

Oracle Coherence 3.6: Share and Manage Data In Clusters

Student Guide - Volume II

D66791GC11

Edition 1.1

October 2012

D79399

ORACLE®

Authors

Mark Lindros
Al Saganich

**Technical Contributors and
Reviewers**

Brian Oliver
Patrick Peralta
Noah Arliss
Christer Fahlgren
David Guy
Robert Lee
Thomas Beerbower
John Speidel
David Leibs
Patrick Fry
Jason Howes
Michele Diserio
Peter Utzschneider
Craig Blitz
Cameron Purdy
Alex Gleyzer
Serge Moiseev
Rob Misek
Randy Stafford
David Felcey
Craig Blitz
Tom Pflaeffle
Madhav Sathe
Rao Bhethanabotla

Editors

Steve Friedburg
Vijayalakshmi Narasimhan

Graphic Designer

Maheshwari Krishnamurthy

Publishers

Sumesh Koshi
Veena Narasimhan

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

I Course Introduction

- Audience I-2
- Class Introductions I-3
- Goal I-4
- Course Objectives I-5
- Prerequisites I-8
- Course Environment I-9
- Course Conventions I-10
- Course Schedule I-12
- How Can I Learn More? I-14

1 Introduction To Coherence

- Objectives 1-2
- Agenda 1-3
- System Performance Degradation 1-4
- When Performance Problems Aren't Performance Problems 1-6
- When Performance Problems Are Capacity Problems 1-8
- Quantifying Capacity 1-9
- Scalability 1-12
- Scaling Example 1-14
- Origins of Performance Problems 1-15
- Definition: Cache 1-16
- Scaling Example Revisited 1-18
- Caching to Scale 1-19
- Notes on Caching 1-20
- Scalability Strategy Comparison 1-22
- Scaling Example Revisited: Caching and Scaling Horizontally 1-24
- Agenda 1-25
- Welcome to Coherence 1-26
- What is Coherence? 1-27
- Coherence Principles 1-29
- Coherence Features 1-30
- What is a Coherence Cluster Node? 1-31
- What is a Coherence Cluster? 1-32
- Coherence Data Grids and Fault Tolerance 1-33

Coherence Cache Topology Examples 1-34
Event and Parallel Processing 1-36
Queries and Filters 1-38
Coherence Transactions 1-39
Coherence Security 1-41
Coherence Management 1-42
Coherence*Extend 1-43
Coherence*Web 1-44
Coherence Data Grid Solution Set 1-46
Standard Edition 1-48
Coherence Enterprise Edition 1-49
Coherence Grid Edition 1-50
Oracle Fusion Middleware 1-52
Summary 1-53

2 Getting Started with Oracle Coherence

Objectives 2-2
Agenda 2-3
Installing Oracle Coherence Server 2-4
Coherence Directory Structure 2-5
Coherence Script Basics 2-6
Starting Coherence 2-8
Understanding Startup 2-9
Coherence Configuration 2-10
Default Configuration 2-12
Command Line Properties 2-13
Coherence Port Use 2-14
Coherence Services 2-15
Service Types 2-16
Coherence Start Sequence 2-17
Example tangosol-coherence-override.xml 2-18
Quiz 2-19
Agenda 2-20
Coherence Programming Model 2-21
Named caches 2-22
Configuring Caches 2-23
Minimal Cache Example 2-24
The Coherence Console 2-25
Console Commands 2-26
Quiz 2-27

