

<<软件体系结构>>

图书基本信息

书名：<<软件体系结构>>

13位ISBN编号：9787308054539

10位ISBN编号：7308054535

出版时间：2008-12

出版时间：浙江大学出版社

作者：邢剑宽,郑翔覃征

页数：337

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>

内容概要

Building software nowadays is far more difficult than it can be done several decades ago. At that time , software engineers focused on how to manipulate the computer to work and then solve problems correctly. The organization of data and implementation of algorithm were the crucial process of software designing then. However , more and more tasks in low level , such as memory management and network communication , have been automatized or at least can be reused with little effort and cost. Programmers and designers , with the help of high level programming languages and wieldy development tools , can pay more attention to problems , rather than bury themselves into the machine code manuals. However , the side effect of these utilities is that more complicated problems are given according to their requirements from military , enterprise and so on , in which the complexity grows rapidly day by day. We believe that software architecture is a key to deal with it.

书籍目录

1 Introduction to Software Architecture
 1.1 A Brief History of Software Development
 1.1.1 The Evolution of Programming Language——'Abstract Level'
 1.1.2 The Evolution of Software Development- Concerns
 1.1.3 The Origin and Growth of Software Architecture
 1.2 Introduction to Software Architecture
 1.2.1 Basic Terminologies
 1.2.2 Understanding IEEE 1471-2000
 1.2.3 Views Used in Software Architecture
 1.2.4 Why We Need Software Architecture
 1.2.5 Where Is Software Architecture in Software Life Cycle
 1.3 Summary
 References
 2 Architectural Styles and Patterns
 2.1 Fundamentals of Architectural Styles and Patterns
 2.2 Pipes Filters
 2.2.1 Style Description
 2.2.2 Study Case
 2.3 Object-oriented
 2.3.1 Style Description
 2.3.2 Study Case
 2.4 Event-driven
 2.4.1 Style Description
 2.4.2 Study Case
 2.5 Hierarchical Layer
 2.5.1 Style Description
 2.5.2 Study Case
 2.6 Data Sharing
 2.6.1 Style Description
 2.6.2 Study Case
 2.7 Virtual Machine
 2.7.1 Style Description
 2.7.2 Study Case
 2.8 Feedback Loop
 2.8.1 Style Description
 2.8.2 Study Case
 2.9 Comparison among Styles
 2.10 Integration of Heterogeneous Styles
 2.11 Summary
 References
 3 Application and Analysis of Architectural Styles
 3.1 Introduction to SMCSP
 3.1.1 Program Background
 3.1.2 Technical Routes
 3.1.3 Function Design
 3.2 System Realization
 3.2.1 The Pattern Choice
 3.2.2 Interaction Mechanism
 3.2.3 Realization of Mobile Collaboration
 3.2.4 Knowledge-based Design
 3.3 Summary
 References
 4 Software Architecture Description
 4.1 Formal Description of Software Architecture
 4.1.1 Problems in Informal Description
 4.1.2 Why Are Formal Methods Necessary
 4.2 Architectural Description Language
 4.2.1 Introduction to ADL
 4.2.2 Comparing among Typical ADLs
 4.2.3 Describing Architectural Behaviors
 4.3 Study Case : WRIGHT System
 4.3.1 Description of Component and Connector
 4.3.2 Description of Configuration
 4.3.3 Description of Style
 4.3.4 CSP-Semantic Basis of Formal Behavior Description
 4.4 FEAL : An Infrastructure to Construct ADLs
 4.4.1 Design Purpose
 4.4.2 FEAL Structure
 4.4.3 FEAL Mapper
 4.4.4 FEAL Application Examples
 4.4.5 Summary

 5 Design Strategies in Architecture Level
 6 Software Architecture IDE
 7 Evaluating Software Architecture
 8 Flexible Software Architecture
 9 A Vision on Software Architecture
 Index

