<<真实世界的Haskell>>

图书基本信息

书名：<<真实世界的Haskell>>

13位ISBN编号：9787564119256

10位ISBN编号：756411925X

出版时间：2010-1

出版时间：东南大学出版社

作者：[美] 沙利文,[美] 戈尔,[美] 斯图尔特

页数：670

<<真实世界的Haskell>>

前言

Have We Got a Deal for You! Haskell is a deep language; we think learning it is a hugely rewarding experience. We will focus on three elements as we explain why. The first is novelty： we invite you to think about programming from a different and valuable perspective. The second is power： well show you how to create software that is short, fast, and safe. Lastly, we offer you a lot of enjoyment： the pleasure of applying beautiful programming techniques to solve real problems. Novelty Haskell is most likely quite different from any language youve ever used before. Compared to the usual set of concepts in a programmers mental toolbox, functional programming offers us a profoundly different way to think about software. In Haskell, we deemphasize code that modifies data. Instead, we focus on functions that take immutable values as input and produce new values as output. Given the same inputs, these functions always return the same results. This is a core idea behind functional programming. Along with not modifying data, our Haskell functions usually dont talk to the external world; we call these functions pure. We make a strong distinction between pure code and the parts of our programs that read or write files, communicate over network connections, or make robot arms move. This makes it easier to organize, reason about, and test our programs. We abandon some ideas that might seem fundamental, such as having a for loop built into the language. We have other, more flexible, ways to perform repetitive tasks. Even the way in which we evaluate expressions is different in Haskell. We defer every computation until its result is actually needed——Haskell is a lazy language. Laziness is not merely a matter of moving work around, it profoundly affects how we write programs. Power Throughout this book, we will show you how Haskells alternatives to the features of traditional languages are powerful and flexible and lead to reliable code. Haskell is positively crammed full of cutting-edge ideas about how to create great software.

Since pure code has no dealings with the outside world, and the data it works with is never modified, the kind of nasty surprise in which one piece of code invisibly corrupts data used by another is very rare. Whatever context we use a pure function in, the function will behave consistently.

<<真实世界的Haskell>>

内容概要

Haskell is most likely quite different from any language youve ever used before. Compared to the usual set of concepts in a programmers mental toolbox, functional programming offers us a profoundly different way to think about software. In Haskell, we deemphasize code that modifies data. Instead, we focus on functions that take immutable values as input and produce new values as output. Given the same inputs, these functions always return the same results. This is a core idea behind functional programming.