Practice.02.01 Overview: Installing, Configuring and Starting a Coherence Cluster	2-28
Practice.02.02 Overview: Using the Coherence Console	2-29
Agenda	2-30
Coherence Application Basics	2-31
Coherence Application Overview	2-32
CacheFactory Useful Methods	2-33
NamedCache Interface Useful Methods	2-34
NamedCache extends various interfaces	2-35
Data Loading Efficiencies	2-36
Data Loading and putAll()	2-37
Reading entries in a cache	2-38
Local or Private Clusters	2-39
Local Storage	2-41
Coherence and TCMP	2-42
Running External Applications in Eclipse	2-44
Launching Cache Servers from Eclipse	2-45
Arguments for running DefaultCacheServer	2-46
Eclipse Environment Variables	2-47
Eclipse Libraries	2-48
Defining Libraries	2-49
Eclipse Run Configurations	2-51
Run Configurations and Coherence	2-52
Summary	2-53
Practice.02.03 Overview: Coherence “Hello World”	2-54

3 Working with Objects

Objectives	3-2
Agenda	3-3
Java Objects and Coherence	3-4
AirPort Object Example	3-5
Objects and Identity	3-7
Identity Types	3-8
Implementing Entities	3-9
Identity generation	3-10
Using Sequence Generators	3-11
Sequence Generators and Entities	3-12
Quiz	3-13
Aggregate Objects	3-14
Value Objects	3-15
Data Affinity	3-16

Properties of Relationships	3-17
Modeling Relationships in Java	3-18
Cardinality Examples	3-19
Modeling Relationships in Coherence	3-20
Solving the getXXX problem	3-21
Customer with Repository Example	3-22
Getting Objects via Repository	3-23
Example AbstractRepository	3-24
Example CreditCardRepository	3-26
Example CoherenceCreditCardRepository	3-27
Quiz	3-28
Practice.03.01 Overview: Developing with Complex Objects	3-29
Agenda	3-30
Serialization Concepts	3-31
Serialization and Performance	3-32
Serialization Options with Coherence	3-33
Serialization Comparison	3-34
Implementing Java Serialization	3-35
Implementing ExternalizableLite	3-36
ExternalizableLite Example	3-37
Portable Object Concepts	3-38
PortableObject Requirements	3-39
PortableObject Example	3-40
POF Indices Requirements	3-41
Registering Portable Objects	3-42
Selected PofWriter Methods	3-43
Selected PofReader Methods	3-44
Quiz	3-45
External Serialization	3-46
PofSerializer Basics	3-47
Registering POF Serializers	3-48
Evolvable Objects	3-49
Implementing Evolvable	3-50
Evolvable and POF	3-51
Serialized versus Unserialized Caches	3-52
Serialization Testing Support	3-53
Quiz	3-54
Serialization Testing Example	3-55
Practice.03.02 Overview: Serialization using ExternalizableLite	3-56
Practice.03.03 Overview: Serialization using Portable Object Format	3-57
Summary	3-58

