C++指针机制与源文件关联关系的可视化研究①

古 辉，乔凯旋

(浙江工业大学 计算机科学与技术学院，杭州 310023)

摘 要：研究了 C++中的指针机制、以及指针类型对象(变量)在多个源程序代码文件中关联关系。基于信息提取和结果整理，计算机可视化实现和表示 C++中的指针机制和多源程序代码文件的关联关系。研究了抽取结果的存储机制和基于该机制的自动排序源文件引用关系的方法，最后提出了一种手工调整图元布局的算法，作为自动排序算法的补充。对实际代码分析的结果表明该方法利于程序分析并支持对源代码的辅助理解。

关键词：组件；图元；层次化；引用关系；可视化

Pointer Mechanism and Source Files Association's Visualization in C++

GU Hui, QIAO Kai-Xuan

(College of Computer Science & Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract: Source code as object for research in this paper, mechanism of pointer and relation of association about pointer type object (variable) within multiple source files is studied. Based on the extraction of message and packing up the results, automatically visualize the elation about source files and mechanism of pointer in C plusplus. The storage mechanism on extraction and an algorithm of automatically disposing the include-relation on files are proposed. As a supplement, an algorithm of manual adjustment for meta-graph is introduced. The actual analysis results on codes show that the research is conducive to support for program analysis and understanding of source code.

Key words: component; metafile; stratifying; include-relation; visualization

1 概述

面向对象程序设计过程中，类通常被封装在组件中，通过引用组件名，在当前作用域中就可以使用组件的方法。组件既能提高代码的复用，也有利于程序结构的清晰化[1]。

本文所述的组件关联关系图是在指针机制和信息流可视化研究的过程中提出的，是相对于程序控制流程图的一种更粗粒度的程序结构图形表示。它有助于在更高层次上观察程序的总体结构，对程序的进一步分析有很大的帮助，特别是在基于格局识别[2]的程序分析时，可以辅助目标信息流从开始到结束的可持续性过程的建立。

下面将叙述关联关系可视化过程中的信息提取、存储以及层次化算法，最后介绍手工调整图元布局的算法，结合自动排序，可以得到较为平衡的视图。

2 文件引用信息抽取与存储

C/C++源文件的引用是通过 include 语句实现的。源文件分为两种：用户自定义源文件、库文件。库文件都有完善的文档说明，本文在处理引用关系时会自动屏蔽对系统库文件的处理，以使得结果表示简洁。

2.1 文件引用关系格式解析

用户自定义源文件（以下简称用户文件）的引用格式是：

#include “filename.h”

库文件的引用格式是：

#include <filename>

编译器编译源文件时，预先定义了搜索源文件的

① 收稿时间:2011-11-13;收到修改稿时间:2011-12-02

顺序。用引号括起的源文件，先从用户的工作路径搜索，如果搜索失败，转到标准库路径继续搜索；如果仍未搜索到，则编译器会提示“No such file or directory”。用一对尖括号<>标起的源文件，编译器从标准库路径开始搜索，如果未搜索到，同样会提示“No such file or directory”[3]。

依赖双引号和尖括号还不足以判断引用文件的类型，还需要结合源文件的位置，才能得到正确的结果。标准库的位置与注册表中记录的编译器程序的安装路径相同。通过 CRegKey 类的成员函数 Open 得到安装路径，再经过字符子串替换，就可以得到标准库的路径[4]。

2.2 文件引用关系存储结构设计

上节中分析了文件引用关系的格式，从源文件中搜索包含的#include 语句并将结果存储起来，我们使用下面的数据结构存储源文件之间的引用关系：

```
struct NodeFile{
   string filename_h;
   string filename_cpp;
   string filepath;
   vector<string>     vec_ud_hfile;
   vector<string>     vec_ud_cppfile;
   vector<string>     vec_stl_hfile;
   vector<string>     vec_stl_cppfile;
};
```

结构解释：一个 NodeFile 代表一个组件，其成员包括头文件名、实现文件名、组件存放路径以及头和实现文件中分别包含的库文件和用户文件[1]。为区别存储在头文件和实现文件中的用户文件和库文件，NodeFile 定义了四个数组，分别存放这些引用关系[5]。

2.3 异常情况分析

在应用 NodeFile 结构存储源文件的引用关系时会出现一些异常，如.cpp 文件不存在，#ifndef…#define…#else…等预编译指令的使用引起的引用关系的二义性[3]。因此，还需要分析与#include 语句关联的预编译指令，确保文件引用关系的准确性。

