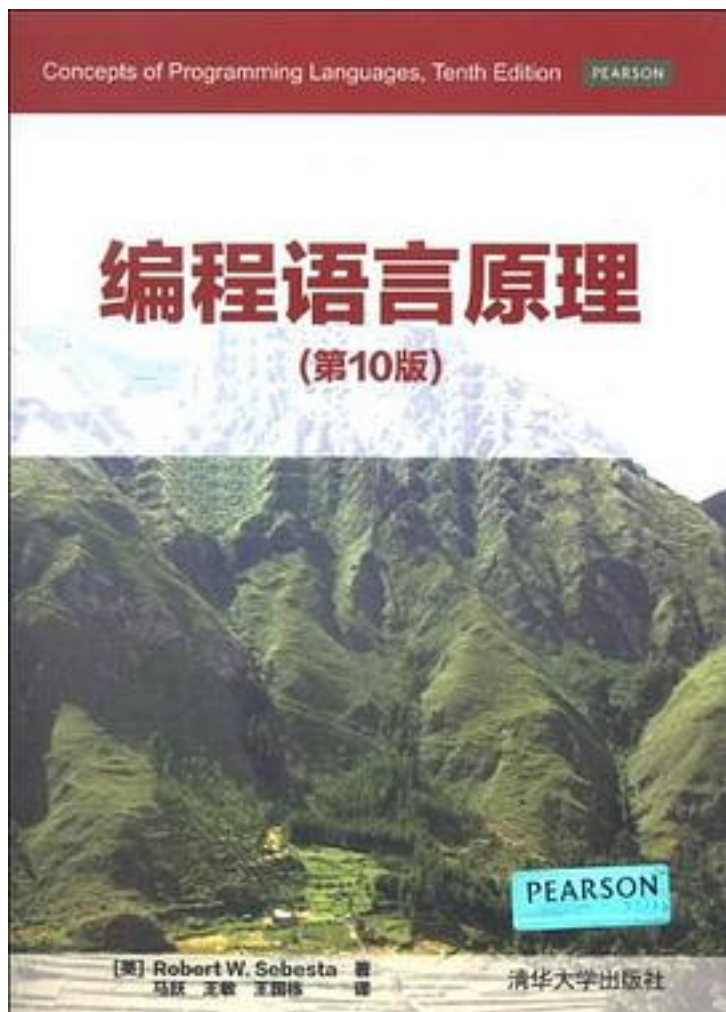


编程语言原理



[编程语言原理_下载链接1](#)

著者:Robert W. Sebesta

出版者:清华大学出版社

出版时间:2013-3

装帧:

isbn:9787302311126

塞巴斯塔编写的这本《编程语言原理(第10版)》从为什么学习程序设计语言入手，深入

细致地讲解了命令式语言的主要结构及其设计与实现，内容涉及变量、数据类型、表达式和赋值语句、控制语句、子程序、数据抽象机制、对面向对象程序设计的支持(继承和动态方法绑定)、并发、异常处理和事件处理等方面。最后两章介绍了函数式程序设计语言和逻辑程序设计语言。

《编程语言原理(第10版)》内容丰富，讲解透彻，既可用做高等院校计算机及相关专业本科生程序设计语言课程的教材和参考书，也可供程序设计人员参考。

作者介绍:

Robert

W.Sebesta，宾夕法尼亚州立大学获得计算机科学博士，拥有30多年的教授计算机科学课程的经验。目前担任科罗拉多大学科罗拉多斯普林斯分校计算机科学系的副教授、ACM和IEEE计算机学会的会员，主要研究方向是设计和评估程序设计语言、编译器设计以及软件测试方法和工具。