4 Configuring Coherence Caches

Objectives 4-2

Agenda 4-3

What Happens to Cached Data? 4-4

Local and Replicated Cache Semantics 4-5

Partitioned and Near Cache Semantics 4-8

Partitioned Cache Semantics 4-10

Near Cache Semantics 4-11

Java Object Access Semantics 4-12

Pass by Reference and Maps 4-13

Pass by Copy and Maps 4-14

Agenda 4-15

Revisiting coherence-cache-config.xml 4-16

Anatomy of a Cache Configuration 4-17

What is a Scheme? 4-18

Scheme Composition 4-19

Declaring Cache Mappings 4-20

Using System Properties to Override XML Elements 4-21

Using Macro Parameters to Override XML Elements 4-22

Backing Maps 4-23

Agenda 4-25

What is a Local Cache? 4-26

When To Use a Local Cache 4-27

When To Not Use a Local Cache 4-29

Local Cache get Diagram 4-30

Local Cache put Diagram 4-31

Configuring a Local Cache 4-32

Defining Local Cache Parameters 4-33

Quiz 4-35

Practice.04.01 Overview: Configuring a Local Cache Scheme 4-38

Agenda 4-39

What is a Replicated Cache? 4-40

When To Use a Replicated Cache 4-41

When To Not Use a Replicated Cache 4-43

Visualizing Data in a Replicated Cache 4-44

Replicated Cache get Diagram 4-45

Replicated Cache put Diagram 4-46

Configuring a Replicated Cache 4-47

Defining Replicated Cache Parameters 4-48

Quiz 4-49

Practice.04.02 Overview: Configuring a Replicated Cache Scheme 4-53

Agenda	4-54
What is a Partitioned Cache?	4-55
When To Use a Partitioned Cache?	4-57
When To Not Use a Partitioned Cache?	4-58
Visualizing Data in a Partitioned Cache	4-59
Partitioned Cache get Diagram	4-60
Partitioned Cache put Diagram	4-62
Partitioned Cache Fault Tolerance	4-64
Storage Disabled Partitioned Cache	4-66
Configuring a Partitioned Cache	4-68
Defining Partitioned Cache Parameters	4-69
Quiz	4-70
Practice.04.03 Overview: Configuring a Partitioned Cache Scheme	4-75
Agenda	4-76
What is a Near Cache?	4-77
When To Use a Near Cache?	4-78
When To Not Use a Near Cache?	4-79
Visualizing Data in a Near Cache	4-80
Near Cache get Diagram	4-81
Near Cache Concurrency Options	4-82
Near Cache put Diagram	4-83
Near Cache put Diagram with Local Storage Disabled	4-84
Near Cache Invalidation Options	4-85
Configuring a Near Cache	4-86
Defining Near Cache Parameters	4-87
Quiz	4-88
Practice.04.04 Overview: Configuring a Near Cache Scheme	4-92
Agenda	4-93
What is an Overflow Cache?	4-94
When To Use an Overflow Cache?	4-95
When To Not Use an Overflow Cache?	4-96
Configuring an Overflow Cache	4-97
Defining Overflow Cache Parameters	4-98
Agenda	4-99
Other Cache Types	4-100
Choosing the Right Cache	4-101
Advanced Configurations	4-102
Summary	4-103

5 Observing Data Grid Events

- Objectives 5-2
- Agenda 5-3
- Event Concepts 5-4
- Simple Object Lifecycle and Events 5-5
- Application versus Server Side Events 5-6
- Listener Behavior 5-7
- Listeners versus Triggers 5-9
- Agenda 5-11
- Pre-events and MapTriggers 5-12
- Post-events and MapListeners 5-13
- Contract for writing a MapListener 5-14
- Selected MapEvent Methods 5-15
- MapListener Event Contents 5-16
- Synthetic Events 5-17
- Determining if an Event is Synthetic 5-19
- Behind the scenes: ObservableMap 5-20
- Observable Map Workings 5-22
- Quiz 5-23
- Registering MapListeners 5-24
- Map Listeners and Exceptions 5-25
- MapListener Registration Variations 5-26
- MapListener Lite 5-27
- MapListener Implementations 5-28
- Anonymous AbstractMapListener Example 5-29
- Making Maps Observable 5-30
- MapListeners and Filters 5-31
- Provided Filters 5-32
- Transforming Events 5-33
- Transformer Example 5-34
- Registering a Transform Programmatically 5-35
- Quiz 5-36
- Practice.05.01 Overview: Working with Map Listeners 5-37
- Agenda 5-38
- Map Triggers - Review 5-39
- MapTrigger Basics 5-40
- Registering MapTriggers Programmatically 5-41
- MapTrigger Example 5-42
- Registering a MapTrigger Declaratively 5-43
- MapTriggers and Exceptions 5-44
- Quiz 5-45

Practice.05.02 Overview: Working with Map Triggers	5-46
Agenda	5-47
Backing Map Listeners and Events	5-48
Backing Map Listener Basics	5-49
Backing Map Listener Event Conversion	5-50
Selected BackingMapManagerContext Methods	5-51
Keys and Backing Map Ownership	5-52
Registering Backing Map Listeners	5-53
Backing Map Listeners and Exceptions	5-55
Backing Map Listener Best Practices	5-56
Quiz	5-57
Practice.05.03 Overview: Working with Backing Map Listeners	5-58
Agenda	5-59
Continuous Query Cache	5-60
Continuous Query Cache Basics	5-62
Summary	5-63

