DSpace Database Access

- Purpose
- DSO - DSpace Objects
- org.dspace.core.Context - DSpace Context
  - Differences between DSpace 5 and DSpace 6 Context
  - Curation Context (Curator.curationContext())
  - Context Configurations
    - Read Only Context in DSpace 6
    - Batch Context in DSpace 6
- Database Interaction (before DSpace 6.x)
- Data Access Objects (introduced in DSpace 6.x)
- Hibernate (introduced in DSpace 6.x)
  - Hibernate Annotations in DSpace
  - The Hibernate Session (Cache) and the Context Object
    - Development tips regarding Hibernate Session
  - The Life-cycle of a DSO with Hiberate
  - Hibernate Cache Management in DSpace Command Line Tools
  - Hibernate Issues Discovered in DSpace 6.1
  - Recommended use of Hibernate and the Context Object (DSpace 6.2 and beyond)
    - When to Construct a Context Object
    - When to use a Read Only Context
    - When to use a Batch Context
    - What is the Proper Way to Close a Context Object?
    - What is the Proper Way to Close a Read-Only Context Object?
    - What is the Proper Way to Close a Batch Context Object?
    - When to call Context.uncacheEntity()
    - When to call Context.reloadEntity()
- Hibernate Queries
  - Hibernate Criteria Queries
  - Hibernate Query Language (HQL)
- Hibernate Logging
- Common Hibernate Error Messages
  - LazyInitializationException
  - StaleStateException
- Hibernate Resources

Purpose

The purpose of this document is to provide an overview of the way that the DSpace code base interacts with a database.

The mechanisms for interacting with the database layer changed significantly in DSpace 6.x (see also DSpace Service based api). This document will highlight those differences.

This document will also outline additional changes that are anticipated in the development of DSpace 7.x.

DSO - DSpace Objects

A DSO is a DSpace Object (`org.dspace.content.DSpaceObject`). Most everything in DSpace is a DSO (e.g. Site, Community, Collection, Item, Bitstream, EPerson, Group). A DSO is saved to the database. Bitstreams are a special type of DSO that have binary storage (of a file) in addition to data in the database.

Each DSO is represented as a table in the DSpace database. Some additional tables are present to represent relationships between DSOs.

org.dspace.core.Context - DSpace Context

The DSpace Context Object contains information about about the user/session interacting with DSpace code.

The context object can be queried to determine the current user and the current user's locale.

The context object can be set to a privileged mode that can bypass all authorization checks.

The context object manages/maintains a list of "events" to dispatch after a commit() (or when dispatchEvents() is called). These events represent changes to objects in the system, and are responded to by Event Consumers.

The context object interacts with the DBConnection class to manage database commits, connection pooling, transactions, etc. The default DBConnection used is the HibernateDBConnection, which manages the Hibernate Session, Transaction, etc (more on that below).

Differences between DSpace 5 and DSpace 6 Context

In DSpace 5, each `new Context()` established a new DB connection. Context then committed or completed/aborted the connection after it was done (based on results of that request). A single Context could also be shared between methods if a single transaction needed to perform actions across multiple methods.

In DSpace 6, Hibernate manages the DB connection pool. Each **thread** is associated with a unique Hibernate Session (which corresponds to a DB connection in the pool). This means two Context objects may use the same DB connection (if they are in the same thread). In other words, code can no longer assume each `new Context()` is treated as a new/separate database transaction.

- This is controlled by the configuration "`hibernate.current_session_context_class`", which in DSpace is configured to use a separate Session per Thread: https://github.com/DSpace/DSpace/blob/dspace-6_x/dspace/config/hibernate.cfg.xml#L16

Please don't change this the property "hibernate.current_session_context_class" unless you really know what you are doing. It has huge impact on the software architecture. Changing the configuration without changing parts of DSpace's source code will probably result in a malfunctioning installation (and could result in data loss).

Curation Context (Curator.curationContext())

A context object built to be shared between Curation tasks.

Context Configurations

The DSpace Context Object can be constructed as a read-only context or a batch context. This mode determines how hibernate will flush changes to the database. It is good behavior to store the old context mode before changing it and to set it back to the old mode when you're done with your work. This reduces problems when code parts that needs to be able to update stored content calls parts of DSpace's code that uses a read only or a batch mode.

See https://github.com/DSpace/DSpace/blob/dspace-6.1/dspace-api/src/main/java/org/dspace/core/HibernateDBConnection.java#L148-L156

Read Only Context in DSpace 6

The Context object can be set to a read only mode for enhanced performance when processing many database objects in a read-only fashion. Objects read by a read only context are not intended to be modified. The context object should not be committed.

