基于 C 语言实现的 IDL 编译器

李 颖,胡 明
(电子科技大学 通信与信息工程学院,四川 成都 611731)

摘 要:对象管理组织指定的 CORBA 规范是分布式对象计算的中间件标准,它允许透明地访问远程对象,同时支持异构系统的互操作,其中,IDL 编译器是分布式计算环境的基本开发工具。为了开发出具有高效性和可维护性的 IDL 编译器,文中首先提出了用 C 语言设计实现的 IDL 编译器三模块设计模式,然后讨论了 IDL 编译器在实现过程中所涉及到的数据结构、顶层接口、函数执行流程等,最后对文中实现的 IDL 编译器进行了测试。结果表明,文中设计实现的 IDL 编译器能成功地完成 IDL 到 C++的转换;极大地缩减开源 CORBA 产品 ACE/TAO(TAO:The ACE ORB)使用的 IDL 编译器 TAO_idl 编译出来的桩和框架的大小;当给常量进行赋值时,如果它们都属于数值型或者字符型,而赋值运算符两边的类型不一致时,文中设计实现的 IDL 编译器能极好地改善 TAO_idl,在赋值时进行类型转换的问题。

关键词:CORBA;接口定义语言;IDL 编译器;C 语言
中图分类号:TP314 **文献标识码**:A **文章编号**:1673-629X(2013)03-0005-05
doi:10.3969/j.issn.1673-629X.2013.03.002

An IDL Compiler Designed and Implemented by C Language

LI Ying,HU Ming
(School of Communication & Information Engineering,UESTC,Chengdu 611731,China)

Abstract:The CORBA specification,which is standardized by the Object Management Group (OMG),is a middleware standard for distributed object computing. It allows transparently access to remote objects and supports the interoperability of heterogeneous systems. And IDL compiler is a basic development tools for the distributed computing environments. In order to develop a high-efficiency and maintainable IDL compiler,firstly,explore a three-module development model of IDL compiler. Then it discusses some technical problems such as data structures,top-level interface and some function execution flow chart in the process of design and implementation of the compiler. At last,carry out tests. The results show that the IDL compiler designed and implemented by C can translate IDL to C++ successfully,and it can reduce the size of stub and skeleton which were produced by TAO's IDL compiler. And also when it comes to assign a value to a constant variable,the IDL compiler implemented can improve the type conversion problem of TAO's IDL compiler,when two sides of the assignment operator are not the same type but both of them are numeric or character type.

Key words:CORBA;IDL;IDL compiler;Clanguage

0 引 言

公共对象请求代理结构(CORBA:Common Object Request Broker Architecture)致力于使多样化的对象系统一体化[1,2]。它开发在应用程序之间可共享、可重用的软件组件。每一个对象借助一个与外界通信的接口来实现对其内部工作细节的封装。对象请求代理(ORB:Object-Request Broker)是 CORBA 的核心[3],它是整合到终端用户应用程序的网络资源和库的集合。ORB 的作用是跟踪、维持接口,简化接口之间的通信,并且利用接口给应用程序提供服务。对象可以通过 ORB 透明的发送或接收应答,这就保证了分布式异构环境下应用程序之间的互操作性。接口用标准化的 IDL 实现。

IDL 用于描述分布式对象接口,它保证了 CORBA 对象在一系列异构的操作系统、开发商和编程语言上协同工作。用户编写的 IDL 接口和 IDL 编译器的作用如图 1。

服务端程序实现 IDL 文件当中的定义的接口,而客户端程序实体则调用服务端程序实现的接口。客户端调用的接口和服务端所实现的接口采用 IDL 定义。IDL 文件通过 IDL 编译器的编译,可以生成客户端的桩和服务器端的框架。客户端的桩,负责把客户端的请求进行编码,通过客户端的 ORB 发送到服务端,并

收稿日期:2012-07-12;**修回日期**:2012-10-16
基金项目:中央高校基金(ZYGX2010J012)
作者简介:李 颖(1986-),男,四川简阳人,硕士研究生,研究方向为通信与信息系统;胡 明,副教授,硕士生导师,研究方向为通信网与宽带通信技术、光分组传输网、DWDM。

