4 数据抽象化
 - 2.2.5 雏形
 - 2.2.6 找出相似的部分来继承
- 2.3 多重继承的缺点
 - 2.3.1 为什么需要多重继承
 - 2.3.2 多重继承和单一继承不可分离
 - 2.3.3 goto语句和多重继承比较相似
 - 2.3.4 解决多重继承的问题
 - 2.3.5 静态语言和动态语言的区别
 - 2.3.6 静态语言的特点
 - 2.3.7 动态语言的特点
 - 2.3.8 静态语言和动态语言的比较
 - 2.3.9 继承的两种含义
 - 2.3.10 接口的缺点
 - 2.3.11 继承实现的方法
 - 2.3.12 从多重继承变形而来的Mix-in
 - 2.3.13 积极支持Mix-in的Ruby
- 2.4 两个误解
 - 2.4.1 面向对象的编程
 - 2.4.2 对象的模板=类
 - 2.4.3 利用模块的手段 = 继承
 - 2.4.4 多重继承不好吗
 - 2.4.5 动态编程语言也需要多重继承

<<松本行弘的程序世界>>

2.4.6 驯服多重继承的方法

2.4.7 Ruby中多重继承的实现方法

2.4.8 Java实现多重继承的方法

2.5 Duck Typing诞生之前

2.5.1 为什么需要类型

2.5.2 动态的类型是从Lisp中诞生的

2.5.3 动态类型在面向对象中发展起来了

2.5.4 动态类型和静态类型的邂逅

2.5.5 静态类型的优点

2.5.6 动态类型的优点

2.5.7 只关心行为的Duck Typing

2.5.8 避免明确的类型检查

2.5.9 克服动态类型的缺点

2.5.10 动态编程语言

2.6 元编程

2.6.1 元编程

2.6.2 反射

2.6.3 元编程的例子

2.6.4 使用反射功能

2.6.5 分布式Ruby的实现

2.6.6 数据库的应用

2.6.7 输出XML

2.6.8 元编程和小编程语言

2.6.9 声明的实现

2.6.10 上下文相关的实现

2.6.11 单位的实现

2.6.12 词汇的实现

2.6.13 层次数据的实现

2.6.14 适合DSL的语言，不适合DSL的语言

第3章 程序块

3.1 程序块的威力

3.1.1 把函数作为参数的高阶函数

3.1.2 C语言高阶函数的局限

3.1.3 可以保存外部环境的闭包

3.1.4 块的两种使用方法

3.1.5 最终来看，块到底是什么

3.1.6 块在循环处理中的应用

3.1.7 内部迭代器和外部迭代器

3.1.8 在排序和比较大小中的应用

3.1.9 用块保证程序的后处理

3.1.10 用块实现新的控制结构

3.1.11 在回调中使用块

3.1.12 块处理的特别理由

3.2 用块作循环

3.2.1 块是处理的集合

3.2.2 块应用范围的扩展

3.2.3 高阶函数和块的本质一样

<<松本行弘的程序世界>>

3.2.4 用Enumerable来利用块

3.2.5 Enumerable的局限

3.3 精通集合的使用

3.3.1 使用Ruby的数组

3.3.2 修改指定范围的元素内容

3.3.3 Ruby中的哈希处理

3.3.4 支持循环的Enumerable

3.3.5 用于循环的each方法

3.3.6 使用inject、zip和grep

3.3.7 用来指定条件的select方法

3.3.8 排序与比较大小的方法

3.3.9 在类中包含 (include) Enumerable模块

3.3.10 List的内部包和块的区别

第4章 设计模式

4.1 设计模式 (1)

4.1.1 设计模式的价值和意义

4.1.2 设计模式是程序抽象化的延伸

4.1.3 Ruby中的设计模式

4.1.4 Singleton模式

4.1.5 Proxy模式

4.1.6 Iterator模式

4.1.7 外部与内部, 哪一个更好

4.1.8 内部迭代器的缺陷

4.1.9 外部迭代器的缺陷

4.2 设计模式 (2)

4.2.1 模式与动态语言的关系

4.2.2 重复使用既存对象的Prototype模式

4.2.3 亲身体验Io语言

4.2.4 Ruby中的原型

4.2.5 编写抽象算法的Template Method模式

4.2.6 用Ruby来尝试TemplateMethod

4.2.7 动态语言与Template Method模式

4.2.8 避免高度依赖性的Observer模式

4.2.9 Observable模块

4.2.10 Observer模式与动态语言

4.3 设计模式 (3)

4.3.1 软件开发的悲剧

4.3.2 开放— 封闭原则

4.3.3 面向对象的情况

4.3.4 非面向对象的情况

4.3.5 OCP 与Template Method模式

4.3.6 Observer模式

4.3.7 使用Strategy模式

4.3.8 Strategy模式与OCP

第5章 Ajax

5.1 Ajax和JavaScript (前篇)

5.1.1 通信及异步页面更新

<<松本行弘的程序世界>>

- 5.1.2 技术要素之一：JavaScript
- 5.1.3 技术要素之二：XML
- 5.1.4 XML以外的数据表现形式
- 5.1.5 技术要素之三：DHTML
- 5.1.6 JavaScript技术基础
- 5.1.7 原型模式的面向对象编程语言
- 5.1.8 使用prototype.js库
- 5.1.9 prototype.js的功能
- 5.2 Ajax和JavaScript（后篇）
 - 5.2.1 巧妙使用DHTML
 - 5.2.2 获取document节点
 - 5.2.3 获取和更新标签数据
 - 5.2.4 设定事件处理程序
 - 5.2.5 追加标签节点
 - 5.2.6 本地HTML应用
 - 5.2.7 和服务端间的通信
 - 5.2.8 使用Prototype.js的优点
 - 5.2.9 在服务器上保存数据
 - 5.2.10 Web 应用的脆弱性
 - 5.2.11 使用JavaScript的感觉

第6章 Ruby on Rails

- 6.1 MVC 和Ruby on Rails
 - 6.1.1 模型、视图和控制的作用
 - 6.1.2 用秒表的例子来学习MVC模式
 - 6.1.3 生成视图和控制部分
 - 6.1.4 GUI工具箱与MVC
 - 6.1.5 同时使用工具箱和MVC
 - 6.1.6 MVC的优缺点
 - 6.1.7 Web应用中的MVC
- 6.2 开放类和猴子补丁
 - 6.2.1 开放类
 - 6.2.2 猴子补丁的目的
 - 6.2.3 猴子补丁的技巧
 - 6.2.4 灵活使用开放类的库
 - 6.2.5 猴子补丁的几点问题
 - 6.2.6 其他办法
 - 6.2.7 Ruby on Rails和开放类
 - 6.2.8 ActiveSupport带来的扩展
 - 6.2.9 字节单位系列
 - 6.2.10 复数形和序数
 - 6.2.11 大规模开发和Ruby
 - 6.2.12 信赖性模型
 - 6.2.13 猴子补丁的未来

第7章 文字编码

- 7.1 文字编码的种类
 - 7.1.1 早期的文字编码
 - 7.1.2 纸带与文字表现

<<松本行弘的程序世界>>

- 7.1.3 文字是什么
- 7.1.4 走向英语以外的语言（欧洲篇）
- 7.1.5 英语以外的语言（亚洲篇）
- 7.1.6 Unicode的问世
- 7.1.7 统一编码成16位的汉字统合
- 7.1.8 Unicode的两个问题
- 7.1.9 Unicode的文字集
- 7.1.10 文字表示的不确定性
- 7.1.11 Unicode的字符编码方式
- 7.2 程序中的文字处理
 - 7.2.1 文字编码有多个意思
 - 7.2.2 只能处理文字集中包含的文字
 - 7.2.3 纷繁复杂的文字编码方式
 - 7.2.4 影响力渐微的Shift_JIS 与EUC-JP
 - 7.2.5 Unicode有多种字符编码方式
 - 7.2.6 为什么会发生乱码
 - 7.2.7 字符编码方式错误
 - 7.2.8 没有字体
 - 7.2.9 变换为内部码时出错
 - 7.2.10 发生不完全变换
 - 7.2.11 文字集的不同
 - 7.2.12 字节顺序错误
 - 7.2.13 从编程语言的角度处理文字
 - 7.2.14 以变换为前提的UCS方式
 - 7.2.15 原封不动处理的CSI方式
 - 7.2.16 使用UTF-16的Java
 - 7.2.17 使用UTF-8的Perl
 - 7.2.18 用UTF-16的Python
 - 7.2.19 采用CSI方式的Ruby 1.8
 - 7.2.20 强化了功能的Ruby1.9
 - 7.2.21 是UCS还是CSI
- 第8章 正则表达式
 - 8.1 正则表达式基础
 - 8.1.1 检索“像那样的东西”
 - 8.1.2 正则表达式的语法
 - 8.1.3 个陷阱
 - 8.1.4 正则表达式对象
 - 8.1.5 选项
 - 8.1.6 正则表达式匹配的方法
 - 8.1.7 特殊变量
 - 8.1.8 字符串与正则表达式
 - 8.1.9 split的本质
 - 8.1.10 字符串的扫描
 - 8.1.11 置换
 - 8.2 正则表达式的应用实例与“鬼车”
 - 8.2.1 解析日志文件的方法
 - 8.2.2 避免使用\$的方法

