

基于 JSP+ SQL Server 的管理信息系统性能优化

赵 洁^{1,2}, 甘志刚²

(1 广东工业大学 经济管理学院, 广东 广州 510520; 2 华南理工大学 计算机科学与工程学院, 广东 广州 510640)

摘要: 以毕业设计(论文)管理信息系统的开发为基础,对现有的系统性能优化技术和方法进行研究,分析如何准确利用这些技术和方法,并利用相关工具对它们及优化后的结果进行定量分析。

关键词: 管理信息系统; 性能优化; 数据库

中图分类号: TP302 1

文献标识码: A

文章编号: 1007-7162(2007)04-0067-06

在高校基金项目的支持下,开发了基于 JSP+ SQL Server 的毕业设计(论文)工作管理信息系统。对系统性能进行优化时,参考了大量文献,使用了基于 JSP 及优化数据库的经验和方法。而在实践中,发现使用这些经验和方法,尤其是数据库的优化方法,如索引的建立、模式的优化设计时存在一些问题:

- 1) 在未深入了解常用重要方法的内部原理情况下难以正确使用这些方法;
- 2) 多种优化方法是相互矛盾的,难以准确地判断使用其中的哪一种方法;
- 3) 大多数方法没有给出定量的分析,难以分析使用该方法后效率是否提高;
- 4) 采用某些方法时,往往得不到预期的效果,效率反而下降了;
- 5) 很多优化方法的验证基于超大规模数量级的数据库($10^5/10^6$),与实践时数据库的数量级相差甚远,难以直接套用方法。

本文根据管理信息系统开发的实践,结合所采用的 JSP+ MS SQL Server 技术方案,对一些常用的优化性能方法及工具进行深入原理剖析,提出更实用有效的系统综合优化方法,并结合实践给出实例,对同类型的系统的性能优化工作具有很好的参考价值。

1 基于 JSP 的系统性能优化

本系统是采用 JSP + Tomcat 5.0 基于数据库开发的 Web 应用系统,系统性能优化主要从以下两方面考虑。

1.1 基于缓冲池的优化

传统的数据库连接方式(通过 Driver Manager)

中,一般在开发基于数据库的 Web 应用系统时,按以下步骤进行:在主程序(如 Servlet Beans)中建立数据库的物理连接;执行 SQL 操作存取数据;关闭连接。该模式存在很多问题,较明显的缺陷是造成系统性能低下,在多层结构的应用程序环境中这种耗费资源的操作对系统性能有很大影响。

要解决以上问题,可以通过连接池(connection pooling)技术。其基本思想是在应用程序启动时预先建立足够的数据库连接,将之组成一个连接池,连接的建立、断开等由连接池自身管理,动态地对连接进行申请、使用和释放。当并发请求多于连接数,将在请求队列中等待。该技术可以帮助解决基于数据库的 Web 系统中所存在的瓶颈,并实现动态伸缩、负载均衡等功能。通过设置参数来控制连接数的上限、下限、每个连接的最大使用次数和最长空闲时间等。通过其自身管理机制可监视连接的数量、使用情况等^[1]。工作原理如图 1 所示。

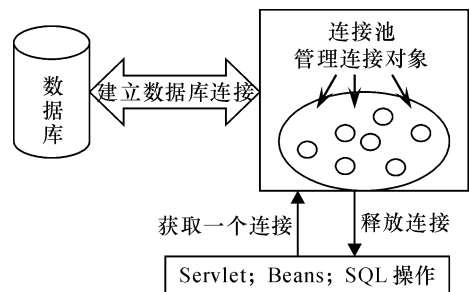


图 1 缓冲池工作原理

本系统采用 Tomcat 5.0 服务器,直接使用服务器提供的连接管理服务,这只需要在应用服务器中做适当的配置就能使用连接池:

- 1) 在 admin 页面设置数据源基本参数;