把返回的应答解码后,传送给客户服务器端的框架负责把客户端经过 ORB 发送来的请求解码,定位对象方法并执行,然后把结果编码后,作为应答经服务端 ORB 返送给客户端[2,3]。由于客户端的桩和服务端的框架需要完成 C/S 模型中请求与应答的编解码,所以,IDL 编译器成为了基于 CORBA 标准的分布式计算环境中最基本的开发工具。正是基于此,文中提出了 IDL 编译器的 C 语言实现方法,并且对其进行了实现。最后,文中对实现的 IDL 编译器进行了测试。

测试结果表明,文中实现的 IDL 编译器与 TAO_idl 相比有两点改善:

A. 当给常量赋值时,如果赋值运算符两边的类型不同,但是,如果运算符两侧都属于数值型或者字符型数据时,文中设计并实现的 IDL 编译器能改善 TAO_idl 的类型转换问题。

B. 相同接口的 IDL 文件经由文中设计实现的 IDL 编译器编译后,生成的客户端桩和服务端框架要比经由 TAO_idl 编译后生成的客户端桩和服务端框架要小。这样就可以有效地节省存储空间。

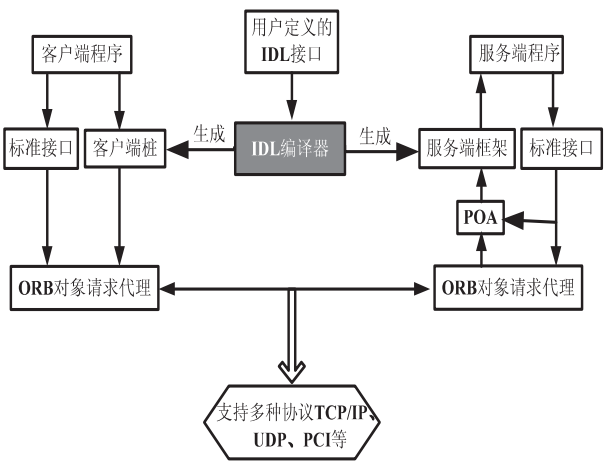

图 1　IDL 编译器在分布式计算应用中的位置端

1　IDL 编译器的设计与实现

1.1　IDL 编译器的三模块设计模式

由于分布式计算环境支持异种平台与异种语言间的应用,从而导致了多种 IDL 编译器的出现[4~6],针对这个特点,文中在对编译器进行设计的时候,采用了如图 2 的三模块设计模式。

图 2　IDL 编译器的三模块设计模式

(a)输入信息处理模块,完成对输入的 IDL 文件的预编译处理、词法分析和语法分析[7~11]等操作。

(b)中间信息存储模块,为 IDL 文件中每种语法单位定义一种数据结构,并且对 IDL 文件进行语法分析和语义分析,同时将 IDL 文件中的信息进行分类存储,生成一棵语法树[7~11]。

(c)目标代码生成模块,遍历中间信息存储模块生成的语法树[12],按照 IDL 向 C++的映射规则,生成目标代码。

文中使用的 IDL 编译器三模块设计模式是一种具有扩展性和继承性的开发设计方法。例如,如果要实现由 IDL 到其他语言的映射[4,5],那么仅仅需要修改目标代码生成模块中的目标代码生成函数;如果将来 IDL 的语法规则发生变化,那么只需要修改中间信息存储模块中语法单位解析函数[11,12]。

1.2　IDL 编译器的内部数据结构

文中为 IDL 语言的每种语法单位建立了一种对应的数据类型的节点,以存储该种语法单位的信息。

1.2.1　输入信息处理模块使用地数据结构

输入信息处理模块只使用到一种类型的节点,即 Global_In 节点。该节点是对输入 IDL 文件进行封装。参数 pd_fp,为输入 IDL 文件的文件指针,通过它对 IDL 文件进行读取;参数 pd_line,记录当前已经分析代码的行数,以便在 IDL 文件出错的时候,进行错误的报告。

