第16卷 第5期
2006年5月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol. 16 No. 5
May 2006

关联规则挖掘算法介绍

陈玉婷[1],王 斌[1],刘 博[1],宋 斌[1,3],李 颉[2,3]

(1.北京邮电大学 电信工程学院,北京 100876;
2.中国人民解放军国防大学 研究生院,北京 100091;
3.中国人民解放军61062部队,北京 100091)

摘 要:数据挖掘是一个多学科交叉融合而形成的新兴的学科,它利用各种分析工具在海量数据中发现模型和数据间的关系。而在大规模事务数据库中,挖掘关联规则是数据挖掘领域的一个非常重要的研究课题。文中介绍了关联规则挖掘的研究情况,描述了经典Apriori算法的实现,并对该算法进行了分析和评价,指出了其不足和原因。描述了FP树挖掘最大频繁项集的算法,通过实例对该算法进行了性能评估,并得到结论:数据库中潜在的最大频繁模式越多,运行时间越长。

关键词:数据挖掘;关联规则;频繁项集;FP树

中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2006)05-0021-05

Introduction of Mining Association Rules Algorithm

CHEN Yu-ting[1],WANG Bin[1],LIU Bo[1],SONG Bin[1,3],LI Jie[2,3]

(1. Institute of Telecom Project,Beijing University of Posts and Telecommunications,Beijing 100876,China;
2. Graduate School,National Defence University of China,Beijing 100091,China;
3. Corps 61062,PLA,Beijing 100091,China)

Abstract:Data mining is an emerging subject that composed and amalgamated by multiple subjects. It is an analytic process designed to explore data in search of consistent patterns and/or systematic relationships between variables. Mining association rules in business transaction databases is one of the important topic of research on data mining. This paper introduced the research complexion of the association rules mining algorithm, describes the classical Apriori algorithm, analyses and evaluates it. The author emphasizes FP tree mining maximum frequent item sets algorithm specially. And evaluates performance of the algorithm through instance. At the end, the paper gives the conclusion: the more maximum frequent item pattern in the database, the longer run time is needed.

Key words: data mining; association rules; frequent item sets; FP tree

0 引 言

数据挖掘(Data mining),又称数据库中的知识发现(Knowledge discovery in database),在最近几年已被数据库界所广泛研究,其中关联规则挖掘是一个重要的问题[1]。关联规则是发现交易数据库中不同商品(项)之间的联系,这些规则找出顾客购买行为模式,如购买了某一商品对购买其他商品的影响。发现这样的规则可以应用于商品货架设计、货存安排以及根据购买模式对用户进行分类[2]。

文中分析了Apriori算法存在的不足及原因并且重点介绍了FP树挖掘最大频繁项集的算法。

1 关联规则的基本概念

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是二进制文字的集合,其中的元素称为项(item)。记 D 为交易(transaction)T 的集合,这里交易 T 是项的集合,并且 $T\subseteq I$。对应每一个交易有惟一的标识,如交易号,记作TID。设 X 是一个 I 中项的集合,如果 $X\subseteq T$,那么称交易 T 包含 X。

一个关联规则是形如 $X\Rightarrow Y$ 的蕴涵式,这里 $X\subset I$,$Y\subset I$,并且 $X\cap Y=\varnothing$。规则 $X\Rightarrow Y$ 在交易数据库 D 中的支持度(support)是交易集中包含 X 和 Y 的交易数与所有交易数之比,记为 support($X\Rightarrow Y$),即

$$\text{support}(X\Rightarrow Y)=|\{T:X\cup Y\subseteq T,T\in D\}|/|D|$$

规则 $X\Rightarrow Y$ 在交易集中的置信度(confidence)是指包含 X 和 Y 的交易数与包含 X 的交易数之比,记为 confidence($X\Rightarrow Y$),即

$$\text{confidence}(X\Rightarrow Y)=|\{T:X\cup Y\subseteq T,T\in D\}|/|\{T:X\subseteq T,T\in D\}|$$

项的集合称为项集(itemset)。包含 k 个项的项集称为 k_项集。项集的出现频率是包含项集的事务数,简称为项集的频率、支持度或计数。如果项集满足最小支持度(由用户或领域专家设定),则称它为频繁项集。给定一个交易集

收稿日期:2005-09-01

作者简介:陈玉婷(1975-),女,江西人,硕士研究生,研究方向为计算机网络理论与技术;导师:宋俊德,教授,博士生导师,研究方向为移动互联网、个人通信、未来通信、CTI/CRM、VLSI、CAD。