收稿日期: 2007-01-19

基金项目: 广东工业大学高教研究基金资助项目(2006C02)

作者简介: 赵 洁(1979-),女,讲师,博士研究生,主要研究方向为计算机应用技术。

2) 更改: \Tomcat 5 0\conf\Catalina\localhost目录下相应的.xml 把原来的改为

<ResourceLink name="数据源名字" global="数据源名字" type="javax.sql.DataSource"/>;

3) 更改: \Tomcat 5 0\conf下的 web.xml 加入图 2 语句;

4) 在相应的 webapp 下面的 web.xml 也加入图 2 语句.

```
<resource-ref>
<description>DB Connection</description>
<res-ref-name>数据源名字</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

图 2 数据源配置

1.2 降低网络流量的考虑

考虑到系统基于 Web, 如果应用程序设计不当会造成过多的网络 I/O 操作, 严重影响整个系统的性能. 应尽量减少对数据库的传输, 使数据处理尽可能在数据库端批量执行, 然后只传送处理后的结果, 达到减少网络 I/O 的目的^[2]. 方法有多种, 例如使用存储过程, 它执行速度快, 可减少网络传输、处理及存储的工作量.

在系统中, 在服务器端大量使用存储过程、触发器和 SQL 函数, 例如计算学生综合评定成绩. 为提高效率, 关系中保留了 lastScore(综合评定成绩)、s1(评分 1)、s2(评分 2)、s3(评分 3) 4 项冗余: (student_no, s1, s2..., s1..., s3, s4, lastScore). 综合成绩是由多项成绩按比例计算的导出项, 计算公式如下:

$$s1 = s11 + s12 + s13 + s14 + s15, s2 = s21 + s22 + s23 + s24 + s25, s3 = s31 + s32 + s33 + s34 + s35$$

(s_{ii} 为评分细项), lastScore = s1 × 比例 1 + s2 × 比例 2 + s3 × 比例 3 + s4 × 比例 4

如果前端写嵌入式 SQL, 综合成绩的计算使用上面的公式, 每次查询需要传送 20 项数据 (类型设置为 tinyint), 大小为 20 字节, 计算成绩所需要传输的数据量为 20 × n. (n 为学生数量) 为避免更新异常, 设置了大量触发器, 当成绩 s_{ii} 更新, 将触发 s_i 更新, 进而触发 lastScore 的更新. 编写存储过程 updateLastScore 供触发器调用, 读出 s_i 所占比例, 按公式重新计算元组的综合评定成绩. 更新操作非集中式, 所以此方式下的更新的时间非常短, 基本可以忽略不计, 查询所有学生综合评定成绩时只需要传送 1 项数据, 所需传数据量为 1 × n, 此传输量仅为上述

方法的 1/20. 无论是传送时间还是计算时间, 都是远远少于前端嵌入式 SQL.

2 基于数据库的系统性能优化

SQL Server 数据库的优化通常分为 3 大类, 第一类主要是硬件方面的优化, 包括对网络、硬件调整, 最常用手段是升级硬件, 要考虑服务器的硬件性能及网卡、交换机等; 第二类是系统参数的优化, 主要对操作系统、数据库等参数进行配置. 第三类主要集中在合理建立数据模型, 优化 SQL 语句, 合理建立索引等方面. 下面主要针对后两类进行讨论.

2.1 系统参数的优化

MS SQL Server 的设计目标是使数据库服务器成为一种几乎不需要优化就可以达到最佳性能的自优化系统, 它的内存管理组件消除了对可用内存进行手工管理的需要, 可动态分配它与其它程序应分配的内存量. 一般只需为服务器打好相关补丁, 关闭不需要的服务, 使数据文件的分布尽可能合理, 减少 I/O 的争用. 而通过修改内存配置选项可在一定程度上优化服务器性能, 如 min server memory, max server memory, max worker threads 等参数, 可用于配置内存使用并影响服务器性能. 但对于这些参数, 尤其是高级参数, 应该深入了解参数的含义和修改带来的潜在影响. 如配置有误, 会影响系统性能^[3].