遇到.cpp 文件不存在的情况时，如果仍就读取该文件会导致文件打开异常，使程序崩溃。此种情况可以通过引入文件存在性验证机制予以解决，基本算法如下：

(1) 提取文件路径 path 与文件名 filename，组合 path 与 filename，得到文件的实际存储位置字符串 posstring。

(2) 搜索 posstring 指向的文件是否存在；

(2.1)若存在，继续读取文件内容；

(2.2)否则，提示文件不存在的位置，返回。

在工作路径和标准库路径都搜索不到指定文件时，可以断定引用了错误的源文件或者源文件已经不存在，这时需要删除或修改相应的引用语句。

为了避免源文件的重复编译，每个源文件通常都用一个守卫卫哨[3]包围源程序代码，其格式为：

```
#if !defined 宏标志
#define 宏标志
//源程序代码
#endif[3]
```

编译器记录源文件的宏标志，避免重复编译同一个源文件。本文以源文件路径和名称的组合作为索引，能够保证源文件集合的唯一性，即不会重复分析同一文件，也不会遗漏不同文件夹下的同名源文件。因此，引用关系语句能够被安全的分析。

3 文件引用关系的可视化和调整算法

选择一个用户文件，抽取其包含的#include 语句，得到用户文件引用关系，递归解析 include 引用的源文件，可以产生抽象的用户文件引用关系表，存储结构参见 2.2 节。每个源文件节点都存储其所引用的源文件名。

我们使用下面的图 1 表示文件图元[6]：

图 1 文件图元基本形状

3.1 文件引用关系建立及层次化

文件引用关系可视化表示需要先在源文件之间建立起连接，通过当前组件能直接索引到目标源文件。封装 NodeFile 结构，增加新成员，可以解决连接过程中遇到的问题。用新名词 CFileUnit 表示新的封装结构，下面仅列出结构部分属性：

```
class CFileUnit{
   NodeFile     m_nf;
```

```
    CPoint  m_pos;
    CSize   m_size;
    vector <CFileUnit*> next;
    int m_numfile;
};
```

每一个 NodeFile 结构都对应一个 CFileUnit 对象。被引用的源文件，把它对应的 CFileUnit 对象地址存放在 CFileUnit::next 中，即减小对象的体积，也方便了文件查找。

源文件的基本信息存储在类型为 vector <CFileUnit>的数组 vec_CFU 中，源文件之间通过持有引用文件在数组中的地址，表示文件之间的引用关系。基本步骤描述如下：

(1) 按顺序遍历源文件数组 vec_CFU，取得当前源文件对象为 CFileUnit *cfu=vec_CFU[loop], loop=0~vec_CFU.size();

(2) 提取 cfu 成员 m_nf 中 vec_ud_hfile，按顺序遍历数组的每个元素 elem；

(2.1) 取得 elem 的值，按顺序与源文件数组 vec_CFU[loop] 的文件名比较，若匹配成功则将 vec_CFU[loop]的地址存入 cfu 的成员 next 数组中，取 next 的下一个元素；否则直接取下一个元素；

(2.2) 匹配完 vec_ud_hfile 的元素转步骤 3。

(3) 提取 cfu 的成员 m_nf 中 vec_ud_cppfile，按顺序遍历数组的元素 elem；

(3.1) 取得 elem 的值，按顺序与源文件数组 vec_CFU[loop] 的文件名比较，若成功则将 vec_CFU[loop]的地址存入 cfu 的成员 next 数组中，取 next 的下一个元素；否则直接取下一个元素；

(3.2) 匹配完 vec_ud_cppfile 的元素，转步骤 4。

(4) 结束

源文件之间的引用关系建立后，可视化过程还需要对引用关系进行层次化和再排序，以免文件之间的引用关系交错而影响视图效果。

文件关联关系的层次化必须建立在文件引用关系基础之上，以第一个输入的源文件作为顶层，遍历它的成员 next，其存储的节点作为第二层，层次化后的结果保存到动态数组 vector<vector <CFileUnit*> >m_Stratify；再以 m_Stratify 的最后一层作为新的开始，递归搜索下一层节点。详细的步骤描述如下：

(1) 以 vec_CFU 的第一个节点初始化 m_Stratify。以 m_Stratify[lastline]表示最后一层；临时层次节点集合为 vector<CFileUnit*> line_cfu;

(2) 取 m_Stratify 的最后一层节点集 m_Stratify [lastline]，如果不为空，转步骤 3，否则结束。

(3)顺序遍历 m_Stratify[lastline]的节点 CFileUnit *cfu_t= m_Stratify[lastline][loop];

(3.1) cfu_t 的成员 next 如果不为空，按顺序遍历 next 的元素 elem；判断该元素是否已经在层次化数组 m_Stratify 中；若已经存在，则继续处理下一个元素 elem； 否则，将当前元素的地址存入 line_cfu，继续处理下一个元素 elem；

