Polyglot Persistence: NoSQL, Relational, and Beyond

Doug Clarke
Oracle
Ottawa, Canada

Keywords:

Java, Persistence, JPA, NoSQL

Introduction

Java data access today isn't just about reading and writing from relational databases. It's also about being able to persist your objects in NoSQL databases and being able to cache them in data grids so you can scale out your application to hundreds of servers. It's about mapping your objects to XML and to JSON for on disk storage or for use in RESTful web services. As Martin Fowler described in his blog posting entitled 'PolyglotPersistence', the persistence needs of applications are evolving from predominantly relational to a mixture of heterogeneous data sources. In response to these needs EclipseLink, the Java Persistence API 2.0 and 2.1 reference implementation, is making support for NoSQL and other non-relational databases a first class citizen. With EclipseLink it's possible to construct applications that mix entities sourced from many types of databases and vendors in a single persistence unit. It's also possible to define and navigate relationships between entities persisted in different database technologies.

The Problem Space

Java Persistence is most commonly associated with Relational Database usage with Object-Relational mapping standardized by the Java Persistence API (JPA) standard. However, the scope of the problem space most developers actually deal with in today's applications is much larger. It includes not only physical storage of the data but also in transformations dealing with XML and JSON and the service interfaces that require these (Figure 1).

