Innovations by InterSystems

Plain Old Java Persistence With Caché

InterSystems Jalapeño Technology

A white paper by:

Andreas Dieckow
Principal Product Manager, Strategic Planning
InterSystems Corporation

InterSystems CACHÉ

Plain Old Java Persistence With Caché

Introduction

Because of its "write once, deploy anywhere" characteristic, the Java programming language has many devotees and practitioners. But Java is an object-oriented language, and its data objects are inherently non-persistent. Even the most ardent fan of Java will admit that the object-relational mapping required to persist Java objects in a relational database is a chore. By some estimates, it can consume 60% – 70% of the total development effort.

In addition to supporting JDBC like any relational database, InterSystems Caché has always offered object-oriented ways to provide data persistence for Java applications. Caché classes can be projected as Java proxy classes, or as Enterprise Java Beans with high-performance bean-managed persistence. These approaches both eliminate object-relational mapping, but they do require Java developers to build their data objects within Caché, before projecting and using them within their Java development environment.

This paper introduces InterSystems' Jalapeño technology, which allows a "Java-in", rather than a "Caché-out" approach to application development. It will discuss how to persist "plain old Java objects" (POJOs) within Caché, and the benefits associated with using object technology for data storage. It will also outline the necessary steps to make an application created with Jalapeño technology run against a relational database.

What is Jalapeño technology?

InterSystems' Jalapeño (**JA**va **LA**nguage **PE**rsistence with **NO** mapping) technology allows Java developers to define object classes within whatever Java development environment they favor and persist those classes in Caché, without object-relational mapping, and without needing to learn or use Caché Studio. The high-performance object data access methods automatically generated by Caché exist independently of the Java class, so developers can use their POJOs within their application, and never think about the mechanics of how to store the data.

Persisting Java objects as objects, thus eliminating object-relational mapping, dramatically reduces development time. But it provides a performance benefit too, since there is no processing required to disassemble and reassemble objects as they are stored or retrieved.

Even though it is preferable to store data as objects, there are times when it may be necessary to query the database using SQL, either within the Java application, or through a third-party data analysis and reporting tool. Through its Unified Data Architecture, Caché automatically exposes data as relational tables. Again, no mapping is required. Caché is JDBC-compliant, and SQL queries in the application can share the same database connection as the object persistence methods used by Jalapeño technology.

Using Jalapeño technology

There are basically two things a Java developer needs to do to persist POJOs in Caché. The first is to create and compile Caché classes from the POJO definitions, the second is to include and use the Jalapeño Object Manager within the Java application.

Create the Caché classes

Jalapeño provides a utility (dubbed the Schema Builder) that automatically creates and compiles persistent Caché classes based on Java class definitions. (The Schema Builder is a Java class included within CacheDB.jar.) Developers can exercise control over how Caché structures the object model by inserting "annotations" in their POJO class definitions. Annotations (introduced in JDK 1.5) do not affect the runtime behavior of the Java class. They merely provide additional metadata about the desired Caché object model. An annotation might, for example, tell Caché that one class inherits from another, or that an index should be built on a given property of a class.

Examples 1A and 1B (next page) show a simple Java class definition, and the Caché class that is derived from it. Note that the Caché class inherits from an InterSystems-provided class called %Library.Persistent, and that an index has been created on the "Name" property.

Unless they wish to, Java developers never have to look at or think about the Caché classes created by the Schema Builder. They can program using POJOs in their favorite IDE. If their object model changes, all they need to do is to re-run the Schema Builder to have those changes reflected in the corresponding persistent Caché classes.

Using the Object Manager

In Caché's other Java binding mechanisms, when Caché classes are projected as Java proxy classes, the persistence methods (inherited from %Library.Persistent) are transformed into Java "accessor" methods. Every Java class includes its own set of accessor methods. By contrast, when using Jalapeño technology, the POJOs are not altered to include accessor methods. Instead, the Java application uses an element called the Object Manager, which establishes the database connection, instantiates the appropriate Caché classes, and constructs and runs the accessor methods.

The Object Manager is a Java class, provided by InterSystems as part of CacheDB.jar. The steps required to use it will be familiar to any Java programmer:

- Include CacheDB.jar in the CLASSPATH statement of the application
- Import the "pojo" package. (There may be other packages in CacheDB.jar that you will also wish to import.)
- Instantiate the Object Manager

Example 2 (page 4) shows an instantiation of the Object Manager. Note that the Object Manager establishes a JDBC connection with the database server. With Caché, object and JDBC database access can share the same connection. Therefore, the Object Manger can use high-performance persistence methods but the data can also be queried using SQL.