D,挖掘关联规则问题就是产生支持度和置信度分别大于最小支持度(minsupp)和最小置信度(minconf)的关联规则。

2 Apriori 算法介绍

2.1 算法描述

Apriori 算法是一种最有影响的挖掘布尔关联规则频繁项集的算法[3],算法使用频繁项集性质的先验知识,用逐层搜索的迭代方法来获得频繁项集。K_ 项集用于探索$(k+1)$项集。首先找到频繁 1 项集的集合,记为 L_1。L_1 用于找频繁 2 项集的集合 L_2,如此下去,直到不能找到频繁 K 项集[4]。

Apriori 性质:

频繁项集的所有非空自己都必须也是频繁的。这是因为根据定义,假设项集 I 不满足最小支持度(minsupp),则 I 不是频繁的,如果把项 A 添加到 I,则结果项集(即 $I \cup A$)不可能比 I 更频繁出现。因此,结果项集也不是频繁的。

利用 Apriori 性质,通过连接和剪枝两步过程来实现频繁项集的挖掘。

a. 连接步:

通过对 L_{k-1} 中的每个元素执行连接,得到了 L_k 的候选集合 C_k。

b. 剪枝步:

C_k 是 L_k 的超集,即它的成员中也有不是频繁的。首先,根据 Apriori 性质,缩小 C_k 的范围,然后扫描数据库,确定 C_k 中每个候选的计数,从而确定 L_k。

算法伪码:

算法:Apriori 使用根据候选生成的逐层迭代找出频繁项集。

输入:事务数据库 D;最小支持度 minsupp

输出:D 中的频繁项集 L

(1) L_1 = find_ frequent_ 1 - itemsets(D);

(2) FOR ($k = 2; L_{k-1} \neq F; k++$) {

(3) C_k = apriori_ gen(L_{k-1}, minsupp);

(4) FOR each transactions $t \in D$ {

(5) C_t = subset(C_k, t);

(6) FOR each candidates $c \in C_t$ DOc. count ++;

(7) }

(8) $L_k = \{c \in C_k \mid c.\text{count} \geqslant \text{minsupp}\}$

(9) }

(10) 返回 $L = \cup kL_k$;

procedure apriori_ gen(L_{k-1}, minsupp)— 候选集产生

(1) FOR each itemset $p \in L_{k-1}$ DO

(2) FOR each itemset $q \in L_{k-1}$ DO

(3) IF ($p[1] = q[1]$) $\wedge$ ($p[2] = q[2]$) $\wedge \cdots \wedge$ ($p[k-2] = q[k-2]$) $\wedge$ ($p[k] < q[k]$) {

(4) $c = p \oplus q$;

(5) IF has_ infrequent_ subset(c, L_{k-1}) THEN delete c;

(6) ELSE add c to C_k;

(7) }

(8) Return C_k;

procedure has_ infrequent_ subset(c, L_{k-1})— 判断候选集的元素

(1) FOR each $(k-1)$ - subset s of c DO

(2) IF Not $s \in L_{k-1}$ THEN Return TURE;

(3) Return FALSE;

2.2 算法分析

尽管 Apriori 算法的候选产生—检查方法大幅度压缩了候选项集的大小,并且导致了很好的性能,然而,有两种可能导致了这个算法开销很大。

* 它可能要产生大量候选项集。例如,如果有 10^4 个频繁 1 项集,那么 Apriori 算法需要产生 10^7 个候选 2 项集,并且累计和检查它们的频繁性。而且如果发现长度为 100 的频繁模式,它必须产生多达 10^{30} 个候选。

* 它可能需要重复扫描数据库,对于挖掘长模式扫描的次数更多。

产生这种情况的原因是,在候选多项集中可能有大量的,甚至是绝大多数的项集在事务数据库中是不存在的。这样就考虑是否能有一种方法不需要产生这么多的候选项集就能挖掘出频繁项集,而 FP 树算法的思想是先通过两次扫描数据库来构建 FP 树,然后在 FP 树的基础上进行频繁项集的挖掘。因此这里考虑应用 FP 树算法。

3 FP 树算法

3.1 定 义

对于项集 $X \subseteq T$,如果 $X.\text{sup} \geqslant s$,并且对于任意 $Y \supset X$,均有 $Y.\text{sup} < s$,则称 X 为 D 中的最大频繁项集。

