

Slice Documentation

Slice Documentation

1. Introduction	1
1.1. What is Slice?	1
1.2. Background	1
1.3. Why Slice?	1
1.4. Purpose of this Document	1
1.5. Intended Audience	1
2. Features and Limitations	2
2.1. Salient Features	2
2.1.1. Transparency	2
2.1.2. Custom Data Distribution Policy	2
2.1.3. Heterogeneous Database	2
2.1.4. Parallel Execution	2
2.1.5. Distributed Transaction	2
2.2. Limitations	2
2.2.1. Collocation Constraint	2
2.2.2. No Sorting	3
3. User Manual	4
3.1. Configuration	4
3.1.1. How to activate Slice Runtime?	4
3.1.2. How to configure each database slice?	4
3.1.3. Implement DistributionPolicy interface	5
4. Download Instruction	6
4.1. Get the Jar	6
4.2. Source Code Download	6
4.3. Build from Source	6
5. Release Notes	7
5.1. Changes in version 0.4.0	7
5.2. Changes in version 0.3.0	7
5.3. Changes in version 0.2.0	7
5.4. Changes in version 0.1.0	7
6. Project Information	8
6.1. Development Team	8
6.2. Mailing List	8

List of Tables

4.1.	6
-----------	---

Chapter 1. Introduction

1.1. What is Slice?

Slice extends OpenJPA for distributed databases. Slice plugs-in to OpenJPA runtime as a single library and can be activated by configuring a persistence unit for multiple databases.

Once configured for Slice, an existing OpenJPA application can transact against multiple databases in the same transaction. The application queries will be executed in parallel against all the databases and any update will be committed to the appropriate databases.

1.2. Background

Java Persistence API (JPA) is a industry-standard specification for persistence of Java objects to relational database. JPA standardizes the complex object-relational mapping problem and is supported by mature implementations from vendors such as BEA Kodo, JBoss Hibernate or Oracle Toplink. JPA is also the persistence service for EJB 3.0 in JEE environment.

OpenJPA - a top level Apache Open Source Project - is a feature-rich, production-quality implementation of JPA. OpenJPA supports numerous extended features over JPA standards and its well-crafted architecture allows a sophisticated plug-in framework for addition of new features.

Slice is one such plug-in extension to enable any OpenJPA-based application to work with distributed, horizontally-partitioned, possibly heterogeneous databases *without* any change to application code or the persistent domain model.

1.3. Why Slice?

Enterprise applications are increasingly using a distributed databases. The reasons for distributed, often horizontally-partitioned databases can be to counter massive data growth, or to support multiple external clients on a hosted platform and in many other scenarios that may benefit from data partitioning.

Any JPA-based user application has to address serious technical and conceptual challenges if it tries to directly interact with a set of physical databases within a single transaction. Slice encapsulates this complexity of distributed database interaction via the abstraction of a virtual database which internally manages multiple physical databases henceforth referred as *database slice*. This *virtualization* of distributed databases approach makes OpenJPA's object management kernel and the user application to work in the same way as in case of a single physical database. Via this database virtualization approach, Slice effectively provides the user application an object-oriented view over a distributed set of, possibly heterogeneous, databases,

1.4. Purpose of this Document

This document describes how to use Slice to enable OpenJPA applications for distributed database environment. This document also discusses the architecture and critical design aspects of Slice.

1.5. Intended Audience

This document is intended for developers interested in building distributed database applications using OpenJPA. This document assumes familiarity with OpenJPA usage and distributed database environment.

This document is also targeted for developers who are interested in extending OpenJPA or its basic architectural aspects.

Top

Chapter 2. Features and Limitations

2.1. Salient Features

The primary objective of Slice is to make building distributed database applications with OpenJPA simple.

2.1.1. Transparency

The existing application or the persistent domain model requires *no change* to upgrade from a single database to a distributed database environment.

