一种实现持久层的新方法①

A New Solution Of The Persistence Layer

郑 华 (广西财经学院计算机与信息管理系 广西南宁 530003)
开金宇 莫 林 (广西大学计算机与电子信息学院 广西南宁 530004)

摘要:基于 J2EE 的多层 WEB 应用开发过程中,持久层的设计是一项重要的工作。本文讨论了目前主流的持久层实现类型的优缺点,并提出一种使用数据库存储过程来实现持久层的新方法,最后通过一个具体实例证明了该方法的可行性和优越性。

关键词:J2EE 持久层 存储过程

1 前言

面向对象的体系结构应该分层进行设计,层次系统结构中,每一层为上层服务,并作为下层客户。这就将一个复杂问题分解成一个增量步骤序列的实现,这不仅允许每层使用不同的实现方法,而且为软件重用提供了强大的支持。

Web 应用开发的当前阶段是构架设计,它包括将应用划分为多个功能组件并将这些组件分割组合成层,高层的构架设计应该中立于所选用的特定技术。多层架构将系统清晰的分为多个功能单元:客户端、表现层、业务逻辑层、持久层和数据库(参见图 1),使得系统更易于维护和扩展。具有三层或等多层的系统被证明比 C/S 模型具有更好的伸缩性和灵活性。

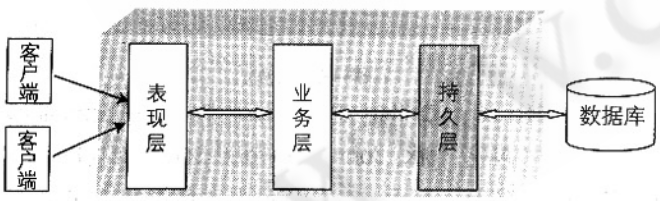

图 1 持久层是分层结构的基础

客户端(Client)是使用和表示数据模型的地方,对于一个 Web 应用,客户端通常是浏览器,基于浏览器的瘦客户端不包含任何表示逻辑,它依赖于表现层。表现层(The Presentation Layer)是最高层的用户接口逻辑。负责显示、页面控制和屏幕导航的代码构成了表现层。业务层(The Business Layer)包含应用的 business objects 和 business services。它接受来自于表现层的请求、基于请求处理业务逻辑。数据库(Database)位于 Java 应用之外,它是系统状态持续性的实际表示,如果使用了一个 SQL 数据库,它会包含关系模式或者可能有存储过程。

持久层(The Persistence Layer)是负责从一个或多个数据库中存入和取出数据的一组类和组件,用以定义、维护、访问和更新数据并管理和满足应用服务对数据的请求。典型的 Web 应用的另一个末端就是持久层,这里通常是程序最容易失控的地方。开发者总是低估构建他们自己的持久框架的挑战性。系统内部的持久层不但需要大量调试时间,而且还经常缺少功能使之变得难以控制,这是持久层的通病。

2 持久层实现类型的发展

一般来讲,持久层实现有三种类型:

2.1 基于 JDBC 的持久层的实现模型 – –混杂模式

持久化功能的原始实现模式。即在业务类中混杂 JDBC 访问代码,从而提供所需的持久化功能。

这种模式的优点在于开发的迅速便捷。对于原型系统或者小型应用显得别具意义。但基于这种模式开

① 基金项目:2005 年广西自然科学基金项目(桂科攻 0537020 – 5A),广西财经学院 2005 年度校内科研项目(2005C27)

发的系统,其维护性和扩展性较差,对象属性、数据库结构的变动都将直接导致业务逻辑代码的修改。

2.2 基于 Data Class 的持久层实现模式

在这种模式中,数据类(Data Class)作为业务类与持久层沟通的桥梁,起着承上启下的作用。

DAO(Data Access Object)实际上就是这种模式的一个典型实现。Data Class 实际上包含了 DAO 模式中的 Domain Class/Object 和 Data Accessor Class。Domain Class 作为对现实世界的抽象,起着信息携带者的作用。而 Data Accessor Class 则通过 JDBC 代码将 Domain Class 与数据库表相关联。