1.2.2　中间信息存储模块使用地数据结构

中间信息处理模块使用地数据结构为:Node_Typedef 节点,Node_Const 节点,Node_Attribute 节点,Node_Sequence 节点,Node_Array 节点,Node_Enum 节点,Node_Union 节点,Node_Struct 节点,Node_Exception 节点,Node_Function 节点,Node_Interface 节点,Node_Module 节点,Global_Out 节点。表 1 以 Global_Out 节点为例,说明中间信息存储模块中各个节点内部信息的设置。

表 1　节点 Global_Out 内部信息

节点成员	说明
char * pd_name	存储 idl 文件的名字
Node_Typedef * pd_tpdef	别名信息链表的头指针
Node_Const * pd_const	常量信息链表的头指针
Node_Sequence * pd_seq	序列信息链表的头指针
Node_Enum * pd_enum	枚举信息链表的头指针
Node_Union * pd_union	联合信息链表的头指针
Node_Struct * pd_struct	结构体信息链表的头指针
Node_Exception * pd_exception	异常信息链表的头指针
Node_Interface * pd_interface	接口信息链表的头指针
Node_Module * pd_module	模块信息链表的头指针
Node_String * pd_para	存储全局环境中各成员顺序

说明:表 1 给出了节点 Global_Out 的内部细节。该节点的设置参照 IDL 中 IDL 文件的语法规则。中间信息存储模块中的其他节点类型也是参照相应语法单

位在 IDL 中的语法规则进行设置。一个 IDL 文件经输入信息处理模块和中间信息存储模块处理以后会生成一棵语法树。每棵语法树都会以一个 Global_Out 节点作为该语法树的根节点。该语法树可能包含九棵子树。这九棵子树分别以以下九种数据类型:别名信息,常量信息,枚举信息,联合信息,结构体信息,异常信息,接口信息,模块信息,序列信息的节点作为其子树根节点。节点 Global_Out 里分别用 pd_typedef,pd_const,pd_enum,pd_union,pd_struct,pd_exception,pd_interface,pd_module,pd_seq 来指向这九棵子树的根节点。另外,Global_Out 里还有另外两个数据成员,一个是 pd_name,它用于存储 idl 文件的名字,另外一个是 pd_para,它存储全局环境中各成员的顺序。

1.2.3　目标代码生成模块使用的数据结构

目标代码生成模块使用地数据结构为:栈 Stack,结构体 FileStack,结构体 FileStub,结构体 FileSkeleton。表 2 以结构体 FileStub 为例说明了目标代码生成模块所使用的数据结构的内部设置。

表 2　结构体 FileStub 的内部信息

节点成员	说明
FileStack StubH	包含桩的头文件指针与管理该文件缩进量的栈
FileStack StubCpp	包含桩的源文件指针与管理该文件缩进量的栈

说明:表 2 给出了结构体 FileStub 的内部细节。该数据类型包含客户端桩(Stub)的文件指针和管理该指针所指文件每行缩进量的栈。结构体 FileSkeleton 的内部细节与结构体 FileStub 的区别仅仅在于,结构体 FileStub 存储的是客户端桩的文件指针,而结构体 FileSkeleton 存储的是服务端框架的文件指针,其他的都是一样。其中,结构体 FileStack 内部包含两个成员,第一个成员的数据类型是栈 Stack,;第二个数据成员的数据类型是一个文件指针,它与栈 Stack 配对。

1.3　IDL 编译器的内部函数接口

在这一小节,文中将给出在编译器设计过程中,所使用的主要函数接口。

1.3.1　输入信息处理模块中使用地顶层函数接口

int scanner(Global_In * global_in,char * word,int word_len);

说明:

(a)参数 global_in 包含已分析的代码行数和输入文件的文件指针。

(b)参数 word 存储从 IDL 文件中取出一个合法标识符。

(c)参数 word_len 指定从 IDL 文件中取出的合法标识符的最大长度。

(d)该接口负责从输入 IDL 文件中取出一个合法字符串,同时在取出字符串的过程中,对输入 IDL 文件进行词法和语法检查,如果有错,则报错,并返回。

1.3.2　中间信息存储模块使用的顶层函数接口

int attribute_parser(Global_In * global_in,Node_Attribute * attribute_ptr);

int const_parser(Global_In * global_in,Node_Const * const_ptr);

