AC 2007-1400: EXPLORATION OF JAVA PERSISTENCE

Robert E. Broadbent, Brigham Young University

Michael Bailey, Brigham Young University

Joseph Ekstrom, Brigham Young University

Scott Hart, Brigham Young University

Scott is an IT undergraduate working in the areas of information retrieval, and document management.

Exploration of Java Persistence

Abstract

Data persistence in a relational database is one of the core requirements of many applications. There are a variety of methods for implementing data persistence; however the advantages and disadvantages of persistence strategies are not always easily identifiable.

Typical introductory courses establish an understanding of relational databases. The more advanced student can find a practical application of his knowledge in variety of available open-source, persistence frameworks.

This paper explores Java data persistence within constraints that make it useable for the student researcher to use as an independent guide, or alternatively for an instructor to use as a resource for adding data persistence to coursework.

Introduction

Most computing programs offer a database principles and applications course to undergraduate students. Such a course often covers relational algebra, entity relationship diagrams, database normalization, standard query language, installation and use of relational database management systems, and use of the Java Database Connector API[1].

The course often provides an introduction to basic database theory and usage. Anyone interested in learning more about data persistence often must explore it on their own initiative. This paper presents options for exploring this subject.

Selection Criteria

There are hundreds of Java-based persistence frameworks, both open-source and proprietary-source, available for experimentation and development. It can be difficult to choose which ones to work with, so the following criteria were used for selecting which technologies and frameworks to use.

- No-cost
- Standards-based
- Readily available documentation, books, and active user forums
- Expert Recommendation
- Current technology

No-Cost

Technologies and frameworks available at no cost were chosen, not to avoid spending money but to avoid the psychological commitment to a purchased technology. Proprietary products were not excluded, as long as there was a cost-free developer version.

Standards-based

Technologies that were based on community standards or that had been accepted as de facto standards were used. Standards-based technologies tend to have wider-spread adoption and stronger vendor support. These technologies also tend to have more formal documentation (such as developer documentation and books) and informal documentation (such as user discussion forums and blogs). There usually is also greater demand for developers who are familiar with standards-based technologies.

Fortunately, many standards-based technologies have open-source, free implementations that are readily available.

Readily available documentation, books, and active user forums

Support material for learning a technology as quickly as possible was important to being able to explore and learn a technology with minimal frustration. In particular, books usually provide the most well-thought out explanation and discussion regarding their subject matter.

In the case of many open source frameworks the primary author of the tool provides excellent documentation, but reserves the most thorough discussion of a technology for a published book. Many libraries have books on Java-based technologies, and in the cases where a library does not have any related texts, the purchase of a relevant book aids in the ability to learn a technology.

Expert Recommendation

Due to the wide range of available technologies, the recommendations of others experienced in the field were sought out. Books that cover the subject of enterprise technologies and frameworks make recommendations regarding the persistent technologies with which they have found the most success. For this study, the technologies chosen for exploration were those recommended by someone with an authoritative opinion in the field.

Methods from Typical Coursework

Data Class Serialization

Many database principles and applications courses address elements of Java-based persistence. The first persistence technique presented is Java class serialization. When given more exploration, the simple serialization of data-containing Java classes provides a very simple and capable method of data persistence. Most importantly, it eliminates the need of programmatically converting the data into another form of data representation such as SQL and avoids the overhead of a relational database. Relationships can be represented as class properties and related

classes can be automatically serialized to disk without specific handling of each class. The use of hash classes provides the ability to search using a key.

By using serialization, some of the functionality of a relational database management system can be provided to a single client without the associated overhead of an entire database server. This technique works particularly well for static data that can be held entirely in memory.

Java Database Connectivity API

Database principles and applications courses also often address the use of the Java Database Connectivity (JDBC) API. The JDBC API provides a standard interface for interacting with a SQL-based relational database management system (RDBMS), through which SQL operations are issued and results programmatically retrieved using the API. The individual RDBMS vendors usually provide the interface implementation for their respective systems; developers are forced to provide the SQL and managed the non-bound results themselves.

