基于Java的数据库访问的优化研究

摘要: 为了实现快速而准确地访问数据库，本文剖析了Java与数据库的接口—JDBC—访问数据库的原理，分析了造成访问性能问题的原因，提出了基于Java的解决数据库访问优化问题的多种思路和方法。

关键词: Java 数据库 JDBC 优化

高阳 刘永强

(湖南长沙中南大学商学院 410083)

1 引言

在Java的技术体系中，应用程序通过JDBC（Java database connection）接口来访问数据库。但是有些时候，查询数据的效率低下，在企业级的应用中表现尤为突出，明明根据规范编写的程序，执行起来却要花费很长时间，得不到预期的效果，造成了整个系统的运行效率不高。因此，在当今硬件环境比较优越的条件下，有必要对数据库访问的软件环境进行优化研究。

2 JDBC数据库访问机制

JDBC是Java与数据库的接口规范，由一些Java语言编写的类和界面组成，定义了一个支持SQL功能的通用低层的应用程序编程接口（API）。JDBC API又定义了若干Java中的类，表示数据库连接、SQL指令、结果集、数据库元数据等，它在功能上和ODBC相同，旨在让开发人员提供一个统一的、标准的数据库访问接口。以下是它的重要接口 [1,4]：

java.sql.DriverManager用来处理加载驱动程序并且为创建新的数据库直接提供支持。

java.sql.Statement在一个给定的连接中作为SQL语句执行的容器，它具有两个字类。

java.sql.ResultSet从数据库服务器返回的结果集。

JDBC的使用方法如下：在网页中使用<applet code=' ' ></applet>标记来应用Java小程序，在小程序中用importjava.sql. 来调用SQL包，这样就可以利用上述接口来访问Web数据库。

3 Java数据库访问的优化

3.1 JDBC接口的选择

目前，Java应用程序通过JDBC接口访问数据库有四种驱动模式 [2,5] 因此在不同的应用中，需要采用不同模式的JDBC接口。选择合适的模式，使之符合数据库程序设计，是提高访问速度必须考虑的一个重要方面。

3.1.1 JDBC—ODBC BRIDGE 模式

JDBC通过JDBC—ODBC BRIDGE使用ODBC驱动程序来访问数据库，由于JDBC—ODBC BRIDGE和JDBC不能从服务器上直接下载，所以使用时必需在客户端安装与服务器相同的ODBC驱动程序和JDBC—ODBC BRIDGE的本地库。其优点是因为大多数RDBMS平台都支持ODBC驱动程序，所以，所有ODBC的数据库不加任何修改就能与JDBC协同工作；缺点是用户受底层ODBC驱动程序的功能限制，而且不能用于applet中，因为applet不能加载本地调用。对于小型数据库开发人员来说，采用此模式作为程序的设计基础只能是一个过渡性的解决方案，由于它需要首先把JDBC操作翻译成对应的ODBC调用，然后这些调用又被传递给ODBC驱动程序，最后才执行数据库的相关操作，很明显其性能要削弱很多。

3.1.2 Native API partly—Java Drive 模式

这种方式将用户应用程序的JDBC调用转

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

换成对数据库（Oracle、Sybase、Informix、DB2等）客户端接口的调用。该驱动程序也要求在每台客户机上预先安装，类似于JDBC—ODBC BRIDGE。其优点是不要转换成ODBC调用，它利用多层结构，上层用Java实现，利于平台应用和支持多数据库，下层则改为本地代码，加速执行速度。这种开放和高性能的特征得到了业界的肯定，因而被主要的数据库厂商强烈推荐。尽管下载需要一定的时间，但这相对于访问数据库速度的提高，就不足为虑了。

3.1.3 JDBC—Net pure Java driver模式

这种驱动程序将JDBC调用转换为与DBMS无关的中间网络协议，之后这种协议又被某个特定的数据库服务器转换为一种DBMS协议，因而是最灵活的JDBC模式。其优势在于对多种数据库的支持，广泛应用于Internet/Intranet的开发，安全性和性能都十分显著。缺点是进行数据库操作时，需要花费较长的时间。不过，此模式的JDBC驱动往往提供许多企业级的特征，例如，SSL安全、支持分布式事务处理和集中管理等，因而会对用户特殊的用途有很大的帮助。是否选用，还在于用户扩展应用是否需要对多DBMS进行支持。

