EclipseLink: Beyond Relational and NoSQL to Polyglot and HTML5

Shaun Smith  shaun.smith@oracle.com
Gunnar Wagenknecht  gunnar@wagenknecht.org
About Us

- **Shaun Smith**
  - Eclipse committer on EclipseLink and related projects
  - Oracle TopLink Product Manager
  - OO developer since 1987, so I’m old enough to know Smalltalk ;)
- **Gunnar Wagenknecht**
  - CTO @ AGETO
  - Committer and contributor @ Eclipse
  - Java since 1999; Eclipse since 2001
Agenda—Database to Browser (and back again)

- Browser
- HTTP / REST
- Binding Persistence
  - XML/JSON
- Java
- Database Persistence
  - Relational/NoSQL
- Relational DB | NoSQL DB
Agenda

Browser

HTTP / REST

Binding Persistence
XML/JSON

Java

Database Persistence
Relational/NoSQL

Relational DB | NoSQL DB
NoSQL Databases

- EclipseLink is best known for relational persistence but…
- NoSQL database are increasingly popular
- No common definition (document, graph, columnar)
  - Differing feature sets
  - Some offer query language/API—some not
- No standards
- Every database offers a unique API
  - Cost in terms of learning
  - Zero portability across databases
EclipseLink NoSQL

- Support JPA-style access to NoSQL databases
  - Leverage non-relational database support for JCA (and JDBC when available)
- Define annotations and XML to identify NoSQL stored entities (e.g., @NoSQL)
- Support JPQL subset for each
  - Key principal: leverage what’s available
- Initial support for MongoDB and Oracle NoSQL.
- Support mixing relational and non-relational data in single composite persistence unit (“polyglot persistence”)

Support JPA-style access to NoSQL databases

- Leverage non-relational database support for JCA (and JDBC when available)

Define annotations and XML to identify NoSQL stored entities (e.g., @NoSQL)

Support JPQL subset for each

- Key principal: leverage what’s available

Initial support for MongoDB and Oracle NoSQL.

Support mixing relational and non-relational data in single composite persistence unit (“polyglot persistence”)
Applicability of JPA to NoSQL

- Core JPA concepts apply to NoSQL:
  - Persistent Entities, Embeddables, ElementCollection, OneToOne, OneToMany, ManyToOne, Version, etc.
- Some concepts apply with some databases:
  - JPQL, NamedNativeQuery
- Pure relational concepts don’t apply:
  - CollectionTable, Column, SecondaryTable, SequenceGenerator, TableGenerator, etc.
Querying NoSQL with JPA

- Two kinds of queries
  - JQPL—portable query language defined by the spec
  - Native query—lets you leverage database specific features
  - Dynamic or static @NamedQuery

- JPQL translated to underlying database query framework.
Example MongoDB Mapped Entity

```java
@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>();
```
MongoDB Query Examples

- **JPQL**

  ```java
  Select o from Order o
  where o.totalCost > 1000
  ```

  ```java
  Select o from Order o
  join o.orderLines l where l.cost > :cost
  ```

- **Native Queries**

  ```java
  query = em.createNativeQuery("db.ORDER.findOne({_id:" +
  oid + "}), Order.class);
  Order order =
  (Order) query.getSingleResult();
  ```
Demo EclipseLink NoSQL
“...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...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.”

**Martin Fowler**
ThoughtWorks
Composite Persistence Unit

Composite PU

Relational PU  NoSQL PU
Demo Polyglot Persistence
Agenda

Browser

HTTP / REST

Binding Persistence
XML/JSON

Java

Database Persistence
Relational/NoSQL

Relational DB | NoSQL DB
Binding Persistence

XML and JSON Binding

- EclipseLink implements
  - JAXB for Java/XML binding—covert Java to/from XML
  - Java/JSON binding—convert Java to/from JSON

- Currently no Java/JSON binding standard
  - Java API for JSON Processing (JSR 535) is parsing, not binding