目录: 目录

第1章 预备知识

1.1 学习程序设计语言原理的原因

1.2 程序设计领域

1.2.1 科学应用

1.2.2 商务应用

1.2.3 人工智能

1.2.4 系统程序设计

1.2.5 网络软件

1.3 语言评价标准

1.3.1 可读性

1.3.2 可写性

1.3.3 可靠性

1.3.4 成本

1.4 影响语言设计的因素

1.4.1 计算机体系结构

1.4.2 程序设计方法学

1.5 程序设计语言的分类

1.6 语言设计中的权衡

1.7 实现方法

1.7.1 编译

1.7.2 完全解释

1.7.3 混合实现系统

1.7.4 预处理器

1.8 编程环境

第2章 主要程序设计语言的发展

2.1 Zuse的Plankalkül语言

2.1.1 历史背景

2.1.2 语言概述

2.2 伪代码

2.2.1 Short Code语言

2.2.2 Speedcoding系统

2.2.3 UNIVAC “编译”系统

2.2.4 相关工作

2.3 IBM 704计算机与Fortran语言

2.3.1 历史背景

- 2.3.2 设计过程
- 2.3.3 Fortran I概述
- 2.3.4 Fortran II
- 2.3.5 Fortran IV、77、90、95、2003和2008
- 2.3.6 评价
- 2.4 函数式程序设计：LISP语言
 - 2.4.1 人工智能的起源和表处理
 - 2.4.2 LISP语言的设计过程
 - 2.4.3 语言概述
 - 2.4.4 评价
 - 2.4.5 LISP的两种后代语言
 - 2.4.6 相关语言
- 2.5 迈向成熟的第一步：ALGOL 60
 - 2.5.1 历史背景
 - 2.5.2 早期设计过程
 - 2.5.3 ALGOL 58概述
 - 2.5.4 对ALGOL 58报告的响应
 - 2.5.5 ALGOL 60的设计过程
 - 2.5.6 ALGOL 60概述
 - 2.5.7 评价
- 2.6 商务记录的计算机化：COBOL语言
 - 2.6.1 历史背景
 - 2.6.2 FLOW-MATIC语言
 - 2.6.3 COBOL语言的设计过程
 - 2.6.4 评价
- 2.7 分时处理的开始：BASIC语言
 - 2.7.1 设计过程
 - 2.7.2 语言概述
 - 2.7.3 评价
- 2.8 满足所有人的需要：PL/I
 - 2.8.1 历史背景
 - 2.8.2 设计过程
 - 2.8.3 语言概述
 - 2.8.4 评价
- 2.9 两种早期的动态语言：APL和SNOBOL
 - 2.9.1 APL语言的起源与特点
 - 2.9.2 SNOBOL语言的起源与特点
- 2.10 数据抽象的开始：SIMULA 67
 - 2.10.1 设计过程
 - 2.10.2 语言概述
- 2.11 正交设计：ALGOL 68
 - 2.11.1 设计过程
 - 2.11.2 语言概述
 - 2.11.3 评价
- 2.12 ALGOL系列语言的早期后代语言
 - 2.12.1 为简单性而设计：Pascal语言
 - 2.12.2 可移植的系统语言：C语言
- 2.13 基于逻辑的程序设计：Prolog语言
 - 2.13.1 设计过程
 - 2.13.2 语言概述
 - 2.13.3 评价
- 2.14 历史上规模最大的设计工作：Ada语言
 - 2.14.1 历史背景
 - 2.14.2 设计过程

