第33卷 第21期 计 算 机 工 程 2007年11月
Vol.33 No.21 Computer Engineering November 2007
·软件技术与数据库· 文章编号：1000—3428(2007)21—0066—03 文献标识码：A 中图分类号：TP393.04

基于编译技术的协议解析方法

董 立，赵恒永
（北京化工大学信息科学与技术学院，北京 100029）

摘 要：在过程工业的控制中存在着大量的通信协议，这些协议的结构差别很大。要进行上层应用开发，必须对这些协议进行解析和处理。该文讨论了用形式化描述的方法对协议进行描述，实现了与协议无关的协议解析和处理，从而避免了针对不同通信协议均要编写相应的解析和处理程序，使协议的解析和处理具有更好的灵活性和普适性。

关键词：协议分析；数据帧；编译技术

Protocol Parsing Method Based on Compiling Technology

DONG Li, ZHAO Heng-yong
(School of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029)

【Abstract】 In process industry, there are many communication protocols, but they have many differences in structure. In order to develop up-layer application, it needs to parse and process these protocols. This paper discusses the protocol in which parsing and processing program can be implemented when the protocol is described with the formalize-description, and the program based on formalize-description is independent of the protocol. This implementation can avoid coding parse and processing program for different protocols separately, and makes the program more flexible and adaptable.

【Key words】 protocol analysis; data-frame; compiling technology

在过程工业中，工业自动化技术日趋成熟，自动化控制系统的应用已经非常普及，但是各类控制系统绝大多数是封闭的系统，控制装置自身存在封闭性。不同的厂家为各自的产品提供不同的协议和通信方式，缺乏统一的、标准的开放式接口。如果要集成在一起，必须为它们开发专用的通信接口。随着工业自动化系统功能要求越来越复杂，完全用一个厂家的产品来构成整个系统是很困难的。用户在进行上层应用程序开发的时候，通常需要对协议进行解析，从中提取出所需数据项，并进行相应的存储、加工计算和显示等处理，这就要求用户不得不针对特定的控制系统开发特定的、专用的通信接口。

这些通信接口程序之间的结构非常类似，主要的区别在于对网络接口收到数据帧的理解和解释不同，极易造成整个应用系统的复杂和冗余。更为不利的是，一旦底层设备发生变化，就可能造成整个应用程序的重新编写和编译，大大增加了系统的维护量和开发人员的负担[1]。

1 控制设备协议分析

对于不同控制系统中的控制协议，其数据帧格式往往不同。但是在特定的控制系统中所涉及到的数据帧通常都有明确的格式定义，并且数据帧与数据帧之间往往是相对独立的。更为重要的是控制系统中的设备要完成通信功能必须在物理链路上建立逻辑连接。这里的物理链路包括以太网、串行口等，相应的逻辑连接是TCP/IP协议、串行通信协议，在此基础上各种设备再定义自己的应用层协议。虽然应用层协议的类别会因设备和厂家的不同而有很大的变化，但是仔细分析就会发现，这些协议都各自具有确定的帧格式，如哪些字节表示命令、哪些字节表示参数名、哪些字节表示设备位号、哪些字节表示数值等。

既然所有的通信协议都遵守一定的帧格式，那么就可以根据已知的帧格式构造此协议的解析程序，针对每一种通信协议手工编写这样的解析程序是没有问题的，本文讨论的是能不能将通信协议数据帧格式的描述与协议的解析代码相分离，用稳定的程序来处理不稳定的数据帧格式。如果答案肯定，那么就能对不同设备采用同一个接口采集数据，如果底层设备的通信协议发生变化，只需要改变协议相应的描述，而不必重新编写和编译协议的解析程序，可以降低系统的复杂度，便于管理和扩充。

2 协议解析的解决方案

如前所述，编译原理中的基本词法分析和语法分析的思想为解决所面临的问题提供了理论基础，特别是词法分析器自动生成器和语法分析器自动生成器提供了很好的借鉴方法[2]。词法分析器自动生成器的基本实现思想是：(1)对输入流进行形式化描述生成一个状态转换图，根据此状态图可以识别此种输入流的所有形式；(2)由一个主控程序根据状态转换图对输入流进行扫描分析以识别符合描述的短语。

