InterSystems IRIS Demo: Java Object Persistence with XEP

Version 2024.1
2024-05-16

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

InterSystems IRIS Demo: Java Object Persistence with XEP
InterSystems IRIS Data Platform Version 2024.1 2024-05-16

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

InterSystems IRIS Demo: Java Object Persistence with XEP

This article introduces the XEP API (`com.intersystems.xep`), which provides support for extremely fast Java object storage and retrieval on the InterSystems IRIS® data platform. It gives you a high level overview of the XEP approach to Java object persistence, and walks you through a simple scenario that demonstrates the main features of the API.

These activities are designed to use only the default settings and features, so that you can acquaint yourself with the fundamentals of XEP without having to deal with details that are beyond the scope of this overview. For the full documentation on XEP, see *Persisting Java Objects with InterSystems XEP*.

1 Rapid Java Object Storage and Retrieval

Java is an object oriented language, so it is natural for Java applications to model data as objects. However, this can cause problems when the application needs to store that data in a database. If you use JDBC to store and retrieve objects, you are faced with the problem of transforming your object data into a set of relational tables, and then transforming query resultsets back into objects. The usual solution to this problem is to use an object-relational mapping (ORM) framework such as Hibernate to automate the process. InterSystems IRIS does offer a standard Hibernate interface, and InterSystems recommends it for large, complex object hierarchies.

On the other hand, for applications that perform tasks such as real-time data acquisition, the main problem is speed rather than data complexity. Although Hibernate is by no means slow (if properly optimized), XEP is a better alternative for tasks that require extremely fast storage and retrieval of simple or moderately complex data. In most cases, it will be considerably faster than either Hibernate or JDBC.

2 How Does XEP Work?

XEP is a lightweight Java API that projects Java object data as *persistent events*. A persistent event is an instance of an InterSystems IRIS class (normally a subclass of `%Persistent`) containing a copy of the data fields in a Java object. Like any such instance, it can be retrieved by object access, SQL query, or direct global access.

Before a persistent event can be created and stored, XEP must analyze the corresponding Java class and *import a schema* into the database. A schema defines the structure of the persistent event class that will be used to store the Java objects. Importing the schema automatically creates a database extent for the persistent event class if one does not already exist. The information from the analysis may be all that XEP needs to import a simple schema. For more complex structures, you can supply additional information that allows XEP to generate indexes and override the default rules for importing fields.

After the schema has been created for a class, you can use various XEP methods to store, update, or delete events, run SQL queries, and iterate through query resultsets.

3 Try It! Hands-on with XEP

Now it’s time for you to try out XEP for yourself. This XepSimple demo is a very small program, but it provides examples for most of the key XEP features and will give you an overview of how the XEP API is used.

3.1 Before You Begin

To use the procedure, you will need a system to work on, with version 1.8 of the JDK and a Java IDE of your choice installed, and a running InterSystems IRIS instance to connect to. Your choices for InterSystems IRIS include several types of licensed and free evaluation instances; the instance need not be hosted by the system you are working on (although they must have network access to each other). For information on how to deploy each type of instance if you do not already have one to work with, see Deploying InterSystems IRIS in *InterSystems IRIS Basics: Connecting an IDE*. Connect your IDE to your InterSystems IRIS instance using the information in InterSystems IRIS Connection Information and Java IDEs in the same document.

3.2 Adding the Sample Code

The following demo shows you how to work with XEP and InterSystems IRIS. The XepSimple demo is a very small program, but it provides examples for most of the key XEP features and will give you an overview of how the XEP API is used.

You can just read through the code listed here, or you can download it from the InterSystems GitHub repository and run it for yourself. The download includes a ReadMe that contains all the information you will need to get started.

The demo program is divided into four sections, each of which demonstrates one of the four main XEP classes, EventPersister, Event, EventQuery, and EventQueryIterator:

class XepSimple

Java

```
package xepsimple;
import com.intersystems.xep.*;
import xep.samples.SingleStringSample;

public class XepSimple {
  public static void main(String[] args) throws Exception {
// Generate 12 SingleStringSample objects for use as test data
    SingleStringSample[] sampleArray = SingleStringSample.generateSampleData(12);

// EventPersister
    EventPersister xepPersister = PersisterFactory.createPersister();
    xepPersister.connect("127.0.0.1",51773,"User","_SYSTEM","SYS"); // connect to localhost
    xepPersister.deleteExtent("xep.samples.SingleStringSample");  // remove old test data
    xepPersister.importSchema("xep.samples.SingleStringSample");  // import flat schema

// Event
    Event xepEvent = xepPersister.getEvent("xep.samples.SingleStringSample");
    for (int i=0; i < sampleArray.length; i++) {
      SingleStringSample sample = sampleArray[i];  // array initialized on line 8
      sample.name = "Sample object #" + i;
      xepEvent.store(sample);
      System.out.println("Persisted " + sample.name);
    }

// EventQuery
    String sqlQuery = "SELECT * FROM xep_samples.SingleStringSample WHERE %ID BETWEEN ? AND ?";

    EventQuery<SingleStringSample> xepQuery = xepEvent.createQuery(sqlQuery);
    xepQuery.setParameter(1,3);    // assign value 3 to first SQL parameter
    xepQuery.setParameter(2,12);   // assign value 12 to second SQL parameter
    xepQuery.execute();            // get resultset for IDs between 3 and 12

// EventQueryIterator
    EventQueryIterator<SingleStringSample> xepIter = xepQuery.getIterator();
    while (xepIter.hasNext()) {
      SingleStringSample newSample = xepIter.next();
      newSample.name = newSample.name + " has been updated";
      xepIter.set(newSample);
      System.out.println(newSample.name);
    }

    xepQuery.close();
    xepEvent.close();
    xepPersister.close();
  } // end main()
} // end class XepSimple
```

XepSimple performs the following tasks:

- First, some sample objects are generated by calling a method of our sample data class, xep.samples.SingleStringSample.
- EventPersister *is the main entry point for XEP, and provides the connection to the database.*

  It creates an instance named *xepPersister*, which establishes a connection to the database and deletes any existing sample data. It then calls **importSchema()** to analyze the sample class and send the schema to the database, thus creating an extent to hold persistent SingleStringSample objects.

