Enterprise Java™ Platform Data Access

Seth White
Sun Microsystems, Inc.
JavaSoft Division
901 San Antonio Road
Palo Alto, CA 94303
seth.white@eng.sun.com

Rick Cattell
Sun Microsystems, Inc.
JavaSoft Division
901 San Antonio Road
Palo Alto, CA 94303
rick.cattell@eng.sun.com

Shel Finkelstein
Sun Microsystems, Inc.
JavaSoft Division
901 San Antonio Road
Palo Alto, CA 94303
shel@eng.sun.com

1. Abstract

This paper describes alternative methods for data access that are available to developers using the Java™ platform and related technologies to create a new generation of enterprise applications. The paper highlights industry trends and describes Java technologies that are responsible for a new paradigm in data access. Java technology represents a new level of portability, scalability, and ease-of-use for applications that require data access.

2. Introduction

The Java Platform allows a new paradigm for data access, particularly for web-based and intranet enterprise applications. In this paper, we describe the industry trends that are responsible for this important shift in the way that database applications are architected and developed. We also describe some of the Java technologies that are enabling this transformation: JDBC™, SQLJ, Java™ Blend™, and Enterprise JavaBeans™ (EJB).

3. Industry Trends Affecting Data Access

The inclusion of an embedded Java Virtual Machine in popular web browsers, such as Netscape Navigator, has been a major factor in making Java technology a de facto Internet standard on the client desktop. There are two related trends which we believe will make Java technology a standard in the database server and intranet or application server arenas as well.

First, in the near future major relational database management systems will contain an embedded Java Virtual Machine in the database server. This will make it possible for developers to extend the functionality of a DBMS server with application specific behavior using a standard, portable programming language—the Java language. Several forms of server extensions will be possible, including stored procedures which are written in the Java language, new data types defined using the Java language, and Enterprise JavaBeans components that execute in the DBMS server. Stored procedures and EJB components can use the JDBC API, embedded static SQL, or object/relational mappings such as Java Blend to access stored data.

The second major trend is the adoption of Java technology by vendors of application server products; these include traditional TP monitors who are adding Java language support, as well as web servers that support the Java servlet API, and web-based application servers. Widespread industry support for the Enterprise JavaBeans architecture for constructing, deploying, and executing server-side components is a significant force behind this trend.

The end result of these developments is that Java technology will become universal on both the client and server, and across all tiers of web-based multi-tier applications. This represents a new paradigm for applications that access data, providing them with a level of portability and architectural flexibility that has never before existed. For example, a data access component written using the Java language can not only be deployed on multiple operating systems, but also in multiple execution environments, including a web browser, intranet application server, or database server.

4. Java Technologies for Data Access

4.1 The JDBC API

The foundation for database connectivity and dynamic SQL access in the Java Platform is provided by the JDBC API. The JDBC API is the Java platform core call-level API for data access. Unlike other data access APIs, JDBC can be used to create portable data access components that execute in both a client and server environment. The most recent version of the JDBC API—JDBC 2.0—has been enhanced with the ability to support scrollable and updatable result sets, and with a number of new performance features, including a bulk update facility, connection pooling, and enhanced row prefetching.

The JDBC 2.0 API is the first widely available data access API to support the advanced data types introduced in the SQL3 standard, including binary and character large objects (BLOBs and CLOBs), arrays, references, distinct and structured types. In addition to providing a default mapping of these new SQL3 data types into Java language types, the JDBC 2.0 API also provides a way for a Java application to customize the mapping of SQL3 structured types into Java classes. For example, an SQL CUSTOMER structured type can be mapped to a Java class, Customer, which is specific to an application. CUSTOMER instances which are stored in the database can then easily be retrieved using the JDBC

API. The single line of code below shows how a Customer instance is retrieved from a result set that contains the output of an SQL query.

```
Customer c = (Customer)rs.getObject(1);
```

In the code sample, the result set's getObject() method is called to produce a generic Java Object which the application can narrow to an object of type Customer. The JDBC API's custom type mapping facility provides a more intuitive and object-oriented way for applications to access SQL3 data.

The JDBC 2.0 API also supports relational extensions which add Java language types to the type system of an RDBMS and allow SQL queries to contain references to these Java types. For example, the same code sample shown above can be used to retrieve an instance of the Customer class that is being stored natively in an extended relational DBMS. The new capabilities provided by the JDBC 2.0 API support the efforts of the SQLJ consortium which has defined a specification for the use of Java classes as data types in SQL, the use of Java methods as SQL stored procedures, and embedded static SQL for the Java language.

4.2 Java Blend

In addition to providing a powerful, yet easy to use API for data access, JDBC technology acts as a foundation layer for higher-level data access APIs and components.

Sun's Java Blend product is implemented on top of JDBC. Java Blend allows developers to access relational data as Java objects, thus avoiding the well-known impedance mismatch between the relational and object data models. The conceptual simplicity offered to developers by Java Blend can result in substantial reductions in development time and cost for data-driven applications. Java Blend also has powerful mapping capabilities to define Java classes from relational tables or relational tables from Java classes.

4.3 Enterprise Java Beans

The Enterprise JavaBeans architecture is the paradigm for developing transactional server-side components using Java technology. When combined with data access technology, such as Java Blend, JDBC, or embedded SQL, the Enterprise JavaBeans API enables a flexible application architecture. In this architecture data access can easily be configured or reconfigured to provide better performance and scalability. For example, a data-centric bean could be deployed in a DBMS server environment initially, and later moved to an application server without requiring any additional coding or recompilation of the application. Moreover, EJB components can expose the behavior (e.g., business methods) of existing database or application systems, not just their data.

5. Summary

This paper has highlighted the industry trends and Java technologies that are responsible for a new paradigm in data access. We have shown how these developments will impact data driven applications, and pointed out the new opportunities that they offer to customers and users of database technology. Java technology represents a new level of portability, scalability, and ease-of-use for applications that require data access.