2.2 充分利用系统优化工具

大多数文献提出要利用系统的优化工具, 但没有清晰说明具体的方法. 下面说明如何利用 SQL Server 所提供的优化工具, 如 showplan、I/O 统计和索引优化向导等工具, 它们可以为优化系统性能给出具体的量化数据分析.

1) 解读 showplan 与 I/O 统计数据

showplan 中有多种图标, 表示扫描表、哈希表探测以及聚簇、非聚簇索引查询等, 所以使用该工具应该具备 DBMS 查询处理和优化的基本知识^[4]. 运行查询时, 所使用的数据可能首先载入数据高速缓存中, 如再执行该查询, 使用这些数据可减少物理读入操作数量. I/O 操作越少, 查询执行速度越快. I/O 统计工具包括扫描计数、逻辑读、物理读、预读 4 项统计. 下面结合例子, 说明如何使用 showplan 和 I/O 统计来理解 SQL 语句的处理.

student_info 表中只有 student_no 索引, teacher_info 表中只有 teacher_no 索引.

方案 1: select * from student_info where student_no = (select teacher_info_no from teacher_info

where teacher_name= '张明'). 如图 2所示.

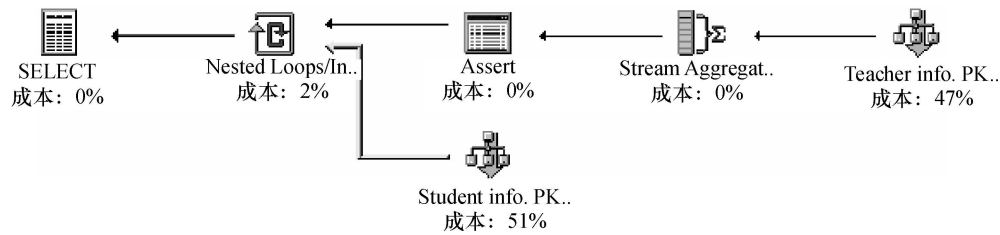


图 2 方案 1的 showplan图示

方案 2 select* from student_t_inf as s teacher_inf as twhere s student_tch_no= t teacher_info_no and teacher_name= '张明'. 如图 3所示.

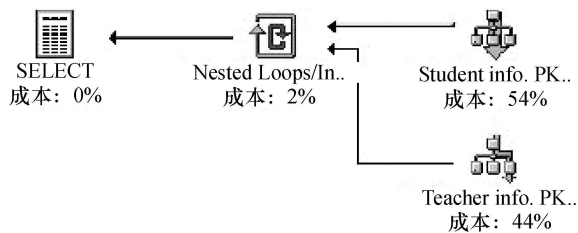


图 3 方案 2的 showplan图示

方案 1从 teacher表中选出符合要求的记录, 并验证确保为一条, 作为外循环; 方案 2从 student中选出的所有元组作为外循环, 从 showplan的图形可见方案 1大大优于方案 2

从下面的 I/O统计也可以看出方案 1的扫描次数与逻辑读远远少于方案 2 优劣立见.

方案 1的 I/O统计: 表 'student_info'. 扫描计数 1、逻辑读 6次、物理读 0次、预读 0次; 表 'teacher_inf'. 扫描计数 1、逻辑读 2次、物理读 0次、预读 0次.

方案 2的 I/O统计: 表 'teacher_info'. 扫描计数 378、逻辑读 756次、物理读 0次、预读 0次; 表 'student_info'. 扫描计数 1、逻辑读 6次、物理读 0次、预读 0次.

2)使用索引优化分析器

查询优化器负责为查询产生最优化的执行计划, 可用于决定最有效率的执行计划: 确定是否存在索引并评估索引对查询的有用性; 确定可使用哪些索引和列来减少查询所检查的行的数目; 确定处理联接操作的最佳策略, 例如以何种顺序联接表和使用何种联接策略; 对给定查询的多种计划使用基于成本的评估, 以选择出最有效率的计划; 创建列统计以提高查询性能.

创建对查询优化器有用的索引的原则: 确定所有查询的优先权; 确定每个查询的 WHERE 子句的每个部分的选择性; 确定是否创建索引; 确定应该被

索引的列; 确定组合索引的最佳列顺序; 确定对其他索引的需求程度; 创建索引后, 测试每个优先级最高的查询的性能.

对于某些查询, 需要使用优化器多次, 否则向导无法推荐可使用的索引. 为多个查询选择合适的索引比为单个查询困难, 因为对一条查询是最佳的索引对其他查询未必是最佳的. 目标是通过评估 I/O, 使所有高优先级的查询获得可接受的性能^[3].

2 3 基于数据库索引设计的优化