显然,任何频繁项集都是某最大频繁项集的子集,所以可以把发现所有频繁项集的问题转化为发现所有最大频繁项集的问题[5]。

频繁模式树 FP - tree

频繁模式树 FP - tree 是一个树结构,定义如下:

(1)它由一个被标记为"null"的树根节点、作为根节点孩子的子节点集合和一个频繁项头表组成。

(2)每个子节点由 6 个域组成:项 ID itemId、节点计数 support、父节点指针 parent、兄妹节点指针 sibling、孩子节点指针 child、指向下一个具有相同 itemId 的指针 next。其中,sibling 和 child 两个域不是必需的,仅仅为方便树的遍历而建立。

(3)频繁项头表 headerTable 由频繁项结点 headerTableNode 组成。headerTableNode 包括 5 个域:项 ID itemId、支持度 support、指向前一频繁项结点的指针 prev、指向后一频繁项结点的指针 next、指向 FP - tree 树中与之 itemId 相同的第 1 个节点的指针 headerLink。所有的

频繁项结点通过 prev 指针和 next 指针形成链表结构,从而构成频繁项头表。

3.2 算法描述

基于 FP 树的最大频繁模式挖掘算法,主要包括两大步骤:

Step1:构造频繁模式树 FP－tree。

Step2:利用 FP－tree 挖掘最大频繁模式。

下面分别描述针对以上两个步骤的频繁模式树的构造算法和挖掘最大频繁模式算法。

3.2.1 频繁模式树 FP－tree 的构造算法

输入:事务数据库 D;最小支持数 support

输出:事务数据库 D 的频繁模式树,FP－tree

方法:

(1) 扫描 D 一次,产生频繁项集合 F 及其支持数。按其支持数降序排列 F,生成频繁项列表 L;

(2) 创建 FP－tree(记为 T) 的根节点,标号为"null"。对于 D 中的每个事务 Trans作如下处理:① 按 L 中的次序排列 Trans 中的频繁项,设排列后的结果为[p | P],其中 p 是第 1 个项,而 P 是剩余项的列表;② 调用 insert_tree([p | P], T);③ 如果 P 非空,递归调用 insert_ tree(P, N)。

过程 insert_ tree([p | P], T) 的执行情况如下:如果 T 有孩子节点 N 使得 N.itemId = p,则 N 的节点计数增加 1;否则创建一个新节点 N,将其 itemId、节点计数 support 分别设置为 p、1,由父节点指针 parent 链接到它的父节点 T,并通过相同 itemId 间的 next 链,将其链接到具有相同 itemId 的节点。

3.2.2 利用 FP－tree 挖掘最大频繁模式算法

输入:D 的频繁模式树 FP－tree;频繁项头表 hTable;最小支持数 support

输出:D 中的最大频繁模式集

方法:调用过程 FP－Mine(FP－tree,hTable,null)

Procedure FP－Mine(Tree,hTable,alpha)

{

(1)如果 Tree 只有一个根节点,则返回;

(2)if (Tree 是单一路径)

(3)then 输出该单一路径上的所有最大频繁模式

(4)else 对于项头表 hTable 中的每一个结点 a_i, do{

(5)输出最大频繁模式 beta= $a_i \cup$ alpha,beta 的支持度等于 a_i 的支持度;

(6)调用过程 constructConditionalPatternBase($a_i \rightarrow$ headerLink)//构造 beta 的条件模式基,结果存入临时数据库文件 tempDB

(7)构造事务数据库 tempDB 的频繁模式树 $Tree_{beta}$,即 beta 的条件 FP 树

(8)递归调用 FP－Mine($Tree_{beta}$, $Tree_{beta}$ 的项头表, beta)}

}

● 其中,标号(3)从单一路径寻找所有最大频繁模式的执行情况如下:

a.初始化叶子节点 leaf＝null;

b.while (leaf 不等于根节点) do{

c.从叶子节点 leaf 到根节点的所有 item,构成频繁项集 beta;

d.输出最大频繁模式 beta∪alpha,其支持度等于叶节点 leaf 的支持度;

e.叶子节点 leaf 向根节点上移若干步,上移的步数等于该单一路径上由下至上,连续具有相同支持度的节点总数;}

● 过程 constructConditionalPatternBase($a_i \rightarrow$ headerLink)的执行情况为:

a.顺着 $a_i \rightarrow$headerLink 的 next 链,找到 FP 树中所有与 $a_i \rightarrow$headerLink 的 itemId 相同的树节点,记下这些树节点到树根之间的所有分支,各分支中项的计数分别等于这些树节点的支持计数 support;

b.将每一个分支作为 support 条数据库事务记录,保存到临时数据库文件 temDB 中。

上面的算法比较抽象,下面以一个具体的例子来说明。

例:设事务数据库 D 如表 1 所示,最小支持数 support ＝1。

表 1　事务数据库 D

TID	Items	排了序的频繁项集
100	1,2,5	2,1,5
200	2,4	2,4
300	2,3	2,3
400	1,2,4	2,1,4
500	1,3	1,3
600	2,3	2,3
700	1,3	1,3
800	1,2,3,5	2,1,3,5
900	1,2,3	2,1,3

第 1 次扫描数据库得到 D 的按项的支持数递减排序的频繁项列表 L〈(2:7),(1:6),(3:6),(4:2),(5:2)〉(冒号后面的数字为支持数),由于树的每一条路径都按照这种顺序排列项,把每一个事务中的频繁项按 L 的顺序排列,得到表 1 右端的频繁项集列表。按照上述的频繁模式树 FP－tree 的构造算法,得到数据库 D 的 FP－tree 如图 1 所示。

● 调用过程 FP－Mine(FP－tree,hTable,null)的具体过程如下:

(1)hTable 赋值为频繁项头表中的最后一个结点,即 5:2。根据算法,首先输出最大频繁模式{5:2},接着构造其条件模式基{(2:1,1:1),(2:1,1:1,3:1)},将其存放到 tempDB 中。构造 tempDB 的 FP－tree 如图 2 所示。对图 2 所示的 FP 树,递归调用 FP－Mine(FP－tree, hTable,5:

2),由于该树是单一路径,输出该路径上的所有最大频繁模式{(5 1 2 3:1),(5 1 2:2)}。至此,得到了所有包含 itemId 等于 5 的所有最大频繁模式集{(5:2),(5 1 2 3:1),(5 1 2:2)};

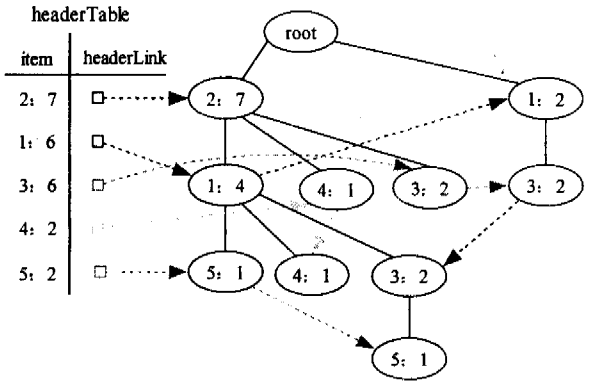

图 1 数据库 D 的 FP-tree

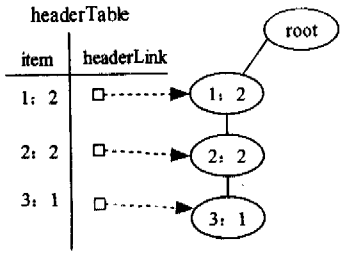

图 2 itemId=5 的项的条件 FP-tree

(2)hTable 赋值为项头表中 5:2 前面的结点,即 4:2。同(1),算法将输出所有包含 itemId 等于 4 的所有最大频繁模式集{(4:2),(4 2 1:1),(4 2:2)};

(3)hTable 赋值为项 3:6,算法输出所有包含 itemId 等于 3 的所有最大频繁模式集{(3:6),(2 3:4),(2 3 1:2),(1 3:4)};

(4)hTable 赋值为项 1:6,算法输出所有包含 itemId 等于 1 的所有最大频繁模式集{(1:6),(1 2:4)};

(5)hTable 赋值为项 2:7,算法输出所有包含 itemId 等于 2 的所有最大频繁模式集{(2:7)}。到此,算法全部执行结束。

最终得到的数据库 D 的所有最大频繁模式集合为:

{(5:2),(5 1 2 3:1),(5 1 2:2),(4:2),(4 2 1:1),(4 2:2),(3:6),(2 3:4),(2 3 1:2),(1 3:4),(1:6),(1 2:4),(2:7)},其中冒号前面的数字为最大频繁模式,冒号后面的数字为该最大频繁模式的支持数。