- 2.14.3 语言概述
- 2.14.4 评价
- 2.14.5 Ada 95和Ada 2005
- 2.15 面向对象的程序设计：Smalltalk
 - 2.15.1 设计过程
 - 2.15.2 语言概述
 - 2.15.3 评价
- 2.16 结合命令式和面向对象的特性：C++
 - 2.16.1 设计过程
 - 2.16.2 语言概述
 - 2.16.3 评价
 - 2.16.4 一种相关语言：Objective-C
 - 2.16.5 另一种相关语言：Delphi
 - 2.16.6 一种关系不大的语言：Go
- 2.17 基于命令式的面向对象语言：Java
 - 2.17.1 设计过程
 - 2.17.2 语言概述
 - 2.17.3 评价
- 2.18 脚本语言
 - 2.18.1 Perl的起源与特点
 - 2.18.2 JavaScript的起源与特点
 - 2.18.3 PHP的起源与特点
 - 2.18.4 Python的起源与特点
 - 2.18.5 Ruby的起源与特点
 - 2.18.6 Lua的起源与特点
- 2.19 一流的.NET语言：C#
 - 2.19.1 设计过程
 - 2.19.2 语言概述
 - 2.19.3 评价
- 2.20 标记与程序设计混合的语言
 - 2.20.1 XSLT
 - 2.20.2 JSP
- 第3章 描述语法和语义
 - 3.1 概述
 - 3.2 描述语法的普遍问题
 - 3.2.1 语言识别器
 - 3.2.2 语言生成器
 - 3.3 描述语法的形式化方法
 - 3.3.1 巴科斯-诺尔范式和上下文无关文法
 - 3.3.2 扩展的BNF
 - 3.3.3 文法与识别器
 - 3.4 属性文法
 - 3.4.1 静态语义
 - 3.4.2 基本概念
 - 3.4.3 属性文法定义
 - 3.4.4 本质属性
 - 3.4.5 属性文法的例子
 - 3.4.6 计算属性值
 - 3.4.7 评价
 - 3.5 描述程序的意义：动态语义
 - 3.5.1 操作语义
 - 3.5.2 指称语义
 - 3.5.3 公理语义
- 第4章 词法分析和语法分析

- 4.1 概述
- 4.2 词法分析
- 4.3 语法分析问题
 - 4.3.1 语法分析概述
 - 4.3.2 自顶向下的语法分析器
 - 4.3.3 自底向上的语法分析器
 - 4.3.4 语法分析的复杂度
- 4.4 递归下降的语法分析
 - 4.4.1 递归下降的语法分析过程
 - 4.4.2 LL文法类
- 4.5 自下而上的语法分析
 - 4.5.1 自下而上的语法分析器的分析问题
 - 4.5.2 移进-归约算法
 - 4.5.3 LR语法分析器
- 第5章 名字、绑定和作用域
 - 5.1 概述
 - 5.2 名字
 - 5.2.1 设计问题
 - 5.2.2 名字形式
 - 5.2.3 特殊字
 - 5.3 变量
 - 5.3.1 名字
 - 5.3.2 地址
 - 5.3.3 类型
 - 5.3.4 数值
 - 5.4 绑定的概念
 - 5.4.1 属性与变量绑定
 - 5.4.2 绑定类型
 - 5.4.3 存储绑定和生存期
 - 5.5 作用域
 - 5.5.1 静态作用域
 - 5.5.2 块
 - 5.5.3 声明的次序
 - 5.5.4 全局作用域
 - 5.5.5 静态作用域的评价
 - 5.5.6 动态作用域
 - 5.5.7 动态作用域的评价
 - 5.6 作用域和生存期
 - 5.7 引用环境
 - 5.8 命名常量
- 第6章 数据类型
 - 6.1 概述
 - 6.2 基本数据类型
 - 6.2.1 数值类型
 - 6.2.2 布尔类型
 - 6.2.3 字符类型
 - 6.3 字符串类型
 - 6.3.1 设计问题
 - 6.3.2 字符串及其操作
 - 6.3.3 字符串长度的设计选项
 - 6.3.4 评价
 - 6.3.5 字符串类型的实现
 - 6.4 用户定义的序数类型
 - 6.4.1 枚举类型