把这种思想运用到本文所面临的问题上，问题的解决方案就可描述为：(1)把设备的通信协议形式化描述(描述的方式即采用正规式集的方式)；(2)由一个状态图生成程序产生对应这些正规式的状态图(在下文中把状态图称为驱动表、相应的状态图生成程序称为驱动表自动构造器)；(3)由一个唯一的主控程序根据此驱动表对接收到的数据帧进行分析识别，输出用户希望的数据格式。

2.1 总体设计方案

本文提出一种根据用户配置文件自动进行协议数据识别

作者简介：董 立(1981－)，男，硕士研究生，主研方向：编译技术；赵恒永，教授

收稿日期：2006-11-15 **E-mail：**dongli81@126.com

的解决方案，该方案主要由以下部分构成，即用户配置文件、词法分析器自动生成器、语法分析器自动生成器、数据帧识别器，方案结构如图 1 所示。

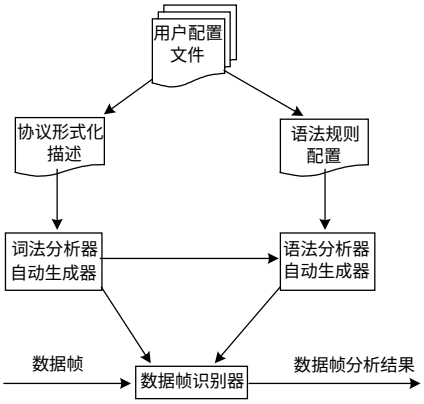

图 1　总体方案结构

程序运行时，首先根据用户配置文件自动生成词法分析器和语法分析器，数据帧识别器把接收到的数据帧进行词法分析，然后进行语法分析，最后得到分析结果。这样做的好处是，一旦底层应用设备协议发生变化，就可以适当更改用户配置文件，而不必把整个程序重新编译。

2.2　用户配置文件设计

用户配置文件是本文方案的基础所在，一个良好的配置文件设计，能够大大简便以后词法分析器和语法分析器的生成。由图 1 可以看出，在设计时采用了 2 个文件单独配置的方法：(1)底层设备协议的形式化描述，该配置文件主要用来自动生成词法分析器；(2)用户定义的语法规则，用来自动生成语法分析器。

协议的形式化描述实际上就是通过分析设备的协议定义，用正规式集的方式表现出来，以便进行词法分析。

一条命令帧或响应帧的基本格式为：

命令头_命令_参数_数据

命令头：Gnn, Ann

G：代表命令

A：代表响应

nn：命令或响应的序号

命令：由两个字母表示的命令或响应的内容

参数：参数表示了数据的来源、个数、大小、类型等

数据：数据值，有的命令和响应没有数据部分

_：代表一个或多个空格

此类消息的形式化描述，即正规式集如下所示：

```
digit          [0-9]
commandtype    (a|g|p){digit}{digit}
interrupt      (p){digit}{digit}
id             [a-z][a-z0-9_ ]*
integer        {digit}{digit}*
exp            [Ee][+-]?{digit}+
float          ({digit}*"."{digit}+({exp})?)|
               ({digit}+"."{digit}*({exp})?)
token          [a-z0-9_]+
command        (si)|(mn)|(sp)|(fg)|(fp)|(fs)| (ms)
                 |(bf)| (bg)|(bd)|(bk)|(pm)|(sc)|(ac)
datatype       (int)|(flt)|(chr)|(der)
bool           (on)|(off)
value          {integer}|{float}|{token}
string
```

语法规则定义是用户配置文件的主要部分，它既可以让用户定义一定的语法规则，还能够让用户决定在符合规则的情况下什么样的数据帧接受，什么样的数据帧抛弃，以及接受的数据帧哪一部分需要存储等一系列动作。

该配置文件首先应该给出词法分析所能识别的协议关键字，这样用户就可以根据这些关键字和协议手册实现自己的配置。这些关键字的形式化描述上面已经给出。另外还应该包括已经提前定义的动作。本文在设计中采用了提前定义动作函数的方法。再就是规定用户配置条目的格式，具体设计包括两部分：