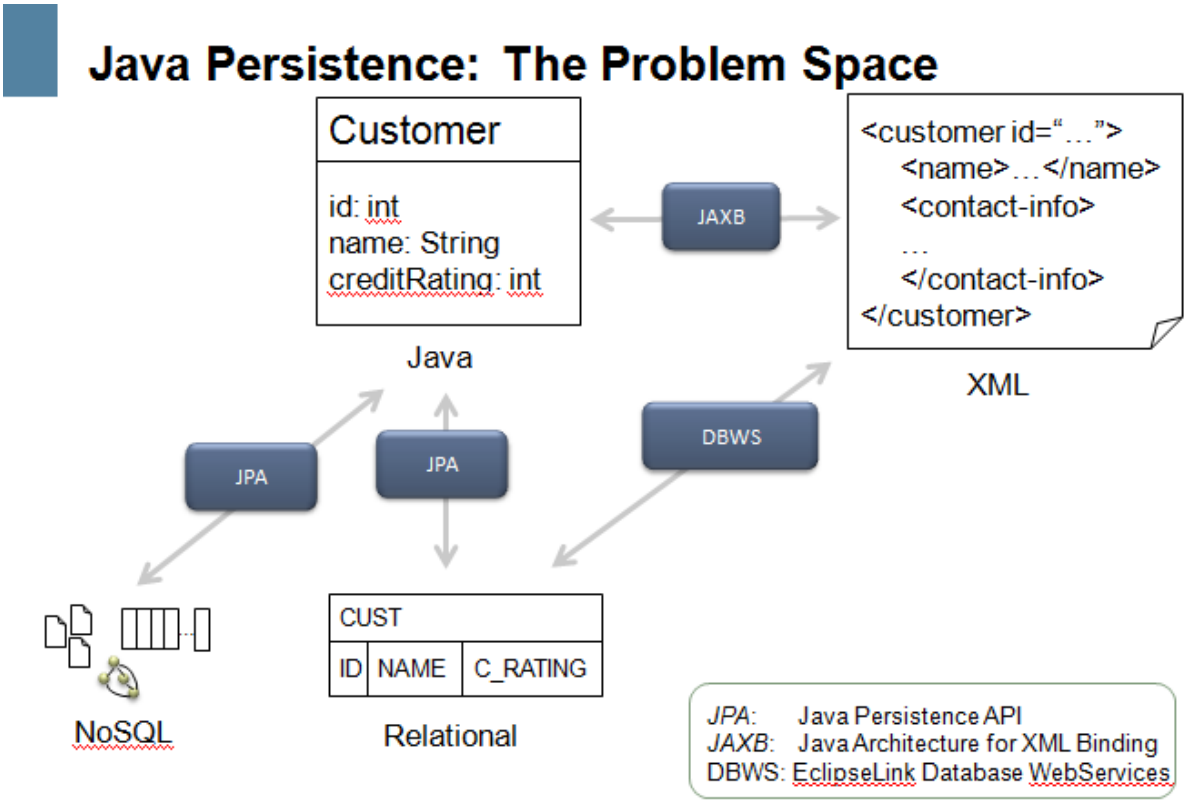

Figure 1: Problem Space

JPA & NoSQL

The Java Persistence API defines how Java objects are stored in relational database, a programmer API for reading, writing, and querying these persistent Java objects ("Entities"), a full featured query language in JP QL, and a container contract that supports plugging any JPA runtime in. This standard allows developers to leverage JPA providers and the relational databases they back with a common knowledge base and tooling support and allows persistence units to be written that can be run on different JPA providers and different backing databases.

NoSQL database are increasingly popular but they do present some challenges when it comes to aligning with JPA. There are many categories or types of NoSQL data stores with no common definition (e.g., document, graph, columnar). Each offers differing feature sets with some providing a query language/API and some not. There are no standards which means that every database offers a unique API. This introduces a cost in terms of learning and offers zero portability across databases.

EclipseLink NoSQL

Starting with EclipseLink 2.4 there is now support for NoSQL databases provided by the NoSQL component. Features include:

- JPA access to NoSQL databases
    - Leverage non-relational database support for JCA (and JDBC when available)
- Annotations and XML to identify NoSQL stored entities (e.g., @NoSQL)
- JPQL subset for each database technology

    - Key principal: leverage what's available
- Initial support for MongoDB and Oracle NoSQL.
- Mixing relational and non-relational data in single composite persistence unit ("polyglot persistence")

Mapping to NoSQL

Here is an example of an entity class mapped to MongoDB.

```
@Entity
@NoSql(dataFormat=DataFormatType.MAPPED)
public class Order {
    @Id // Use generated OID (UUID) from Mongo.
    @GeneratedValue
    @Field(name="_id")
    private String id;
    @Basic
    private String description;
    @OneToOne(cascade={CascadeType.REMOVE, CascadeType.PERSIST})
    private Discount discount;
    @ElementCollection
    private List<OrderLine> orderLines = new ArrayList<OrderLine>();
```

Querying

- JPQL

```
Select o from Order o
  where o.totalCost > 1000
Select o from Order o
  where o.description like 'Pinball%'
Select o from Order o
  join o.orderLines l where l.cost > :cost
```

- Native Queries

```
query = em.createNativeQuery(
  "db.ORDER.findOne({\"_id\":\"" +
  oid + "\"})", Order.class);
Order order =
  (Order) query.getSingleResult();
```

Polyglot Persistence

Although the term was not first coined by Martin Fowler his description is a good starting point.

> *One of the interesting consequences of this is that we are gearing up for a shift to polyglot persistence - where any decent sized enterprise will have a variety of different data storage technologies for different kinds of data. There will still be large amounts of it managed in relational stores, but increasingly we'll be first asking how we want to manipulate the data and only then figuring out what technology is the best bet for it.*
>
> *http://martinfowler.com/bliki/PolyglotPersistence.html*

With the EclipseLink support for JPA using relational databases and its new NoSQL functionality allowing a similar approach we can now more easily adopt a polyglot persistence strategy within our application development.

EclipseLink's approach allows:

- Relational and NoSQL databases each have their strength - choose the right one for the job
- A single application may have need for both relational and NoSQL data
- EclipseLink JPA supports use of multiple database technologies in the same application

- Relationships can span databases and database technologies

What's Next?

There is currently no formal standard for accessing NoSQL databases through a common Java API. While some believe that JPA provides a good starting point for a common API there are still many technical challenges ahead. The EclipseLink NoSQL component has been introduced to gain initial feedback from the community and help drive how this space may evolve. We are eager to have developers try out our approach and provide feedback to both the EclipseLink project and the broader Java community to help shape a Java Persistence API for NoSQL.

Contact address:

Doug Clarke
Oracle Canada ULC
45 O'Connor Street
K1P 1A4, Ottawa
Canada

Email douglas.clarke@oracle.com