int enum_parser(Global_In * global_in,Node_Enum * enum_ptr);

int exception_parser(Global_In * global_in,Node_Exception * exception_ptr);

int function_parser(Global_In * global_in,Node_Function * function_ptr);

int interface_parser(Global_In * global_in,Node_Interface * interface_ptr);

int module_parser(Global_In * global_in,Node_Module * module_ptr);

int sequence_parser(Global_In * global_in,Node_Sequence * sequence_ptr);

int struct_parser(Global_In * global_in,Node_Struct * struct_ptr);

int typedef_parser(Global_In * global_in,Node_Tpdef * typedef_ptr);

说明:

(a)接口 attribute_parser 的参数 global_in,包含已分析的代码行数和输入文件的文件指针。

(b)接口 attribute_parser 的参数 attribute_ptr,存储当前正在分析的 attribute 属性的信息。

(c)接口 attribute_parser 负责对属性 attribute 进行解析,然后把属性的信息用参数 attribute_ptr 返回,并插入到语法树中。

(d)中间信息存储模块其他的接口,将会对其他相应的语法单元进行解析,并且把相应的信息返回。在这些接口中,它们的第一个参数和接口 attribute_parser 相同,第二个参数将用于存储当前正在分析的语法单元的信息,最后把解析得到的信息插入到语法树。

1.3.3　目标代码生成模块中,使用地顶层函数接口

int gen_stub_h(Global_Out * global_out,FileStack * stub_h);

int gen_stub_cpp(Global_Out * global_out,FileStack * stub_cpp);

int gen_skeleton_h(Global_Out * global_out,FileStack * skeleton_h);

int gen_skeleton_cpp(Global_Out * global_out,

FileStack * skeleton_cpp);

说明:

(a)接口 gen_stub_h 的参数 global_out,是中间信息存储模块生成的语法树的根节点。

(b)接口 gen_stub_h 中的参数 stub_h,包含了客户桩(Stub)的头文件的文件指针和用于控制桩文件每行缩进量的栈。

(c)接口 gen_stub_h 负责产生客户端桩的头文件。

(d)接口 gen_stub_cpp 与接口 gen_stub_h 区别在于,接口 gen_stub_h 生成的是客户端桩的头文件,接口 gen_stub_cpp 产生的是客户端桩的源文件。

(e)接口 gen_skeleton_h,接口 gen_skeleton_cpp,这两个接口与接口 gen_stub_h,gen_stub_cpp 的区别是,前两者产生的是服务端框架的头文件和源文件。而后两者产生的是客服端桩的头文件和源文件。

1.4 三模块执行流程分析

1.4.1 输入信息处理模块和中间信息存储模阶段

前两个模块的执行流程如图3。

step1:判断文件指针是否到达文件末尾,如果到达文件末尾,那么执行 step6;如果没有达到文件末尾则执行 step2。

step2:从输入 IDL 文件中提取一个字符串。

step3:判断该字符串是否为合法的字符串,如果为合法字符串,则执行 step4,否则执行 step5。

step4:判断如果该字符串为异常,即字符串为"exception",调用 exception_parser 完成异常信息的词法分析、语法分析、信息提取、信息的存储,然后返回 step1 继续执行;如果字符串为常量(const)、结构体(struct)、别名(typedef)、联合(union)、枚举(enum)、接口(interface)、模块(module),则调用相应的解析函数,完成相应信息的词法分析、语法分析、信息提取、信息的存储,然后返回 step1 继续执行。如果字符串不是上边提到的任何一种字符串,那么,执行 step5。

step5:报告错误类型和错误行数,然后执行 step6。

step6:结束。

当对常量进行赋值的时候,文中设计实现的 IDL 编译器参照标准 C 编译器进行类型扩展的方法,以达到当赋值运算符两侧类型不同时,也可以进行赋值。

1.4.2 目标代码生成模块阶段

中间信息存储模块中的 Global_Out 节点, Node_Module 节点和 Node_Interface 节点,这三个节点里边都有一个数据成员 pd_para,它记录了当前节点内部,各个数据成员的先后顺序。文中设计实现的 IDL 编译器在目标代码生成模块阶段,将会对中间信息存储模块所生成的语法树进行遍历,同时基于 pd_para 记录的各数据成员的先后顺序以生成客户端的桩和服务端的框架。