2.1.2. Custom Data Distribution Policy

User application decides how the newly persistent instances be distributed across multiple database slices. The data distribution policy across the slices may be based on the attribute of the data itself. For example, all Customer whose first name begins with character 'A' to 'M' will be stored in one database while names beginning with 'N' to 'Z' will be stored in another slice. It is also common to find data distribution policies based on temporal attributes.

Slice tracks the original database for existing instances. When an application issues a query, the instances may be loaded from from different database slices. This tracking is important as update to any instance gets committed to the appropriate database.

2.1.3. Heterogeneous Database

Each slice can be configured independently with its own JDBC driver. Hence data can be distributed across heterogeneous databases.

2.1.4. Parallel Execution

All database operations such as query, commit or flush operates in parallel across the database slices.

2.1.5. Distributed Transaction

The database slices participate in a global transaction provided each slice is configured with a XA-complaint JDBC driver, even when the persistence unit is configured for `RESOURCE_LOCAL` transaction.

Warning

If any of the configured slices are non-XA-complaint *and* the persistence unit is configured for `RESOURCE_LOCAL` transaction then each slice is committed without any two-phase commit protocol. If commit on any slice fails, then atomicity of the transaction is not ensured.

2.2. Limitations

2.2.1. Collocation Constraint

No relationship can exist across database slices. In O-R mapping paradigm, this translates to the limitation that the closure of an object graph must be *collocated* in the same database. For example, consider a domain model where Person relates to Adress. Person X refers to Address A while Person Y refers to Address B. Collocation Constraint means that *both* X and A must be stored in the same database slice. Similarly Y and B must be stored in a single slice.

Slice, however, helps to maintain collocation constraint automatically. The instances in the closure set of any newly persistent instance reachable via cascaded relationship is stored in the same slice. The user-defined distribution policy requires to supply the slice for the root instance only.

2.2.2. No Sorting

If a query specified `ORDER BY` clause, then the results from individual database slices are sorted but the result is not sorted across multiple slices.

Top

Chapter 3. User Manual

Slice is designed for ease of use. The user application or the persistent entity model requires no change. A OpenJPA based application working against a single database can be upgraded to work against a set of databases by configuring a persistence unit in `META-INF/persistence.xml`.

3.1. Configuration

Slice is activated via the following property settings:

3.1.1. How to activate Slice Runtime?

The basic configuration property is

```
<property name="openjpa.BrokerFactory" value="slice"/>
```

This configuration activates a specialized factory class aliased as `slice` to create object management kernel that can work against multiple databases.

3.1.2. How to configure each database slice?

The next task is to configuration each database slice. Each slice is assigned a logical name. For example, the following configuration will register two slices with logical name `One` and `Two`.

```
<property name="slice.One.ConnectionURL" value="jdbc:mysql:localhost//slice1"/>
<property name="slice.Two.ConnectionURL" value="jdbc:mysql:localhost//slice2"/>
```

Any OpenJPA specific property can be configured per slice basis. For example, the following configuration will use two different JDBC drivers for slice `One` and `Two`.

```
<property name="slice.One.ConnectionDriverName" value="com.mysql.jdbc.Driver"/>
<property name="slice.Two.ConnectionDriverName" value="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource"/>
```

Any property if unspecified for a particular slice will be defaulted by corresponding OpenJPA property. For example, consider following three slices

```
<property name="slice.One.ConnectionURL" value="jdbc:mysql:localhost//slice1"/>
<property name="slice.Two.ConnectionURL" value="jdbc:mysql:localhost//slice2"/>
<property name="slice.Three.ConnectionURL" value="jdbc:oracle:localhost//slice3"/>

<property name="openjpa.ConnectionDriverName" value="com.mysql.jdbc.Driver"/>
<property name="slice.Three.ConnectionDriverName" value="oracle.jdbc.Driver"/>
```

In this example, Three will use slice-specific `oracle.jdbc.Driver` driver while slice One and Two will use the driver `com.mysql.jdbc.Driver` as specified by `openjpa.ConnectionDriverName` property value.