The use of the JDBC API has very strong vendor support, but its use is error prone and tedious. These drawbacks have led to the creation of a variety of persistence frameworks based on the JDBC API standard.

Exploration Beyond JDBC

Object Relation Mapping

Persistence frameworks were developed to increase the ease of use and reduce the error in the use of JDBC by providing a layer of functionality on top of it. These frameworks are not as simple as using the JDBC API directly, but they provide benefits in terms of functionality and reduction of work by the developer.

The most common persistence frameworks use a technique called Object Relation Mapping (ORM). The ORM technique provides the conversion between Java data classes and the corresponding relational representation.

There are four primary levels of ORM sophistication that are found in persistent frameworks[1].

1. Pure relational

A pure relational mapping operates with a relational representation only. The developer must do the data conversion outside of the framework[2]. An example of a pure relational system in Java would be the use of the JDBC API.

2. Light Object Mapping

Light object mapping performs the conversion between the object model and the relational model via hand-coded SQL statements. This level of object mapping provides an abstraction of

the data persistence to the rest of the application[2]. An example of a framework that provides light object mapping is the iBatis Data Mapper framework.

3. Medium Object Mapping

The medium level of ORM tools provide object mapping between the object model and the relational model via dynamically generated queries. Relationships are automatically managed, and queries made in object model terms are converted to relational queries[2]. An example of a framework that provides medium object mapping is EJB 2.1 container managed persistence.

4. Full Object Mapping

Full object mapping is very similar to medium object mapping, but it provides additional support for more complex object models that include inheritance and composition. This level of object mapping also provides more sophisticated querying capabilities, such as in-memory queries of cached objects, and combining simple object queries into batch queries[2]. An example of a framework that provides full object mapping is Hibernate.

Enterprise Java Beans 2.1 API

The first Java persistence framework explored is the Enterprise Java Beans (EJB) 2.1 API. The EJB API was the first attempt to standardize much of the common functionality required by enterprise applications such as transaction management, security, and data persistence. This standard continues to evolve and is in wide-spread use[3].

The EJB 2.1 standard provides two means of persistence. One is called bean managed persistence (BMP). The BMP specification of data persistence treats a Java class that implements a specific interface as a persistent unit of data. The EJB container, which creates the persistent Java class, calls specific life cycle methods to indicate when persistence is to occur. The developer is required to implement the life cycle methods which perform the actual persistence. The developer can then use whatever method he prefers to implement the persistence mechanism and persist the data contained within the class[3].

During exploration of the EJB 2.1 specification, the BMP persistence mechanism was not investigated. It does not provide any significant benefit, other than transaction management of access to the data class that did not exist in using JDBC directly.

The other mechanism of data persistence provided in the EJB 2.1 specification is called container managed persistence (CMP). When using CMP, a Java class that implements a specific interface is still treated as a persistent unit of data. The persistent Java class must, again, implement a specific interface that contains methods that were called by the managing container[3].

When using CMP, the interface no longer contains methods for performing the data persistence. The CMP method of persistence requires the managing container to interrogate the data class by accessing its data access methods (getter and setter functions). The container is responsible for

keeping the data-containing Java classes and the database synchronized. CMP prevents the developer from having to interact with the database directly[3].

Unfortunately, our use of CMP was unsuccessful and very frustrating. The Sun EJB 2.1 Application Server was configured through extensive use of XML, and manual creation and modification of configuration XML was very error prone. Sun Microsystems provided a utility for XML generation, but it required an understanding of the underlying configuration data. In cases where the application server was misconfigured it was very difficult to discover where the problem lay.

Very basic attempts at data persistence using CMP were successful but more sophisticated attempts required precise coordination of the Java class, the XML configuration data and the database schema. CMP was eventually abandoned due to the error-prone complexity of the system.