3.1.4 Native—protocol pure Java driver模式

这种驱动程序将JDBC调用直接转换为与DBMS所使用的中间网络协议，允许客户端使用该类驱动程序直接访问数据库系统。由于这样的协议都是专用的，因此数据库产品提供者自己将是这种驱动程序的主要来源。其优势在于和数据库本身结合比较紧密，而且是纯Java的实现，性能很高，在企业级的软件应用中，应该是首选。一般来说，商业DBMS的提供者往往会为自己的数据库提供一个JDBC接口。

3.1.5 四种模式实现JDBC接口选择的特征归结如表1：

3.2 Java数据库批量更新机制的引入

传统的数据库更新中，只能使用一个Statement接口上或其子接口上的executeUpdate()或者execute()方法调用来提交更新，而且一次只能提交一个更新。随着需要更新的数据量的增大，这就会大大影响对数据库更新速度。我们可以在JDBC核心API引入一种新的更新机制，可使用户能够创建一批与一个语句对象相关联的更新，通过一次调用将整批更新命令提交给数据库，然后数据库一次处理所有的更新，这比将同一组更新中的每一个逐一地提交给数据库的性能要高得多，大大提高了对数据库的访问速度。支持或不支持这种成批更新是数据库与Java驱动程序开发过程中都必须考虑的问题。在使用过程中，用户需要用DatabaseMetaData对象上的support BatchUPdates()方法来确定他所使用的数据库及驱动程序配置是否支持批量更新，否则会引起数据冲突。下面是一个批量更新的例子：

```
public void insertBatchRowsUsingStatement
()
throws SQLException,BatchUpdateException,
Exception
{
Connection connection=super.getConnection
(); //create connection as usual
DatabaseMetaData dbMetaData=connection.
getMetaData();
if(dbMetaData.supportsBatchUpdates()){
...
connection.setAutocommit(false);
Statement statement=this.getStatement();
statement.addBatch(" INSERT INTO STATE
VALUES(\`VA\`,\`VIRGINIA`)" );
statement.addBatch(" INSERT INTO STATE
VALUES(\`MD\`,\`MARYLAND`)" );
int [] nResults=statement.executeBatch();
connection.commit();
for(int i=0;i<nResults.length;i++){
system.out.println(i+" the statement executed
with" +nResults [i] +" results" )
}
}
...
}
```

3.3 访问数据库中间件Servlet技术的引入

除了从JDBC的角度考虑数据库访问优化问题，我们还可以对数据库中间件Servlet技术进行研究。Servlet是用Java语言编写的运行在服务器端用来扩展服务器功能的Java小程序，通过建立在Web上提供请求和响应服务的运行框架来扩展Web server的功能，具有公共网关接口CGI（Common Gateway Interface）的所有功能。使用Servlet可以很容易实现管理多线程、处理客户请求以及提高安全保障。

Servlet技术源自于请求/响应（Request/Response）模式。用户通过浏览器（Browser）访

表1

选择顺序	所适应的环境	建议
4<3<2<1	实验环境下	选择易于配置的驱动，利于Java程序的开发，后期在对应用环境判断后，再对JDBC模式进行选择。
1<3=2<4	小型企业环境下，不需要对多数据库的支持	模式2和3的有些优点并不能体现出来，建议选择模式4。
1<4<3<2	大型企业环境下，需要对多数据库的支持	模式2和3各有千秋，大多情况下，建议用速度较快的模式2。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