<<真实世界的Haskell>>

书籍目录

Interface from ImplementationMultiparameter TypeclassesFunctional DependenciesRounding Out Our ModuleProgramming to a Monad ’s InterfaceThe Reader MonadA Return to Automated DerivingHiding the IO MonadUsing a newtypeDesigning for Unexpected UsesUsing TypeclassesIsolation and TestingThe Writer Monad and ListsArbitrary I/O Revisited16. Using ParsecFirst Steps with Parsec ： Simple CSV ParsingThe sepBy and endBy CombinatorsChoices and ErrorsLookaheadError HandlingExtended Example ： Full CSV ParserParsec and MonadPlusParsing a URL-Encoded Query StringSupplanting Regular Expressions for Casual ParsingParsing Without VariablesApplicative Functors for ParsingApplicative Parsing by ExampleParsing JSON DataParsing a HTTP RequestBacktracking and Its DiscontentsParsing Headers17. Interfacing with C ： The FFIForeign Language Bindings ： The BasicsBe Careful of Side EffectsA High-Level WrapperRegular Expressions for Haskell ： A Binding for PCRESimple Tasks ： Using the C PreprocessorBinding Haskell to C with hsc2hsAdding Type Safety to PCREBinding to ConstantsAutomating the BindingPassing String Data Between Haskell and CTyped PointersMemory Management ： Let the Garbage Collector Do the WorkA High-Level Interface ： Marshaling DataMarshaling ByteStringsAllocating Local C Data ： The Storable ClassPutting It All TogetherMatching on StringsExtracting Information About the PatternPattern Matching with SubstringsThe Real Deal ： Compiling and Matching Regular Expressions18. Monad TransformersMotivation ： Boilerplate AvoidanceA Simple Monad Transformer ExampleCommon Patterns in Monads and Monad TransformersStacking Multiple Monad TransformersHiding Our WorkMoving Down the StackWhen Explicit Lifting Is NecessaryUnderstanding Monad Transformers by Building OneCreating a Monad TransformerMore Typeclass InstancesReplacing the Parse Type with a Monad StackTransformer Stacking Order Is ImportantPutting Monads and Monad Transformers into PerspectiveInterference with Pure CodeOverdetermined OrderingRuntime OverheadUnwieldy InterfacesPulling It All Together19. Error HandlingError Handling with Data TypesUse of MaybeUse of EitherExceptionsFirst Steps with ExceptionsLaziness and Exception HandlingUsing handleSelective Handling of ExceptionsI/O ExceptionsThrowing ExceptionsDynamic ExceptionsError Handling in MonadsA Tiny Parsing Framework20. Systems Programming in HaskellRunning External ProgramsDirectory and File InformationProgram TerminationDates and TimesClockTime and CalendarTimeFile Modification TimesExtended Example ： PipingUsing Pipes for RedirectionBetter PipingFinal Words on Pipes21. Using DatabasesOverview of HDBCInstalling HDBC and DriversConnecting to DatabasesTransactionsSimple QueriesSqlValueQuery ParametersPrepared StatementsReading ResultsReading with StatementsLazy ReadingDatabase MetadataError Handling22. Extended Example ： Web Client ProgrammingBasic TypesThe DatabaseThe ParserDownloadingMain Program23. GUI Programming with gtk2hsInstalling gtk2hsOverview of the GTK+ StackUser Interface Design with GladeGlade ConceptsEvent-Driven ProgrammingInitializing the GUIThe Add Podcast WindowLong-Running TasksUsing Cabal24. Concurrent and Multicore ProgrammingDefining Concurrency and ParallelismConcurrent Programming with ThreadsThreads Are NondeterministicHiding LatencySimple Communication Between ThreadsThe Main Thread and Waiting for Other ThreadsSafely Modifying an MVarSafe Resource Management ： A Good Idea， and Easy BesidesFinding the Status of a ThreadWriting Tighter CodeCommunicating over ChannelsUseful Things to Know AboutMVar and Chan Are NonstrictChan Is UnboundedShared-State Concurrency Is Still HardDeadlockStarvationIs There Any Hope?Using Multiple Cores with GHCRuntime OptionsFinding the Number of Available Cores from HaskellChoosing the Right RuntimeParallel Programming in HaskellNormal Form and Head Normal FormSequential SortingTransforming Our Code into Parallel CodeKnowing What to Evaluate in ParallelWhat Promises Does par Make?Running Our Code and Measuring PerformanceTuning for PerformanceParallel Strategies and MapReduceSeparating Algorithm from EvaluationSeparating Algorithm from StrategyWriting a Simple MapReduce DefinitionMapReduce and StrategiesSizing Work AppropriatelyEfficiently Finding Line-Aligned ChunksCounting LinesFinding the Most Popular URLsConclusions25. Profiling and OptimizationProfiling Haskell ProgramsCollecting Runtime StatisticsTime ProfilingSpace ProfilingControlling EvaluationStrictness and Tail RecursionAdding StrictnessUnderstanding CoreAdvanced Techniques ： FusionTuning the Generated AssemblyConclusions26. Advanced Library Design ： Building a Bloom