- 6.4.2 子界类型
- 6.4.3 实现用户定义的有序类型
- 6.5 数组类型
 - 6.5.1 设计问题
 - 6.5.2 数组和索引
 - 6.5.3 下标的绑定和数组的种类
 - 6.5.4 数组的初始化
 - 6.5.5 数组操作
 - 6.5.6 矩形数组和不规则数组
 - 6.5.7 切片
 - 6.5.8 评价
 - 6.5.9 数组类型的实现
- 6.6 关联数组
 - 6.6.1 结构和操作
 - 6.6.2 关联数组的实现
- 6.7 记录类型
 - 6.7.1 记录的定义
 - 6.7.2 记录域的引用
 - 6.7.3 评价
 - 6.7.4 记录类型的实现
- 6.8 元组类型
- 6.9 列表类型
- 6.10 联合类型
 - 6.10.1 设计问题
 - 6.10.2 判别式联合与自由联合
 - 6.10.3 Ada的联合类型
 - 6.10.4 F#的联合类型
 - 6.10.5 评价
 - 6.10.6 联合类型的实现
- 6.11 指针和引用类型
 - 6.11.1 设计问题
 - 6.11.2 指针操作
 - 6.11.3 指针的相关问题
 - 6.11.4 Ada中的指针
 - 6.11.5 C和C++中的指针
 - 6.11.6 引用类型
 - 6.11.7 评价
 - 6.11.8 指针和引用类型的实现
- 6.12 类型检查
- 6.13 强类型化
- 6.14 类型等价
- 6.15 理论和数据类型

第7章 表达式与赋值语句

- 7.1 概述
- 7.2 算术表达式
 - 7.2.1 运算符的运算顺序
 - 7.2.2 操作数的运算顺序
- 7.3 运算符重载
- 7.4 类型转换
 - 7.4.1 表达式中的强制类型转换
 - 7.4.2 显式类型转换
 - 7.4.3 表达式中的错误
- 7.5 关系表达式和布尔表达式
 - 7.5.1 关系表达式

- 7.5.2 布尔表达式
- 7.6 短路求值
- 7.7 赋值语句
 - 7.7.1 简单赋值
 - 7.7.2 条件赋值
 - 7.7.3 混合赋值运算符
 - 7.7.4 一元赋值运算符
 - 7.7.5 赋值表达式
 - 7.7.6 多重赋值
 - 7.7.7 函数式编程语言中的赋值
- 7.8 混合模式赋值
- 第8章 语句级控制结构
 - 8.1 概述
 - 8.2 选择语句
 - 8.2.1 双路选择语句
 - 8.2.2 多重选择结构
 - 8.3 迭代语句
 - 8.3.1 计数控制循环
 - 8.3.2 逻辑控制循环
 - 8.3.3 用户自定义的循环控制机制
 - 8.3.4 基于数据结构的迭代
 - 8.4 无条件分支
 - 8.5 防护命令
 - 8.6 结论
- 第9章 子程序
 - 9.1 概述
 - 9.2 子程序的基本原理
 - 9.2.1 子程序的一般性质
 - 9.2.2 子程序的基本定义
 - 9.2.3 参数
 - 9.2.4 过程与函数
 - 9.3 子程序的设计问题
 - 9.4 局部引用环境
 - 9.4.1 局部变量
 - 9.4.2 嵌套子程序
 - 9.5 参数传递方式
 - 9.5.1 参数传递的语义模型
 - 9.5.2 参数传递的实现模型
 - 9.5.3 参数传递方法的实现
 - 9.5.4 常见语言的参数传递方法
 - 9.5.5 参数的类型检查
 - 9.5.6 多维数组作为参数
 - 9.5.7 设计考虑
 - 9.5.8 参数传递的例子
 - 9.6 子程序作为参数
 - 9.7 间接调用子程序
 - 9.8 重载子程序
 - 9.9 泛型子程序
 - 9.9.1 C++中的泛型函数
 - 9.9.2 Java 5.0中的泛型方法
 - 9.9.3 C# 2005中的泛型方法
 - 9.9.4 F#中的泛型函数
 - 9.10 函数的设计问题
 - 9.10.1 函数的副作用