6 Querying and Aggregating Data in the Cache

Objectives	6-2
Agenda	6-3
Filters and Caches	6-4
Filter Execution	6-6
Developing Filters	6-7
Accessing Object Properties	6-8
Out-of-the-box Filters	6-9
Example: Filtering with EqualsFilter	6-10
Sorting	6-11
Example: Sorting with a Comparator	6-12
Paging and LimitFilters	6-13
ValueExtractors	6-14
Provided ValueExtractors	6-15
Value Extractors and Dot notation	6-16
Practice.06.01 Overview: Filtering and Sorting Data	6-17
Aggregating Results	6-18
Understanding Aggregation Execution	6-19
Developing Aggregators	6-20
Main versus Parallel Execution	6-21
Working with isFinal	6-22
Example Aggregator	6-23
Using Aggregators	6-25
Out-Of-the-Box Aggregators	6-26

Practice.06.02 Overview: Developing and using Aggregators 6-27
Filters and Indexes 6-28
Index: Example with Ordering 6-30
Index: Example without Ordering 6-32
Quiz 6-33
Data Affinity 6-35
Specifying Data Affinity 6-36
Implementing KeyAssociations 6-37
Booking.Id Example 6-38
Implementing a KeyAssociator 6-39
Registering KeyAssociators 6-40
Quiz 6-41
Agenda 6-43
What is CohQL? 6-44
CohQL Statements 6-45
Working with Queries 6-46
General SELECT Syntax 6-47
Path Expressions 6-48
Filtering with WHERE 6-49
QueryFilter Example 6-51
key() and value() Pseudo functions 6-52
Aggregating in SELECT 6-53
Parameters 6-54
Quiz 6-55
Managing Cache Lifecycle 6-56
UPDATE Syntax 6-57
INSERT Syntax 6-58
DELETE Syntax 6-59
Index Management 6-60
Starting QueryPlus 6-61
QueryPlus Arguments 6-62
Summary 6-63

7 Performing In-place Processing of Data with Entry Processors

Objectives 7-2
Agenda 7-3
Managing Data Consistency and Transactions: Databases 7-4
Managing Data Consistency and Transactions: Coherence 7-5
ConcurrentMap 7-7
ConcurrentMap: Example 7-9
Issues with Locking 7-10

Updating Data in Coherence	7-12
What is an EntryProcessor?	7-13
Entry Processor Execution Diagram	7-15
Out-of-the-Box EntryProcessors	7-17
Developing EntryProcessors	7-19
InvocableMap Interface	7-20
InvocableMap.EntryProcessor Interface	7-22
InvocableMap.Entry Interface	7-23
EntryProcessor Semantics	7-25
EntryProcessor Behavior	7-26
Contract for Writing EntryProcessors	7-28
Example: Creating a Custom EntryProcessor	7-29
Quiz	7-31
Practice.07.01 Overview: In-Place Processing of Data	7-34
Agenda	7-35
What is an Invocation Service?	7-36
Configuring an Invocation Service	7-37
Implementing the Invocable Interface	7-38
Implementing the AbstractInvocable Interface	7-39
Example: AbstractInvocable Implementation	7-40
Examining the InvocationService Interface	7-41
Executing an Invocation Agent Synchronously	7-42
Executing an Invocation Agent Asynchronously	7-43
Quiz	7-45
Practice.07.02 Overview: Implementing Invocable Agents	7-48
Summary	7-49

8 Transactions and Coherence

Objectives	8-2
Agenda	8-3
What is a Transaction?	8-4
Transaction Benefits	8-5
ACID Properties	8-6
Agenda	8-8
Coherence Transactions Overview	8-9
Configuring Transactional Caches	8-10
NamedCache API and Transactions	8-11
Connection API	8-12
Using the Connection API	8-13
Multiple Caches and Transactions	8-14
OptimisticNamedCache API	8-15