问数据库时，提出HTTP请求，Servlet会动态地生成响应，然后发送包含处理结果的HTTP响应到浏览器，如图1：

Servlet可以很好地代替CGI提供动态数据，与之相比，Servlet主要有以下优势：

· 可扩展性。由于是Java编写的，它就具备了Java所带来的优点，其中包括作为一种面向对象的变成语言所具有的可扩展性。

· 与平台和协议无关性。由于Servlet是使用Java来编写的，所以一开始是与平台无关的，可就任何平台运行的承诺在服务器运行，同时无需在网络上传输所用的协议。

· 快速响应性。在响应请求后，能够在后台持续运行，能够在多线程中高效地处理多个请求。

· 持久性。Servlet只需WEB服务器加载一次，而且可以在不同的请求之间保持服务，始终保持与数据库的连接，而CGI程序是瞬间的，需要每次加载，一旦运行结束，会从内存中清除。

3.4 J2EE软件模型中对数据库访问的设计模式的引入

J2EE（Java 2 Enterprise Edition）[3]是用于可扩缩企业应用的Java平台，具有Web功能、应用使能、基于组件的环境和API。其软件模型的设计模式是一组类、接口与其关系的集合，提供通用问题类型的一般设计方案。其数据库访问的设计模式根据使用要求、要试图解决的通用问题、类/接口的设计结构及其关系、参与合作的对象、期望的解决结果以及实现约束等来进行定义。通过确定解决问题时的依赖模式，开发人员可以根据一组设计方案很快地认识和选择问题解决方案。在这里主要是对其中的数据库相关类进行演示说明，它分为实体类（Entity Class）和会话类（Session Class）。

实体类对应于一个表的记录的封装，也就是该类的一个实例对应于表中的一个记录。而且，该类中的属性和记录中的字段是一一对应的。必须注意的是，实体类的实例是每个记录在内存中的对应，在程序中对实例的操作并不马上反应到数据库的记录中。在该类中，只是对数据的包装，仅需要一些基本的方法，即setXX()和getXX()方法。

会话类对应于对一个表中的所有记录的操作，比如增加、删除、查找和更新记录等。这些操作的结果是将表中的记录和内存中的实体类的实例对应起来，或将实例与表中的记录对应起来。

我们可以用一个实例进行说明，看一下对Book表的封装。

```
class BookTable{
    void Add(Book book);
    void Delete(Book book);
    void Update(Book book);
    Collection findbyID(int iID);
    Collection findbyXXXX(XX,XX);
    Collection findbyPulisherName(String sPublisherName);
};
```

上面的类的申明中，Add()用于将内存中的一个Book实例映射到数据库中，Delete()用于删除数据库中的某一个记录，Update()用于更新表中的一个记录，而findbyXXXX()则对应于SELECT语句。

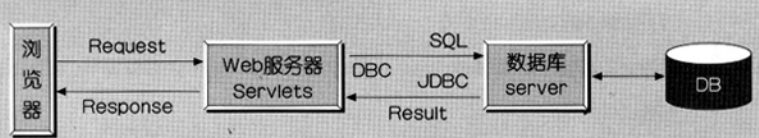
对于涉及到多个表操作时，可以有两种方式。一种是象BookTable一样，专门封装一个类，另一个方法是，直接在BoolTable中写一个findbyPublisherName()，这个方法设计成返回一个Book的集合。

通过使用这种数据库访问的设计模式，可以使程序更加模块化、集成化。特别在企业环境下使用，更便于对数据库的访问。

4 结束语

除了以上的一些解决方案外，基于Java数据库访问的优化还可以从其他方面进行考虑。例如数据库连接池高效策略模式、硬件环境的进一步改善和实际操作中的经验积累都是提高Java数据库访问性能的手段。而我们在进行系统功能的优化时，一般要求对其原理和运作机制进行分析，方能找出解决方案。

图1 .Servlet Request/Response模式



参考文献

1 Getting Started with the JDBC API.http://java.sun.com/j2se/1.3/docs。

2 B C Desai, R pollock.MDAS:Hetrogeneous Distribued Database Managerment System [J]. Information and Software, 1992, 34(10):28-43。

3 Paul J.Perrone, et al 著，张志伟、谭郁松译，J2EE构建企业系统专家级解决方案，清华大学出版社。

4 孙元，Java语言SQL接口：JDBC技术 [M]，清华大学出版社，1999。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn