基于分组技术的工作流活动多实例的调度研究*

杨 光，张春海
(中国海洋大学 计算机科学系，山东 青岛 266071)

摘 要：在分析静态和动态活动多实例模型的基础上，提出了一种机制——实例池和多实例分组技术，并且提出相应的算法，解决了静态和动态多实例的调度问题。该机制可以处理静态和动态多实例的生成、初始化、运行和调度以及销毁。

关键词：工作流；活动多实例；实例池；分组；调度
中图法分类号：TP311 **文献标识码**：A **文章编号**：1001-3695(2006)09-0060-04

Study of Multiple Instances Schedule Based on Instances Groups

YANG Guang, ZHANG Chun-hai
(Dept. of Computer Science, Ocean University of China, Qingdao Shandong 266071, China)

Abstract: Analysing the multiple instances executing models, based on the models, instances pool and groups are proposed to manage the life of static and dynamic multiple instances, which consists of creation, initialization, execution, schedule and destruction. Accordingly the algorithms are presented to realize the multiple instances schedule.

Key words: Workflow; Multiple Instances; Instances Pool; Groups; Schedule

工作流是业务过程的模型，它将现实中的业务逻辑和规则以合适的模型在计算机中进行表示并对其实施计算[1,2]。工作流活动多实例，是由一个工作流活动定义而产生的多个实例，它在多个工作流参与者的参与下共同完成。活动多实例问题是在实际中经常遇到的问题，如调查问卷、投票选举等。在工作流管理系统中引入对活动多实例的支持，可以为现实环境中的业务流程处理带来极大的灵活性和方便性。但是由于活动多实例本身的调度控制复杂性，使得大多数工作流管理系统对工作流活动多实例的调度基本不支持或仅提供有限的支持。

1 工作流活动多实例

1.1 工作流活动多实例的描述

定义：工作流活动[3]。

工作流活动 N 是一个四元组 $N<\mathrm{ID}, U, T, V>$，ID 是活动区分于其他活动的唯一标志；U 是活动参与者的集合；T 是活动属性的集合；V 是属性的值域。

活动实例 $<id, u, t, v>$ 则是活动定义 $<\mathrm{ID}, U, T, V>$ 的实例化。称一个活动 N 是多实例，指在同一过程实例中至少存在两个活动实例 $n_1<id_1, u_1, t_1, v_1>$ 和 $n_2<id_2, u_2, t_2, v_2>$ $(id_1 \neq id_2)$ 具有相同的活动定义 $<\mathrm{ID}, U, T, V>$。

根据运行语义的不同，文献[3]提出了几种典型的多实例运行模型。在此基础上，我们归纳出活动多实例的前后触发情况：

(1) One-Join。前序活动 A 的某个实例 a 完成，就触发活动 A 的后继活动 B 的实例的执行。根据 B 被触发的实例个数，可以分为 1-1 触发和 1-n 触发两种触发方式。

(2) All-Join。活动 A 的 N 个实例均完成后，才触发 A 的后继活动 B 的实例的执行。同样可以分成 n-1 触发和 n-n 触发。

(3) Part-Join。活动 A 的一定数量的实例完成后，就可以触发 A 的后继活动 B 的实例的执行。可以分成 n-1 触发和 n-n 触发。对这种汇聚使用 m/n 连接符表示，如图 1 所示。

1.2 工作流活动多实例的分类

工作流活动多实例从其产生的方式来看，可以分为两种：

(1) 静态多实例(又称为设计期多实例)。它在设计期就能够确定。某个活动在流程定义时就已经确定了要产生的实例的数目。

业务流程中的问卷调查通常就是静态多实例的典型代表。问卷的发放数量在定义流程时就可以确定，这种情况只需要在设计流程时指定数目就可以了。

根据静态多实例的定义，在工作流中出现次数最频繁的活动单实例也可以归属到静态多实例的范畴中。

(2) 动态多实例(又称为运行期多实例)。它的多实例个数在运行期决定。一般是因为多实例的数目要依据运行期的一些信息而定，而这些信息在设计期是不确定的或者是模糊的。如工作流参与者不明确时，就无法在设计期决定到底应该产生多少个活动实例。

但是在动态多实例活动要被引擎执行前，动态多实例应该被确定化为静态多实例，否则工作流将无法继续执行。这就要求每个动态多实例都应该有一个实例确定点，流程实例执行路径一旦跨越了该实例确定点，则表示相应的动态多实例所需要的产生条件和初始化信息均已经满足，可以将动态多实例静态

收稿日期：2005-08-13；修返日期：2005-09-28
基金项目：国家“863”计划资助项目(2004AA112070)

化了。但如果有两个动态多实例的实例确定点产生了相互依赖, 则本流程就会失败。

流程实例执行路径跨越了一个实例确定点后, 根据相应的动态多实例一次所能产生的多实例个数, 动态多实例又可以被分为两类, 即当条件满足时, 一次只产生 $m(1 \leq m < n)$ 个实例; 当条件满足时, 一次性产生所有的活动实例。

在线投票流程如图 2 所示。对一项决议进行投票, 在截止时间前登录注册的人员均有权投票, 只有当有效票数超过所有注册人员的 80% 后, 投票结果才算有效。

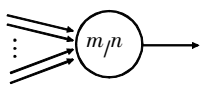


图 1　Part-Join

m/n 连接符

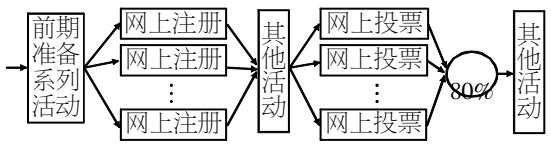


图 2　在线投票流程

"网上注册"和"网上投票"是两个动态多实例。大多数情况下, 注册一般是一个单实例的活动, 有一个活动处理所有的注册信息。如果要记录其他的一些和与会者个人相关的信息, 可以作为一个多实例来实现。

每当一个用户登录后, 就会产生一个"网上注册"的活动实例, 当到达截止时间或是注册人数达到一定数量时, 这个多实例活动就可以完成了。当投票人数确定后, 所有的投票活动实例会一次性地全部产生。如网络会议的注册等, 只能是登录一个工作流参与者就产生一个注册活动实例。

无论是哪种情况, 都需要一种机制来控制活动多实例的生成、运行和销毁。在此我们引入了活动的实例池, 对活动多实例进行管理。

2　实例池中活动多实例的分组

2.1　活动实例池

活动实例池定义如下:

(1) 活动实例的生存期。工作流活动实例的生存期起始于活动的某个实例被创建时, 终止于该实例被销毁。

(2) 活动实例池。工作流的活动实例池隶属于一个流程定义, 对应着流程定义中某个活动定义。它是一个七元组 POOL < PoolID, ProcessID, ActivityID, Input, Output, InputControlBlock, OutputControlBlock >, 其中: PoolID 为活动实例池的唯一标志; ProcessID 为所隶属的流程定义标志; ActivityID 为所对应的活动定义; Input 为活动实例池的输入; Output 为活动实例池的输出; InputControlBlock 为输入控制块 ICB, 用于处理 Input 信息并控制活动的生成和激活; OutputControlBlock 为输出控制块 OCB, 决定多实例活动的汇聚, 用于生成 Output 信息。

在工作流系统运行时, 根据某个流程定义可以产生多个流程实例。组成流程实例的活动实例的生存期均是由与 ActivityID 相对应的活动实例池来管理的。活动所需要的资源等均由实例池在创建活动实例时负责申请。当资源不足时, 将活动转入休眠队列; 待到资源申请成功后, 继续创建活动实例; 创建完毕后, 执行活动或分派工作项。一个简单的示意图如图 3 所示。

2.2　活动多实例分组

为了管理活动多实例, 我们提出活动多实例的分组技术。对一个活动定义的多个实例, 根据其不同的执行路径和触发条件, 分成各不相同的组, 每个组含有一个或多个活动实例。首先, 我们对工作流活动及其实例的定义进行如下扩展:

(1) 扩展的工作流活动。工作流活动 N 是一个五元组 N < ID, U, T, V, JoinStyle >, ID 是活动区分于其他活动的唯一标志; U 是活动参与者的集合; T 是活动属性的集合; V 是属性的值域, JoinStyle 是活动的汇聚属性, 取值为{ One-Join(1), All-Join(n), Part-Join(m/n) }。

