IssueLog

IssueLog contains changes requested by the community for JSR-243

As changes are requested by the community (for example, on mailing lists or JIRA) they will be added to the PROPOSED list here. When they are implemented, they are moved to ACCEPTED. When a maintenance release is being shipped, unimplemented items will be moved to the DEFERRED list. When a maintenance release is being planned, items from the DEFERRED list are reviewed.

ACCEPTED

1. Maybe we should look at another constructor for the SingleFieldIdentity classes that embeds the knowledge that the class name is exact? That way, the jdo implementation could construct identity instances and the user could also construct identity instances that represent the exact class.
2. JDO doesn't have a mechanism to stop queries from overrunning. JPA2 now allows a persistence property to allow timing them out, and most JDO implementations have allowed this as an extension since JDO1. It would make sense for JDO (2.3) to have the same or a variation. Some ideas

Option 1 : Simple PMF property "javax.jdo.option.queryTimeout" to specify the number of millisecs (or secs) before any query is timed out. Throw a QueryTimeoutException (extends JDOException) when the timeout happens

Option 2 : as Option1, plus setTimeout() on Query to define it on a per query basis.

Option 3 : as Option2, plus we add cancel() on Query so that users can cancel long running queries programmatically, throwing a QueryInterruptedExceptio
(extends JDOUserException). The cancel would apply to all currently running invoked queries from that Query instance.

PROPOSED

DEFERRED

1. It would be nice to get List.get(int) into JDOQL, in other words access of indexed list elements. Querydsl has support for that and for consistency it would be nice if JDOQL supported it as well.
2. I just took a look at the JDO javadoc and saw that it says there, the declared parameters and the parameters passed to the execute method have to match exactly:

<http://db.apache.org/jdo/api23/apidocs/javax/jdo/Query.html#executeWithMap%28java.util.Map%29>

IMHO this is a feature making the use of JDO with dynamic queries (and implicit parameter declarations) more complicated than necessary. It would be more developer-friendly, if it was legal to pass more parameters than the ones actually used, because they don't harm and it removes the need for additional tracking logic (which parameter is used, which one isn't).

I therefore recommend to relax the rule from exact match to "super-set", i.e. only these 2 requirements:

- All declared parameters must exist in the params-map passed to the

method executeWithMap(...).

- All types of the declared parameters must match the ones passed to

executeWithMap(...).

For downward compatibility, I recommend introducing an option that can be passed to the PersistenceManagerFactory (in the properties). As a first idea, I recommend: javax.jdo.option.QueryParameterExactMatch (with values "true" and "false").

1. There is increasing interest in NoSQL datastores (Google BigTable, Apache HBase, VMWare Redis, etc), which not only do not support SQL, but also do not necessarily provide support for traditional consistency or queriability features or guarantees, instead offering features like eventual consistency, key-value storage mechanisms, etc.

This request is to modify the JDO specification (and TCK & RI) so that it relaxes certain portions of the specification, perhaps in the form of profiles similar to JavaEE 6 profiles, to allow datastores that may not support queries in general, do not support the ACID requirements, or that support key-value-based storage mechanisms to be JDO-compliant. Additions to the specification may also be needed in order to directly address NoSQL-type datastores, in a manner similar to its treatment of relational persistence mechanisms. Additionally, this request would serve to better support persistence on micro platforms where consistency, queriability, etc, may not be supported.

1. Consider extending JDOQL in JDO3.1 to include a series of new methods, following the contracts of existing Java classes

String.charAt(int) String.startsWith(String, int) String.length() String.trim()

Enum.ordinal() Enum.toString()

The following are obviously deprecated in the JDK, but worth consideration Date.getHour() Date.getMinutes() Date.getSeconds() Date.getDay() Date.getMonth() Date.getYear()

We already have JDOHelper.getObjectId(Object), so why not add JDOHelper.getVersion(Object)

On a related topic JDO-633 has "List.get(int)".

Note that all of these are already implemented in the RI.

We already have Math.abs, Math.sqrt, so why not also have

Math.acos(number) Math.asin(number) Math.atan(number) Math.ceil(number) Math.cos(number) Math.exp(number) Math.floor(number) Math.log(number) Math.sin(number) Math.tan(number)

1. With an RDBMS datastore, when generating the schema, it is desirable to be able to specify the positioning of the column(s) of a field in the generated table.

With spreadsheet documents, it is critical to be able to define which column number is used for a particular field. With other datastores it is also likely desirable.

To achieve this I propose adding an attribute to the XML <column> called "position". Similarly for the @Column annotation. This can take integer values with 0-origin.

Complications :

- how this is handled when we have inheritance. If a root class has subclass-table inheritance, then those fields are persisted into the tables of all subclasses, and we may want to override the positions for those individually - to that end the user can override the metadata of the superclass fields.
- anything else ?
    1. Create an implementation of the API in javascript.
    2. I propose to split the standardization of the statically typed query capability into a general part and a JDOQL specific part. A fork of certain Querydsl parts could provide a foundation for both.

The reason for the splitted approach is that a standardization of the general model would benefit also other frameworks and could become a foundation for LINQ in Java.

The general part could be formed of the contents of the following Querydsl packages :

com.mysema.query com.mysema.query.types
com.mysema.query.types.custom com.mysema.query.types.expr
com.mysema.query.types.path com.mysema.query.types.query

The JDO specific elements could consist of the following elements :

com.mysema.query.jdoql.JDOQLSubQuery -> javax.jdo.query.SubQuery ?!? com.mysema.query.jdoql.JDOQLQuery \-> javax.jdo.query.Query ?!?

of course with some modifications and a standardization of the code generation (probably also via APT).

What do you think about this approach?

Are there any plans to add Query by Criteria to the JDO standard? I'm referring to a programmatic API to build queries, similar to what Hibernate already has and JPA is planning to add.

If not, does anyone know of another route to achieving this functionality? Are there any libraries that can be layered over JDO to get this? Should I write my own?

QueryDSL already does that for JDOQL http://source.mysema.com/display/querydsl/Querydsl I'm sure the guys who wrote it would love to hear feedback 😉

1. Aggregate type specification at page 175 claims that avg(expression) return type is the same of the enclosed expression. This is wrong from a strictly mathematical point of view. In fact, if you select an avg on an int field member the result is truncated. The result type should always be double.
2. I am implementing the shortcut form for if-then-else (x ? y : c) in Querydsl. Is there support for it in JDOQL? If not, could it be added?

1. There is renewed interest (e.g. cloud computing) in persistence APIs that don't have a fully ACID compliant, fully featured SQL back end, and yet the data is not simply key-value blob storage.

A few years ago some of us remember dark clouds on JDO's horizon. Recently there were some of a different kind http://www.datanucleus.com/news/google_app_engine.html

1. "QueryDSL" (documented at http://source.mysema.com/display/querydsl/Querydsl) now provides beta support for JDOQL using type-safe syntax. Some basic docs

are found here http://source.mysema.com/display/querydsl/JDOQL+Guide

Would be great if JDO supported streams, similar to OPENJPA-130