3.1.3. Implement `DistributionPolicy` interface

Slice needs to determine which slice will persist a new instance. The application can only decide this policy (for example, all `PurchaseOrders` before April 30 goes to slice One, all the rest goes to slice Two). This is why the application has to implement `org.apache.openjpa.slice.DistributionPolicy` and specify the implementation class in configuration

```
<property name="slice.DistributionPolicy" value="com.acme.foo.MyOptimalDistributionPolicy"/>
```

The interface `org.apache.openjpa.slice.DistributionPolicy` is simple with a single method. The complete listing of the documented interface follows:

```
public interface DistributionPolicy {
    /**
     * Gets the name of the slice where a given instance will be stored.
     *
     * @param pc The newly persistent or to-be-merged object.
     * @param slices name of the configured slices.
     * @param context opaque context for future use.
     *
     * @return identifier of the slice. This name must match one of the
     *         configured slice names.
     * @see DistributedConfiguration#getSliceNames()
     */
    String distribute(Object pc, Set<String> slices, Object context);
}
```

While implementing a distribution policy the most important thing to remember is **collocation constraint**. Because Slice can not establish or query any cross-database relationship, all the related instances must be stored in the same database slice. Slice can determine the closure of a root object by traversal of cascaded relationships. Hence user-defined policy has to only decide the database for the root instance that is the explicit argument to `EntityManager.persist()` call. Slice will ensure that all other related instances that gets persisted by cascade is assigned to the same database slice as that of the root instance. However, the user-defined distribution policy must return the same slice identifier for the instances that are logically related but not cascaded for `persist`.

Top

Chapter 4. Download Instruction

4.1. Get the Jar

Slice is available as a single jar and can be downloaded from the following location.

Table 4.1.

Version	Download	Release Notes
0.4.0	openjpa-slice-0.4.0.jar	Release Note
0.3.0	openjpa-slice-0.3.0.jar	Release Note
0.2.0	openjpa-slice-0.2.0.jar	Release Note
0.1.0	openjpa-slice-0.1.0.jar	Release Note

4.2. Source Code Download

Source code is maintained in Apache Subversion Repository. The following command will fetch the repository content to local machine, in `slice` subdirectory relative to the current directory.

```
svn co https://svn.apache.org/repos/asf/labs/fluid/slice slice
```

4.3. Build from Source

To build the project and run the tests, **Maven 2.0.x** is required. The following command will compile the source code, run the tests and create a `openjpa-slice-{VERSION}.jar`.

```
mvn package
```

Note

The test configuration `src/test/resources/META-INF/persistence.xml` currently assume that two MySQL databases named `slice1` and `slice2` are available in `localhost`.

To build the project without running the tests, run the following command

```
mvn package -Dtest=false
```

Top

Chapter 5. Release Notes

5.1. Changes in version 0.4.0

- Uses a native and naive internal Distributed Transaction Manager to enable 2-phase commit when underlying slices are XA-complaint.

5.2. Changes in version 0.3.0

- Automatic Collocation Constraint : Identifies the correct slice for instances that are newly persistent by cascade. Collocation constraint requires that the closure of any object graph be stored in the same slice. In earlier version, user-defined DistributionPolicy has to ensure this constraint by returning the same slice for all the instances in the same closure. Now DistributionPolicy only needs to be implemented for root instances i.e. the instances that are explicit argument of persist() call. The cascaded instances will be automatically assigned the same slice identifier of the root instance.

5.3. Changes in version 0.2.0

- Per-slice Configuration: Now each slice can be configured with separate database driver or database dictionary or any other property.
- The previous multi-URL configuration is no more supported.

5.4. Changes in version 0.1.0

- First working version for extending OpenJPA to horizontally-partitioned, distributed database.

[Top](#)

Chapter 6. Project Information

6.1. Development Team

Slice has been developed by Pinaki Poddar of **BEA Systems** under the aegis of **Apache Labs**.

6.2. Mailing List

Apache Labs mailing list : <labs-subscribe@labs.apache.org>

Pinaki Poddar can be reached at <pinaki.poddar@gmail.com>