- EclipseLink interprets JAXB XML bindings for JSON
  - Content-type selectable by setting property on Marshaller/Unmarshaller
XML and JSON from JAXB Mappings

```java
@XmlRootElement
public class Customer {
    private long id;
    private String firstName;
    private String lastName;
    private Address address;
    @XmlElementWrapper(name = "phone-numbers")
    @XmlElement(name = "phone-number")
    private Set<PhoneNumber> phoneNumbers;
}

{  
    "phone-numbers": [{  
        "id": 2,  
        "num": "512-555-9999",  
        "type": "mobile"  
    }],  
    "address": {  
        "city": "New York",  
        "id": 1,  
        "street": "Central Park East"  
    },  
    "firstName": "Woody",  
    "id": 1,  
    "lastName": "Allen"  
}

<?xml version="1.0" encoding="UTF-8"?>
<customer>
  <phone-numbers>
    <phone-number>
      <id>2</id>
      <num>512-555-1234</num>
      <type>home</type>
    </phone-number>
  </phone-numbers>
  <address>
    <city>New York</city>
    <id>1</id>
    <street>Central Park East</street>
  </address>
  <firstName>Bill</firstName>
  <id>1</id>
  <lastName>Allen</lastName>
</customer>
```
Challenges – Binding JPA Entities to XML/JSON

- Bidirectional/Cyclical Relationships
- Composite Keys/Embedded Key Classes
- Byte Code Weaving

```xml
<?xml version="1.0" ?>
<employee>
    <first>Mark</first>
    <last>Twain</last>
    <id>1</id>
</employee>
```
Bidirectional Relationship

```java
@Entity
public class Project{
    ...
    @OneToMany(mappedBy="project")
    private List<Employee> members;
}

@Entity
public class Employee{
    ...
    @ManyToOne
    private Project project;
}
```
Bidirectional Relationships in JAXB

- JAXB specification does not support bidirectional relationships. One side must be marked `@XmlTransient`.
- But that loses the relationship!

```xml
<?xml version="1.0" ?>
<employee>
  <first>Mark</first>
  <last>Twain</last>
  <id>1</id>
</employee>
```

Marshall

Unmarshall
EclipseLink XmlInverseReference

```java
@Entity
public class Project{
    ...
    @OneToMany (mappedBy="project")
    private List<Employee> members;
}

@Entity
public class Employee{
    ...
    @ManyToOne
    @XmlInverseReference (mappedBy="members")
    private Project project;
```
EclipseLink XmlInverseReference

- EclipseLink restores relationships on unmarshall!

```xml
<?xml version="1.0" ?>
<employee>
  <first>Mark</first>
  <last>Twain</last>
  <id>1</id>
</employee>
```

Marshall

Unmarshall
Demo

JAXB/JPA Fidelity

JSON Binding
JPA-RS

GET http://.../persistence/Accounting/Invoice/...

JAX-RS http://.../persistence/Accounting/Invoice/... mapped to JPA-RS service

JPA-RS maps URI http://.../persistence/Accounting/Invoice/... to Accounting PU and Invoice entity

JPA
JPA-RS Features

- Access relational data through REST with JSON or XML
- Provides REST operations for entities in persistence unit (GET, PUT, POST, DELETE)
- Automatic generation of XML and JSON bindings
- Supports invocation of named queries via HTTP
- Server Caching—EclipseLink clustered cache
- Dynamic Persistence also supported
  - Entities defined via metadata—no Java classes required
  - Enables persistence services for HTML 5/JavaScript applications
Resource Model & Links

{
    "firstName": "Frank",
    "gender": "Male",
    "id": 1,
    "lastName": "Smith",
    "responsibilities": [],
    "salary": 1,
    "version": 13,
    "address": {
        "_link": {
            "href": "http://localhost:7001/employee.web-js/persistence/employee/entity/Address/18",
            "method": "GET",
            "rel": "self"
        }
    }
}

...
Demo EclipseLink JPA-RS
Releases & Roadmap

EclipseLink 2.1 Helios
- Query Extensions
- TREAT AS
- FUNC
- Batch IN and EXISTS
- Attribute Group
  - Load
  - Fetch
  - Copy
  - Merge
- eclipselink-oxm.xml
- Dynamic MOXy

EclipseLink 2.3 Indigo
- Tenant Isolation
  - SINGLE_TABLE
- Extensible
- External Metadata
- Multiple DBs
- Data Partitioning

EclipseLink 2.4 Juno
- JSON Binding
- JPA-RS
- Tenant Isolation
  - Tenant per Table/Schema
- ALTER Schema
- NoSQL

EclipseLink 2.5 Kepler
- JPA 2.1
- JPA-RS++ …
Summary

- Java is evolving—and EclipseLink is evolving too!
  - JSON Binding
  - JAXB/JPA Fidelity
  - JPA-RS
  - NoSQL
  - Polyglot Persistence