The read only context is intended for data security. Implementations using this context should not be able to accidentally save changes.

Batch Context in DSpace 6

Hibernate provides a mechanism to submit a large batch of changes to a database in a memory-efficient manner.

See https://docs.jboss.org/hibernate/orm/3.3/reference/en-US/html/batch.html

Database Interaction (before DSpace 6.x)

All Database actions require the presence of a Context object.

All DSOs are constructed with a Context object. The context object provides access to the database connections to create/retrieve/update/delete DSOs.

The context object is used to authorize access to particular actions.

Individual DSOs implement an update() method. This method calls org.dspace.storage.rdbms.DatabaseManager.update(). This is a helper class that helps to construct the SQL for a DSO.

Data Access Objects (introduced in DSpace 6.x)

The concept of a Data Access Object (DAO) was introduced in DSpace 6 to provide an optimization layer between the DSpace code and the DSpace database.

In DSpace 6, Hibernate was implemented as the DAO. The DAO concept would allow for a framework other than Hibernate to be implemented in the DSpace code base.

Here is the interface for the GenericDAO in DSpace 6: https://github.com/DSpace/DSpace/blob/master/dspace-api/src/main/java/org/dspace/core/GenericDAO.java

Hibernate (introduced in DSpace 6.x)

DSpace 6 introduced hibernate (http://hibernate.org/orm/) as an object relational mapping layer between the DSpace database and the DSpace code.

Objects accessed by hibernate are registered in the hibernate.cfg.xml file. DSO properties and relationships can be inferred from the database schema.

The following class provides a hibernate implementation of the GenericDAO interface.

- See https://github.com/DSpace/DSpace/blob/master/dspace-api/src/main/java/org/dspace/core/AbstractHibernateDAO.java

Because hibernate has a mechanism for automatically updating content that has changed, the save() method is not implemented.

- See https://github.com/DSpace/DSpace/blob/dspace-6.1/dspace-api/src/main/java/org/dspace/core/AbstractHibernateDAO.java#L43-L45

The save to the database is invoked when Context.commit() is called.

- See https://github.com/DSpace/DSpace/blob/dspace-6.1/dspace-api/src/main/java/org/dspace/core/Context.java#L404-L437

The hibernate commit is implemented in the following manner

- https://github.com/DSpace/DSpace/blob/dspace-6.1/dspace-api/src/main/java/org/dspace/core/HibernateDBConnection.java#L80-L86

Hibernate Annotations in DSpace

Additional relationships can be explicitly declared using Hibernate annotations.

The Hibernate Session (Cache) and the Context Object

Hibernate will intelligently cache objects in the current Hibernate Session, allowing for optimized performance. Each Hibernate Session opens a single database connection when it is created, and holds onto it until the session is closed. A Session may consist of one or more Transactions.

In DSpace, the Hibernate Session (and its Transactions) is managed by the `HibernateDBConnection` object. (NOTE: This class is perhaps unfortunately named as it manages the process of obtaining a database connection from Hibernate, via a Session. *It does not represent a single database connection.*)

- https://github.com/DSpace/DSpace/blob/dspace-6_x/dspace-api/src/main/java/org/dspace/core/HibernateDBConnection.java

The DSpace Context object has methods (like uncacheEntity() and reloadEntity()) which can manage objects cached within this Hibernate Session (via `HibernateDBConnection`).

Some care is needed to properly utilize the Hibernate cache. Objects are loaded into the Session cache on access. Objects are not removed from the cache until one of the following occurs:

- The Hibernate Session's Transaction is committed (e.g. via a call to `Context.commit()` or `Context.complete()`)
- The Hibernate Session's Transaction is rolled back (e.g. via a call to `Context.abort()`)
- The object is specifically "evicted" (i.e. uncached) from the Hibernate Session (e.g. via a call to `Context.uncacheEntity()`)

Be aware, once an object is removed (detached) from the Session cache, it will need to be reloaded from the database before it can be used again! This can be achieved via `Context.reloadEntity()` or by querying for the object again via its Service.