3.3 算法的性能评估

从 FP-tree 的构造过程来看,只需要两次扫描数据库:第一次扫描搜集频繁项集,第二次扫描形成 FP-tree。将一个事务 Trans 插入到 FP-tree 的时间开销为 $O(|Trans|)$,其中 $|Trans|$ 为 Trans 中的频繁项的数量。空间开销方面,由于 FP-tree 的大小由数据库中频繁项出现的总次数界定,FP-tree 的高度由数据库事务中最大的频繁项集中频繁项的个数界定,所以 FP-tree 不会超过原始的数据库大小。事实上,由于 FP-tree 只包含数据库中的频繁项,而且事务之间存在许多共享的频繁项,FP-tree 的构造过程又使得越频繁的项越靠近 FP-tree 的树根,进一步增强了频繁项被多个事务共享的性质,这些因素导致 FP 树的大小通常比原始的数据库小很多,也就是说 FP-tree 是一种高度压缩的结构。实验证明一颗小的 FP-tree 就可以压缩存放一些相当大的数据库,比如,对于一个包含 67 557 个事务,每个事务包含 43 个项的数据库,当最小支持度是 50%时,数据库中频繁项出现的总次数为 2 219 609,而其 FP-tree 中的结点总数为 13 449,压缩比高达 165。

从挖掘过程来看,需要扫描数据库 D 的 FP-tree 一次,并对每一个频繁项 a_i,产生一个小的条件模式基 B_{a_i},其中每个 B_{a_i} 由若干经变换后的 a_i 的前缀路径组成。接着,频繁模式挖掘通过构造 B_{a_i} 的条件 FP-tree,递归地在这些小的条件模式基 B_{a_i} 上进行。通常,FP-tree 要比数据库 D 小很多,相似地,基于 B_{a_i} 构造的条件 FP-tree 也要比 B_{a_i} 小很多,而且,由于模式基 B_{a_i} 只包含频繁项 a_i 的变换后的前缀路径,使得 B_{a_i} 通常又要比最初的 FP-tree 小很多。这样以来,后来的每一个挖掘过程都在小很多的模式基和条件 FP-tree 上进行。不仅如此,由于挖掘的操作主要包含前缀计数调整、计数和模式片断的连接,这些操作的代价要远远小于候选频繁项集的产生和测试操作,因此算法具有很高的执行效率。

3.4 应用实例的效果

本节将给出实验结果。所有的实验都在一台主存为 256MB 的 PC 机上运行,操作系统为 Window2000,程序用 Microsoft/Visual C++ +6.0 编写。数据来源一种是手工输入,一种是通过程序产生。数据库文件格式为:

```
T100
该事务中的 item 数
itemId1(正整数)
itemId2
...
T200
...
```

所采用的数据生成算法如下:

```
输入数据库中的事务总数|Trans|;
for (int i=1; (i<=|Trans|); i++)
{
新建并打开一个写文件;
将事务 ID 写入数据库文件;
随机产生一个小于最长事务长度的正整数|TranItemCount|;
  for(int j=0; (i<|TranItemCount|); i++)
    {
      随机产生一个小于数据库中不同 item 总数的正整数 itemId;
      将 itemId 写入数据库文件;
    }
```

笔者选择了 1 个手工输入的数据库文件,和 5 个由上述数据生成算法产生的数据库文件进行了实验,实验结果如表 2 所示。

表 2　实验结果

实验数据说明					输入参数		运行情况	
数据集	数据来源	事务数	item 个数	最长事务长度	支持数	支持度	运行时间	挖掘出的最大频繁模式个数
Database1	手工输入	9	5	4	1	11.10%	0.5 秒	13
Database2	程序产生	50	20	10	1	2%	3 秒	1738
					2	4%	2 秒	871
					3	6%	1.5 秒	455
Database3	程序产生	50	100	10	1	2%	2.5 秒	1300
					2	4%	1.5 秒	179
					3	6%	1 秒	67
Database4	程序产生	50	100	20	1	2%	14 秒	6758
					2	4%	4 秒	1819
					3	6%	1.5 秒	421
Database5	程序产生	250	100	10	1	0.4%	10 秒	6254
					2	0.8%	3.5 秒	1365
					3	1.2%	2 秒	386
Database6	程序产生	250	100	20	1	0.4%	>6 分钟	>65211
					2	0.8%	160 秒	39153
					3	1.2%	35 秒	12267