FilterIntroducing the Bloom FilterUse Cases and Package LayoutBasic DesignUnboxing， Lifting， and BottomThe ST MonadDesigning an API for Qualified ImportCreating a Mutable Bloom FilterThe Immutable APICreating a Friendly InterfaceRe-Exporting Names for ConvenienceHashing ValuesTurning Two Hashes into ManyImplementing the Easy Creation FunctionCreating a Cabal PackageDealing with Different Build SetupsCompilation Options and Interfacing to CTesting with QuickCheckPolymorphic TestingWriting Arbitrary Instances for ByteStringsAre Suggested Sizes Correct?Performance Analysis and TuningProfile-Driven Performance Tuning27. Sockets and SyslogBasic NetworkingCommunicating with UDPUDP Client Example：syslogUDP Syslog ServerCommunicating with TCPHandling Multiple TCP StreamsTCP Syslog ServerTCP Syslog Client28. Software Transactional MemoryThe BasicsSome Simple ExamplesSTM and SafetyRetrying a TransactionWhat Happens When We Retry?Choosing Between AlternativesUsing Higher Order Code with TransactionsI/O and STMCommunication Between ThreadsA Concurrent Web Link CheckerChecking a LinkWorker ThreadsFinding LinksCommand-Line ParsingPattern GuardsPractical Aspects of STMGetting Comfortable with Giving Up ControlUsing InvariantsA. Installing GHC and Haskell LibrariesB. Characters, Strings， and Escaping RulesIndex

章节摘录

In this section, weve discussed how Haskell, unlike most languages, draws a cleardistinction between pure code and I/O actions. In languages such as C or Java, thereis no such thing as a function that is guaranteed by the compiler to always return thesame result for the same arguments or a function that is guaranteed to never have sideeffects. The only way to know if a given function has side effects is to read its docu-mentation and hope that its accurate.Many bugs in programs are caused by unanticipated side effects. Still more are causedby misunderstanding circumstances in which functions may return different results forthe same input. As multithreading and other forms of parallelism grow increasinglycommon, it becomes more difficult to manage global side effects.Haskells method of isolating side effects into I/O actions provides a clear boundary.You can always know which parts of the system may alter state and which wont. Youcan always he sure that the pure parts of your program arent having unanticipatedresults. This helps you to think about the program. It also helps the compiler to thinkabout it. Recent versions of ghc, for instance, can provide a level of automatic paral-lelism for the pure parts of your code——something of a holy grail for computing.For more discussion on this topic, refer to "Side Effects with Lazy I/O" on page 188.

媒体关注与评论

“现代软件的最大问题在于性能、模块化、可靠性和并发性。
在《真实世界的Haskell》中，作者很好地讲授了如何使用HaskeII这一超前于当今主流的语言，来逐一化解这些问题。
”　　——Trim Sweeney，Epic Games创始人，同时也是Unreal　游戏引擎设计者　　“这是第一本涵盖了现实世界程序员所需一切技术的书籍。
当读罢此书，你将能够用当前所钟爱的语言写出更优秀的代码。
”　　——Simon Peyton Jones.Microsoft Research Haskell语言架构师，GlasgowHaskell 编译器设计者

<<真实世界的Haskell>>

编辑推荐

《真实世界的Haskell(影印版)》是一本上手快且易于使用的指导书，它向你介绍这门日趋流行的编程语言。
你将学习如何将Haskell应用于不同实践当中，从简短的脚本到要求苛刻的大型应用。
《真实世界的Haskell(影印版)》向你讲解了函数式编程的基础，帮助你加深对如何在现实世界中应用Haskell的理解，例如输入／输出性能、数据处理、并发等等。
《真实世界的Haskell》能帮助你：·理解过程式与函数式编程之间的差异·学习Haskell的特性，以及如何使用它来开发有用的程序·与文件系统、数据库和网络服务交互·编写可以进行自动测试、代码覆盖和错误处理的代码·通过并发和并行编程发挥多核系统的威力在《真实世界的Haskell(影印版)》中你将发现大量的实用习题和真实的Haskell程序示例，你可以修改、编译及运行它们。
无论是否曾经使用过函数式语言，如果想要了解Haskell为何成为众多组织所选用的实用语言，《真实世界的Haskell》是你的首选。

<<真实世界的Haskell>>

版权说明