(2) 扩展的活动实例。扩展的活动实例是活动定义 < ID, U, T, V, JoinStyle > 的实例化, 它是一个六元组 < gid, id, , u, t, v, joinstyle > 。其中, gid 是活动实例在实例池中的组号。在不引起混淆的情况下, 下文将直接称满足以上两个定义的实体为工作流活动和活动实例。

(3) 实例组。一个实例组隶属于一个活动实例池, 是一个四元组 < Gid, ACTs, State, Priority/ JoinStyle > 。Gid 是实例组的唯一标志; ACTs 是实例组中活动的集合; State 是实例组的状态, 在 ICB 分组中可以表示此实例组是否已被触发过, 在 OCB 分组中表示实例组中的活动是否满足了汇聚条件; Priority 出现在 ICB 分组中, 表示触发优先级; JoinStyle 出现在 OCB 分组中, 表示汇聚类型。

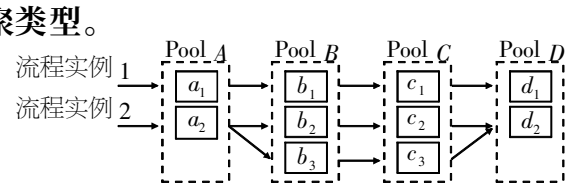


图 3　使用活动实例池管理活动的生存期

2.3　ICB/OCB 分组表和 OCB-ICB 触发映射表的生成

活动实例池的管理将以组为单位进行, 分组操作在 ICB 和 OCB 中分别进行, 最终形成一个流程定义的 ICB 分组表、OCB 分组表和 OCB-ICB 触发映射表。ICB 分组被实例池用于生成某一组的活动; OCB 分组被用于判断某个分组是否满足汇聚条件以触发后续活动; 触发映射表则包含了组与组之间的触发对应关系。将实例分组后可以形式化地表示为 PoolID{ gid{ inst_id, inst_id, …}, gid{ inst_id, …}, …}, 其中 PoolID 表示具体的实例池, gid 表示实例池中的某个分组, inst_id 表示组中的活动实例。

相关算法如下:

算法 1　建立 ICB/OCB 分组表

输入: 流程定义

输出: ICB/OCB 分组表

(1) 对起始节点的实例池, ICB 分组设为 PBegin。

(2) 对终止节点的实例池, OCB 分组设为 PEnd。

(3) 对每个实例池: ①实例池中的活动按其触发源进行分组, 相同触发源的活动分为同一组, 形成当前实例池的 ICB 分组; ②实例池中的活动按其汇聚点进行分组, 相同汇聚点的活动分为同一组, 形成当前实例池的 OCB 分组。

(4) 将所有活动实例池的 ICB 分组组合成 ICB 分组表。

(5) 将所有活动实例池的 OCB 分组组合成 OCB 分组表。

(6) 输出 ICB/OCB 分组表。

由算法 1 可知, 当流程定义了动态多实例后, ICB/OCB 分组表的生成无法在定义期一次完成。将动态多实例所涉及的分组保留, 待运行期动态多实例确定后再填入。

算法 2 建立 OCB-ICB 触发映射表

输入: 流程定义

输出: OCB-ICB 触发映射表

(1) 取流程定义图的有向边的集合 E。

(2) 从 E 中取一条边 $<a, b>$, a 是弧尾, b 是弧头。a 属于 OCB 分组表中组 OA, b 属于 ICB 分组表中的组 IB。

①如果 a 是 PBegin, 则表示 b 是初始节点。如果 b 不在 PBegin 的触发序列中, 则在 PBegin 的触发序列中填入 b, 转(3)。

②如果 b 是 PEnd, 则表示 a 是终止节点。如果 PEnd 不在 a 的触发序列里, 则填入 PEnd, 转(3)。

③如果 IB 不在 OA 的触发序列中, 则将 IB 填入 OA 的触发序列, 转(3)。

(3) 如果 E 中还有其他的边未处理, 则转(2)。

(4) 对各个触发序列中的实例组按触发优先级排序。

(5) 输出 OCB-ICB 触发映射表。

首先是因为起始节点和终止节点比较特殊, 所以要特殊处理。再者, 实例组的触发优先级不同, 即当实例组 g_1 和 g_2 均满足触发条件, 但又不能同时触发时, 则必须根据触发优先级来决定 g_1 和 g_2 的触发次序。因此, 触发映射表中某个 OCB 组的 ICB 触发序列应该是一个偏序集合 $<$ICBTrigger, $\leq>$。若 ICBTrigger $=\{g_1, g_2, g_3, \cdots, g_n\}$ 则实例组的触发优先级关系应满足 $g_1 \geq g_2 \geq g_3 \geq \cdots \geq g_n$。因此生成触发序列时, 需要按实例组的触发优先级进行排序。