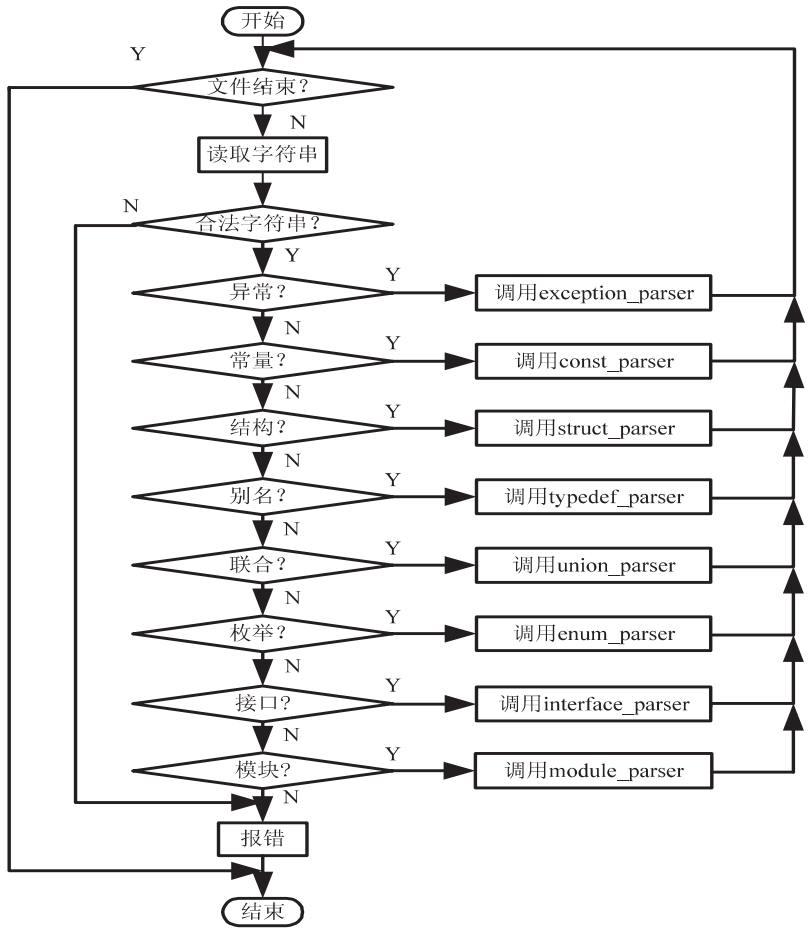

图3 输入信息处理和中间信息存储模块执行流程

为了提高效率和减少生成客户端桩和服务端框架的大小,文中在设计实现 IDL 编译器的时候,尽量采用模板类、前置自增等方法。

2 测 试

2.1 测试内容

文中设计实现的 IDL 编译器在测试的时候,主要集中在两个方面,一个是功能测试,一个是性能测试:

(a)功能测试。

由于输入 IDL 文件是无法穷举的,因此文中采用了等价类测试的方式进行测试。通过将输入文件的语句分成多个能够覆盖 IDL 语法各种情况的等价类,并

分别针对各个等价类完成测试。

经过细致的功能测试,文中设计实现的 IDL 编译器能够正确的对输入 IDL 文件进行语法解析,并对错误的语句进行报错。文中设计和实现的模板能够实现《IDL 到 C++语言映射》中规定的语义。当对常量类型进行赋值时的可以有效地改善 TAO_idl 在赋值运算符两端类型不一致时的类型转换问题。同时,该编译器能够将输入的 IDL 数据进行正确的映射,对 IDL 文件中定义的类型正确地编解码,生成的桩和框架能够正确地完成与本课题设计实现的 CORBA 中间件进行配合。

到目前为止,本课题组利用基于自己开发的分布式平台为“XX 工程学院”开发的三个波形已经成功交付使用。在波形开发过程中,使用的便是由文中设计实现的 IDL 编译器。