Example: OptimisticNamedCache	8-16
Using the Coherence Resource Adapter	8-17
Coherence Resource Adapter and User-Managed Transactions	8-18
Multiple Resource Adapters	8-19
Transaction Isolation	8-20
Transaction Phenomena	8-21
Coherence Transaction Isolation Levels	8-22
Quiz	8-24
Practice.08.01 Overview: Configuring, Running, and Reviewing Transactional Clients	8-28
Agenda	8-29
What are Distributed Transactions?	8-30
Two-Phase Commit (2PC) Protocol	8-32
Open Group XA (Extended Architecture) Interface	8-33
Transactions and Resource Managers	8-34
A Successfully Committed 2PC	8-35
A Successfully Aborted 2PC	8-36
Summary	8-37

9 Integrating a Data Source with Coherence

Objectives	9-2
Agenda	9-3
Persisting Data to Storage	9-4
O/RM Integration	9-6
Data Integration Patterns	9-8
Cache-Aside Pattern	9-10
Read-Through Pattern	9-11
Read-Through: Example	9-12
Write-Through Pattern	9-13
Write-Through: Example	9-14
Write-Behind Pattern	9-15
Write-Behind: Example	9-16
Refresh-Ahead Pattern	9-17
Agenda	9-18
The CacheLoader Interface	9-19
CacheLoader and CacheStore Lifecycle	9-20
CacheLoaders and Initialization	9-21
CacheLoader: Example	9-22
Configuring CacheLoaders	9-23
The CacheStore Interface	9-24
CacheStore: Store Example	9-25

CacheStore: Load Example	9-26
CacheStore: *All Example	9-27
CacheStore Architecture	9-28
Refresh-Ahead Caching	9-29
Refresh-Ahead Configuration	9-31
Write-Through	9-32
Write-Behind	9-33
Write-Behind Configuration	9-36
What Happens If a Write-Behind Transaction Fails?	9-39
Write-Behind Caveats	9-40
Idempotency	9-41
Write-Behind Factor Can Be Set Dynamically	9-43
Practice.09.01 Overview: Integration using a CacheStore	9-44
Agenda	9-45
The Java Persistence Architecture	9-46
JPA and Persistence Requirements	9-47
The JPA Approach	9-48
What Are JPA Entities?	9-50
Entity Class Requirements	9-52
JPA and CacheStores	9-53
Integrating JPA and Coherence	9-54
Obtaining a JPA Implementation	9-55
Mapping Entities	9-56
Configuring JPA	9-57
Coherence JPA Configuration	9-58
Cachestore-scheme	9-59
Cache Mapping	9-60
Summary	9-61

10 Understanding Typical Caching Architectures

Objectives	10-2
Agenda	10-3
Single Application Instances	10-4
Multiple Application Instances	10-5
Local Caching Pattern	10-7
Distributed Coherent Caching Pattern	10-8
Introducing the Caching Infrastructure Layer	10-9
Cache-Aside Patterns	10-10
Read-Through Pattern	10-11
Write-Aside Pattern	10-12
Write-Through Pattern	10-13

Write-Behind Pattern	10-14
External Update Anti-Pattern	10-15
External Update Pattern Using Messaging	10-16
External Update with Direct Access	10-17
Near Caching Pattern	10-18
Client-Side Event Processing Pattern	10-19
Server-Side Event Processing Pattern	10-20
Server-Side Processing Pattern: Queries, Map Reduction, Computation...	10-21
Combined = Scalable Platform!	10-22
Quiz	10-23
Summary	10-27
Practice.10.01 Overview: Examining Sample Topologies	10-28