- 9.10.2 返回值的类型
- 9.10.3 返回值的个数
- 9.11 用户定义重载运算符
- 9.12 闭包
- 9.13 协同程序
- 第10章 实现子程序
 - 10.1 调用和返回的一般语义
 - 10.2 实现“简单”的子程序
 - 10.3 通过栈动态局部变量实现子程序
 - 10.3.1 更复杂的活动记录
 - 10.3.2 一个不含递归调用的例子
 - 10.3.3 递归调用
 - 10.4 嵌套子程序
 - 10.4.1 基础知识
 - 10.4.2 静态链
 - 10.5 块
 - 10.6 动态作用域的实现
 - 10.6.1 深层访问
 - 10.6.2 浅层访问
- 第11章 抽象数据类型与封装结构
 - 11.1 抽象的概念
 - 11.2 数据抽象简介
 - 11.2.1 抽象数据类型之浮点型
 - 11.2.2 用户自定义的抽象数据类型
 - 11.2.3 示例
 - 11.3 抽象数据类型的设计问题
 - 11.4 语言示例
 - 11.4.1 Ada中的抽象数据类型
 - 11.4.2 C++中的抽象数据类型
 - 11.4.3 Objective-C中的抽象数据类型
 - 11.4.4 Java中的抽象数据类型
 - 11.4.5 C#中的抽象数据类型
 - 11.4.6 Ruby中的抽象数据类型
 - 11.5 参数化的抽象数据类型
 - 11.5.1 Ada
 - 11.5.2 C++
 - 11.5.3 Java 5.0
 - 11.5.4 C# 2005
 - 11.6 封装结构
 - 11.6.1 引言
 - 11.6.2 C中的封装
 - 11.6.3 C++中的封装
 - 11.6.4 Ada包
 - 11.6.5 C#程序集
 - 11.7 命名封装
 - 11.7.1 C++命名空间
 - 11.7.2 Java包
 - 11.7.3 Ada包
 - 11.7.4 Ruby模块
- 第12章 面向对象程序设计的支持
 - 12.1 概述
 - 12.2 面向对象编程
 - 12.2.1 引言
 - 12.2.2 继承

- 12.2.3 动态绑定
- 12.3 面向对象语言的设计问题
 - 12.3.1 对象的排他性
 - 12.3.2 子类是子类型吗
 - 12.3.3 单继承与多继承
 - 12.3.4 对象的分配和释放
 - 12.3.5 动态绑定与静态绑定
 - 12.3.6 嵌套类
 - 12.3.7 对象的初始化
- 12.4 Smalltalk对面向对象编程的支持
 - 12.4.1 一般特征
 - 12.4.2 继承
 - 12.4.3 动态绑定
 - 12.4.4 Smalltalk的评价
- 12.5 C++对面向对象编程的支持
 - 12.5.1 一般特征
 - 12.5.2 继承
 - 12.5.3 动态绑定
 - 12.5.4 评价
- 12.6 Objective-C对面向对象编程的支持
 - 12.6.1 一般特征
 - 12.6.2 继承
 - 12.6.3 动态绑定
 - 12.6.4 评价
- 12.7 Java对面向对象编程的支持
 - 12.7.1 一般特征
 - 12.7.2 继承
 - 12.7.3 动态绑定
 - 12.7.4 被嵌套的类
 - 12.7.5 评价
- 12.8 C#对面向对象编程的支持
 - 12.8.1 一般特征
 - 12.8.2 继承
 - 12.8.3 动态绑定
 - 12.8.4 被嵌套的类
 - 12.8.5 评价
- 12.9 Ada 95对面向对象编程的支持
 - 12.9.1 一般特征
 - 12.9.2 继承
 - 12.9.3 动态绑定
 - 12.9.4 子包
 - 12.9.5 评价
- 12.10 Ruby对面向对象编程的支持
 - 12.10.1 一般特征
 - 12.10.2 继承
 - 12.10.3 动态绑定
 - 12.10.4 评价
- 12.11 面向对象构造的实现
 - 12.11.1 存储实例数据
 - 12.11.2 方法调用到方法的动态绑定
- 第13章 并发
 - 13.1 概述
 - 13.1.1 多处理器体系结构
 - 13.1.2 并发的种类