(b)性能测试。

在性能测试环节,文中采用的仍然是功能测试中的相同的等价类测试方法。文中设计实现的 IDL 编译器,编译出来的桩和框架,与文中相配合的 CORBA 调用所需要的时间与 TAO 通用版 COBRA 调用由它自己的编译器编译出来的桩和框架的时间相比,略微有所减少,大约能节省几毫秒。同时由表 3,不同的数据类型的 IDL 接口文件,经由文中设计实现的 IDL 编译器编译生成的桩和框架,要比由 TAO_idl 编译出来的客户端桩和服务端框架的大小要小很多,这样就可以有效的节省存储空间,并且这种存储空间大小的节省并不是以牺牲 IDL 编译器的功能或者是客户端桩和服务端框架的功能为代价的。

表 3　不同类型 IDL 接口文件编译生成的桩和框架大小对比

测试内容	TAO_idl	文中设计的 IDL 编译器
基本类型	69.2KB	43.2KB
基本类型定长结构体	87.2KB	53.3KB
构造类型定长结构体	100KB	78KB
基本类型变长结构体	112.3KB	81.4KB
构造类型变长结构体	129KB	83.2KB
基本类型属性	56KB	34KB
构造类型属性	80KB	57KB
基本类型定长序列	78KB	44KB
构造类型定长序列	91KB	73KB
基本类型变长序列	83KB	49KB
构造类型变长序列	99.4KB	69KB
前向声明类型	58.7KB	34.1KB

2.2　测试结果及结论

经过细致的测试,文中设计实现的 IDL 编译器能够正确的对输入 IDL 文件进行词法和语法解析,并对 IDL 文件当中的错误进行报告。文中设计实现的 IDL 编译器能够对 IDL 当中的数据类型进行映射,而且生成的桩和框架能够正确地完成与 CORBA 的配合。当对常量进行赋值时,若赋值运算符两侧类型不同,文中设计实现的 IDL 编译器能有效地改善 TAO_idl 的类型扩展问题,同时该编译器在可以达到和 Tao_idl 相同的性能指标的同时,还可以缩减 TAO_idl 编译出来的客户端桩和服务端框架的大小。

3　结束语

文中设计实现的 IDL 编译器是用 C 语言来设计实现的。正如前面测试部分描述的那样,该编译器目前已调试通过,并进行了试运行。文中的工作将为 CORBA 中间件和 SCA 的进一步研究奠定了基础,将继续进行 CORBA 和 SCA 的研究工作,以期望获得更大的突破。

参考文献:

[1] 汪　芸. CORBA 技术及其应用[M]. 南京:东南大学出版社,1999.

[2] 韦乐平. CORBA 系统结构、原理与规范(美)[M]. 北京:电子工业出版社,2000.

[3] Object Management Group. The Common Object Request Broker Architecture Specification Version 2.6[S]. 2001.

[4] Object Management Group. The C++ Language Mapping Specification Version 1.1[S]. 2003.

[5] 韦乐平. CORBA 语言映射(美)[M]. 北京:电子工业出版社,2001.

[6] Henning M, Vinoski S. Advanced CORBA Programming with C++[M]. [s.l.]:Addison-Wesley,2000.

[7] 童亚拉. 分布式编译的方法和系统研究[J]. 计算机技术与发展,2010,20(5):79-82.

[8] 夏　铭,陆　阳,盛业兴,等. 嵌入式数据库中利用 Lex, Yacc 设计 SQL 编译器[J]. 计算机技术与发展,2006,16(11):121-124.

[9] 刘　阳,王　静. 编译技术原理及其实现方法[M]. 北京:学苑出版社,1994.

[10] 孙悦红. 编译原理及实现[M]. 北京:清华大学出版社,2005.

[11] Holmes J. Building Your Own Compiler with C++[M]. [s.l.]:Prentice Hall,1994.

[12] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures[M]. [s.l.]:Morgan Kaufmann,2001.

基于C语言实现的IDL编译器

作者： 李颖， 胡明
作者单位： 电子科技大学 通信与信息工程学院,四川 成都 611731
刊名： 计算机技术与发展
英文刊名： Computer Technology and Development
年，卷(期)： 2013(3)