High Performance JPA mit Oracle Coherence

Agenda

1. What is JPA?
2. What is Coherence?
3. Why Coherence with JPA?
4. „JOTG“ - JPA On The Grid
    1. JPA Backed Caches
    2. JPA 2nd Level Cache
    3. JPA 2nd Level Cache with JPA Backed Cache
5. Summary

http://blog.eisele.net

http://twitter.com/myfear

markus.eisele@msg-systems.com

What is JPA?

- The Java Persistence API (JPA) is a Java programming language framework managing relational data in applications
- The Java Persistence API originated as part of the work of the JSR 220 Expert Group. JPA 2.0 is the work of the JSR 317 Expert Group.
- Persistence in this context covers three areas:
  - the API itself, defined in the javax.persistence package
  - the Java Persistence Query Language (JPQL)
  - object/relational metadata

- There are different JPA providers available:

What is Coherence?

ORACLE FUSION MIDDLEWARE

TopLink Grid

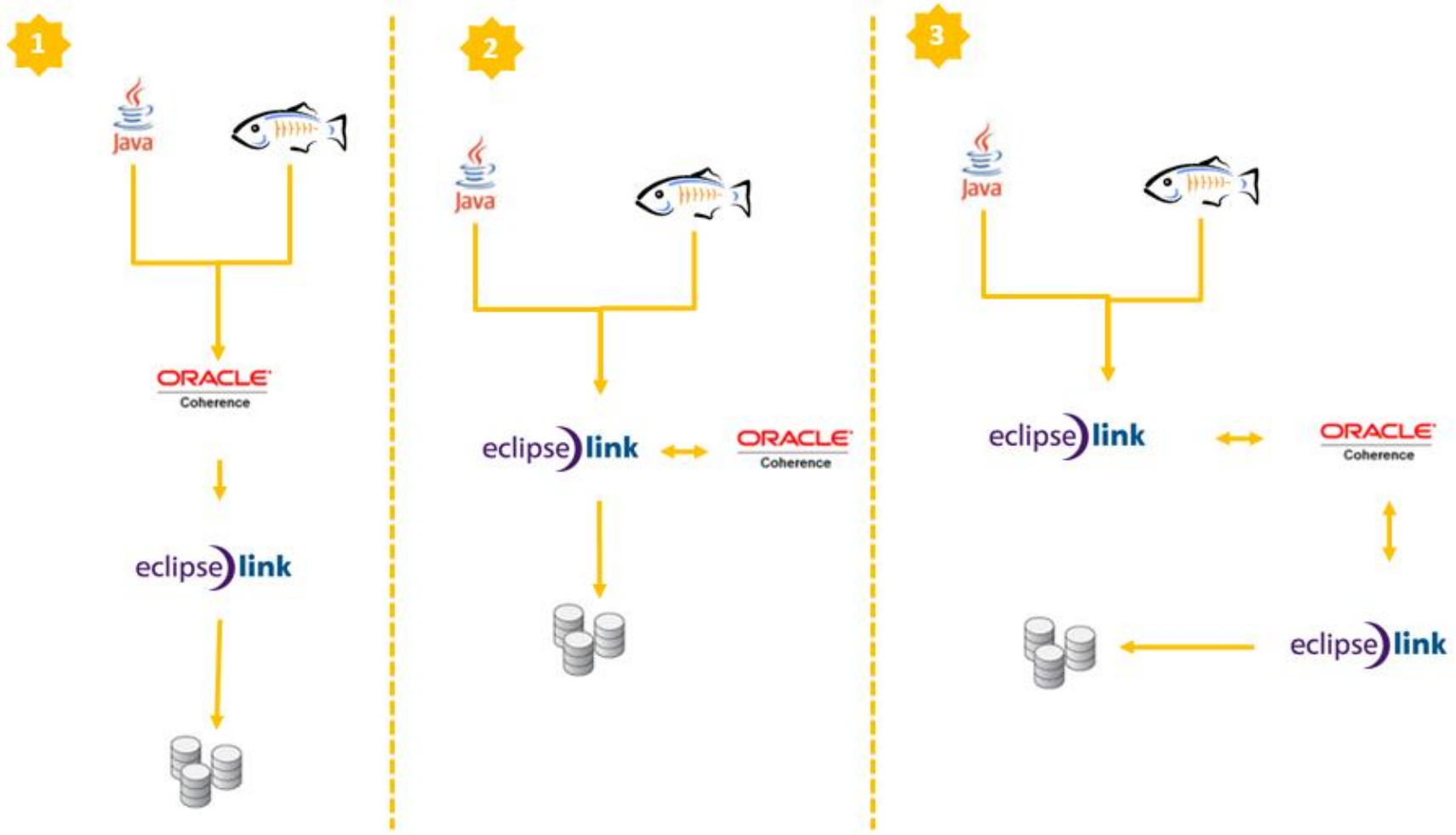
- TopLink Grid is a component of Oracle TopLink
- TopLink Grid allows Java developers to transparently leverage the power of the Coherence data grid
- TopLink Grid combines:
  - the simplicity of application development using the Java standard Java Persistence API (JPA) with
  - the scalability and distributed processing power of Oracle's Coherence Data Grid.
- Supports 'JPA on the Grid' Architecture
  - EclipseLink JPA applications using Coherence as a shared (L2) cache replacement along with configuration for more advanced usage
  - TopLink Grid integrates EclipseLink JPA and Coherence
  - Base configuration uses Coherence data grid as distributed shared cache
  - Updates to Coherence cache immediately available to all cluster nodes
  - Advanced configurations uses data grid to process queries to avoid database access and decrease database load