- 13.1.3 使用并发的目的
- 13.2 子程序级并发概述
 - 13.2.1 基本概念
 - 13.2.2 为并发而设计的语言
 - 13.2.3 设计问题
- 13.3 信号量
 - 13.3.1 概述
 - 13.3.2 合作同步
 - 13.3.3 竞争同步
 - 13.3.4 评价
- 13.4 管程
 - 13.4.1 概述
 - 13.4.2 竞争同步
 - 13.4.3 合作同步
 - 13.4.4 评价
- 13.5 消息传递
 - 13.5.1 概述
 - 13.5.2 同步消息传递的原理
- 13.6 Ada对并发的支持
 - 13.6.1 基本原理
 - 13.6.2 合作同步
 - 13.6.3 竞争同步
 - 13.6.4 任务终止
 - 13.6.5 优先级
 - 13.6.6 受保护对象
 - 13.6.7 评价
- 13.7 Java线程
 - 13.7.1 Thread类
 - 13.7.2 优先级
 - 13.7.3 信号量
 - 13.7.4 竞争同步
 - 13.7.5 合作同步
 - 13.7.6 非阻塞同步
 - 13.7.7 显式锁定
 - 13.7.8 评价
- 13.8 C#线程
 - 13.8.1 基本线程操作
 - 13.8.2 同步线程
 - 13.8.3 评价
- 13.9 函数式语言中的并发
 - 13.9.1 Multilisp
 - 13.9.2 并发ML
 - 13.9.3 F#
- 13.10 语句级并发
- 第14章 异常处理和事件处理
 - 14.1 异常处理概述
 - 14.1.1 基本概念
 - 14.1.2 设计问题
 - 14.2 Ada中的异常处理
 - 14.2.1 异常处理程序
 - 14.2.2 将异常绑定到处理程序
 - 14.2.3 继续
 - 14.2.4 其他设计选择
 - 14.2.5 例子

- 14.2.6 评价
- 14.3 C++中的异常处理
 - 14.3.1 异常处理程序
 - 14.3.2 异常与处理程序的绑定
 - 14.3.3 继续
 - 14.3.4 其他设计选择
 - 14.3.5 例子
 - 14.3.6 评价
- 14.4 Java中的异常处理
 - 14.4.1 异常类
 - 14.4.2 异常处理程序
 - 14.4.3 异常与处理程序的绑定
 - 14.4.4 其他设计选择
 - 14.4.5 例子
 - 14.4.6 finally子句
 - 14.4.7 断言
 - 14.4.8 评价
- 14.5 事件处理概述
- 14.6 Java的事件处理
 - 14.6.1 Java Swing的GUI组件
 - 14.6.2 Java事件模型
- 14.7 C#中的事件处理
- 第15章 函数式程序设计语言
 - 15.1 概述
 - 15.2 数学函数
 - 15.2.1 简单函数
 - 15.2.2 函数形式
 - 15.3 函数式程序设计语言基础
 - 15.4 第一种函数式程序设计语言LISP
 - 15.4.1 数据类型和结构
 - 15.4.2 第一个LISP解释器
 - 15.5 Scheme概述
 - 15.5.1 Scheme的起源
 - 15.5.2 Scheme解释器
 - 15.5.3 基本数值函数
 - 15.5.4 定义函数
 - 15.5.5 输出函数
 - 15.5.6 数值谓词函数
 - 15.5.7 控制流
 - 15.5.8 表函数
 - 15.5.9 用于符号原子和表的谓词函数
 - 15.5.10 Scheme函数示例
 - 15.5.11 LET
 - 15.5.12 Scheme中的尾递归
 - 15.5.13 函数形式
 - 15.5.14 构建代码的函数
 - 15.6 Common LISP
 - 15.7 ML
 - 15.8 Haskell
 - 15.9 F#
 - 15.10 基本命令式语言对函数式编程的支持
 - 15.11 函数式语言和命令式语言的比较
- 第16章 逻辑程序设计语言
 - 16.1 概述