在这种模式中,我们实现了业务逻辑与底层数据结构之间的分离。Data Class 作为一个相对独立的逻辑层次,较为清晰的体现了所谓持久层的概念。底层关系数据的结构变化,可以较好的屏蔽在数据类这个层次,从而避免对上层的业务逻辑造成影响。实际上,在目前正在运行的很多 Java 企业应用系统中,基于这种类型的设计占了很大比重。

但随着设计层次的增多,编码量的增加相当可观。相对第一种混杂模式而言,系统结构上得到较大提升的同时,项目设计和开发成本也相应增高。

2.3 基于现有持久层框架的实现模式

实际上,这种模式是第二种模式的延伸。Data Classes 所包含的 Data Accessor 和 Domain Class 数量并没有减少。只是,把其中最为繁琐的工作——基于 JDBC 的 OR 映射工作,交由第三方组件完成。Data Accessor 中的繁琐编码工作因此得到了空前简化,而于此同时,伴随持久层框架而来的辅助工具也大大减轻了 Domain Class 的编码负担。

在 Java 发展的初级阶段,直接调用 JDBC 几乎是数据库访问的唯一手段。随着近年设计思想和 Java 技术本身的演化,出现了许多 JDBC 的封装技术,这些技术为我们的数据库访问层实现提供了更多的选择,目前主流的几套 JDBC 封装框架包括:Hibernate、Apache OJB、iBatis,JDO 以及 J2EE 框架中的 CMP 等。这些框架以优良的设计大大提高了数据库访问层的开发效率,并且通过对数据访问中各种资源和数据的缓存调度,实现了更佳的性能。

3 存在的问题

客户机/服务器数据传送:当使用 JAVA 语言编程实现查询的时候,大量的数据在应用服务器和数据库之间传送,这会极大地影响性能。

开放式数据库事务:一个事务被创建给每一个查询并用 java 代码来执行在一个应用服务器。当一个单个逻辑操作需要多查询,很多单一事务。

数据库模式和 Java 代码的紧密结合:数据库结构暴露于 Java 层。因此,对于数据库结构的改变同时需要修改 Java 代码。考虑到 Java 类结构和成员数据,数据库结构应该被从软件中其它层抽象出来。

软件版本协调性:因为 Java 代码依赖数据库结构,数据库版本包含了模式改变的必须和相应的 Java 代码版本相一致。但常常这是很难协调的。

4 一种新的方法——存储过程

存储过程的使用有助于解决上述所列的问题。存储过程是指保存在数据库并在数据库端执行的程序。你可以使用特殊的语法在 Java 类中调用存储过程。在调用时,存储过程的名称及指定的参数通过 JDBC 连接发送给 DBMS,执行存储过程并通过连接(如果有)返回结果。存储过程运行在 DBMS 自身,这可以帮助减少应用程序中的等待时间。不是在 Java 代码中执行 4 个或 5 个 SQL 语句,而只需要在服务器端执行 1 个存储过程。网络上的数据往返次数的减少可以极大地优化性能。图 2 是在多层开发中把与数据相关的业务逻辑转移到数据库中的存储过程来实现的示意图。

所有关联的 SQL 操作都可以在数据库内部实现。对于多表数据查询和更新等操作,导致的查询压缩细节内置于一个存储过程中。这样设计的好处是配置了数据相关的业务逻辑(数据存在的地址以及如何操作这些数据),这些逻辑属于数据库内部。

4.1 存储过程

存储过程(Stored Procedure)是一组完成特定功能的 SQL 语句集,经编译后存储在数据库中。存储过程能够通过接收参数向调用者返回结果集,结果集的格式由调用者决定;能够返回状态值给调用者,指明调用是成功或是失败;包括针对数据库的操作语句,并且可以在一个存储过程中调用另一存储过程。

存储过程具有以下优点:

(1) 存储过程允许标准组件式编程。存储过程在

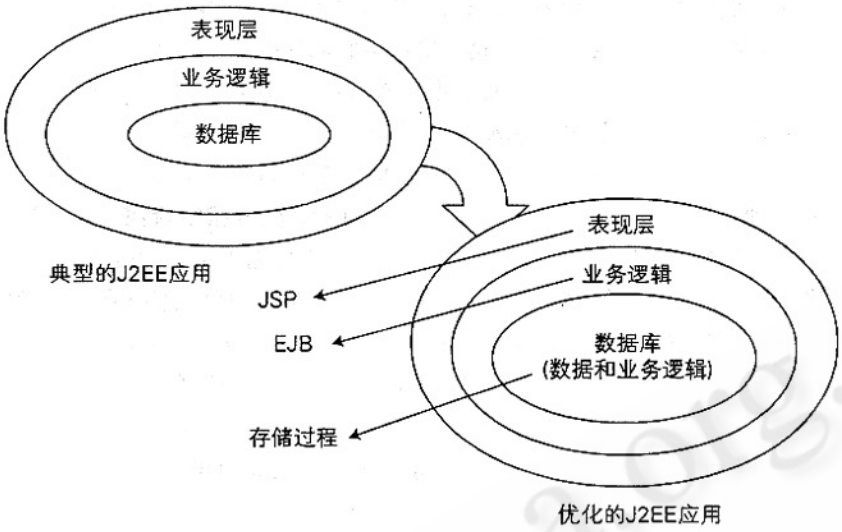

图 2　移动业务逻辑到数据库层

被创建以后可以在程序中被多次调用，而不必重新编写该存储过程。而且数据库专业人员可随时对存储过程进行修改，但对应用程序源代码毫无影响（因为应用程序源代码只包含存储过程的调用语句），从而极大地提高了程序的可移植性。

（2）存储过程能够实现较快的执行速度。因为存储过程是预编译的，在首次运行一个存储过程时，查询优化器对其进行分析、优化，并给出最终被存在系统表中的执行计划。而批处理的 SQL 语句在每次运行时都要进行编译和优化，因此存储过程要比批处理的执行速度快很多。

（3）存储过程能够减少网络流量。对于同一个针对数据库对象的操作（如查询、修改），这一操作如果所涉及到的 SQL 语句被组织成一个存储过程，当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而大大减少了网络流量，降低了网络负载。

（4）存储过程可被作为一种安全机制来充分利用。系统管理员通过对执行某一存储过程的权限进行限制，从而能够实现对相应的数据访问权限的限制，避免非授权用户对数据的访问，保证数据的安全。

4.2 在 Java 中调用存储过程

下面以一个 WEB 应用的用户登录功能为例介绍存储过程的优势。

4.2.1 基于 SQL/JDBC 的登录步骤

（如图 3 所示）如下：

（1）通过 USER_DATA 表执行 SQL 查询获取用户帐户信息，并检索出用户帐户的数据（用户名，密码，登陆计数，最大登录数）。核查输入的密码是否和数据库里的密码一致。

（2）如果密码正确，对 USER_LOGIN 表执行查询，检查是否该用户已经登录。

如果数据没找到，意味着该用户没有登录。转到步骤 3。如果数据找到，对比用户的 cookie 和查询操作返回的 cookie。如果匹配，意味着该用户已经登录并能够继续使用该系统。如果 Cookie 不匹配，意味着同样的用户名已经在其它计算机上使用了。既然这样，必须核查用户的登陆次数还没有超过系统限制（一个系统可以限制一个帐户只能登陆一次、多登录或无限的登录）。如果最大登录数还没被超过，继续下一步，否则，退出登录。