章节摘录

(3) It is easy to implement grammar. It is similar to define the class of every node in the abstract grammar tree, and these classes are easy to write directly. Generally, they can be automatically generated by compiler or grammar analysis generator. In this part, we will describe the roles in the Boolean expression system. Generally speaking, there are five roles in this type of system. The first one is BooleanExpression. This role declares an abstract Evaluate operation, this interface is shared by all the nodes of the Boolean expression abstract grammar tree. The second role is TerminalExpression (such as VariableExpression and Constant). This type of role implements the Evaluation operation in the BooleanExpression that are related to terminals, every terminal in the Boolean Expression needs an object instance of this class. The third role is NonterminalExpression (such as AndExpression, Or Expression and NotExpression). Every rule in the Boolean expression grammar needs an object instance of NonterminalExpression, and we must maintain object instance of Boolean Expression for every symbol in every rule in the Boolean expression grammar. We also need to implement the Evaluate operation for every NonterminalExpression in the grammar. In the NonterminalExpression Evaluate operation, we must call the Evaluate operation for every symbol in the grammar. The fourth role is context (this is "the inner state of interpreter engine") It includes the global information besides the interpreter. The fifth role is client. Client will construct a special Boolean expression's abstract grammar tree in the Boolean expression's definition, and this abstract grammar tree is composed by the instance objects of TerminalExpression and NonterminalExpression. The client will also call the Evaluate operation. The collaboration relationship between these five roles can be simply described as follows: At first, the client constructs a Boolean Expression, which is an abstract grammar tree which is composed by instances of TerminalExpression and NonterminalExpression. Then the client initiates the context and calls interpret operation. Then every NonterminalExpression defines the Evaluate operation of the according expression, and all the Evaluate operation of the expression forms the basis of recursive evaluation. At last, the Evaluate operation of every node uses the context to store and access the states of interpreter system. In this and the following part, we will introduce the implementation methods of Boolean expression evaluation system. When encountering the real implementation of Boolean expression, we have many details to deal with, and the process quality of these details directly influences the whole system's performance. These problems are mainly incarnated in the following aspects: The first problem is to construct the abstract grammar tree. The interpreter style does not specify how to construct an abstract grammar tree in detail, that is to say, the interpreter style does not involve syntax analysis. But when we are constructing an abstract grammar tree, we need to use a table-driven grammar analysis program to finish this task; we can also use the recursive decline grammar analysis program to construct the abstract grammar tree. The second problem is how to define the Evaluate operation. In fact, Evaluate operation does not need to be defined and implemented in the expression's classes. If we need to construct a new interpreter frequently, we can use the Visitor style in design pattern theory, put the Evaluate operation in an independent "Visitor" object, this method may be better. For instance, a program design language has many operations on abstract grammar tree, such as type check, code optimization and code generation, etc. A proper way is to use a visitor, so as to avoid defining this operation in every class. The third problem is the shared terminals. In some grammars, many terminals may occur in the same sentences (such as true and false in Boolean expression evaluation system). In this case, it is better to share the copy of that symbol. The terminal nodes usually do not store their positions in the grammar tree, in the process of evaluation, any context information they required is transferred by their parent nodes. So, the inner state and outer state in the terminal node are explicitly different. We can implement those using Flyweight design patterns. In the implementation of Boolean expression evaluation system, we define two operations in the Boolean expression. The first operation is Evaluate, which evaluates the value of the specified Boolean expression in the context, and this context must provide "true" or "false" for every variable. The second operation is Replace, which replaces a variable with an expression so as to generate new Boolean expression. The Replace operation makes the system not only finish the evaluation of Boolean expression, but also do the grammar

<<软件体系结构>>

analysis of the Boolean expression. Because of the manuscript length constraint , we will not describe the implementation details of each subclass. The interpreter style has an important characteristic : we can use many operations to "interpret" the same sentence. Among the three operations we defined in the BooleanExpression , the Evaluate operation is the basic operation in the process of computing Boolean Expression. It interprets a Boolean expression and returns a simple result. But in the above system , we do not only have the Evaluate operation , the replacement and copy can also be treated as interpreter , and the only difference is the interpretation for the sentence.

<<软件体系结构>>

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>