提高数据库速度最快的方法是建立索引. 文献给出数十种经验及方法^[4 6], 但大多没有分析原因和给出适用的条件, 容易在使用时出现引言所出现的问题. 必须理解 SQL Server数据库的内部数据结构, 才能结合实际合理选择簇集索引以及非簇集索引, 极大地提高 SQL Server的查询性能, 提高用户的响应时间. 下面针对簇集与非簇集索引进行讨论^[3 5].

1)簇集与非簇集索引的原理

簇集索引是 B- 树结构, 最底层 (叶子层) 为带有表的数据页, 表创建簇集索引时, 表内的行将基于簇集索引键的逻辑顺序物理排列存储. 每表只能有一个簇集索引, 其他的索引称为非簇集索引. 如果表没有簇集索引, 其数据按堆 (heap) 方式存储. 非簇集索引也是 B 树结构, 与簇集索引有两个重大差别: 数据行不按非簇集索引键的顺序排序和存储; 非簇集索引的叶层不包含数据页, 具有完全独立于数据行的结构.

对于非簇集索引来说, 叶层页面中带有索引键值外, 还带有行指示器. 行指示器根据是否存在簇集索引分为两种情况. 如果表使用了簇集索引, 则该表的行指示器就是该表的簇集索引. 对于堆, 行指示器就是行的 ID 标识 (RID), RID 由文件 ID、页面号和该行页数构造而成, 即指向行的指针.

选择不同字段作为簇集索引关键字时, 其非簇集索引大小也跟着变化, 因为非簇集索引的行指示器中包含簇集索引的关键字值. 因此建立簇集索引

时,尽量使用短的关键字值。

2)作用与成本

簇集索引对于经常要搜索范围值的列特别有效。使用簇集索引找到包含第一个值的行后,便可以确保包含后续索引值的行在物理相邻。例如,应用程序执行的一个查询经常检索某一日期范围内的记录,则使用簇集索引可以迅速找到包含开始日期的行,然后检索表中所有相邻的行,直到到达结束日期。

但簇集索引只对索引键有关的查询有利,对其他与簇集索引键无关的查询不利。建立的开销很大:整个文件要重新组织,重组时记录要重新定位,原来建立在文件上的索引都需重建;使用簇集索引时非簇集索引也会变大;使用簇集索引要保持数据页按照索引关键字值的逻辑顺序物理排列存储。但是,除了表很小的情况下,一般都推荐每个表有一个簇集索引。

3)合理使用索引

(1)避免在自增 ID 列上建立簇集索引:一般情况下,SQL Server 默认在主键上建立簇集索引,但有时候这样会使簇集索引成为浪费。特别当在表中建立一个 ID 列来唯一标识每个元组时(ID 列是自动增大的,步长一般为 1),如果将此列设为主键,SQL Server 会默认此列为簇集索引。而实际上,ID 号是自动生成的,不能确定每条记录的 ID 号,难以使用 ID 号来进行查询,这就难以发挥簇集索引的效用。

(2)建立索引的基本准则:在频繁修改的数据表上,最好限制该表使用一个簇集索引和不超过 3 到 4 个非簇集索引。涉及大量事务处理的表索引应限制在 3 个以内。适合建立簇集索引的列:①最频繁使用的、用以缩小查询范围的字段,例如日期字段;②最频繁使用的、需要排序的字段;③长度比较短的索引关键字;④数据取值范围不大的列。不适合建立簇集索引的列:①数据取值范围很大的列;②频繁更新的列。

适合建立非簇集索引的列:①用于表联合使用的列;②用户范围查询的列;③用于 order by queries 的列;④用于 group by queries 的列;⑤用于合计功能的列。不适合非簇集索引的列:①行数很少的表;②数据取值范围大的列;③很宽的列(索引列的宽度要小于 25 个字节);④事务负载大(大量的插入删除操作)但决策支持操作很少的表;⑤与查询列无关的列。

4)适当建立复合索引

适当建立复合索引,可减少 I/O,快速地检索数据。复合索引只能是非簇集索引,包含常用查询引用

的所有列。当查询被索引覆盖,查询优化器不需要访问数据页,索引视图可以预聚合数据,提高查询性能。

影响索引覆盖查询能力的因素:索引必须包含所有被引用的列;索引中的非首列对覆盖查询也有用的;WHERE 子句不是必需的,查询优化器会扫描所有叶级页;若使用包含 WHERE 子句中所引用列的非簇集索引比簇集索引需要更少的 I/O 操作,则使用非簇集索引覆盖查询;可以联接多个索引以覆盖查询。

建立复合索引的原则:用合适的列组成索引;尽量减小索引键的大小;维护行和键值大小的比率。

