文章
Louis Lu · 十一月 2, 2021 阅读大约需 分钟

IRIS 2021 摘文档 First Look 15 -- 使用 XEP 实例化存储 Java 对象

本文档介绍了 XEP API（com.intersystems.xep），它提供了在 InterSystems IRIS® 数据平台上极其快速的对 Java 对象存储和检索的能力。文档概述了使用 XEP 将 Java 对象持久化的方法，并引导您通过一个简单的场景来演示该 API 的主要功能。

这些演示的内容、只使用默认设置和功能，这样您就可以熟悉 XEP 的基本原理，而不必处理超出本概述范围的细节问题。有关 XEP 的完整文档，请参见 *Persisting Java Objects with InterSystems XEP*（*使用 InterSystems XEP 持久化 Java 对象*）。

要浏览所有的摘要（First Look），包括可以在 InterSystems IRIS 免费的评估实例 上执行的那些，请参见 InterSystems First Looks（InterSystems 摘要）。

1. 快速的 Java 对象存储和检索

Java 是一种面向对象的语言，因此对于 Java 应用程序来说，将数据建模为对象是很自然的。然而，当应用程序需要将数据存储在数据库中时，这可能会出现问题。如果您使用 JDBC 来存储和检索对象，您就面临着将对象数据转换为一组关系表，然后再将查询结果集转化为对象的问题。这个问题的通常解决方案是使用对象-关系映射（ORM）框架——如 Hibernate——来自动化这个过程。InterSystems IRIS 提供了一个标准的 Hibernate 接口，InterSystems 推荐它用于大型、复杂的对象层次结构。

另一方面，对于执行实时数据采集等任务的应用程序，主要问题是速度而不是数据复杂性，虽然 Hibernate 绝对不慢（如果适当优化的话），但对于需要极其快速地存储和检索简单或中等复杂数据的任务来说，XEP 是一个更好的选择。在大多数情况下，它将比 Hibernate 或 JDBC 快得多。

2. XEP 是如何工作的？

XEP 是一个轻量级的 Java API，它将 Java 对象数据作为 *持久化事件*（*persistent event*）来处理。持久化事件（persistent event）是 InterSystems IRIS 类（通常是 %Persistent 的子类）的一个实例，包含 Java 对象中数据字段的副本。与任何此类实例一样，可以通过对象访问、SQL 查询或直接 global 访问来检索它。

在创建和存储持久化事件（persistent event）之前，XEP 必须分析相应的 Java 类，并将 *schema* 导入数据库。Schema 定义了用于存储 Java 对象的持久化事件类的结构。如果持久化事件类还不存在，导入 schema 将自动为其创建数据库定义。来自分析的信息可能是 XEP 导入 schema 所需全部信息。对于更复杂的结构，您可以提供额外的信息，允许 XEP 生成索引并覆盖导入字段的默认规则。

在为类创建 schema 之后，您可以使用各种 XEP 方法来存储、更新或删除事件、运行 SQL 查询，并遍历查询结果集。

3. 试一试！ XEP 的实际操

现在是您自己尝试 XEP 的时候了。这个 XepSimple 演示是一个非常小的程序，但它提供了大多数重要的 XEP 功能的示例，并概述了如何使用 XEP API。

想试试 InterSystems IRIS Java 开发和互操性能的在线视频演示吗？请查看 Java QuickStart（Java 快速入门）！

用前须知

要使用该程序，您需在一个系统上安装 JDK 1.8 版本和任意的 Java IDE，并连接一个正在运行的 InterSystems IRIS 实例。您对 InterSystems IRIS 的选择包括多种已授权的和免费的评估实例；实例不需要您正在工作的系统（尽管它们必须具有网络访问权限）。关于如何部署每种类型的实例的信息（如果您还没有可使用的实例），请参见 *InterSystems IRIS Basics: Connecting an IDE*（*InterSystems IRIS 基础：连接一个 IDE*）中的 Deploying InterSystems IRIS（部署 InterSystems IRIS）。使用同一文档中的 InterSystems IRIS Connection Information（InterSystems IRIS 连接信息）和 Java IDE 中的信息，将 IDE 连接到您的 InterSystems IRIS 实例。

添加示例代码

下面的演示向您演示如何使用 XEP 和 InterSystems IRIS。XepSimple 演示是一个非常小的程序，但它提供了大多数关键 XEP 功能的示例，并概述了如何使用 XEP API。

您可以阅读这里列出的代码，或者您可以从 InterSystems GitHub 存储库下载并自己运行它。下载的内容包括一个 ReadMe，其中包含开始时需要所有信息。

演示程序分为四个部分，每个部分演示四个主要 XEP 类中的一个：EventPersister、Event、EventQuery 和 EventQueryIterator：