The iBatis Data Mapper Framework

The iBatis Data Mapper Framework provides object relational mapping between an object model and an equivalent relational model. This mapping is done via hand-coded SQL statements provided by the developer, with the SQL queries and the relating object model Java classes specified in an XML file. The iBatis-provided data access classes were used to execute the pre-defined SQL queries and perform the object data mapping[4].

Use of the iBatis Data Mapper Framework is aided by excellent documentation, which includes examples of how to use the iBatis ORM capabilities. During experimentation, the iBatis framework functioned as expected. Use of the iBatis framework formalized access to JDBC, but its use of JDBC is still error prone especially when working with a changing data model.

The Hibernate Persistence Framework

The Hibernate Persistence Framework is a mature ORM tool with full object mapping capabilities. SQL queries to the relational model are dynamically generated based on the properties and relationships of the data model. Metadata about the object model is used to dynamically generate SQL queries for object model-relational model synchronization[5].

Object model metadata is specified in two ways. Traditionally Hibernate metadata is provided by XML files that contain mapping information about the relational model and the object model. Hibernate XML metadata files are not difficult to keep consistent with the data model and the relational model, but it is still a tedious process[5].

The alternative method for specifying metadata is made possible through the addition of annotation support in Java 5.0. When running Java 5.0 or greater, Hibernate version 3.0 and greater allow the specification of metadata directly in a Java class source file using annotations.

Hibernate annotation-based metadata is specified using configuration by exception. This causes Hibernate to assume an object model class has certain characteristics unless otherwise specified.

For instance, Hibernate assumes that the relational model table name and the data model class name are identical, and that the class property names are to be used as the table column names[5].

Experimentation with the annotation-based metadata configuration provided the most flexible method of configuration between the data model classes and the relational model. Hibernate also has the ability to dynamically update the relational model to match changes in the data object model, and Hibernate provides the most sophisticated functionality of any of the ORM frameworks. It also does not require the implementation of interfaces as EJB 2.1 does. However, Hibernate also is conceptually the most difficult to learn.

Future Work

EJB 3.0

A newer persistence framework to be considered for investigation is the EJB 3.0 Java Persistence API. The EJB 3.0 specification has been recently accepted. It takes a dramatically different approach to persistence than the EJB 2.1 specification. EJB 3.0 provides full object mapping similar to Hibernate, annotation-based configuration, and transactional capabilities beyond those provided by Hibernate.

Tools

There are a wide variety of tools to be explored that simplify the use of persistence frameworks in the development process. The Borland TogetherSoft IDE is an example of such a tool. It has an editor for creating entity-relationship diagrams. From these diagrams it will then generate the associated Java classes and the SQL-based table creation statements for the associated tables.

Conclusion

Experimentation with Java persistence frameworks provides students with functionality beyond that which is available through the JDBC API. Persistence frameworks can make data persistence less error prone than JDBC alone, but it is time consuming and can add additional complexity and frustration. However, in many situations a persistence framework may be worth the additional effort.

Bibliography

1. Riccardi, G. Principles of Database Systems with Internet and Java Applications, Addison Wesley, 2001.
2. Fussel, M. Foundations of O-R Mapping: Conlcusion, 2000, available Jan. 16, 2007 at: http://www.chimu.com/publications/objectRelational/part0008.html
3. Armstrong, E., J. Ball, S. Bodoff, Et al., Dec. 5, 2005, available Jan. 16, 2007 at: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/J2EETutorial.pdf
4. iBatis Data Mapper Version 2.0 Developer Guide, Aug 9, 2006, available Jan. 16, 2007 at: http://ibatis.apache.org/docs/java/pdf/iBATIS-SqlMaps-2_en.pdf

5. Hibernate Reference Documentation 3.2.0ga, Oct 15, 2006, available Jan. 16, 2007 at: http://www.hibernate.org/hib_docs/v3/reference/en/pdf/hibernate_reference.pdf