CBF平台设计之数据访问框架

目录

[7. 数据访问框架 4](#_Toc504489474)

[7.1. ORM引擎 4](#_Toc504489475)

[7.2. 静态数据访问框架 11](#_Toc504489476)

# 数据访问框架

数据访问框架（Data Access Framework，简称DAF）模块致力于通过ORM等引擎、框架帮助应用中的数据访问对象类（Data Access Object，简称DAO）以极其简洁的方式为上层应用提供统一的、无需编写SQL的数据访问接口。

数据访问框架模块由：1、ORM引擎；2、综合数据访问框架；共2个子模块构成：

ORM引擎子模块提供了一个帮助DAO类实现对象-关系数据库映射（Object Relational Database Mapping，简称ORM）方法的引擎。



## ORM引擎

ORM引擎子模块提供了一个帮助DAO类实现持久化方法（即为上层应用提供统一的数据访问接口，Select，Update，Delete，Insert，等等，使得数据库结构对上层应用透明，应用无需编写SQL语句来获取、写入数据）的引擎。

#### 需求背景和目标

按传统方式开发业务应用，需要编写不少数据访问层的含有SQL语句的代码，用来从数据库读取信息，或者更新、删除数据库中的信息，这些代码很大部分是重复的，并且，如果程序员素质良莠不齐，就很容易出现一些错误的数据库使用方式，造成性能问题，甚至影响整个应用系统的可靠性和健壮性，或者使应用程序变得难以维护。因此，我们希望在业务应用中，专门定义一类对象，这些对象和数据库中的数据之间有既定的对应关系，能够提供Update/Insert/Delete/Select方法，依据对象的状态更新、添加、删除数据库中的数据，以及依据数据库中的数据更新对象的状态（所谓持久化，主要就是指这个过程），这类对象的存在，使得上层应用与数据库隔离，对数据库的访问操作被约束在一个很小的代码范围内，既有利于应用程序的维护，也有利于数据库使用方式审查和问题分析。从应用分层的角度，我们把这类对象称为数据访问对象，即DAO；从数据模型的角度，我们把这类对象称为业务实体数据对象。

但让所有相关业务数据对象具备持久化能力成为DAO，不是一件简单的事情。如果要求业务数据类的开发者自行编写代码实现持久化接口，开发工作量较大，质量也难以保证。为此，我们提出ORM引擎设计，这是用于支持业务数据对象以极其简易的方式获得持久化能力的一种框架设计。

一般情况下，一个DAO类和一个表对应，类的实例可对应表中的一条记录，类的属性则对应表的域。

例如，数据库中有如下表和记录：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table:Staffer | | | | |
| Field:Identifier | Field:FirstName | Field:LastName | Field:Phone | Field:PostCode |
| 1001 | Zhang | san | 80001001 | 100010 |
| 1002 | Li | si | 80001003 | 100020 |

又有如下DAO类和应用代码：



执行后，s对象的FirstName、LastName、Phone、PostCode等属性的值将分别是zhang、san、80001001和100010。

#### 持久化接口和可持久化类

尽管持久化框架核心的运作机制并不要求统一的持久化接口，但统一持久化接口的形态仍然是有意义的——一些面向切面编程的框架和工具就需要依赖一个统一的持久化接口，因此，我们提出如下接口（IPersistence），作为持久化接口的标准形态：



只有实现了该接口的类，我们才称之为可持久化类（相应的实例称为可持久化对象）。借助于持久化图和持久化器的支持，我们能够让拥有ISqlData接口实现对象（在组件对象模块中提供的最常用的基本数据类型：String、Integer、Long、Bolean、Float等等，都实现了ISqlData接口）作为其成员的、结构复杂的业务数据对象类很轻松地实现这个持久化接口，成为可持久化类。

#### 关系图、持久化图和持久化器

所谓关系图（RelationMap）就是一个ISqlData接口实现对象的登记表，其中，每个ISqlData接口实现对象都要标明其对应的数据库中域名（注意，关系图并不确定对应的表名）：

 所谓持久化图（PersistenceMap）就是一个关系图对象的登记表，其中，每个关系图对象都要标明其对应的数据库中表名：

 显然，持久化图提供了充分的信息，指出了一组ISqlData接口实现对象与具体表、表中的具体域之间的映射关系。

我们只需在业务数据对象类中实现一个接口（IPersistenceHelper），填写一个持久化图，标明其有哪些成员对象（必须都是ISqlData接口实现对象）会加入到该类对象的持久化过程中（注意，这意味着一个对象的持久化过程可能会涉及多张表），同时，再提供一个主键域名组，标明其中哪些域名是数据库中相关各表的主键域名（注意，这意味着相关各表的主键域名必需相同），随后，即可将持久化图和主键域名组交给持久化器（PersistenceEngine），由持久化器完成该类对象的持久化操作。

业务数据对象类填写持久化图和主键域名组的接口的实现非常简单：



再借助持久化器，业务数据对象类就可以轻松实现持久化接口：



持久化器本身的工作机制并不复杂，它将依据持久化图和主键域名表的信息自动生成SQL语句，并调用SqlService对象的Execute方法执行这些SQL语句，如果是SELECT，在持久化图中登记的ISqlData接口实现对象则用来接收这些SQL语句执行后从数据库中查询到的数据；如果是INSERT、UPDATE，在持久化图中登记的ISqlData接口实现对象则用来提供这些SQL语句执行时需要送往数据库的参数。无论如何，主键域名对应的ISqlData接口实现对象中事先都必须有准确的数据，不管是SELECT，INSERT，UPDATE，还是DELETE，都要依据主键域名表生成WHERE子句，该子句总是会使用主键域名对应的ISqlData接口实现对象作为执行时需要送往数据库的参数。