class XepSimple

```
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 loc
alhost
    xepPersister.deleteExtent("xep.samples.SingleStringSample");    // remove old tes
t data
    xepPersister.importSchema("xep.samples.SingleStringSample");   // import flat sch
ema

    // 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 执行以下任务：

- 首先，通过调用示例数据类（sample data class）xep.samples.SingleStringSample 的方法生成一些示例对象。
- EventPersister 是 *XEP 的主要入口点（main entry point），并提供到数据库的连接。*

它创建一个名为 *xepPersister* 的实例，该实例建立到数据库的连接并删除任何现有的示例数据。然后调用 **importSchema()** 来分析示例类并将 schema 发送到数据库，进而扩展空间保持久化 SingleStringSample 对象。

EventPersister 包含连接 InterSystems IRIS 实例所需信息——主机名、端口、IRIS命名空间、用户名和密码。使用实例的正确信息更新它们，如 *InterSystems IRIS Basics:Connecting an IDE*（《*InterSystems IRIS 基础：连接一个 IDE*》中的 InterSystems IRIS Connection Information（InterSystems IRIS 连接信息））所述，这与您用来连接 IDE 和实例的信息相同。可以指定使用 **USER** 命名空间，或者使用您在已安装的实例中创建的另一个命名空间。如果实例是本地安装的，并且连接使用 localhost 作为服务器地址，程序将使用本地共享内存连接，这比标准的 TCP/IP 连接还要快。

- Event *封装了（encapsulate）Java 对象和相应的数据库对象之间的接口。*

一旦生成 schema，*xepPersister* 就可以为示例类创建一个名为 *xepEvent* 的 Event 对象。在循环中，SingleStringSample 的每个实例都将被修然后被持久化到数据库中。*xepEvent* **store()** 方法使用了 *xepPersister* 中定义的连接和 schema。

- EventQuery *用于准备和执行 SQL 查询。*

名为 *xepQuery* 的 EventQuery<SingleStringSample> 对象是通过将查询字符串传递给 *xepEvent* 对象的 **createQuery()** 方法创建的。该字符串定义了一个接受两个参数（? 字符）的 SQL 查询。参数值是通过调用 **setParameter()** 定义的，调用 **execute()** 可查询结果集。

- EventQueryIterator *用于从结果集中读取行，并更新或删除相应的持久化对象。*

现在 *xepQuery* 包含了查询结果集，可以通过调用 **getIterator()** 为它创建一个名为 *xepIter* 的迭代器。在循环中，迭代器的 **next()** 方法被用来获取每一行数据并将其分配给

SingleStringSample 对象。然后修改对象，迭代器的 **set()** 方法更新数据库中相应的持久化对象。

- 当处理完成，它通过调用 XEP 对象的 **close()** 方法进行清理。

SingleStringSample 类

如果您感兴趣，下面是我们的示例类列表：

了解有关 XEP 和对象持久化的更多信息

xep.samples.SingleStringSample

```
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;
  }
}
```

之所以选择这个类，部分原因是在 XEP 生成 schema 之前不需添加标 注 annotation。大多数 类都需一个或多个标 注 来进行 schema 优化（这超出了本文的讨论范围）。

下一步

XepSimple 演示的目的是让您体验 XEP，而又不陷入细节困境，它并非用于实际生产代码的模型——它甚至没有进行异常检查。最重要的简化是在示例数据中。您不需标 注 或其他机制来帮助 schema 的生成优化，类持久化了一些很小的 Java 对象。实际应用程序通常需一些标 注（尽管通常比 Hibernate 少）。

当您把 XEP 引入生产系统时，您将需了解 XEP 为提要优化、索引控制、批量加载和其他重要任务提供的全部工具。主要的 XEP 书籍 *Persisting Java Objects with InterSystems XEP*（《使用 InterSystems XEP 持久化 Java 对象》）对这些功能进行了全面描述。本文档末尾列出的参考资料描述了 InterSystems IRIS 对 Java 支持的其他方面。

4. 了解有关 XEP 和对象持久化的更多信息

要了解有关 Java 对象持久化和其他 InterSystems Java 互操作的更多信息，请参见：

- *First Look: JDBC and InterSystems Databases*（概要：JDBC 和 InterSystems 数据库）

介绍了如何通过 JDBC 连接到 InterSystems IRIS
：它提供了产品简介、特殊功能介绍，以及一个亲自尝试的机会。这是一个简单了解并熟悉 InterSystems IRIS 对 Java 支持的起点。

- *Using Java with the InterSystems JDBC Driver（《在 InterSystems JDBC 驱动程序中使用 Java》）*中的“InterSystems Java Connectivity Options（InterSystems Java 连接选项）”概述了 JDBC 驱动程序支持的所有 InterSystems IRIS Java 技术。
- InterSystems IRIS 提供了 Java API，通过 SQL 表、对象和多维存储轻松访问数据库。有关每种类型访问的详细信息，请参见以下书籍：
  - *Using Java with the InterSystems JDBC Driver（《在 InterSystems JDBC 驱动程序中使用 Java》）*进行 SQL 表访问。InterSystems JDBC 驱动程序允许 InterSystems IRIS 建立到外部应用程序的 JDBC 连接，并通过 SQL 提供对外部数据源的访问。
  - *Using the Native API for Java（使用 Native API for Java）*进行本机多维存储访问。InterSystems IRIS 本机 API 允许您直接访问本机基树的多维存储数据结构，这些结构是 InterSystems IRIS 对象和 SQL 表接口的基础。

    *Persisting Java Objects with InterSystems XEP（使用 InterSystems XEP 持久化 Java 对象）*进行对象访问。XEP 针对事务处理应用程序进行了优化，这些应用程序处理简单到中等复杂的对象层次结构，并需要极快的对象数据持久化和检索。
- *Implementation Reference for Java Third Party APIs（《Java 第三方 API 的实施参考》）*中的 "Hibernate Support（Hibernate 支持）"描述了 InterSystems IRIS 的 Hibernate 实现。这个实现了对 Java Persistence Architecture（JPA）的支持，这是推荐用于 Java 项目中大型、复杂对象层次结构的持久化技术。

#InterSystems IRIS #InterSystems IRIS for Health

源 URL: https://cn.community.intersystems.com/post/iris-2021-%E6%8A%80%E6%9C%AF%E6%96%87%E6%A1%A3-first-look-15-%E4%BD%BF%E7%94%A8-xep-%E5%AE%9E%E4%BE%8B%E5%8C%96%E5%AD%98%E5%82%A8-java-%E5%AF%B9%E8%B1%A1