  EventPersister contains variables for the information needed to connect your InterSystems IRIS instance — `host`, `port`, `irisnamespace`, `username`, and `password`. Update them with the correct information for your instance, as described in InterSystems IRIS Connection Information in *InterSystems IRIS Basics: Connecting an IDE*; this is the same information you used to connect your IDE to the instance. Specify the **`USER`** namespace, as shown, or use another that you have created in your installed instance. If the instance is locally installed and the connection uses `localhost` as the server address, the program will use a local shared memory connection, which is even faster than the standard TCP/IP connection.

- Event *encapsulates an interface between a Java object and the corresponding database object.*

  Once the schema has been generated, *xepPersister* can create an Event object named *xepEvent* for the sample class. In the loop, each instance of SingleStringSample is modified and then persisted to the database. The *xepEvent* **store()** method takes advantage of the connection and schema defined in *xepPersister*.

- EventQuery *is used to prepare and execute SQL queries.*

  An EventQuery<SingleStringSample> object named *xepQuery* is created by passing a query string to the *xepEvent* object's **createQuery()** method. The string defines an SQL query that accepts two parameters (the `?` characters). The parameter values are defined by calls to **setParameter()**, and a call to **execute()** fetches the query resultset.

- EventQueryIterator *is used to read rows from the resultset and update or delete the corresponding persistent objects.*

  Now that *xepQuery* contains the query resultset, an iterator named *xepIter* can be created for it by calling **getIterator()**. In the loop, the iterator's **next()** method is used to get each row of data and assign it to a SingleStringSample object. The object is then modified, and the iterator's **set()** method updates the corresponding persistent object in the database.

- When processing is done, it cleans up by calling the **close()** methods for the XEP objects.

3.3 The SingleStringSample Class

In case you're curious, here's a listing of our sample class:

xep.samples.SingleStringSample

Java

```
public class SingleStringSample {
  public  String name;
  public SingleStringSample() {}
  SingleStringSample(String str) {
      name = str;
  }

  public static SingleStringSample[] generateSampleData(int objectCount) {
      SingleStringSample[] data = new SingleStringSample[objectCount];
      for (int i=0;i<objectCount;i++) {
          data[i] = new SingleStringSample("single string test");
      }
      return data;
  }
}
```

This class was chosen in part because annotations don't need to be added before XEP can generate a schema from it. Most classes will require one or more annotations for schema optimization (which is a subject beyond the scope of this document).

4 Next Steps

The XepSimple demo is designed to give you a taste of XEP without bogging you down in details, and is obviously not a model for real production code — it didn't even bother with exception checking. The most important simplification was in the sample data. You persisted a few tiny Java objects from a class that doesn't require annotation or other mechanisms to aid in schema generation and optimization. Real world applications will usually require some annotation (although typically less than Hibernate).

For an introductory online video, see Using XEP with Java Applications.

When you bring XEP to your production systems, you will need to understand the full range of tools that XEP provides for schema optimization, index control, batch loading, and other important tasks. The main XEP book, *Persisting Java Objects with InterSystems XEP*, contains a comprehensive description of these features. The sources listed at the end of this document describe other facets of InterSystems IRIS Java support.

To learn more about Java object persistence and other InterSystems Java interoperability technologies, see tand the following documentation:

- *InterSystems IRIS Basics: JDBC and InterSystems Databases* provides an introduction to connecting to InterSystems IRIS via JDBC: it offers quick facts, a special feature, and a chance to try it for yourself. This is the simplest starting point for becoming familiar with InterSystems IRIS Java support.
- "InterSystems Java Connectivity Options" in *Using Java with the InterSystems JDBC Driver* provides an overview of all InterSystems IRIS Java technologies enabled by the JDBC driver.
- InterSystems IRIS provides Java APIs for easy database access via SQL tables, objects, and multidimensional storage. See the following books for detailed information on each type of access:
  - *Using Java with the InterSystems JDBC Driver* for SQL table access. The InterSystems JDBC driver allows InterSystems IRIS to establish JDBC connections to external applications, and provides access to external data sources via SQL.
  - *Using the Native SDK for Java* for native multidimensional storage access. The InterSystems IRIS Native SDK allows you to directly access the native tree-based multidimensional storage data structures that underlie the InterSystems IRIS object and SQL table interfaces.

  - *Persisting Java Objects with InterSystems XEP* for object access. XEP is optimized for transaction processing applications that work with simple to medium complexity object hierarchies and require extremely high speed object data persistence and retrieval.

- "Hibernate Support" in the *Implementation Reference for Java Third Party APIs* describes the InterSystems IRIS dialect of Hibernate. This dialect implements support for Java Persistence Architecture (JPA), which is the recommended persistence technology for large, complex object hierarchies in Java projects.