#### 衍生支持

在面向对象设计的应用系统中，很多时候业务数据对象类经常需要采用衍生设计，例如，VIP类衍生自客户类，VIP类将继承客户类中定义的所有成员，同时，VIP类又可以定义一些客户类所不具备的成员。在物理数据模型层面，为合理存储相关信息，通常都会设计两张数据库表，客户表和VIP表，客户表包含对应客户类定义的成员的域，VIP表则包含对应VIP类扩充定义的成员的域，两张表可以通过主键（例如客户号）建立关联（通常，客户表的记录数要远大于VIP表的记录数）。

在持久化器的支持下，我们也可以对DAO类运用这种衍生设计，并与上述物理数据模型对应：

 在上述示例中，VIP类实例的持久化接口方法被调用时，都会同时访问VIP表和客户表。

#### 筛选（绑定SqlDataTable和SqlCuror）支持

应用设计时需要基于过滤条件获取所有符合条件的DAO集合。持久化器提供的接口能够让DAO与一个按照过滤条件产生的SqlDataTable或者SqlCursor绑定起来——设置SqlDataTable/SqlCursor的当前位置，该DAO就会依据SqlDataTable/SqlCursor当前位置上的数据，自动更新其内部保存的数据（实际上，持久化器在调用SqlService的接口产生SqlDataTable/SqlCursor时，所传入的ISqlData接口实现数据对象就是绑定的DAO持久化图中的ISqlData接口实现数据对象）：



为方便应用开发，DAO通常也都会在自己的类型中提供CreateSqlDataTable和OpenSqlCursor方法，其实现就是调用持久化器提供的相应接口。

#### 临时表支持

在使用筛选功能时，存在两个问题，一是从数据库返回的结果集可能非常大，由此进一步产生了内存消耗巨大的SqlDataTable对象，容易引发应用内存溢出异常；二是某一个应用处理过程中产生的检索结果，无法被另一个应用处理过程重用，而这种重用是有些应用场景所期望的。

为此，持久化器还提供了临时表接口，即基于过滤条件先生成临时表（而非SqlDataTable），再基于先前生成的临时表产生绑定的SqlDataTable：



#### 分页检索支持

持久化器提供了分页检索接口，即在符合过滤条件的结果集中，再依据给定的分页参数产生绑定的SqlDataTable。对此SqlDataTable进行遍历，只会获得指定分页中的数据：

分页检索接口还可以基于先前生成的临时表工作：



#### 事务级缓存支持

业务系统应用程序基本都采用高度组件化的设计模式，这种设计模式降低了开发复杂度，提升了开发效率，但是，容易导致运行时性能低下——应用程序运行过程中会产生过多的数据库访问，其中，很多数据库访问是重复的，比如客户信息验证组件，可能在交易主处理过程中直接调用执行了，然后，在交易主处理过程调用执行账户信息验证组件时，在账户信息验证组件的处理过程中又调用执行了一次客户信息验证组件——客户信息验证组件运行过程中，最大的开销就是访问数据库中的客户信息，因此，如果平台能够提供数据库访问缓存支持，即可大大提高这些业务系统的性能。

数据访问框架提供了事务级的缓存支持。调用DAO的Select方法，或CreateSqlDataTable方法，内存中即会保存相关的数据，此后，只要是在同一个数据库事务中，再次调用同类DAO的Select方法，或CreateSqlDataTable方法，并且主键域值相同，或者过滤条件相同，这些方法就会使用内存中先前保存的数据来设置DAO，或者需要返回的SqlDataTable对象中的数据，而不会向数据库服务器提交SQL。

事务级的缓存会由于执行相关DAO的Insert方法、Update方法和Delete方法而自动得到更新。

#### ORM引擎的特性和优势

与常见的开源的ORM引擎（例如Hibernate）相比，本ORM引擎具有很鲜明的特性和很显著的优势——

首先，它没有一个用IDL或类似IDL的文件描述持久化数据结构的过程，自然也不需要通过工具依据IDL描述的信息来自动生成一个数据类，而是需要可持久化类实现一个设置持久化图的接口，这就意味着可持久化类的开发、使用都很方便（接口实现比编写IDL描述更加简洁，部署时无需部署配套的IDL脚本），运行速度也会快很多（运行时不会再访问IDL描述的信息，也不会通过反射来操作对象的属性），同时也意味着使用java语言编写的可持久化类如果需要转换为其他语言，需要重新编程——但是，请注意！在核心系统开发环境配备的统一的架构设计和管理工具上，我们实际上已经完成了与使用IDL定义持久化数据结构非常类似的工作，通过这个工具就可以自动生成各种语言版本的可持久化类！

其次，它完全基于接口，而非基于类型工作，这就意味着应用更加灵活，能够很好地应对为现有数据类添加持久化能力的需求。

最后，它是独创的完全适应分布式数据库和核心系统应用层面的特性要求的ORM引擎——不仅支持基于主键和过滤条件的访问，还支持基于索引的访问，支持分片键转换。

## 静态数据访问框架