第36卷 第20期 计 算 机 工 程 2010年10月
Vol.36 No.20 Computer Engineering October 2010
·软件技术与数据库· 文章编号：1000—3428(2010)20—0071—03 文献标识码：A 中图分类号：TP311.52

集中式系统的分布式通用查询框架设计

白剑斐，叶绿宽，杨文钧，丁 宁
(广东省国家税务局信息中心，广州 510080)

摘 要：针对集中式系统的查询功能存在设计复杂、查询速度慢、时效性差等问题，提出一种建立在分布式数据源基础上的通用解决方案，将大量数据进行分发，并采用反向的数据访问方式，以提高查询效率，增强扩展性，降低程序复杂度。实现一种可继承的组件装配式查询方案，可以简化开发部署工作，对不断变化的各种形式的业务需求做出快速响应。

关键词：集中式系统；通用查询框架；依赖注入；模板化设计

Design of Distributed General Query Framework in Centralized System

BAI Jian-fei, YE Lv-kuan, YANG Wen-jun, DING Ning
(Information Center of Guangdong Provincial Office SAT, Guangzhou 510080, China)

【Abstract】 The query function for a centralized system is usually hard to design and costs a large number of server resources. A solution based on distributed data source is discussed to solve these problems, which uses an inverse data access method to improve querying efficiency and reduce programming complexity. It also contains an inheritable module-assembling query scenario to simplify the process of development and deployment, in order to meet the complexity of business requirement.

【Key words】 centralized system; general query framework; dependency injection; template design

1 概述

随着计算机硬件性能的提高和业务模式的成熟，越来越多的组织选择建设大集中式业务系统。这种模式将业务系统集中部署在总部高性能的应用服务器和数据库上，各地分支机构用户通过浏览器远程访问进行业务操作。相比以前分支机构自己搭设业务系统的做法，这种模式提升了业务规范性，保证了数据安全，并大大降低了系统维护的成本和风险。广东国税 2006 年上线全国统一的中国税收征管信息系统(CTAIS)后，征管水平得到了大幅提升[1]。由于集中式业务系统的数据存放在统一的数据库中，随着业务的持续开展，该模式在查询功能和数据应用方面的弊端逐步显现出来，主要体现在以下 3 个方面：

(1)查询速度越来越慢。部分核心表的记录数快速增长，甚至达到千万乃至亿级，导致基于这些表的查询速度无法忍受。为了解决这一问题，往往需要投入大量资金不断升级硬件，并大量采取索引、表分区等数据库优化技术以及加工表、数据仓库等 BI(商业智能)技术。这大大增加了系统的复杂性，却只能将查询时间控制在分钟级[2]。

(2)查询时效性差。由于查询功能大量采用加工表、数据仓库等技术，生成加工数据需要大量占用系统资源和时间，因此数据加工往往在晚上和周末进行，生成的数据可能滞后一天、一周甚至一月，这经常造成用户苦等数据的状况，不利于缩短业务反应时间。

(3)数据的生产者拿不到第一手的数据。基层是数据的生产者，集中式系统上线后，出于安全考虑，基层用户往往无法直接访问数据库，只能使用系统提供的查询功能。这些功能往往无法满足批量统计等各种个性化查询需求，用户需要从分散的功能模块中获取数据并手工进行计算，大大增加了工作量。新的查询需求从需求分析、设计、测试到最终部署，需要一个非常长的周期。

为了解决这些问题，广东国税在征管辅助系统二期设计开发的过程中，对查询模块进行攻关，设计出了一套分布式、方案化的通用查询框架。该框架的主要设计目标是：

(1)高性能。实现全局统计到明细数据的多用户并发查询，并将查询时间缩短到秒级。

(2)通用性。能够满足各种查询类型的需求，为用户提供丰富的交互手段。

(3)快速开发部署。开发者可以通过拖拽和配置的方式短时间内开发出新的查询功能，并上传至服务器完成部署，无须进行编译和服务器重新配置。

本文查询框架在 J2EE 平台上实现，采用分层式设计思想，涉及 Spring、Groovy、Ajax 等主流服务器和客户端技术。

2 分布式数据源

为了从根本上解决数据查询的效率问题，需要建立一组分布式数据源。应用服务器在将数据写入生产数据库的同时，将同样的数据异步写入主查询数据库。主查询数据库定期通过 ETL 的方式将数据按生产者的区域进行增量分发[3]。由于每次分发时传输的数据量较小，因此可以每半天甚至每小时同步一次，大大提升了数据时效性。查询服务器维护到查询

作者简介：白剑斐(1985－)，男，工程师、硕士，主研方向：大数据量核心业务系统设计，B/S 系统的用户体验优化；叶绿宽、杨文钧、丁 宁，工程师

收稿日期：2010-04-20 **E-mail：**kanferbai@163.com

数据库和各级分发库的连接池。系统运行时，查询框架从主查询数据库中获取用户信息和系统配置信息，根据查询的内容从各级分发库中获取数据。当某一级分发库不可用或会话数过多时，查询框架自动从它的上一级获取数据。数据分发过程如图 1 所示。

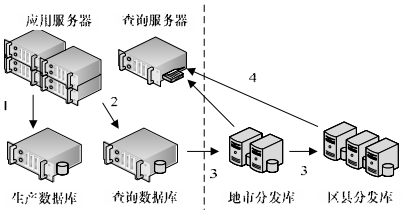

图 1　数据分发过程

在建立分布式数据源后，由于每个数据库中的记录数较少，可以直接进行访问，不再需要设计复杂的加工表和数据仓库，提升了查询的性能和时效性。用户可以在一个功能中进行从宏观统计到微观明细的各种粒度查询，大大简化了用户操作，提升了查询功能实用性。数据分发是一个单向的过程，因此，可以将分发数据库的访问权限开放给本级的技术部门，以开展进一步的数据应用工作。

3　反向数据访问

数据访问层是查询框架的核心，承担着快速、高效获取和传输数据库数据的任务。为了更高效地在服务器与客户端之间传递数据，满足用户对交互性的更高要求，需要设计 Dataset，它是一个跨越服务器和客户端的对象化数据集，其核心是一组元数据，包括数据集的标示符、列的定义、参数及属性等。由于服务器和客户端都由元数据定义了数据结构，因此数据可以直接连接成以逗号分隔的字符串，通过长轮询的流式方式压缩传输，无需再将每一条记录转换成对象，再通过 XML 或 JSON 的方式传输[4]。客户端可以直接使用 Dataset 的元数据解析接收到的字符串，接收完一组记录即可在页面上展现出来，无需等待 XML 或 JSON 传输完毕后再遍历其中的对象属性以填充记录集。这大大加快了数据展现的时间，降低了传输的数据量，减少了对服务器资源的占用。在大量数据传输测试中，用户只需花费以往 1/5 的时间就看到了一屏数据，剩下的数据在用户向下拖动滚动条时继续加载并展现出来。在整个数据加载过程中，服务器与客户端的通信量降低了 30%以上。客户端的 Dataset 还实现了多种数据操纵方法，用户对数据进行排序、筛选、查找、统计等交互性操作时，无需再向服务器发送任何请求。Dataset 在客户端被创建，一个页面可以有多个独立的 Dataset。用户修改查询表单中的条件时，首先引起 Dataset 中参数的改变(如图 2 中的 1)。用户发出查询请求后，客户端 Dataset 元数据而不是查询条件本身被提交(如图 2 中的 2)。服务器接收到提交的 Dataset 元数据后，根据其中的信息构造一个同样的 Dataset。查询框架根据服务器端 Dataset 标示符查找访问数据源的方式，将其注入到 Dataset 中(如图 2 中的 3)。

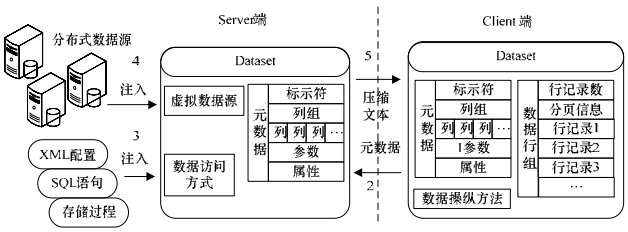

图 2　反向数据访问过程

查询框架提供 3 种数据访问方式：

(1)XML 配置文件。查询框架提供一种向导式、图形化的数据访问配置方式，开发人员可以通过鼠标选取的方式设置查询表、字段、连接关系等信息。

(2)SQL 语句。开发人员可以编写一个十分复杂的 SQL 语句来获得查询内容。

(3)存储过程。开发人员通过配置的方式访问数据库中的存储过程。存储过程需要实现预先定义的接口以供查询框架调用。通过这种方式可以利用数据库强大的功能，通过临时表、游标、缓存等特性进一步优化性能。

数据访问方式只存在于服务器端 Dataset 中，用户无法在客户端中查看到数据库表结构，也无法通过篡改元数据的方式获得查询结果之外的数据。对于经常使用的功能，查询框架还可以自动将查询结果缓存起来，当用户再次提交数据访问请求时，直接将缓存注入到 Dataset 中，瞬时返回数据。

服务器端 Dataset 得到数据访问的方式后，将查询请求发送到虚拟数据源。查询框架根据查询条件和用户机构权限信息将连接池中的实际数据源注入到虚拟数据源中以获得数据(如图 2 中的 4)。这个过程对用户和开发者都是透明的。用户可能在从总部统计数据到基层明细数据的下钻过程中连接了多个实际数据源以保证系统持续的快速响应。而对开发者来说，只有一个统一的数据源，不需要为每一个机构层次设计一套访问逻辑，也不需要创建、连接、维护和断开数据源。

在整个数据访问的过程中，访问数据的模式被反转了，条件被提交到了客户端，数据结构由客户端向服务器传送，访问方式被注入进来，而数据源在获取数据时才得到。这种方式将数据访问的各个环节之间解耦，以充分利用分布式数据源的优势，并大大方便了整个框架的修改和扩展。服务器端只需要设计成一个精简高效的通用数据引擎，无须包含任何业务逻辑。数据引擎的实例在完成一次数据访问任务后就可以销毁，不需要贯穿整个客户端页面生命周期。这非常适合并发数高、数据量大的查询功能。

4　装配式查询方案

查询功能往往界面、逻辑上相似，因此，可以将相同的部分抽取出来形成通用查询处理器，而将与业务紧密相关的数据查询语句和展现样式以 XML 配置文件的方式分离出来，形成查询方案。很多系统将查询方案分为条件、查询语句、展现表格等几部分，运行时采用“填空”的方式在页面上展现出来。随着业务的快速发展，这种方式死板的缺点愈发明显，让用户形成通用等于不好用的观念。很多主从表、下钻、统计明细清册等类型的查询难以固化成某一种模板，业务需要更灵活的展现形式，更多交互特性。

本次项目改变了“填空”型查询方案的做法，而是定义了一个视图模型，它是 HTML 页面的超集，其中封装了界面逻辑和操作逻辑的对象，如用于数据访问的 Dataset 和一套精心设计的用于查询条件输入及结果展现的控件，包括具有多种交互方式的表格、多选下拉列表、弹出式选择器等[5]。用户可以自由在查询方案中装配一个或多个 Dataset 和控件，它们包含大量的可修改属性和事件响应方式。视图模型还定义了用于访问查询方案解析生命周期各个阶段的监听器，开发者可以在其中编写动态语言脚本，定制服务器端查询方案的解析过程，以满足业务的个性化要求。开发者还可以通过直接向客户端输出 HTML 和 JavaScript 代码，增加客户端的元素和功能。

查询方案可以存储在文件系统或数据库中，当用户请求一个新的查询功能页面时，方案解析器才读入查询方案XML，这样在部署和修改查询配置时，不会对服务器造成任何影响(如图 3 中的 1)，无需停机进行。对于经常被访问的功能，查询框架会自动将它们缓存起来以提升性能。方案解析引擎将查询方案解析为 HTML 文件，在解析过程中，监听器上的动态语言脚本被执行，Dataset 和控件被翻译成 HTML 片段和 JavaScript 代码。解析完成后，服务器将 HTML 文档发送到客户端，方案解析引擎的实例被销毁(如图 3 中的 2)。客户端浏览器根据接收到的 HTML 文件中的代码，创建查询方案中定义的 Dataset 和控件对象(如图 3 中的 3)。数据展现控件绑定在 Dataset 上，当控件值被改变时，Dataset 的参数相应改变，而 Dataset 返回数据时，控件的值也随之改变，这样大大简化了客户端业务逻辑，无须编写任何代码。当 Dataset 中的元数据被提交，服务器端 Dataset 创立后(如图 3 中的 4)，查询框架根据 Dataset 的标示符，向方案解析引擎查询数据访问方式，并将其注入到 Dataset 中(如图 3 中的 5)。Dataset 从分布式数据源获取数据(如图 3 中的 6)，发送到客户端(如图 3 中的 7)。客户端 Dataset 接收到查询结果后，刷新与之绑定的控件(如图 3 中的 8)，将结果展现出来。

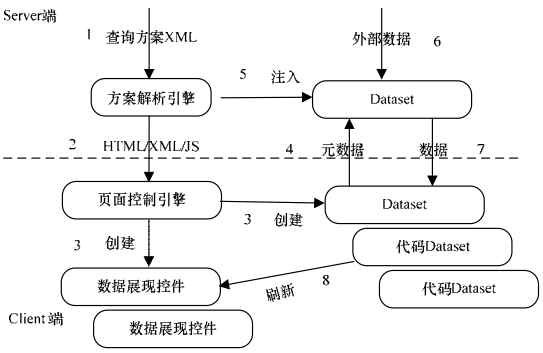

图 3　查询方案解析过程

查询结果展现时往往需要将其中的代码转换为文字。为了简化开发过程的工作量，进一步提升系统性能，本次项目没有采用在编写 SQL 查询语句时连接代码表的方法，而是在系统主界面上定义了一组用于查询代码表的被动加载 Dataset。当查询结果字段在展现前需要进行转码时，对应的代码 Dataset 被加载，并通过哈希连接的方式快速匹配。由于代码 Dataset 加载时可以根据用户的权限和功能模块等信息只加载一小部分代码表内容，因此在涉及到机构、人员等数据的代码转换时，这种匹配往往比在数据库中进行连接操作更省时。当用户翻页或访问包含此字段的其他功能时，由于代码 Dataset 已经被加载过，可以直接进行匹配，进一步提高了性能。