Why Coherence with JPA?

- Historical approach to scaling a JPA application
  - Adding nodes to a cluster
  - Tuning database performance to reduce query time
- Both of these approaches will support scalability but only to a point

- Historical approach to scaling EclipseLink JPA applications into a cluster:
  - **Disable Shared Cache**
    - Each transaction retrieves all required data from the database. Increased database load limits overall scalability but ensures all nodes have latest data.
    - Memory footprint of application increases as each transaction has a copy of each required Entity
    - Every transaction pays object construction cost for queried Entities.
    - **Database becomes bottleneck**
  - **Cache Coordination**
    - When Entity is modified in one node, other cluster nodes messaged to replicate/invalidate shared cached Entities.
    - Creation and/or modification of Entity results in message to all other nodes
    - Messaging latency means that nodes may have stale data for a short period.
    - Shared cache size limited by heap of each node
    - Objects shared across transactions to reduce memory footprint

Strategies for JPA on the grid



JPA Backed Caches – Traditional Approach

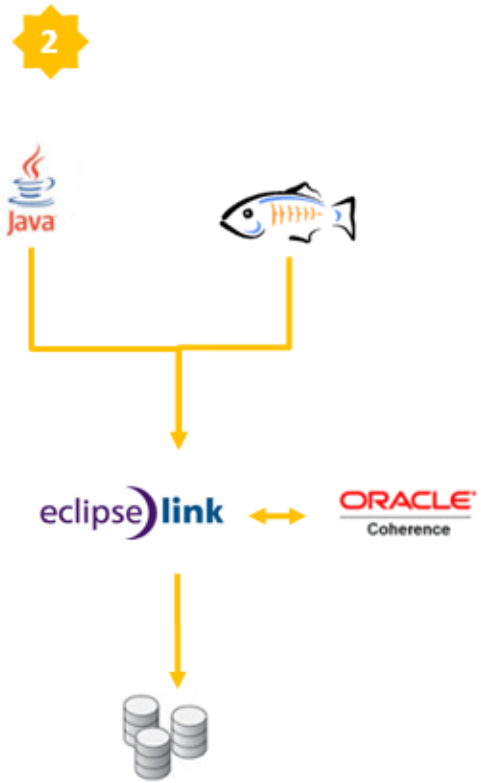
- Coherence API with caches backed by a database mapped through JPA.
- The grid accesses relational data through JPA CacheLoader and CacheStore implementations.
- TopLink Grid provides CacheLoader and CacheStore implementations that are optimized for EclipseLink JPA. (EclipseLinkJPACacheLoader and EclipseLinkJPACacheStore)
- Using the standard JPA run-time configuration file persistence.xml and the JPA mapping file orm.xml.
- The Coherence cache configuration file coherence-cache-config.xml must be specified to override the default Coherence settings and define the CacheStore caching scheme.

Using the Traditional Approach

```
// Get the configured Cache

NamedCache cache =
CacheFactory.getCache("Employee");

//Create a new Employee
Employee emp = new Employee();
emp.setFirstName("Markus");
emp.setLastName("Eisele");
emp.setId(1);

//Put the Employee into the cache
cache.put(1, emp);
```

JPA Second Level Cache – Grid Cache

- Ensures all nodes have coherent view of data.
  - Database is always right
  - Shared Cache is always right—Entities read, modified, or created are available to all cluster members.
- Communication is to primary and backup nodes.
- Coherence cache size is the sum of the available heap of all members—larger cache size enables longer tenure and better cache hit rate
- Can be used with existing applications and all EclipseLink performance features without altering application results