<<松本行弘的程序世界>>

8.2.3 从邮件中取出日期的方法

8.2.4 典型拼写错误的检索方法

8.2.5 Ruby.9的新功能“鬼车”

第9章 整数和浮点小数

9.1 深奥的整数世界

9.1.1 整数是有范围的

9.1.2 尝试位运算

9.1.3 操作特定的位

9.1.4 表示负数的办法

9.1.5 Ruby的整数

9.1.6 挑战公开密钥方式

9.2 扑朔迷离的浮点小数世界

9.2.1 计算机对小数的处理

9.2.2 固定小数点数不易使用

9.2.3 科学计数法也有问题

9.2.4 小数不能完全表示

9.2.5 有不能比较的时候

9.2.6 误差积累

9.2.7 不是数的特别“数”

9.2.8 计算误差有多种

9.2.9 误差导致的严重问题

9.2.10 BigDecimal是什么

9.2.11 能够表示分数的Rational类

第10章 高速执行和并行处理

10.1 让程序高速执行（前篇）

10.1.1 是不是越快越好

10.1.2 高速执行的乐趣与效率

10.1.3 以数据为基础作出判断

10.1.4 改善系统调用

10.1.5 数据可靠吗

10.1.6 只需改善瓶颈

10.1.7 profiler本身成了累赘

10.1.8 算法与数据结构

10.1.9 理解O记法

10.1.10 选择算法

10.1.11 调查算法的性能

10.1.12 高速执行的悲哀

10.1.13 性能优化的格言

10.2 让程序高速执行（后篇）

10.2.1 确认程序概要

10.2.2 发现瓶颈

10.2.3 使用更好的profiler

10.2.4 高速优化之一：削减对象

10.2.5 高速优化之二：利用立即值

10.2.6 高速优化之三：利用C语言

10.2.7 高速优化之四：采用合适的数据结构

10.2.8 全部以C语言计算

<<松本行弘的程序世界>>

10.2.9 还存在其他技巧

10.3 并行编程

10.3.1 使用线程的理由

10.3.2 生成线程

10.3.3 线程的执行状态

10.3.4 传递值给线程的方法

10.3.5 信息共有所产生的问题

10.3.6 数据完整性的丧失

10.3.7 死锁

10.3.8 用锁来实现对资源的独占

10.3.9 二级互斥

10.3.10 用队列协调线程

10.3.11 锁模型与队列模型比较

10.4 前景可期的并行编程技术，Actor

10.4.1 何谓Actor

10.4.2 操作Actor的3种处理系统

10.4.3 Erlang的程序

10.4.4 Pingpong处理的开始

10.4.5 启动pingpong程序

10.4.6 Erlang的错误处理

10.4.7 Erlang的使用场所

10.4.8 面向Ruby的库“Revactor”