边集 E 中只包含已经确定了执行路径的边, 而不包含与动态多实例相关的边。这些不确定的边将随着运行期信息的确定而被确定。当动态多实例“静态化”时, 工作流引擎或者管理员将对边集 E 进行更新, 并重新执行算法 1 和算法 2, 以更新各个表的信息。

此时, 算法的执行将被限制在一个局部范围内, 仅在已经确定化了的动态多实例的前后局部执行。

3 活动多实例的调度控制

我们使用一个示例流程, B, C, E 都是静态多实例, D 则是动态多实例。假设 D 的实例确定点定义在 b_1 和 b_2 的执行完成时刻。流程定义图如图 4 所示。

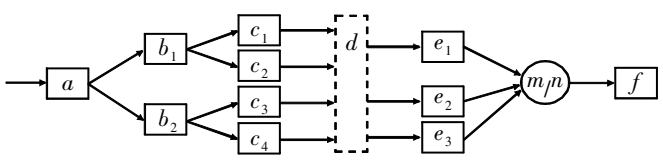


图 4 多实例的实例流程

首先使用表生成算法生成 ICB 分组表(表 1)、OCB 分组表(表 2)和 OCB-ICB 触发映射表(表 3)。

表 1 ICB 分组表

ICB A	{PBegin}
ICB B	$\{IB_1\{b_1, b_2\}\}$
ICB C	$\{IC_1\{c_1, c_2\}, IC_2\{c_3, c_4\}\}$
ICB D	{???}
ICB E	{???}
ICB F	$\{IF_1\{f\}\}$

表 2 OCB 分组表

OCB A	$\{OA_1\{a\}\}$
OCB B	$\{OB_1\{b_1\}, OB_2\{b_2\}\}$
OCB C	{???}
OCB D	{???}
OCB E	$\{OE_1\{e_1, e_2, e_3\}\}$
OCB F	{PEnd}

表 3 OCB-ICB 的触发映射表

OCB 组	PBegin	OA_1	OB_1	OB_2	OE_1	f
触发序列	a	IB_1	IC_1	IC_2	IF_1	PEnd

因为 D 是动态多实例, 所以在设计期算法第一次执行时, 相关的分组不能确定, 暂时空置。

3.1 控制算法描述

为了描述控制算法, 首先给出关键活动实例和 Count 算子的定义。

定义: 关键活动实例。

一个活动实例被挂起时, 整个流程实例必须被挂起。若活动实例执行失败, 流程的执行必须终止。这种活动实例就称为关键活动实例。如 1-1, 1-n, n-1 汇聚中的“1”端一般自动成为关键活动实例, 对于其他活动实例则可以根据实际情况决定。

定义: Count 算子。

Count 算子接收输入元组 $<$OCB 组号 gid, 汇聚条件 joinstyle $>$, 并负责更新 OCB 分组表中组号为 Gid 的实例组 g 的状态 State, 当满足汇聚时, 置 State 为“COMMIT”。

实例池的 ICB 根据 ICB 分组表和 Input 来决定实例化一个或一组活动实例。当活动完成初始化开始执行后, 就由 OCB 来负责。一旦活动完成, 则 OCB 根据活动的完成状态和 OCB 分组表来决定如何操作。如果活动执行失败, 且活动是关键活动, 则终止执行并将信息发送给工作流引擎进行异常处理; 若活动是非关键活动, 则丢弃此活动实例。如果活动执行成功, 则 OCB 使用 Count 算子更新实例组的状态 State, 当其满足了触发条件时, OCB 就产生 Output 发送给工作流引擎去引发下一活动。具体的算法实现如下:

算法 3 实例池 POOL 的 ICB 控制算法

(1) ICB 接收输入信息 Input。

(2) 从 Input 信息中查找前一活动所属的 OCB 分组的组号 OP, 在 OCB-ICB 触发映射表中找到与 OP 对应的触发序列。

(3) 从触发序列中找出组状态 State 为“NULL”的第一个组 g(Priority 值最大者, 因为生成表时已经排序)。

(4) 为组 g 中的活动申请资源生成组 g 中的活动实例集合, 将组 g 的 State 设置为“CREATED”。

此时活动将开始执行, 或调用外部程序, 或调用 Web Service, 或分发工作项给工作流参与者。当活动执行完毕并提交后, 由实例池的 OCB 进行调度控制。

算法 4 实例池 POOL 的 OCB 控制算法

(1) OCB 接收活动的提交信息。

(2) 如果活动是因为发生错误而被提交的, 则若活动是关键活动, 置活动所在组 g 的 State 为“EXCEPTION”, 终止执行, 由工作流引擎处理异常; 否则, 丢弃非关键活动并退出算法。

(3) 如果活动所在的组 g 的 State 已被设置为“COMMIT”或“EXCEPTION”, 则丢弃活动, 退出算法。

(4) 使用 Count(gid, joinstyle) 算子进行实例组的 State 更新。

(5) 如果实例组 g 的 State 状态为“COMMIT”, 将组号放入 Output 输出信息, 发送给工作流引擎。

3.2 多实例控制举例

假设 T_1 时刻, 状态为“b_1, b_2 刚被提交”, 则动态多实例活动 D 可以被实例化。此时的流程定义图被更新为如图 5 所示。

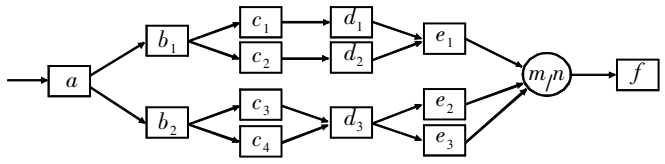


图 5　多实例的实例流程

算法被再次执行,以更新相应的表的内容。相关的更新如表 4 ~表 6 所示。

表 4　ICB 分组表更新

ICB D	$\{ ID_1\{ d_1\} , ID_2\{ d_2\} , ID_3\{ d_3\} \}$
ICB E	$\{ IE_1\{ e_1\} , IE_2\{ e_2, e_3\} \}$

表 5　OCB 分组表更新

OCB C	$\{ OC_1\{ c_1\} , OC_2\{ c_2\} , OC_3\{ c_3, c_4\} \}$
OCB D	$\{ OD_1\{ d_1, d_2\} , OD_2\{ d_3\} \}$

表 6　OCB-ICB 的触发映射表更新

OC_1	ID_1
OC_2	ID_2
OC_3	ID_3
OD_1	IE_1
OD_2	IE_2

此时,流程执行中的不确定因素均已被消除。T_2 时刻,状态变为“e_1 已经开始执行, d_3 刚被提交”,则 Pool E 的 ICB 接收 Input 信息后,扫描触发映射表 OD_2 的触发序列,发现 $IE_2\{ e_2, e_3\}$ 的 State 是“NULL”,因此 Pool E 的 ICB 分组中 IE_2 为当前应该触发的分组。Pool E 的 ICB 为组 IE_2 申请资源后,生成了实例 e_2, e_3,并将 IE_2 的 STATE 设为“CREATED”。T_3 时刻,状态为“e_1, e_2 执行成功,陆续被提交, e_3 还在执行”,则 Pool E 的 OCB 收到了 e_1, e_2 的提交信息。OCB 连续执行了 Count(OE_1, 2/3) 和 Count(OE_1, 2/3) 后,实例组 OE_1 的状态 State 被设置为“COMMIT”。所以 OCB 产生了 Output,交由工作流引擎,触发活动 f。T_4 时刻, e_3 完成,但是因为实例组 OE_1 的状态 State 已经被设置为“COMMIT”,所以 e_3 将被丢弃。

4　相关研究与算法的比较

使用实例池和分组技术进行多实例的工作流调度控制,并将多实例在触发和汇聚两方面进行分组,比其他的一些算法机制中复杂的判断和控制机制更容易理解和实现。而且将多实例的生成和控制与单实例的统一起来,并将活动实例的生存期管理从工作流引擎中分离出来,使工作流引擎以一致的接口仅对实例池进行管理。文献[3] 提出了使用 Shell 来管理和记录多实例信息,与本文的研究方向有些类似,但是实现机制不同;文献[4] 提出了活动多实例的不同类别并进行了分类,但缺乏对多实例同步控制的深入分析; 文献[5] 提出了工作流多实例建模,但未提出具体的表现形式。