(3.2) next 元素处理完毕，结束当前循环，否则处理下一个元素 elem。

(4) 循环处理完 m_Stratify[lastline]后，得到新的节点集合 line_cfu;判断 line_cfu 是否为空；

(4.1) 若是，将 line_cfu 存入 m_Stratify，并清空 line_cfu，转步骤 2；

(4.2) 否则结束。

3.2 文件引用关系的可视化及其实现

源文件引用关系在层次化处理后，还需要计算每一源文件图元的大小，为各个层中的节点分配坐标。源文件图元的大小由多种因素决定，本文统计引用的文件数且限制图元的最大宽度为 105pixel，图元的大小存储在 m_size(参见 3.1 的 CFileUnit 结构)中。由于层次化后的关系表中存储的是指针链接，可以通过指针直接把坐标存储到实际对象的成员 m_pos(参见 3.1 的 CFileUnit 结构)。先初始化第一个源文件的起始坐标 (X,Y)，详细的实现算法描述如下：

(1) 按行处理引用关系表 m_Stratify，从首行开始；

(2) 提取前一个已分配坐标的节点位置 pos(x,y)和大小 size(cx,cy)；下一个节点的坐标 pos_nex- t(x,y)=pos(x+cx+distance_node,y)；把坐标 pos_next 存储到当前节点的 m_pos；继续计算下一个节点直到处理完本层中的所有节点。

(3) 转下一行，计算第一个节点的坐标 pos(X,Ylast +Max(cy)+distance_line)：转步骤(2) 继续计算本层后续节点的坐标。

(4) m_Stratify 的所有行处理完毕，结束。

算法解释：步骤(2)中 distance_node 表示同层之中节点之间的间隔，步骤(3)中 distance_line 表示相邻层之间的间隔，两者都定义为一个常数。Ylast 表示已分

配坐标的最后一行的 y 轴坐标。Max(cy)表示上一行中所有图元高度的最大值。

坐标分配完成之后，文件在物理视图上的相对关系已经确定。在结构 CFileUnit 中封装了图元实现方法 DrawUnit；逐个调用图元对象的 DrawUnit 方法便可以在视图上显示出源文件的引用关系。引用关系图与选择的第一个源文件有关，改变初始的源文件使分析得到的引用关系表产生变化，最后输出的图形也不一样。本文分析一个 MFC 框架下的端口扫描程序，选择两个不同的初始源文件，分别得到图 2 和图 3。

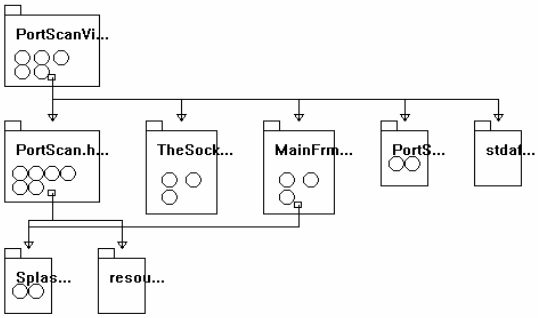

图 2 初始分析目标为视图(View)组件

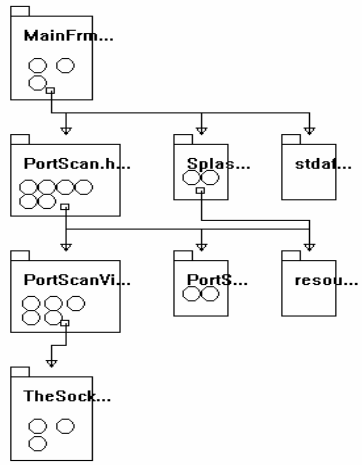

图 3 初始分析目标为框架(Frame)组件

3.3 手工调整图元布局算法

图元在视图中的位置唯一并且每个图元都具有自己的范围，通过点选择并移动图元，为图元排列和调整提供了一种辅助方法。

图元位置与视图上的点一一对应，每个图元依据自己的位置和大小，通过视图与鼠标[7]，直接调整视图上图元的位置。在源文件的引用关系比较复杂时，能够辅助布图算法，保证视图的平衡性，对处理孤立点极其有效。

手工调整图元布局算法：

输入：鼠标指针位置 P

输出：鼠标位置 P’，位于 P’ 点的图元

(1) 移动光标到目标图元，位置为 P；

(2) 选择移动图元指令，移动光标到期望位置 P’；

(3) 图元判断 P 是否在自己的范围内。

(4) 否，则继续下个图元，转步骤 3；

(5) 是，则修改当前图元的位置 m_pos 的值为 P’，结束移动图元指令。

(6) 刷新视图。

通过对图 2 进行手工操作，得到图 4 中较为平衡的视图。

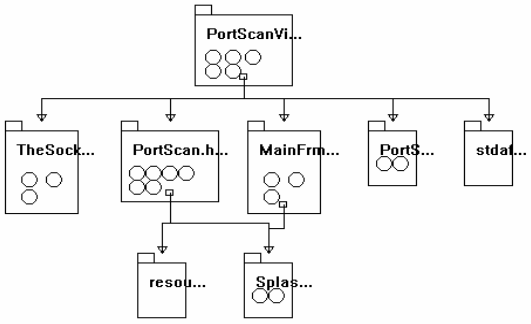

图 4 调整图 1 后的视图

源文件引用关系的可视化需经过信息提取与存储、引用关系建立、层次化、坐标分配、绘制等子过程，得到基本的关系图，在此基础上，根据文件图元与坐标的一一对应关系，手工调整图元布局，使得结果视图的表示自然协调。

4 结语

指针信息的抽取过程需要分析程序的上下文，完整的上下文关系[8]需分析源文件之间的引用关系。良好的源文件引用关系组织和可视化表示方法有助于指针流分析和可视化，本文介绍的可视化过程中图元的组织结构、层次化算法和调整图元的算法，经过适当的修改，亦可以运用于程序理解[9]可视化的其它方面。

参考文献

1 Lakos J. Large Scale C++ Software Design.北京:中国电力出版社,2003.77–84.

(下转第 253 页)

算法的能量消耗均衡性网络能量消耗均衡性是衡量分簇算法性能的重要指标,具体表现在每轮的耗能及网络死亡期方面。显然,网络每轮消耗的能量越小越均匀,网络死亡期越短,网络能量消耗均衡性越好[4]。

协议中采用适用于能量异构情况的簇头选举机制同时采用了合理的簇间通信方式机制，使得平均能量较低的簇不再转发和传送数据包而是仅仅收集数据，这样就有效地均衡网络的整体消耗。改进后的协议始终选择能量高的节点作为簇头和簇间通信转发节点，将簇头剩余能量和簇头转换相关联，从而能很好地均衡网络节点能耗。

5 结语

本算法主要从簇头选择、簇间传输路径两个方面对分簇路由算法进行分析与改进。簇头选择中，剩余能量较高的节点竞争为候选簇头，簇间通信过程中，运用单跳/多跳的数据传输模式，基于通信代价和簇内剩余平均能量选择簇间数据转发模式，有效均衡网络负载。通过分析和仿真证明，算法在网络生存周期、负载均衡等多方面都显示了很好的性能。下一步计划研究路由协议的容错能力和具体的数据融合算法。

参考文献

1 Heinzelman W, Chandrakasan A, Balakrishnan H. An application specific protocolarchitecture for wireless microsensor networks. IEEE Trans. on Wireless Communications, 2002,1(4):660−670.

2 ManjeshwarA, GrawalD P. TEEN: a protocol for enhanced efficiency in wireless sensornetworks. Proc. of the 15th Parallel and Distributed Processing Symp. San Francisco: IEEE Computer Society, 2001,2009−2015.

3 Younis O, Fahmy S. Heed: a hybrid, energy-efficien,t distributed clustering approach for ad-hoc sensor networks. IEEE Trans. on Mobile Computing, 2004,3(4):660−669.

4 刘新华,李方敏,旷海兰,等.基于能量异构的无线传感器网络分布式成簇算法.小型微型计算机系统,2010,1:26−31.

5 w'Chandrakasan HA, Balakrishnan H. An application specific protocol architecture for wireless microsensor networks. IEEE Trans. on Wireless Communications, 2002,l(4):660−670.

6 张强,卢潇,崔晓臣.基于能量高效的无线传感器网络 LEACH 协议改进.计算机工程与设计,2011,32(2):427−429.

7 周钰川,施荣华,周媛媛.WSN 中基于非匀簇的混合多跳路由协议.计算机应用研究,2011,28(2):642−644.

(上接第 239 页)

2 胡磊.基于模板的可视化软件理解研究与实现.武汉:武汉大学,2004.

3 Harbison III SP, Steele Jr GL．C 语言参考手册.北京:机械工业出版社,2003.40−45.

4 CRegKey.[2011-8-31].http://msdn.microsoft.com/zh-cn/library/xka57 xy4 (v=VS.80).aspx..

5 Mayers.STL 高效编程.北京:机械工业出版社,2006.64−80.

6 Booch G, Rumbaug J, Jacobson I.UML 用户指南.北京:机械工业出版社,2006.164−170.

7 Petzold C.方敏等译. windows 程序设计.北京:清华大学出版社,2010.240−245.

8 Hind M, Pioli A. Which Pointer Analysis Should I Use. ACM SIG-SOFT Software Engineering Notes. 2000, 25(5):113−11.

9 李莹,张琴燕.程序理解.计算机应用研究,2001,18(6):40−41.