10.4.9 Revactor的应用场合

10.4.10 另一个库Dramatis

第11章 程序安全性

11.1 程序的漏洞与攻击方法

11.1.1 种软件漏洞

11.1.2 因权限被窃取而成为重大问题

11.1.3 安全问题的根源

11.1.4 “守护神”引起的问题

11.1.5 多样化的攻击手段

11.1.6 缓冲区溢出

11.1.7 整数溢出

11.1.8 SQL注入

11.1.9 Shell注入

11.1.10 跨站点脚本攻击

11.1.11 跨站点伪造请求

11.1.12 社会工程

11.2 用异常进行错误处理

11.2.1 异常的历史

11.2.2 Java的受控异常

11.2.3 Icon的面向目标判断

11.2.4 Ruby的异常

11.2.5 异常发生

11.2.6 异常类

11.2.7 异常处理的设计方针

11.2.8 异常发生的设计原则

<<松本行弘的程序世界>>

第12章 关于时间的处理

12.1 用程序处理时刻与时间

12.1.1 时差与时区

12.1.2 世界协调时间

12.1.3 夏令时 (DST)

12.1.4 改历

12.1.5 日期与时间的类

12.1.6 年问题

12.1.7 DateTime类

12.1.8 Time与DateTime的相互变换

第13章 关于数据的持久化

13.1 持久化数据的方法

13.1.1 保存文本

13.1.2 变换成文本的Marshal

13.1.3 使用Marshal模块

13.1.4 复制有两种方式

13.1.5 仔细看Marshal的格式

13.1.6 不能保存的3类对象

13.1.7 制作面向对象数据库

13.1.8 试用PStore

13.1.9 变换为文本的YAML

13.1.10 用YAML制作数据库

13.2 对象的保存

13.2.1 高速的Object Prevalence

13.2.2 Object Prevalence的问题点

13.2.3 使用Madeleine

13.2.4 访问时刻信息

13.2.5 让Madeleine更容易使用

13.2.6 Madeleine的实用例Instiki

13.3 关于XML的考察

13.3.1 XML的祖先是SGML

13.3.2 XML是树结构的数据表现

13.3.3 优点在于纯文本

13.3.4 缺点在于冗长

13.3.5 不适合重视效率的处理

13.3.6 适合于信息交换的格式

13.3.7 XML的解析

13.3.8 XML处理库REXML

13.3.9 XML的代替

第14章 函数式编程

14.1 新范型——函数式编程

14.1.1 具有多种函数式性质的Lisp

14.1.2 彻底的函数式编程语言Haskell

14.1.3 延迟计算：不必要的处理就不做

14.1.4 灵活的“静态多态性”类型系统

14.1.5 近代函数式语言之父OCaml

14.1.6 强于并行计算的Erlang

<<松本行弘的程序世界>>

- 14.1.7 用Ruby进行函数式编程
- 14.1.8 用枚举器来实现延迟计算
- 14.2 自动生成代码
 - 14.2.1 在商业中利用Ruby
 - 14.2.2 使用Ruby自动生成代码
 - 14.2.3 消除重复代码
 - 14.2.4 代码生成的应用
 - 14.2.5 代码生成的效果
 - 14.2.6 编写代码生成器
 - 14.2.7 也可以使用XML
 - 14.2.8 在EJB中使用代码生成
- 14.3 内存管理与垃圾收集
 - 14.3.1 内存管理的困难
 - 14.3.2 垃圾收集亮相之前
 - 14.3.3 评价垃圾收集的两个指标
 - 14.3.4 垃圾收集算法
 - 14.3.5 引用计数方式
 - 14.3.6 标记和扫描方式
 - 14.3.7 标记和紧缩方式
 - 14.3.8 复制方式
 - 14.3.9 多种多样的垃圾收集算法
 - 14.3.10 分代垃圾收集
 - 14.3.11 保守垃圾收集
 - 14.3.12 增量垃圾收集
 - 14.3.13 并行垃圾收集
 - 14.3.14 位图标记
- 14.4 用C语言来扩展Ruby
 - 14.4.1 开发与执行速度的取舍
 - 14.4.2 扩展库
 - 14.4.3 看例题学习扩展模块
 - 14.4.4 QDBM函数
 - 14.4.5 初始化对象
 - 14.4.6 实现方法
 - 14.4.7 关于垃圾收集的注意事项
 - 14.4.8 其他的RubyAPI
 - 14.4.9 扩展库的编译
 - 14.4.10 扩展库以外的工具
- 14.5 为什么要开源
 - 14.5.1 自由软件的思想
 - 14.5.2 自由软件的历史
 - 14.5.3 Emacs事件的发生
 - 14.5.4 开源的诞生
 - 14.5.5 OSS许可证
 - 14.5.6 开源的背景
 - 14.5.7 企业关注开源的理由
 - 14.5.8 Ruby与开源
 - 14.5.9 选择许可证的方法