(1)语法规则条

语法规则条主要是用户根据具体需要写出程序应该能够识别的语法规则，该语法规则并不要求用户写出所有同该协议有关的条目，仅需要写出用户需要的规则即可。例如某用户仅需要识别返回数据的响应帧，就可以写成：

commandtype_command_integer_parameter_data

不必将并非返回数据的命令帧 commandtype_ command_ _ _写出。

(2)相应动作项

相应动作项是发现符合用户定义的语法规则的数据帧后应该执行什么样的动作，常用的是存储，一般是数据项的存储，也可能是用户感兴趣的命令类型的存储等。

例如，如果定义一个动作 save(string key)可以把 key 对应的实际数据值存到一个全局变量中，那么用户就可以在该分析器以外通过配置和调用从而获得感兴趣的数据。

相应配置条目可以写成：

commandtype_command_integer_parameter_data {save(data);}

表示发现符合上述语法规则的数据帧就接受，并且存储data数据项到全局变量。

2.3　词法分析器自动生成器

词法分析器的主要目的是读取用户的形式化描述，然后自动生成协议驱动表，即本文所说的词法分析器。词法分析器自动生成器的主要部分就是 3 个主要算法，这 3 个主要算法实现了从正规表达式到最小化确定有限自动机的一步步转换。

最后生成的最小化确定有限自动机，既可以包括所有形式化描述的最小化确定有限自动机，也可以是针对每一个形式化描述的最小化确定有限自动机。本文采取的是后者。相应的数据结构如下：

Sword[]单词数组，包含有 3 个域：Name域代表可识别的单词名称；*States*[]代表该单词状态数组；***Trans***[]是状态转移矩阵，Nstate=***Trans***[inChr] [curState]。

(1)从正规表达式到NFA

设计中用状态转换图来分别表示相应的NFA或DFA，而在内存中的表示，则使用状态矩阵表示，即上面提到的***Trans***[]。在手工方式下，可以手工对每一类单词建立一张状态转换图，然后把所有的状态转换图合并成一张统一的状态图。在程序自动生成中，所有状态以及状态转换都需要保存，以便后继处理的使用。

(2)从NFA到DFA

从不确定的有限自动机*N*转换为一个等价的确定有限自动机*M*，可以用子集构造法完成[3]。*M*的每个状态，对应于N

的一个状态的集合。其基本思想是：*M*读入一个给定的输入串之后处于状态{*x*, *y*, *z*}，当且仅当*N*可能处在*x*, *y*, *z*中任何一个状态，视*N*选定哪个状态而定。让*M*记住*N*可能走的全部可能路线，并让*M*同*N*平行地工作。*M*的接收状态将是任何含有*N*的接收状态的任何集合。

(3)DFA的最小确定化

经过上述建立的DFA并不是最优的，其状态数比必要的状态数可能要多，因此需要对这个DFA进行最小化，其基本思想是：在一个状态集合中，还有不等价状态，就把不等价的状态分开，使一个集合中的状态等价。分开后，如果一个集合中的状态还有不等价的，就继续分开，直到每个分开的小集合中的状态都等价为止。

2.4 语法分析器自动生成器

有了以上的词法分析器自动生成器，就可以生成能够识别协议形式化描述中的所定义的单词。而这些单词，正是本文在用户语法规则条中所使用的。利用用户所定义的语法规则，生成相应的语法树。

由于该语法分析器是通过读取用户配置文件生成，因此首先要有一个小的词法分析器，能够识别文件中的单词，该词法分析器是按照文件中配置条的特殊字符进行分割，然后对比形式化描述文件中的定义，识别特定的和已经定义的单词符号。语法分析树自动生成器如图 2 所示。