Using the 2nd Level Cache

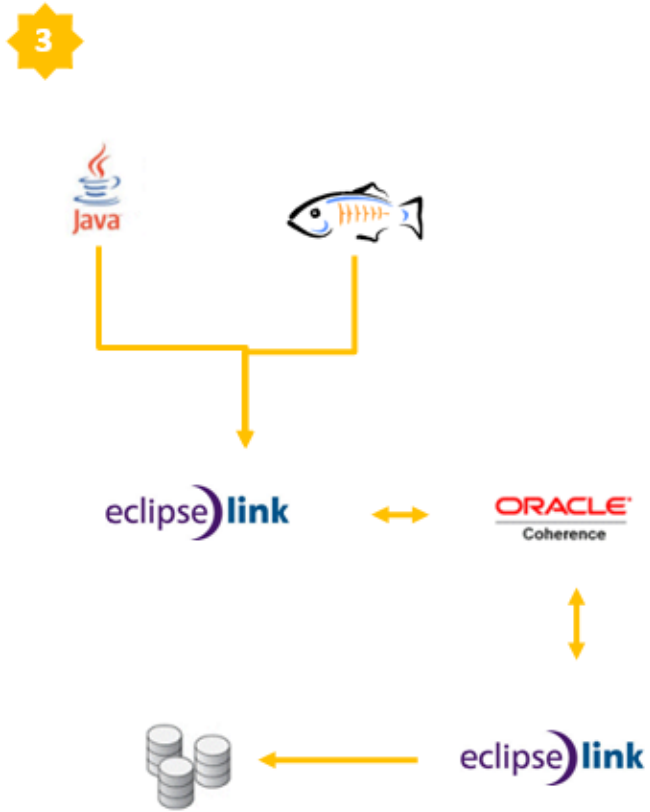
```
...
import
oracle.eclipselink.coherence.integrated.cache.Coherenc
eInterceptor;
import
org.eclipse.persistence.annotations.CacheInterceptor;
...

@Entity
@CacheInterceptor(value = CoherenceInterceptor.class)
public class Employee implements Serializable {
...
}
```

Configuring Optimizations

- Grid Read (`@Customizer(CoherenceReadCustomizer.class))`)
  - In the Grid Cache configuration, all reads (both pk and non-pk) are executed against the grid (by default).
  - For Entities that typically:
    - Need to be highly available
    - Must have updates written *synchronously* to the database; database is system of record
- Grid Entity (`@Customizer(CoherenceReadWriteCustomizer.class))`)
  - The Grid Entity configuration is the same as the Grid Read configuration except that all writes are executed against the grid, not the database.
  - For Entities that typically:
    - May have updates written asynchronously to the database (if CacheStore configured)

JPA 2nd Level Cache with JPA Backed Cache

- Combining both approaches is possible with some combinations
- Grid Entity
  - Can be optionally used with CacheStore to update the database.
- Grid Read
  - Can be optionally used with CacheLoader.



Relationship Support

- Coherence does not provide support for the serialization of complex graphs across caches.
  - Coherence serializes objects/object graphs and places the results in to a single cache under a key.
- TopLink Grid 11gR1 does support storage of complex graphs of Entities with each Entity type stored in a corresponding Coherence cache.
  - Relationship information is stored into Coherence
  - Relationships are reconstituted upon retrieval from Coherence
  - Lazy and eager relationships are supported
  - Relationships between Coherence cached and database persisted objects is supported.
- TopLink Grid wraps Entities with relationships with a byte code generated Wrapper class when put()ing into Coherence
  - Wrapper encodes relationship details
  - Wrapper is stripped off when Entity retrieved from Coherence and relationships are reconstituted
  - Eager relationships will result is retrieval (from either Coherence or database depending on configuration) of target Entity/Entities
- Non-TopLink Grid applications can query Entities from Coherence with Filters and get() wrapped Entities
  - TopLink Grid serializer must be configured on Cache
  - Coherence clients can configure auto-unwrapping
  - Relationships will be null when retrieved by Coherence clients

Summary

- TopLink supports a range of strategies for scaling JPA applications
- TopLink Grid integrates EclipseLink JPA with Oracle Coherence to provide:
  - 'JPA on the Grid' functionality to support scaling JPA applications with Coherence
  - Support for caching Entities with relationships in Coherence
- Both TopLink and Coherence are a part of WebLogic Application Grid

ORACLE FUSION MIDDLEWARE WEBLOGIC

ORACLE TOPLINK

ORACLE Coherence

Links and Readings

- http://blog.eisele.net
- http://www.oracle.com/technetwork/middleware/coherence/overview/index.html
- http://www.oracle.com/technetwork/middleware/toplink/overview/index.html
- http://www.oracle.com/technetwork/middleware/toplink/tl-grid-097210.html
- http://www.eclipselink.org/

Vielen Dank für Ihre Aufmerksamkeit

Markus Eisele

markus.eisele@msg-systems.com

www.msg-systems.com

.consulting .solutions .partnership