<<松本行弘的程序世界>>

章节摘录

版权页：插图：最后，变量和算式分别有自己的类型，这使得我们能够在一开始就认真考虑这些变量应该扮演什么样的角色。

我们在编写程序时就要考虑数据类型，虽然要考虑的东西变多了，但是也不能简单地说这是坏事。显而易见，这是我们开发好的、可靠性高的程序所必需的。

从以上几点来看，静态类型似乎全都是优点。

其实它也有几个缺点，或者说是问题。

其中一个问题是，若不指定类型就写不了程序。

当然，指定类型是静态类型编程语言的特征之一。

但是说到底，数据类型只是一些辅助信息，并不是程序本质。

当我们想把精力集中到程序处理的实际问题时，却要一个个考虑数据类型的定义，这是很烦琐的。

并且，有时会让人觉得，有的类型声明仅仅是为了满足编译器的要求。

程序规模也因为数据类型的定义而变大，重要的部分反而容易被忽视。

另外一个问题是灵活性的问题。

静态类型本身限制了给某个变量只能赋值某种类型的对象，这种限制可能成为妨碍将来变化的枷锁。

前面学过的多重继承和接口会产生令人费解的继承关系，这时怎样设定适当的类型就变得比较困难了。

总结一下，用静态类型编程语言的人通过定义类型，把更多的信息传达了出来，这算是给编译器和将来读程序的人减轻负担的一种方法吧。

2.5.6 动态类型的优点前面介绍了静态类型，那么动态类型又怎样呢？

动态类型编程语言的最大优点是源代码变得很简洁。

编程语言的进化使我们可以用更简单的程序来传达给计算机更多信息。

如果不用指定与程序本质无关的数据类型，程序也完全可以正确执行，也可以检测出来错误的话，这不是一种很好的想法吗？

得益于简洁，我们在编写程序的时候就不用考虑数据类型这些无关本质的部分了，而是可以集中于程序处理的本质部分，编写简洁程序的话，也可以提高生产力。

另一方面，有人会担心，简洁的程序虽然让我们在编程的时候变得简单了，但是因为缺少类型信息，以后读起来是不是就变得难以理解呢？

虽然写的时候容易了，但难读的程序也是不可取的。

对于这样的担心，我的回答是，简洁的程序更突出了程序处理的实质，理解起来反而变得简单了。

实际上，动态类型的编程语言（例如Ruby）的程序规模和静态类型相比，程序行数相差数倍的情况并不少见。

很多人都感觉到动态类型的程序更好理解。

对于简洁程序的另外一个担心是，动态类型语言是否运行缓慢呢？

事实上是这样的。

同样的处理，在大多数情况下，静态类型编程语言运行得要快些。

<<松本行弘的程序世界>>

编辑推荐

《松本行弘的程序世界》为“Ruby之父”经典力作，展现了大师级的程序思考方式。作者凭借对编程本质的深刻认识和对各种技术优缺点的掌握，阐述了Ruby的设计理念，并由此延伸，带领读者了解编程的本质，一窥程序设计的奥秘。

《松本行弘的程序世界》不是为了介绍某种特定的技术，而是从宏观的角度讨论与编程相关的各种技术。

书中第1章介绍了作者对编程问题的新思考和新看法，剩下的内容出自《日经Linux》杂志于2005年5月到2009年4月连载的“松本编程模式讲坛”，其中真正涉及“模式”的内容并不多，大量篇幅都用于介绍技术内幕和背景分析等内容，使读者真正了解相关技术的立足点。

另外，书中还包含许多以Ruby、Lisp、Smalltalk、Erlang、JavaScript等动态语言所写成的范例。Ruby之父佳作，进入不同凡响的程序世界，深入剖析程序设计的道与术，举一反三，触类旁通。

<<松本行弘的程序世界>>

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>