- 16.2 谓词演算简介
 - 16.2.1 命题
 - 16.2.2 子句形式
- 16.3 谓词演算与定理证明
- 16.4 逻辑程序设计概述
- 16.5 Prolog的起源
- 16.6 Prolog的基本元素
 - 16.6.1 项
 - 16.6.2 事实语句
 - 16.6.3 规则语句
 - 16.6.4 目标语句
 - 16.6.5 Prolog的推理过程
 - 16.6.6 简单算术
 - 16.6.7 表结构
- 16.7 Prolog存在的缺陷
 - 16.7.1 归结的顺序控制
 - 16.7.2 封闭世界假设
 - 16.7.3 否定问题
 - 16.7.4 固有的限制
- 16.8 逻辑程序设计的应用
 - 16.8.1 关系数据库管理系统
 - 16.8.2 专家系统
 - 16.8.3 自然语言处理
- 参考文献
 - • • • • ([收起](#))

[编程语言原理_下载链接1](#)

标签

编程语言

计算机

编译原理

编程

计算机科学

计算机科学-语言与编译器

程序设计

PL

评论

读了1、2、6、9、10、11、12章，书中多是不同语言语法特性的罗列，没有我更想知道的why、具体的实现机理。

《程序设计语言原理》

太浅了，但有些关键点说的比较清晰

讲了各种语言的表层的逻辑，没有讲具体的实现原则，结合一定的编译器知识、语言设计知识再看也许更好。

说实话，这本书无论是对编程新手还是碰过三四门语言的人来说，都不是一本值得花时间看的书。我前面看的很细，一开始说程序语言发展的历史我还是比较感兴趣的，后面慢慢就变成针对编程语言的不同特性罗列各种不同语言的设计实现，往往也一笔而过，作者想说的太多，但写得又太少，最后成为介绍性的大杂烩

讲解了和编程语言相关的各种概念

大而全的编程语言介绍

[编程语言原理_下载链接1](#)

书评

作为一个程序员，一般只有精通一门程序设计语言就可以胜任当前的工作了。当往往进入一个新的项目，或者重新选择一份工作，或者自己发现当前的所使用的程序语言对于有些问题的解决，用着不是那么方便，这个时候我们就倾向于去寻求一种合适的语言。比如，我最近在工作过程中...

这本书的名气很大，很多人说是经典之作。
读过后是什么感觉呢？就是没什么感觉。读之前对程序语言有多少困惑和不解，读之后还是有多少困惑和不解。
为什么，因为书里对各种语言的叙述更多停留在语法层面上，是的，不同语言的语法是不一样的。但是为什么新的语言引入了一种东...

我有个“坏习惯”：碰到我买到的书都要评论，呵呵。
《概念》我也买了，不过买的是第5版的英文版。这本书我比较欣赏的是她介绍语言发展的部分《Evolution of the Major Programming Languages》（即：《主流程序设计语言的演化》），从我之前听都没听过的 Zuse Plankalkal ...

在读，不求甚解
只因为很多内容不能在自己的脑海中形成自己的体系，一些基本功有所欠缺所致
读到中间部分了，对于程序设计语言的历史，一些程序语言的特点及一些常见结构的形成原因有所了解，比如字符串，知道为什么会有字符串这个数据类型，在不同的语言中对于这个数据类型的...

清华出烂书，传统一直没有变。
这本书本身是不错的，但是翻译错误实在多了些，甚至有些都是排版引起的逻辑性错误。
看这书，做好和英文版本对比着看的心理准备吧，否者有些章节，比如讲解BNF和EBNF的地方，就让你一头雾水。

[编程语言原理_下载链接1](#)