MRO 中多重 BOM 结构遍历及显示的算法①

叶水生[1], 祝中良[2], 户伟利[3]

[1](南昌航空大学 无损检测技术教育部重点实验室, 南昌 330063)
[2](南昌航空大学 软件学院, 南昌 330063)
[3](南昌航空大学 信息工程学院, 南昌 330063)

摘 要: 为优化 MRO 系统的性能, 提高系统的运行效率, 提出了两种物料清单(BOM)构造的方法, 并指出了两种方法的优缺点, 将 BOM 结构的存储方法由三层扩展到多层, 并使用 Java 语言和 Swing 库对 BOM 结构进行了遍历和显示. 实际应用表明, 文章中提出的 BOM 结构遍历和显示算法能有效提高 MRO 系统的运行效率.

关键词: 设备运行与维护; 物料清单; 遍历; 显示; 算法

Multilayer BOM Structure Ergodicity of MRO and the Display Algorithm

YE Shui-Sheng[1], ZHU Zhong-Liang[2], HU Wei-Li[3]

[1](The Ministry of Education Key Laboratory of Nondestructive Testing Technology, Nanchang Hangkong University, Nanchang 330063, China)
[2](School of Software, Nanchang Hangkong University, Nanchang 330063, China)
[3](School of Information Engineering, Nanchang Hangkong University, Nanchang 330063, China)

Abstract: In order to optimize the performance of MRO system and improve the operating efficiency of this system, two formation methods of BOM have been put forward. In addition, advantages and disadvantages of these two methods are pointed out. The storage means of BOM structure is extended from three layers to multi-layers, and Java language as well as Swing library are applied to go through and display this BOM structure. The practical application indicates that the BOM structure ergodicity and the display algorithm mentioned in this paper can effectively improve the operating efficiency of MRO system.

Key words: MRO; BOM; ergodic; display; algorithm

1 引言

设备运行与维护(Maintenance, Repair & Operations)是指在实际的生产过程不直接构成产品, 只用于维护、维修、运行设备的物料和服务. MRO 是指非生产原料性质的工业用品. 随着工业生产领域设备的日益复杂化, 机械制造业内许多企业已经或打算实施 MRO 系统, 以增强自己在国际化竞争中的应对能力[1]. MRO 系统中的物料清单(Bills of Material, BOM), 与设备运行的各个状态紧密相关, 是 MRO 系统运行的基础. 因此, BOM 的变化与维护将对生产产生直接的影响. BOM 结构遍历和显示的算法设计构成了 MRO 系统的核心. 为优化 MRO 系统的性能, 提高系统运行效率, 笔者研究了 BOM 的遍历和显示算法, 这对 MRO 系统的开发与设计有着重要意义.

2 BOM结构的构造

2.1 BOM 表的构造

物料清单(BOM), 又称为产品结构表或产品结构树, 是一个描述产品结构的技术文件, 是系统中最基本的资料, 是设备运行与维护(MRO)系统中的一个核心部件, MRO 系统中所有设备的运行状况都将通过 BOM 建立逻辑上的关联关系[2].

在 MRO 系统中, BOM 采用有向的单根树结构集合形式表示. 一个多层 BOM 结构, 由多个如图 1 所

① 收稿时间:2013-08-09;收到修改稿时间:2013-09-09

示的单根树结构组成. 其中树根部分 A 表示父组件, 叶子部分表示该父组件的各个子组件. 父组件 A 也可以是其他组件的子组件. 处于顶层的根组件(也就是设备本身)没有父组件[3].

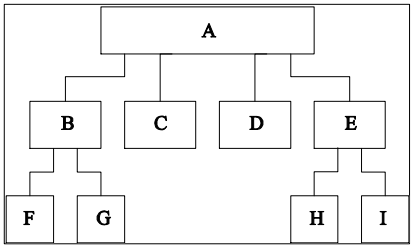

图 1　BOM 中的一个单根树结构

为简化问题, 本文采用归纳法来得出多层 BOM 结构存储中 BOM 表的构造方法. 文章从一个特殊的例子, 即三层BOM结构存储中BOM表的构造方法推导出多层BOM 结构存储中构造 BOM 表的一般性方法.

2.1.1 三层 BOM 结构存储的 BOM 表构造

三层 BOM 结构采用有向单根树的数据结构, 它详细的记录了产品的结构信息. 各个零部件在不同产品中的不同位置以组件 ID 来区别. 即便是同样的零部件结构, 只要是在不同的产品中, 也要重新记录一次. 以图 1 中产品 A 为例, 其 BOM 的结构表达如表 1 和表 2 所示[4].