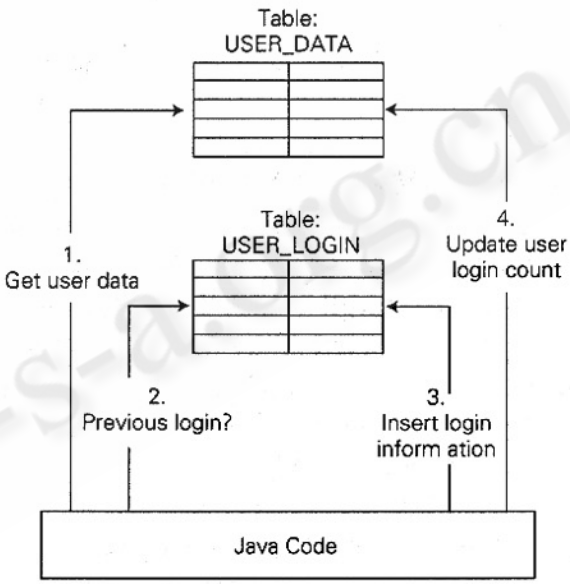

图 3　基于 SQL 查询实现用户登录

（3）插入一行新的数据到 USER_LOGIN 表，包括用户名、cookie 值和时间戳。

(4) 更新 USER_DATA 表,使得用户登录计数自动加1。

以上简单的登录功能实现中,需包含执行登录所需的所有 Java 代码和 SQL 查询语句(具体源代码限于篇幅省略)。要调用数据库四次,数据要来回进行传送。其中执行 SELECT 声明两次,执行一次 Insert 语句,最后调用 UPDATE 语句。如果将这些 SQL 语句转移到一个存储过程中将大大简化代码,仅涉及一次网络调用,所有关联的 SQL 操作都在数据库内部发生。

4.2.2 创建存储过程

把用户登录的业务逻辑转移到存储过程可以解决这些问题。存储过程可以执行所有用户登录操作的处理。

```
create procedure doLogin //创建数据库存储过程实现登录事务逻辑
    @username varchar(32),
    @password varchar(32),
    @cookie varchar(256)
as
begin
    declare @max_logins int //定义变量—最大登录数
    declare @login_count int //定义变量—登录计数
    declare @ins_error int //定义变量 - 登录失败计数
    SELECT @login_count = u.LoginCount, //从基本表中检索出用户计数和最大登录数
      @max_logins = u.MaxLogins
      FROM USER_DATA u
      WHERE ( u.UserName = @username
        AND
        u.Password = @password )
    IF @@ROWCOUNT = 0 BEGIN //检查用户登出
      RAISERROR ('The user was not found',1,1)
      SELECT ' as UserName
      RETURN
    END
      DECLARE @prev_user varchar(64) // - - 基于cookie 检查该账户是否再次登录
    SELECT @prev_user = ul.UserName // - - 如果原先已经登录,则该用户应经登录,可以
      FROM USER_LOGIN ul //跳过剩下的登录过程
      WHERE ul.Cookie = @cookie
      IF @@ROWCOUNT > 0 BEGIN //登录计数器大于1则该用户已登录
      IF @prev_user = @username BEGIN
        select * from USER_DATA
          where Username = @username
        RETURN
      END
    END
      IF @login_count >= @max_logins BEGIN //——检查用户登录数
      RAISERROR ('Max user logins reached',1,1)
      select ' as Username
      RETURN
    END
    BEGIN TRAN
    UPDATE USER_DATA // - - 给该用户的登录计数器自动加1
      SET LoginCount = (LoginCount + 1)
      WHERE Username = @username
    INSERT INTO USER_LOGIN // - - 添加登录信息进入 USER_LOGIN 表
      ( UserName,
        Cookie,
        Timestamp )
    VALUES
      ( @username,
        @cookie,
        GETDATE() )
    SELECT @ins_error = @@ERROR // - - 检查成功信息
    IF @ins_error = 0 BEGIN
      COMMIT TRAN
      select * from USER_DATA where Username = @username
      RETURN
    END
    ELSE BEGIN
```

(下转第 77 页)

(上接第 73 页)

```
        RAISERROR ('User login failed',1,1)
        ROLLBACK TRAN
        RETURN
    END
end
```

4 结论

使用存储过程来实现持久层这种方法可以在多层的 WEB 应用开发中大范围采用。但由于很难界定数据库服务器和应用服务器的比例,而把所有的业务逻辑都通过存储过程来实现是不明智的。然而,如果只把与数据相关的业务逻辑转移到存储过程,不管是在多层 Web 应用中的持久层还是对于数据库服务器而言都能获取巨大的效率。

参考文献

1 夏昕、曹晓钢、唐勇,深入浅出 Hibernate,北京 电子工业出版社,2005 年 6 月:1 -35。

2 田珂、谢世波、方马,J2EE 数据持久层的解决方案,计算机工程,2003 年 12 月,第 29 卷第 22 期:93 -95。

3 董樵、苗放、祖彩霞,实现基于 JDO 的电子政务系统持久层,物探化探计算技术,2004 年 5 月,第 26 卷第 2 期:189 -192。

4 薛兵、曹作良,设计模式和数据持久层框架在 Web 系统中的应用,天津理工学院学报,2004 年 3 月,第 20 卷第 1 期:72 -74。