5　未来的工作

基于以上的控制机制,我们实现了一个多实例工作流的原型。该原型已可对常见的多实例进行调度,并且能够比较好地处理动态多实例。但是由于算法需要随着动态多实例确定而不断重复执行,因此当工作流的流程定义非常复杂时(包含数十个甚至数百个多实例,静态和动态多实例相混杂,执行路径交叉繁杂) 耗时较多。虽然这对于长达数天或数月的流程则可以忽略不计,但对于短时间的流程实例(如生存期只有几小时的流程实例) 的性能影响比较严重。因此,我们以后将重点研究和优化短时间流程实例的多实例调度问题。

参考文献:

[1]　ollingsworth D. Workflow Management Coalition: The Workflow Reference Model[M] . Brussels: WFMC-TC00-1003, 1994.

[2]　WfMC. Workflow Management Coalition Specification[M] . Brussels: Terminology and Glossary, WFMC-TC-1011, 1996.

[3]　孙瑞志, 史美林. 工作流活动多实例的调度控制[J] . 软件学报, 2005, 16(3) : 400-406.

[4]　Van Der Aalst WMP, Ter Hofstede AHM, Kiepuszewski KB, *et al*. Workflow Patterns[M] . BETA Working Paper Series, Working Paper 47. Eindhoven: Eindhoven University of Technology, 2000.

[5]　范玉顺, 吴澄. 一种提高系统柔性的工作流建模方法研究[J] . 软件学报, 2002, 13(4) : 833-839.

作者简介:

杨光(1981-) , 男, 硕士研究生, 主要研究方向为数据库理论及应用、XML 数据库和半结构化数据库、异构数据集成、工作流等; 张春海(1963-) , 男, 副教授, 硕士生导师, 主要研究方向为数据库理论及应用、XML 数据库和半结构化数据库、异构数据集成等。

(上接第 59 页)

[4]　hristor K Georgiadis, Ioannis Mavridis, *et al*. Flexible Team-based Access Control Using Contexts[C] . Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, 2001. 21-27.

[5]　Eve Cohen, Roshan K Thomas. Models for Coalition-Based Access Control (CBAC) [C] . Proceedings of the 7th ACM Symposium on Access Control Models and Technologies, 2002. 97-106.

[6]　Michael J Covington, Wende Long, Srividhya Srinivasan. Securing Context-aware Applications Using Environment Roles[C] . Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, 2001. 10-20.

[7]　Marc Wilikens, *et al*. A Context-related Authorization and Access Control Method Based on RBAC: A Case Study From the Health Care Domain[C] . Proc. of the 7th ACM Symposium on Access Control Models and Technologies, 2002. 117-124.

[8]　Arun Kumar, Neeran Karnik , Girish Chafle. Context Sensitivity in Role-Based Access Control[J] . ACM SIGOPS Operating Systems Review, 2002, 36(3) : 53-66.

[9]　Gustaf Neumann, Mark Strembeck. Design and Implementation of a Flexible RBAC-service in an Object-oriented Scripting Language[C] . Proceedings of the 8th ACM Conference on Computer and Communications Security, 2001. 58-67.

[10]　Mark Strembeck, Gustaf Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environment[C] . Proceedings of the 8th ACM Symposium on Access Control Models and Technologies, 2003. 392-427.

[11]　J M Spivey. The Z Notation: A Reference Manual[EB/ OL] . http: // spivey. oriel. ox. ac. uk / mike / zrm /, 2001.

[12]　丁二玉, 骆斌, 陈世福. 一种面向对象的数据库访问控制模式[J] . 计算机工程与应用, 2004, 40(11) : 171-173.

作者简介:

钱伟(1981-) , 男, 江苏南通人, 硕士研究生, 研究方向为访问控制、工作流; 丁二玉(1980-) , 男, 江苏徐州人, 讲师, 研究方向为数据挖掘、Web 个性化; 骆斌(1967-) , 教授, 研究方向为数据库、人工智能。