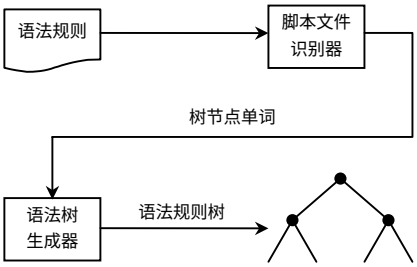

图 2　语法分析树自动生成器

(1)脚本文件识别器

脚本文件识别器主要是识别配置文件中所定义的节点关键词，如 command, integer 等，即在语法配置文件中所出现的 *Sword*[]中的 name 域。该识别器也是一个词法分析器，只不过，它所要进行的动作是按照特定的分割字符，识别关键词。例如配置条目

commandtype_command_integer_parameter_data {save(data);}

中所出现的 commandtype, command, integer, parameter, data 等。

首先是把形式化描述中的名称定义为关键字，然后通过词法分析的方法，识别语法配置脚本中的关键词，并且记下识别的顺序。

(2)语法树生成

考虑到本文所要进行的语法分析，是一种比较简单的对比分析，只要符合配置条目中所出现的语法配置条，就可以执行其后的动作。因为在设计中把配置条目的动作也作为一个单独的节点挂在语法树叶子节点的。这样一旦语法分析走到叶子节点，就可以认为是语法匹配，再进一步走到动作节点，去执行相应的语法动作。

2.5 数据帧识别器

当词法分析器和语法分析器都建立起来以后，就可以对数据帧进行分析识别。数据帧识别器接收数据帧，然后调用词法分析器识别单词，然后根据识别出的单词的顺序去查语法树，最后执行能够识别的语法的对应动作。

当系统取得所要识别的数据帧后，首先把它转换成字符串，然后存到字符缓冲区；词法分析器从缓冲区的头部开始进行状态匹配，如果出现不匹配的情况要进行字符回退，词法分析的结果是把所有的单词识别出来并按顺序存储。随后这些按顺序存储的单词查找语法树，并执行能够匹配的语法树的动作。其结构如图 3 所示。

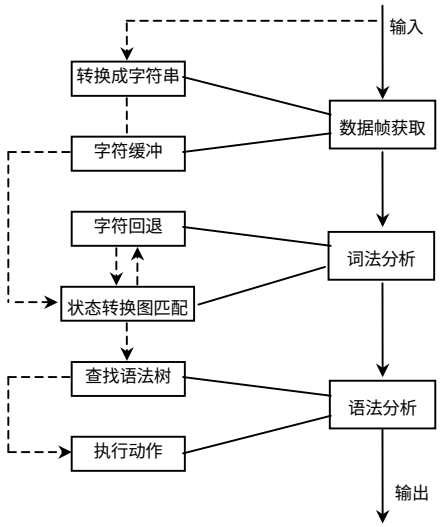

图 3　数据帧识别器结构

3　结束语

该系统如果底层协议发生变化，只需要在形式化描述文件中加入新的协议形式化描述，在用户语法配置文件中加入新的语法配置条。程序启动时，会自动生成相应的词法分析器和语法分析器，此时的词法分析器和语法分析器已经是不同于先前而是适合于现在底层协议的分析器，协议识别主程序就会调用该新生成的词法分析器和语法分析器分别进行词法分析和语法分析，从而进行相应的数据提取。此过程完全避免了应用程序的重新编译，而且，一般用户经过简单培训，也可以方便地对配置文件进行手工配置，解放了开发人员对整个应用程序的维护[4]。

本文所介绍的词法分析和语法分析的自动生成方法，在目前基于网络的应用程序开发中具有很大的意义(如可用于协议分析和协议转换等)。基于状态转换图的模式匹配方法具有一般匹配无可比拟的效率，可用于基于规则的网络入侵监测中等。

参考文献

1 Aho A, Sethi R, Ullman J. Compilers Principles, Techniques, and Tools[M]. MA: Addison-Wesley, 1986.

2 吕映芝, 张素琴, 蒋维杜. 编译原理[M]. 北京: 清华人学出版社, 1998.

3 霍普克罗夫特 J E, 厄尔曼 J D. 形式语言与自动机的关系[M]. 徐美瑞, 译. 北京: 科学出版社, 1979.

4 陈火旺, 钱家骅, 孙永强. 编译原理[M]. 2 版. 北京: 国防工业出版社, 1984.