5　继承式方案设计

装配式查询方案提供了灵活强大的可定制性，因此开发者需要为各个控件设置属性和事件处理代码，开发量较大，且容易造成不同的查询功能间操作方式和界面的差异。为了解决这一问题，将开发者从机械重复劳动中解放出来，本项目参照面向对象的思想，为查询方案设计了继承的特性。这种继承是单向的、基于通配符匹配的、覆盖式的。

首先，设计基方案，其中定义以“*”号命名的查询条件表单、数据展现表格、弹出式子窗口等控件，设置了其大小样式等信息。接着，为每种不同的展现形式设计了样式方案，如部门统计、人员清册等，它继承了基方案，并包含一系列以前缀或后缀加“*”号命名的控件，如用于展现部门统计信息的表格*_BM 等，其中定义了多个名称包含通配符的列，如部门 BM、金额*_JE、比例*_BL 等，列的宽度、对齐方式、代码转换等属性在其中被设置。然后，为相似的功能模块设计了功能方案，它继承了样式方案，并包含与业务相关的属性和事件处理代码，如在表格统计区显示汇总数、单击机构列表可进行下钻等。以上方案为抽象方案，不能被方案解析引擎解析，只能用于继承。最后，当设计实际查询方案时，只需指定继承于一个功能方案，并将控件按方案中的控件名称命名，方案解析时就可以自动获得继承链上所有抽象方案中的属性和事件处理代码。方案继承示例如图 4 所示。

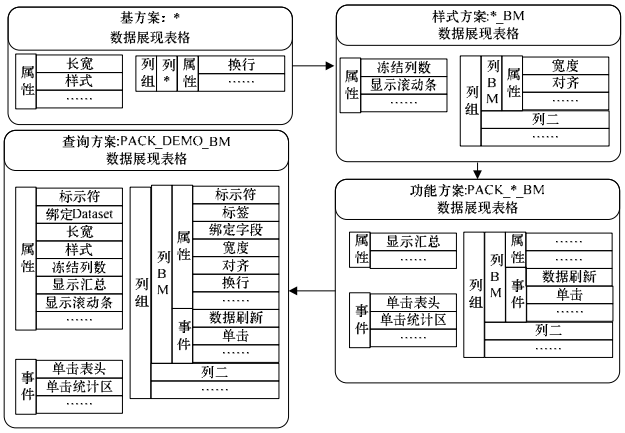

图 4　方案继承示例

通过继承的方式，查询方案的开发变得非常简单。如需要添加一个数据展现表格时，只需要指定标示符和需要绑定的 Dataset，并根据绑定的 Dataset 中的字段自动创建列即可。表格会根据被继承的抽象方案中的样式恰当地展现出来，并按照方案中设置的事件处理方式响应交互。查询方案可以覆盖抽象方案中控件的属性和事件处理代码，以满足个性化的需求。另一方面，当抽象方案中的属性和事件处理代码被修改时，所有继承于该方案的实际查询方案随之改变，降低了代码维护量。

通过继承式的设计方法，利用可视化编辑器带来的便利，开发者几分钟就可以设计出一个新的查询方案，这大大提高了对业务需求的响应能力，解放了开发人员的生产力。熟悉数据结构的业务人员经过简单培训后，就可以自行设计并对本级和下级发布查询方案，提升了整个机构的数据应用水平。

6　性能测试

选择正常纳税人查询这一具有代表性的业务需求进行测试，在同一分局、行业、税收管理员的查询条件下，查询耗时如表 1 所示。

表 1　查询性能对比　　s

方式	省局		市局		区局		分局	
	统计	前 20 条	统计	前 20 条	统计	前 20 条	统计	前 20 条
原始数据	124.65	112.14	21.31	18.75	4.53	4.37	0.50	0.33
预生成统计数	0.06	—	0.04	—	0.03	—	0.03	—

从表 1 可以看到，直接查询省局原始数据时间过长，用户难以接受，将数据分发到区县局，可以有效地缩短查询时间，大大提升用户体验。使用查询框架可以将省、市局的预

(下转第 77 页)