从表 2 可以看出针对每一个数据库文件,随着输入的最小支持度增加,数据库中的频繁模式数随之减少,使得运行时间也随着变短;当数据库文件 item 的种类增加时,而支持度不变时,由于数据库文件生成算法生成数据的随机性,使得每一个 item 出现的几率降低,从而导致频繁模式数的减少和运行时间的缩短(参照 Database2 和 Database3 的结果对比);当数据库中事务的平均长度增加时,会导致数据库中频繁模式数量的急剧增加和运行时间的显著增长(参照 Database3 和 Database4 的结果对比、Database5 和 Database6 的结果对比));另外随着数据库中事务数的增加,运行时间也会有所增加,主要是因为扫描数据库的时间变长;总体上讲,数据库中潜在的最大频繁模式越多,运行时间越长。

参考文献:

[1] 王　燕,李　睿,李　明.数据挖掘技术应用研究[J].甘肃科技,2001,12(1):49-50.

[2] Han Jiawei, Kamber M. Data Mining: Concepts and Techniques[M]. United States of America: Morgan Kaufmann, 2000.267-270.

[3] 孟晓明.浅谈数据挖掘技术[J].计算机应用与软件,2004,21(8):34-36.

[4] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases[A]. Proc 1993 ACM-SIGMOD Int Conf Management of Data(SIGMOD'1993)[C]. Washington, D.C., United States: ACM Press, 1993.207-216.

[5] Han J, Jian P, Yiwen Y. Mining frequent patterns without candidate generation[A]. In: Proceedings of the 2000 ACM SIGMOD International Conference Management of Data[C]. Dallas: [s.n.], 2000.1-12.

(上接第 15 页)

近期的路线图可以参考 FSG 的网站文档。可以预见,LSB 作为 Linux 应用程序开发的二进制接口的国际标准将对 Linux 相关产业的发展产生重大的影响。

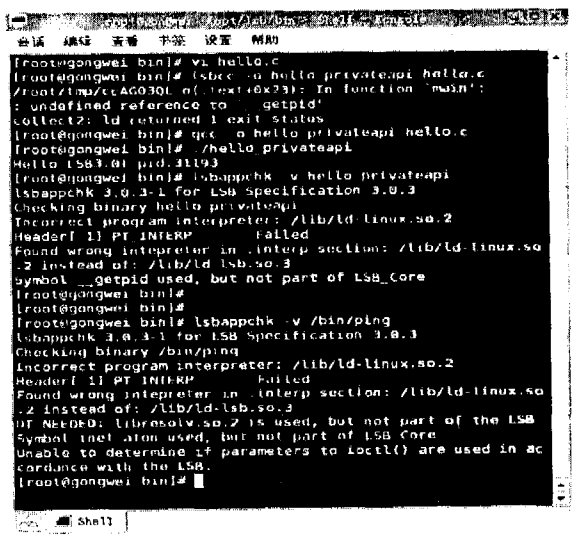

图 4　用 lsbappchk 测试系统工具 ping

7　结束语

文中深入剖析了 LSB3.0 标准的文本规范;基于 LSB 的发行版组件;LSB3.0 认证的应用程序的构建;LSB3.0 的标准认证以及 LSB3.0 的开发环境。由于 LSB 标准是开放标准中最重要的核心,中国 Linux 标准工作组拟议将其直接引用作为中文 Linux 标准体系中的 Linux 应用二进制界面(ABI)规范。深入地理解和应用 LSB 规范对中国 Linux 相关产业的发展和中文 Linux 标准体系的制定具有重大的现实意义。

参考文献:

[1] Linux Standard Base Team. Linux Standard Base Mission Statement[EB/OL]. http://www.linuxbase.org Mission Statement, 2005.

[2] Linux Standard Base Team. LSB3.0 ReleaseNote3[EB/OL]. http://www.linuxbase.org/LSBWiki/ReleaseNotes3, 2005.

[3] Linux Standard Base Team. Linux Standard Base Specifications Archive[EB/OL]. http://refspecs.freestandards.org/lsb.shtml#LSB_3_0_0, 2005-07.

[4] Fink M. Linux 及开发源代码在商业经济中的应用[M].雷之宇,倪志欣,刘　韵译.北京:清华大学出版社,2005.

[5] Yeoh. Building LSB Compliant Applications[Z]. Christopher IBM OzLabs Linux Technology Centre, 2004.

[6] Linux Standard Base Team. Building applications with the Linux Standard Base IBM press[M]. USA: IBM press, 2004.