Development tips regarding Hibernate Session

A few tips on working with Hibernate Sessions (all gleaned from https://developer.atlassian.com/confdev/development-resources/confluence-architecture/hibernate-sessions-and-transaction-management-guidelines)

- **Hibernate sessions are not thread-safe**
  - Therefore, any new DSpace code must ensure it is not attempting to share objects or Sessions between threads. Instead, pass around UUIDs, so the new thread can load the referenced object in a new Session.
- The more objects you load during the lifetime of a Session, the less efficient each query will be
  - So, be sure to use `Context.commit()` or `Context.uncacheEntity()` when you are done with an object
  - (recommendation: offer very specific/limited guidance on when to call uncacheEntity())
- Because Hibernate has built-in Session caching, it is not recommended to cache objects elsewhere in your code. If you must perform other caching, store UUIDs instead
  - Caching objects elsewhere is likely to result in a `LazyInitializationException` if the object (cached elsewhere) outlives its Session. See "Common Hibernate Error Messages" below

The Life-cycle of a DSO with Hiberate

(Explanation needed for the states that an object can be in)

- Retrieved from hibernate
- Retrieved from hibernate, modified, unsaved
- Retrieved from hibernate, modified, saved
- "Detached" object

Hibernate Cache Management in DSpace Command Line Tools

Some DSpace command line tools process a large number of DSOs from a single Context object. In such a case, the hibernate cache can become too large and trigger memory exceptions.

In such a case, it is necessary to explicitly purge items from the DSpace cache.

For instance, when re-indexing DSpace, the entire hierarchy is traversed. DSOs are removed from the cache once they are no longer needed.

- See https://github.com/DSpace/DSpace/blob/dspace-6.1/dspace-api/src/main/java/org/dspace/discovery/IndexClient.java#L175-L176
- See https://github.com/DSpace/DSpace/blob/dspace-6.1/dspace-api/src/main/java/org/dspace/discovery/IndexClient.java#L183-L184
- See https://github.com/DSpace/DSpace/blob/dspace-6.1/dspace-api/src/main/java/org/dspace/discovery/IndexClient.java#L207-L208

Hibernate Issues Discovered in DSpace 6.1

Surprisingly, Hibernate Database Connections are shared between DSpace Context objects. Therefore, database connections used by read only contexts and by editable contexts are shared.

The proper commit/closure of a database connection differs for read only connections and writable connections. Since these connections are shared, unexpected behavior has been discovered when an incompatible database connection is used by a DSpace context.

- See https://git
  - See Unable to locate Jira server for this macro. It may be due to Application Link configuration.
- See https
  - Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Recommended use of Hibernate and the Context Object (DSpace 6.2 and beyond)

When to Construct a Context Object

When to use a Read Only Context

When to use a Batch Context

What is the Proper Way to Close a Context Object?

What is the Proper Way to Close a Read-Only Context Object?

What is the Proper Way to Close a Batch Context Object?

When to call Context.uncacheEntity()

When to call Context.reloadEntity()

Hibernate Queries

In order to take advantage of the hibernate cache and other hibernate features, all queries for DSOs will be performed through the hibernate framework rather than by generating SQL explicitly.

Hibernate Criteria Queries

This allows the construction of a query in an object-oriented fashion.

Hibernate Query Language (HQL)

HQL is a SQL-like query language that references hibernate object properties rather that table column names.

Hibernate Logging

If you wish to see Hibernate queries and their parameters logged in your DSpace log files (dspace.log.*), you can update the `log4j.properties` A1 appender as follows:

```
# Log all Hibernate queries (does not include query params)
log4j.logger.org.hibernate.SQL=DEBUG, A1
# Log all Hibernate query parameters (immediately after query they pertain to)
log4j.logger.org.hibernate.type.descriptor.sql=TRACE, A1
```

Common Hibernate Error Messages

LazyInitializationException

For example: `LazyInitializationException: failed to lazily initialize ... could not initialize proxy - no Session`

- This error means that a Hibernate managed object has outlived its Session. In other words, the object has become "detached" or disconnected from a Session.
- In DSpace, this often means either Context.uncacheEntity() was called on the object too soon, or it a Context.commit() was called (which clears it from Session) and the object was used again later (without first calling Context.reloadEntity())
- More info and tips on avoiding these exceptions: https://developer.atlassian.com/confdev/development-resources/confluence-architecture/hibernate-sessions-and-transaction-management-guidelines

StaleStateException

For example: `StaleStateException: Batch update returned unexpected row count from update`

- This error means that your Hibernate tried to update an object that either no longer exists in the Database, or the update already previous occurred. In other words, the state of this object was "stale" in the Hibernate cache, and its state in the Database was different.
- In DSpace, this may mean that Context.commit() should have been called previously to save the object in question (and ensure the cache and database are synced).

Hibernate Resources

- https://www.lynda.com/Java-tutorials/Java-Database-Access-Hibernate/534635-2.html
  - Requires a lynda.com subscription
- Hibernate Transactions / Sessions / Caching (a guide for Atlassian developers)
  - https://developer.atlassian.com/confdev/development-resources/confluence-architecture/hibernate-sessions-and-transaction-management-guidelines
  - https://developer.atlassian.com/confdev/development-resources/confluence-architecture/hibernate-sessions-and-transaction-management-guidelines/hibernate-session-and-transaction-management-for-bulk-operations