2.4 基于数据库逻辑模式设计的优化

这方面的方法很多,如减少连接运算,减小关系的大小和数据量、常用的统计字段(统计、平均值)加入数据库中等常用方法。同时应充分利用 DBMS 提供的自动维护数据完整性的功能,如 CHECK 等约束^[6]。

下面讨论节省数据库的存储空间对数据库性能的影响。正确选择属性的存储类型,不同的属性长度会影响性能,例如,当把学生班别用 2~3 位数字表示,使用不同类型存储,相关的 I/O 统计输出:

扫描计数 1、逻辑读 11 次、物理读 0 次、预读 0 次。

改为 smallint 类型存储后,运行查询时 I/O 统计输出为:

扫描计数 1、逻辑读 10 次、物理读 0 次、预读 0 次。

对比前后两个统计数据,逻辑读的数量下降了,虽然这种下降并不明显。但当对每个属性作同样考虑时,所提高效率是相当可观的。另结合假属性的使用^[4],能大大节省存储空间。

2.5 查询优化

对查询的微小调整可能导致查询性能的大幅度改善。一般对于应用程序优化特别是 SQL 语句的优化,可以改善系统的性能,提高用户的响应时间,有时甚至可以得到 80% 的系统性能的提升^[7-8]。

1) SQL Server 的自动查询优化原理

SQL Server 具有自动优化功能,它可以规划出 where 子句中的搜索条件并确定哪个索引能缩小表扫描的搜索空间。如果一个查询语句写成有效使用搜索参数(SARG——Search Argument)^[8],查询优化器就可以使用表中字段的索引来查询数据,而不必直接查询记录中的数据,如带有 =、<、<=、>、>= 等操作符的条件语句可以直接使用索引。有效地

SARG 撰写原则: (1)有 WHERE 条件; (2)使用 Exactmatch (=); (3)使用区间查询 (between >, <, >=, <=); (4)使用 AND; (5)适当的 LIKE 使用方式. 使用方式如下:

column inclusive_operator < constant or variable
> 或是 < constant or variable> inclusive_operator column

例如以下都是好的 SARG 使用方式: name = 'jones', salary > 40000, 60000 < salary, name = 'jones' AND salary > 100000, Like 'LEW IS%'.

Non-SARG 使用方式: NOT, !=, < >, ! >, ! <, NOT EXISTS NOT IN, and NOT LIKE, LIKE '%LEW IS%', 其中该种 LIKE 使用方法将会造成全表搜索, 将无法提升性能.

如果一个表达式不能满足 SARG 的形式, 将无法限制搜索的范围, SQL Server 必须逐行扫描来判断是否满足 WHERE 子句中的所有条件. 所以索引对于不满足 SARG 形式的表达式来说是无法起作用的.

2) SQL Server 的自动查询优化

有些文献认为, SQL Server 执行下面的语句, 效率会不一样:

(1) SELECT FROM customer WHERE city = 'beijing' AND fname = 'li'

(2) SELECT FROM customer WHERE fname = 'li' AND city = 'beijing'

假设表中存在 100 万条记录, 其中 city = 'beijing' 有 10 万条, fname = 'li' 的为 2 万条. 单从语句表面看, 确实不一样. (1) 执行第一个条件结果返回一个 10 万行的临时表, 然后再从中进行选择, 从而得到最终结果. (2) 先得到一个 2 万行的临时表, 再得到同样的结果. 然而这种担心不必要. 以上的语句满足 SARG 形式, SQL Server 能实现自动优化. 所以不需考虑如何类似的顺序调整, 而将精力放在其他方面.

3) 优化语句的经验与方法

(1) 按需取字段: 字段选择按“按需所取”原则, 避免大量使用 "select* ". 例如, 表中有 8 000 条记录:

select student_no, student_name, student_class
from student_info (执行需 513 ms);

select student_no, student_name from student_info
(执行需 423 ms);

select student_no from student_info (执行需 220 ms).

每少提取一个字段, 数据的提取速度就会有相应的提升. 提升的速度还依赖于舍弃的字段的大小.

(2) 把 Non-SARG 改变为 SARG

在某些情况下, 可以将 Non-SARG 语法改写成 SARG 的指令, 加强系统对索引的使用, 如:

Non-SARG: WHERE SUBSTR NG (au_hame, 1, 2) = 'Mc'

改写成 SRAGWHERE au_hame >= 'Mc' AND au_hame < 'Md'.

或是 Non-SARG WHERE ABS(change_in_quantity) < 4 改写成

SARGWHERE change_in_quantity < 4 AND change_in_quantity > -4

(3) 注意 Like 的形式: Like 语句是否属于 SARG 取决于所使用的通配符的类型.

如: student_name like '陈%', 这就属于 SARG. 而: student_name like '%张' 或者 tudent_name like '%张%', 不属于 SARG. 原因是通配符% 在字符串的首部使得索引无法使用.

(4) EXISTS 的效率与 IN 一样: 很多文献提出用 EXISTS 替代 IN. 但经实验, 发现两者具同样效率.

select* from task where teacher_no in (select teacher_no from teacher_info where teacher_name = '李清')

表 'task'. 扫描计数 1、逻辑读 2 次、物理读 0 次、预读 0 次.

表 'teacher_info'. 扫描计数 1、逻辑读 21 次、物理读 0 次、预读 0 次.

select* from task where exists (select* from teacher_info where task teacher_no = teacher_info teacher_no and teacher_name = '李清')

表 'task'. 扫描计数 1、逻辑读 2 次、物理读 0 次、预读 0 次.

表 'teacher_info'. 扫描计数 1、逻辑读 21 次、物理读 0 次、预读 0 次.

3 结语

结合基于 JSP+ SQL Server 的高校本科毕业 (论文) 开发, 对多种优化系统的方法进行深入剖析, 如缓冲池的配置使用、减少网络 I/O 的考虑, 并深入探讨基于数据库的系统优化, 介绍如何充分利用 SQL Server 提供的功能、系统优化工具, 并使用它们来挑选、验证繁多的优化方法, 利用它们提供的数据来量化使用优化方法后的效果, 基于系统给出详实的测试数据, 通过了解 SQL Server 内部数据组织

方式,选择合适的索引,编写有利于性能提高的 SQL 语句,在查询优化器中不断测试与优化.实践证明,使用这些措施可大大提高管理信息系统的性能.

参考文献:

- [1] 倪显利. 数据库连接池浅议 [EB/OL]. [2006-08-21]. http://www.netadm.in.com.cn/kt_article_show.php?article_id=7141&categ_code=10011006
- [2] 吕华,杜忠军. 数据库性能优化 [J]. 计算机应用, 2003, 23: 172-174.
- [3] MARK Spenk. SQL Server 2000 数据库管理员指南 [M]. 北京:机械工业出版社, 2002: 323-363.
- [4] 王能斌. 数据库系统原理 [M]. 北京:电子工业出版社, 2000: 412-413.
