

<<编程大师智慧>>

图书基本信息

书名：<<编程大师智慧>>

13位ISBN编号：9787564122621

10位ISBN编号：7564122625

出版时间：2010-6

出版时间：东南大学出版社

作者：（美）比安库利，（美）活登 著

页数：480

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>

前言

PROGRAMMING LANGUAGE DESIGN IS A FASCINATING TOPIC. There are so many programmers who think they can design a programming language better than one they are currently using; and there are so many researchers who believe they can design a programming language better than any that are in current use. Their beliefs are often justified, but few of their designs ever leave the designer's bottom drawer. You will not find them represented in this book. Programming language design is a serious business. Small errors in a language design can be conducive to large errors in an actual program written in the language, and even small errors in programs can have large and extremely costly consequences. The vulnerabilities of widely used software have repeatedly allowed attack by malware to cause billions of dollars of damage to the world economy. The safety and security of programming languages is a recurrent theme of this book.

<<编程大师智慧>>

内容概要

《编程大师智慧》的主要特色在于它是多位编程语言创造者的独家采访，他们创造的这些编程语言极具历史意义，对于当下的信息社会产生了重大影响。

从这部独特的采访集中，你将了解到某些特殊设计决定产生的过程，包括这些先行者当时头脑中的既定目标、他们不得不做的折衷权衡，以及这些宝贵经验至今对编程产生了怎样的影响。

如果你对那些用远见卓识和努力工作造就计算机产业的人们感兴趣的话，那么自然会发现《编程大师智慧》是一本令人着迷的好书。

作者简介

编者：（美国）比安库利（Federico Biancuzzi）（美国）沃登（Shane Warden）比安库利（Federico Biancuzzi），是一位自由采访者，他的采访报道发表在多家在线刊物上。

沃登（Shane Warden），是一位自由软件开发者，他的主要兴趣在于编程语言设计和虚拟机。

在业余时间，他是独立出版社Onyx Neon Press的小说出版部门负责人。

他也是O'Reilly图书《敏捷开发艺术》的合著者之一。

书籍目录

FOREWORD
 PREFACE
 1 C++ Design Decisions Using the Language OOP and Concurrency Future Teaching
 2 PYTHON The Pythonic Way The Good Programmer Multiple Pythons Expedients and Experience
 3 APL Paper and Pencil Elementary Principles Parallelism Legacy
 4 FORTH The Forth Language and Language Design Hardware Application Design
 5 BASIC The Goals Behind BASIC Compiler Design Language and Programming Practice Language Design Work Goals
 6 AWK The Life of Algorithms Language Design Unix and Its Culture The Role of Documentation Computer Science Breeding Little Languages Designing a New Language Legacy Culture Transformative Technologies Bits That Change the Universe Theory and Practice Waiting for a Breakthrough Programming by Example
 7 LUA The Power of Scripting Experience Language Design
 8 HASKELL A Functional Team Trajectory of Functional Programming The Haskell Language Spreading (Functional) Education Formalism and Evolution
 9 ML The Soundness of Theorems The Theory of Meaning Beyond Informatics
 10 SQL A Seminal Paper The Language Feedback and Evolution XQuery and XML
 11 OBJECTIVE-C Engineering Objective-C Growing a Language Education and Training Project Management and Legacy Software Objective-C and Other Languages Components, Sand, and Bricks Quality As an Economic Phenomenon Education
 12 JAVA Power or Simplicity A Matter of Taste Concurrency Designing a Language Feedback Loop
 13 C# Language and Design Growing a Language C# The Future of Computer Science
 14 UML Learning and Teaching The Role of the People UML Knowledge Be Ready for Change Using UML Layers and Languages A Bit of Reusability Symmetric Relationships UML Language Design Training Developers Creativity, Refinement, and Patterns
 15 PERL The Language of Revolutions Language Community Evolution and Revolution
 16 POSTSCRIPT Designed to Last Research and Education Interfaces to Longevity Standard Wishes
 17 EIFFEL An Inspired Afternoon Reusability and Genericity Proofreading Languages Managing Growth and Evolution
 AFTERWORD
 CONTRIBUTORS
 INDEX

章节摘录

This is of course what we have for usual numeric types, such as ints, doubles, complex numbers, and mathematical abstractions, such as vectors. This is a most useful notion, which C++ supports for built-in types and for any user-defined type for which we want it. This contrast to Java where built-in types such as char and int follow it, but user-defined types do not, and indeed cannot. As in Simula, all user-defined types in Java have reference semantics. In C++, a programmer can support either, as the desired semantics of a type requires. C# (incompletely) follows C++ in supporting user-defined types with value semantics. "General resource management" refers to the popular technique of having a resource (e.g., a file handle or a lock) owned by an object. If that object is a scoped variable, the lifetime of the variable puts a maximum limit on the time the resource is held. Typically, a constructor acquires the resource and the destructor releases it. This is often called RAII (Resource Acquisition Is Initialization) and integrates beautifully with error handling using exceptions. Obviously, not every resource can be handled in this way, but many can, and for those, resource management becomes implicit and efficient.

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>