Example 1A – a Java Class

```
import com.intersys.pojo.annotations.CacheClass;
import com.intersys.pojo.annotations.Index;

@CacheClass(name="Person",primaryKey="ID",sqlTableName="PERSON")
@Index(description="Name Index on Person
table",name="PersonIndexOne",propertyNames={"name"},sqlName="PersonIDX")
public class Person {

        public String name;
        public String ssn;
   public String telephone;
}
```

Example 1B – the corresponding Caché Class

```
Class User.Person Extends %Library.Persistent [ ClientName = Person, Not ProcedureBlock,
SqlTableName = PERSON ]
{

Property name As %Library.String(JAVATYPE = "java.lang.String", MAXLEN = 4096);

Property ssn As %Library.String(JAVATYPE = "java.lang.String", MAXLEN = 4096);

Property telephone As %Library.String(JAVATYPE = "java.lang.String", MAXLEN = 4096);

Index PersonIndexOne On name [ SqlName = PersonIDX ];

XData JavaBlock
{
<JavaBlock><Package implementation="CacheRefactor.cache"
pojo="CacheRefactor"></Package><UseSameNames>false</UseSameNames><Name
implementation="Person"
pojo="Person"></Name><ResolveNameCollisions>false</ResolveNameCollisions><
EagerFetchRequired>true</EagerFetchRequired></JavaBlock>
}
```

Example 2 – instantiating the Object Manager

```
public DBService (String[] args)
      throws Exception
   {
      String          host = "localhost";
      String          username="_System";  // null for default
      String          password="sys";  // null for default


      for (int i = 0; i < args.length; i++)
         if (args[i].equals("-user"))
            username = args[++i];
         else if (args[i].equals("-password"))
            password = args[++i];
         else if (args[i].equals("-host"))
            host = args[++i];

      String          url="jdbc:Cache://" + host + ":1972/USER";

      Class.forName ("com.intersys.jdbc.CacheDriver");
      Connection connection =  DriverManager.getConnection (url, username, password);
      objectManager = ApplicationContext.createObjectManager (connection);
}
```

Deploying against a relational database

The Object Manager's ability to handle both object and relational data access becomes especially important when a Java application that was created with Jalapeño technology needs to be deployed against a relational database, rather than Caché. Deploying within a relational architecture is very simple, requiring only two additional steps.

First, an appropriate relational database schema must be created. Caché provides an export utility (as part of CacheDB.jar) that can project the object model – originally derived from POJO class definitions – as DDL files that can be imported into a relational database. It is important to note that this is not the same as Caché's standard relational projection. Because the object schema in Caché was created from Java class definitions, the export utility "knows" some things about the POJOs, and uses that information when it builds the relational data schema.

Once an appropriate relational database schema has been established, all that remains is to configure the Object Manager so that it connects to the relational database instead of Caché. The Object Manager will automatically use object persistence methods (Open, Save, New, Delete) when connecting to Caché, and relational persistence methods (Select, Update, Insert, Delete) when connecting to a relational database.

Although InterSystems' Jalapeño technology makes it easy to deploy a Java application in a relational environment, developers will find that their applications run faster when deployed on Caché. Not only does Caché enable high-performance object persistence, it has also been shown to respond to SQL queries – particularly complex queries – faster than relational databases.

Conclusion

Caché has long supported several ways of providing data persistence to Java applications, both via JDCB and object data access. But heretofore, these approaches have been "Caché-centric". Developers have been required to define objects in Caché, then project them to their Java environment.

Caché's new Jalapeño Technology gives developers the option of using a "Java-in" approach to achieving data persistence. Persistent Caché classes can be defined and compiled from POJO class definitions. At runtime, the database connection and persistence is handled by an Object Manager, which is provided as part of Jalapeño. Developers can use the original POJOs without thinking about how information gets persisted in the database.

In addition to freeing Java developers from the need to use the Caché IDE, it also frees them from the tedious and very time-consuming task of object-relational mapping. Caché allows both object and relational access to data over the same connection, so developers can think in terms of objects. Their Java applications can use high-performance object-oriented persistence methods, and query the Caché database using SQL, if appropriate.

Jalapeño technology does not constrain developers to deploy their applications on Caché. With minimal additional work, a Java application created with Jalapeño technology can run on a relational database, although the resulting performance is not likely to be as good.

InterSystems Corporation

World Headquarters
One Memorial Drive
Cambridge, MA 02142-1356
Tel: +1.617.621.0600
Fax: +1.617.494.1631

www.InterSystems.com