表 1　产品 A 的组件表

组件 ID	组件名称
A0000001	A
B0000001	B
B0000002	C
B0000003	D
B0000004	E
C0000001	F
C0000002	G
C0000003	H
C0000004	I

表 2　产品 A 的组件关系表

父组件 ID	子组件 ID
A0000001	B0000001
A0000001	B0000002
A0000001	B0000003
A0000001	B0000004
B0000001	C0000001
B0000001	C0000002
B0000004	C0000003
B0000004	C0000004

以上三层 BOM 结构存储的特点是产品间结构不互相影响, 各个产品之间的数据记录没有交叉, 因此维护比较方便. 而且采用以 A 开头的字母加组件编号表示底层的根组件 ID; 以 B 开头的字母加组件编号表示第二层组件的 ID; 以 C 开头的字母加组件编号表示第三层组件的 ID[5]. 这样查询起来比较方便. 但这种存储结构也存在缺点, 例如, 当数据库中存在多个设备时会出现多个根组件, 这时这个存储结构很难快速确定当前子组件是属于哪个根组件.

3　多层BOM结构存储的BOM表构造

多层 BOM 结构存储的 BOM 表构造方法在三层 BOM 结构的基础上做了改进, 其主要的改进是加入了根组件 ID 和当前组件层级数字段, 通过根组件 ID 字段可以快速定位当前子组件所属的根组件对象, 通过层级数字段则可定位当前子组件所处的层级数[6]. 通过加入这两个字段, 可以有效加快 BOM 结构遍历的速度. 以图 1 中产品 A 为例, 改进后的 BOM 的结构表达如表 3 和表 4 所示.

表 3　产品 A 的组件表

组件 ID	组件名称	根组件 ID
A0000001	A	A0000001
B0000001	B	A0000001
B0000002	C	A0000001
B0000003	D	A0000001
B0000004	E	A0000001
C0000001	F	A0000001
C0000002	G	A0000001
C0000003	H	A0000001
C0000004	I	A0000001

表 4　产品 A 的组件关系表

ID	父组件 ID	子组件 ID	根组件 ID	层级数
1	A0000001	A0000001	A0000001	1
2	A0000001	B0000001	A0000001	2
3	A0000001	B0000002	A0000001	2
4	A0000001	B0000003	A0000001	2
5	A0000001	B0000004	A0000001	2
6	B0000001	C0000001	A0000001	3
7	B0000001	C0000002	A0000001	3
8	B0000004	C0000003	A0000001	3
9	B0000004	C0000004	A0000001	3

在构造多层 BOM 结构存储的数据表时, 有两个关键性的原则, 一是要清晰的定义产品的结构树, 二

是要能够高效的进行分解. 表 3 和表 4 正是满足上述两个原则的构造. 通过加入根组件 ID 字段可以快速定位当前组件所属的 BOM 树结构, 通过加入层级数字段则可以快速获取当前组件所属的层级数[7]. 通过以上两个字段的加入, 可以加速对 BOM 表的遍历. 在对多层 BOM 树结构的 BOM 表进行构造时, 虽然数据项增多, 产生了一定的数据冗余, 但是此种构造方法可以满足 MRO 系统对于 BOM 结构的两个关键性原则. 因此, 这是一种实用性较强的构造方法.

4 BOM结构的遍历及显示算法

本文中所提到的 BOM 结构遍历及显示算法都是使用 Java 语言基于 Swing 库实现的. 为了实现 BOM 结构在用户界面上的显示功能, 需要使用 Swing 库中的 JTree 控件.

4.1 JTree 控件简介

Swing 是一个用于开发 Java 应用程序用户界面的开发工具包. 它以抽象窗口工具包(AWT)为基础使跨平台应用程序可以使用任何可插拔的外观风格. Swing 开发人员只用很少的代码就可以利用 Swing 丰富、灵活的功能和模块化组件来创建优雅的用户界面.

JTree 是 Swing 库中一种能够显示树状结构信息的类. 它使用一个 TreeModel 类来管理当前所创建的树结构模型, 使用 TreeNode 类来管理树结构中的树节点, 使用 TreePath 类来管理树节点在当前树结构中的路径[8].

使用 JTree 类结合内存中构建的内存树结构可以方便的实现 BOM 结构的遍历和显示.

4.2 三层 BOM 结构的遍历及显示算法

三层 BOM 结构遍历和显示时需要在内存中构建一个与 BOM 结构相对应的内存树结构, 假设存在如图 2 所示的三层 BOM 结构. 则构造的内存树结构形式以 Java class 的形式表示为如下所示[9].

```
class BomStruct{
        Bom    root;
        Vector<Bom>      middleBoms;
        Vector<Vector<Map<string,Bom>>>
lastBoms;
}
```

其中变量 root 保存了根组件泵车的信息.

变量 middleBoms 采用了 java.util 包中的向量容器 Vector 来存储第二层的 Bom 对象, 具体包括底盘、支腿和臂架三个组件; 选用向量容器的原因是在对 BOM 结构进行存储和显示的过程中, 会经常从容器中取出数据和设置容器中某项的数据, 向量作为一种支持随机访问迭代器的容器, 效率较高[10].

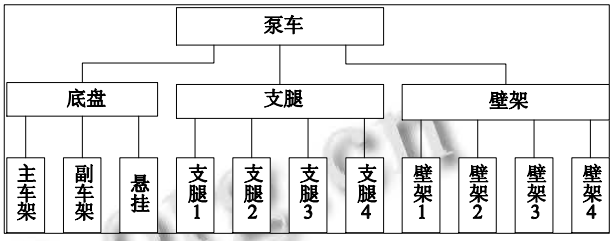

图 2 三层 BOM 结构图

变量 lastBoms 使用嵌套容器(也称容器的容器)来存储第三层的 BOM 对象, 其中内层容器中的第一个向量存储的是主车架、副车架和悬挂组件; 第二个向量存储的是支腿 1、支腿 2、支腿 3 和支腿 4 组件; 第三个向量则存储臂架 1、臂架 2、臂架 3 和臂架 4 组件. 外层的向量容器则用于存储这三个向量. 最里层使用映射容器 Map 来存储组件对象是因为每个三层组件对象都有其对应的父组件信息, 而 Map 中的键值则用来存储父组件的 ID, Map 中的数值则用来存储第三层 BOM 对象.

根据表 1 和表 2 所示的数据库表结构, 三层 BOM 结构遍历及显示的步骤如下:

从数据库组件表和组件关系表中查找出相应组件和组件关系信息.

根据 a)中查找出的组件和组件关系信息, 构造一个 BomStruct 对象. 将组件的相关信息填充到 BomStruct 对象中.

根据 b)中构造的 BomStruct 对象的相关信息, 在 java Swing 界面中使用 JTree 对象构造并显示 BOM 结构树.

在三层 BOM 结构树遍历和显示时, 由与 BOM 结构的层级数较少, 使用 BomStruct 对象能够完成 BOM 结构的遍历和显示. 而且由于 BOM 结构只有三层, 在不存储根组件的情况下仍能够通过遍历快速找到该根组件所对应的所有子组件.

4.3 多层 BOM 结构的遍历及显示算法

多层 BOM 结构遍历和显示时也需要在内存中构建内存结构树, 但与三层 BOM 结构的树构建方法不同的

是, 由于此时层级数未知, 内存树结构的构造必须使用递归的方法来实现. 如果仍然使用表 1 和表 2 的 BOM 结构存储方式, 查找某个子组件对应的根组件将会非常耗时, 所以此时采用表 3 和表 4 的 BOM 结构存储方式. 表 3 和表 4 的设计能在很大程度上提高多层 BOM 结构的存取效率. 此时虽然相对表 1 和表 2 的存储方式来说, 增加了根组件 ID 字段和层级数字段, 增加了数据库的冗余, 但是对于 MRO 这种大型系统来说, 一般采用 Oracle 和 SQL Server 大型数据库服务器, 其存储和处理数据的能力一般都能满足企业级系统的需要[11].

多层 BOM 结构遍历和显示时构造的内存树结构形式以 java class 的形式表示为如下所示.

```
class MyTreeNode<T>{
        public    T   t;
        private   MyTreeNode<T>    parent;
        public    Vector<MyTreeNode<T>>        vec
Node;
}
class   MyTree{
        public    MyTreeNode<T>   root;
        public    MyTree();
        public    void     addNode();
        public    MyTreeNode<T>     search();
}
```

其中 MyTreeNode<T>类是内存树的某一个树节点, 它是一个泛型类, 拥有一个类型参数 T. 将树节点类定义为泛型类是为了使 MyTree 类能够适用于各种编程环境. 其中树节点类的 t 变量用于保存当前的子节点数值. parent 变量用于保存父节点信息. vecNode 变量是一个向量容器, 它用来保存当前节点所拥有的所有子节点信息[12].

MyTree 类表示含有 MyTreeNode<T>类型节点的树对象. 其中 root 表示根节点. MyTree()是该类的默认构造函数. addNode()是一个向树中添加树节点的方法. Search()是一个在当前树中查找树节点的方法.

根据表 3 和表 4 所示的数据库表结构, 多层 BOM 结构遍历及显示的步骤如下:

根据根节点 ID(以下简称 ID)从数据库中查找出根节点, 初始化一个 MyTreeNode<T>对象用于存储根节点, 初始化一个 MyTree 对象, 将根节点设置到 MyTree 对象中.

查询组件关系表中根组件 ID 为 ID 的所有 BOM 的最大层级数(假设此最大层级数为 N), 用于遍历和添加组件到 MyTree 对象中.

查询数据库中层级数为 2 的所有组件, 使用 MyTree 对象中的 search 方法查找到根节点, 使用 addNode 方法添加所有二层子组件到 MyTree 对象中.

查询数据库中层级数为 3 的所有组件, 并使用 MyTree 对象中的 serach 方法查找到其父组件, 使用 addNode 方法将 3 层子组件添加到 MyTree 对象中.

按照 d)中的方法, 依次添加第 4, 5, 6 至 N 级子组件到 MyTree 对象中.

根据 e)中生成的 MyTree 对象, 在界面上使用 JTree 对象显示 BOM 结构树.

5 实验结果

使用多层 BOM 结构遍历及显示算法定制的多层 BOM 结构数据表如图 3 和图 4 所示. 图中使用的是 MySql 数据库管理系统. 图 3 显示的是组件表中的组件信息. 图 4 显示的是组件关系表中的组件关系信息.

id	name	rootid
A0000001	泵车	A0000001
B0000001	臂架系统	A0000001
B0000002	转塔系统	A0000001
B0000003	泵送系统	A0000001
B0000004	泵车底架	A0000001
B0000005	泵车底盘	A0000001
B0000006	液压管系	A0000001
B0000007	电气系统	A0000001
C0000001	1#臂架油缸	A0000001
C0000002	1#臂架	A0000001
C0000003	2#臂架油缸	A0000001

图 3 BOM 组件表图

	mainid	subid	rootid	level
1	A0000001	A0000001	A0000001	1
2	A0000001	B0000001	A0000001	2
3	A0000001	B0000002	A0000001	2
4	A0000001	B0000003	A0000001	2
5	A0000001	B0000004	A0000001	2
6	A0000001	B0000005	A0000001	2
7	A0000001	B0000006	A0000001	2
8	A0000001	B0000007	A0000001	2
9	B0000001	C0000001	A0000001	3
10	B0000001	C0000002	A0000001	3
11	B0000001	C0000003	A0000001	3
12	B0000001	C0000004	A0000001	3
13	B0000001	C0000005	A0000001	3

图 4 组件关系表

根据组件表和组件关系表中的 BOM 结构信息, 采用 java Swing 技术显示的多层 BOM 结构如图 5 所示.

图 5 使用 Swing 显示的 BOM 结构示意图

6 结语

本文采用归纳法分析和比较了两种不同的 BOM 结构遍历和显示算法. 其中三层 BOM 结构遍历和显示算法由于其层数较少, 查询速度也较快, 所以在组件关系表中只存储了父组件 ID 和子组件 ID 字段, 没有存储根组件 ID 字段, 数据冗余较少. 但其也存在明显的缺陷---只能处理三层或三层以下的 BOM 结构. 为了扩展算法, 让其支持多层 BOM 结构的高效准确查询, 本文提出了多层 BOM 结构的遍历和显示算法. 多层 BOM 结构的遍历和显示算法弥补了只能显示三层 BOM 结构的缺陷, 但由于加入了根组件 ID 字段, 也带来了一定的数据冗余. 但这些数据冗余都在系统应用许可的范围内. 多层 BOM 结构的遍历和显示算法在实际应用中取得了明显的效果, 极大的提高了 MRO 系统的运行效率.

参考文献

1 Davenport TH. Putting the Enterprise into the Enterprise System. Harvard Business Review, 1998.

2 Yen DC. A Synergic Analysis for Web-based Enterprise Resources Planning System. Computer Standards, 2002.

3 褚士震,贾晓亮,耿俊浩,等.面向制造的重型装备产品 BOM 管理方法研究.制造业自动化,2013,35(6):34–38.

4 杜杰,陆金桂.一种改进的多级型 BOM 遍历算法.工程设计 CAD 与智能建筑,2002,(9):65–67.

5 彭克勤,岳清.BOM 的关系型数据库设计及算法研究.计算机与数字工程,2009,37(12):14–16.

6 黄改娟,张仰森,刘武雷.基于关系数据库的复合型 BOM 的设计与实现.北京信息科技大学学报(自然科学版),2012,6:014.

7 任荣升,徐建良.制造业中 BOM 建模及算法解决方案.微计算机信息,2006,9(3):23–25.

8 李钟蔚.Java 开发实战宝典.北京:清华大学出版社,2010:513–515.

9 李维斯.数据结构(Java 版).北京:清华大学出版社,2010.10:350–354.

10 拉佛.Java 数据结构和算法.北京:中国电力出版社,2009.10:35–37.

11 徐怀平.优化 Oracle 的查询性能.电脑编程技巧与维护,2012,23:008.

12 埃克尔.Java 编程思想.北京:机械工业出版社,2010.8:500–503.