- [5] FREEDK. SQL Server2000索引结构及其使用 [EB/OL]. [2006-12-04]. <http://www.hacker.cn/Get/gjrz/0622710310892794.shtml>
- [6] 尹萍. SQL Server数据库性能优化 [J]. 计算机应用与软件, 2005, 22(3): 51-53.
- [7] 海量数据库的查询优化及分页算法方案 [EB/OL]. [2006-12-4]. <http://news.softhouse.com.cn/news/show/4881.html>
- [8] 李增强,管文强,王海勇. Microsoft SQL Server2000数据库系统设计优化 [J]. 计算机系统应用, 2006(10): 88-90.

The Performance Optimization of Management Information System (MIS) Based on JSP+ SQL Server

ZHAO Jie^{1,2}, GAN Zhigang²

(1. School of Economics and Management, Guangdong University of Technology, Guangzhou 510520, China)

2. School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China)

Abstract Based on the development of the University Graduation Project(Thesis) MIS, this paper goes further into the theory of present optimizing technologies and methods, analyzes how to precisely use these technologies and methods, and uses relative tools to make quantitative analysis of them and the results after optimization.

Key words management information system; performance optimization; database

简讯

全国第 12 届可拓学年会征文启事

全国第 12 届可拓学年会定于 2008 年 8 月 26 日~8 月 28 日在北京大学召开。会议将交流可拓学的理论成果和应用成果,讨论可拓学发展的有关问题。新学科可拓学的建设和发展,需要众多学科的学者参与,我们欢迎各个领域的专家的专家学者、管理人员和工程技术人员参加这次会议。现就会议征文的有关问题通知如下。

1. 征文内容 1)理论研究: (1)基元理论; (2)可拓数学; (3)可拓逻辑; (4)可拓学的哲学基础; (5)可拓学与其他学科的交叉融合。 2)应用研究: 与各专业领域相结合产生的可拓工程理论、方法与应用的研究,如(但不限于此); 人工智能、计算机、自动化、控制、检测、管理、经济等。 3)应用成果: (1)工程技术、企业管理、经济领域、科学研究以及各行各业利用可拓工程方法的应用成果; (2)关于可拓工程的应用软件和硬件产品的研制报告。

2. 征文要求 1)论文必须文笔精练、资料可靠,字数不超过 7 千字,并附中英文摘要和关键词; 2)论文应是新的研究成果,且未在国内外学术会议或刊物上发表过; 3)严格按照科技论文的撰写论文; 4)优秀论文将推荐到核心期刊正式发表; 5)征文截止日期: 2008 年 3 月 31 日。请于此日期前把论文全文用电子邮件发送到年会筹备组,以便安排审稿。

会议筹备组设在广东工业大学研究所。

email extenic@vip.163.com; extenics2003@gdut.edu.cn

电话: 020-37626179 传真: 020-37626276

地址: 广东东风东路 729 号广东工业大学可拓工程研究所 (邮编: 510090)

联系人: 杨春燕研究员

中国人工智能学会可拓工程专业委员会