11 Coherence*Extend

Objectives	11-2
Agenda	11-3
What Is Coherence*Extend?	11-4
Coherence*Extend Architecture	11-5
Coherence*Extend Capabilities	11-7
Coherence*Extend Advantages	11-8
Coherence*Extend Disadvantages	11-9
When to Use Coherence*Extend?	11-10
Coherence*Extend Clients	11-11
Data Replication	11-12
Agenda	11-13
Configuring Coherence*Extend	11-14
Client-side Cache Configuration Descriptor	11-15
Cluster-side Cache Configuration Descriptor	11-16
Launching a Coherence*Extend-Enabled DefaultCacheServer Process and Java Client Application	11-18
Quiz	11-20
Agenda	11-21
What is Coherence*Web?	11-22
Coherence*Web Architecture	11-24
Configuring and Running Coherence*Web	11-26
Quiz	11-27
Practice.11.01 Overview: Configuring and running Coherence*Extend	11-28
Practice.11.02 Overview: Writing a Coherence*Extend Java client	11-29
Practice.11.03 Overview: Writing a Coherence*Extend C++ Client	11-30
Summary	11-31

12 Coherence Administration

Objectives	12-2
Agenda	12-3
Overview	12-4
Management Architecture	12-5
Configuring Coherence JMX	12-6
Accessing the Coherence MBean using HTTP and JMX RI	12-7
Accessing the Coherence MBean using Java bundled JMX Implementation	12-8
Accessing the Coherence MBean using the Coherence MBeanConnector	12-9
System MBeans to Watch	12-10
Quiz	12-12
Agenda	12-13
What is the Reporter?	12-14
Configuring Basic Settings	12-15
Managing Reporter MBean Attributes	12-16
Managing Reporter MBean Operations	12-17
Finding Reporter Log Data	12-18
Viewing Reporter Data	12-19
Creating Custom Reports	12-21
Running Reporter in a Distributed Environment	12-23
Quiz	12-24
Practice.12.01 Overview: Configuring and Running the Reporter	12-25
Agenda	12-26
Understanding Coherence Administrator Needs	12-27
Complete Coherence Management: Overview OEM 11gR1	12-28
Oracle Coherence Support	12-30
Proactive Monitoring Using Alerts and Notification	12-31
Coherence Monitoring and Dashboard Complete cluster visibility	12-32
Monitoring Cache Quick resolution of cache performance issues	12-34
Monitoring Nodes	12-36
Diagnosing JVM Issues	12-37
Complete Coherence Grid Management Drastically improve productivity and reliability	12-38
Provisioning Cut cost and reduce risk by automation	12-39
Configuration Management Reduces time to diagnose and repair issues	12-40
Monitor Complete Infrastructure as a Single System Single console drastically reduces total cost of ownership	12-41
Integrated Fusion Middleware Management	12-42
End-to-End Management Manage performance and change across all tiers	12-43
Third-Party Management Tools	12-44
Quiz	12-45

Agenda	12-46
Production versus Development Modes	12-47
Hardware Recommendations	12-49
JVM Recommendations	12-52
JVM Deployment Concerns	12-53
JVM Heap Sizing	12-55
Tuning Garbage Collection	12-57
Sizing a Coherence System	12-59
General Sizing Guidelines	12-60
Size-Limiting Storage JVMs	12-62
Binary Calculator: Example	12-63
Network Configuration	12-64
Operating System Tuning	12-68
Setting Buffer Sizes	12-69
Testing Multicast	12-71
Datagram Test	12-74
Service Configuration	12-77
Setting Thread Count	12-78
Partition Count	12-79
Tuning the OS for Coherence*Extend	12-80
Tuning the Coherence*Extend Client Side	12-81
Tuning the Coherence*Extend Cluster Side	12-82
Cluster Quorum	12-83
Partitioned Cache Quorum	12-84
Extend Proxy Quorum	12-86
Coherence Logs	12-87
Performance Testing: General Advice	12-88
Quiz	12-90
Summary	12-92
Practice.12.01 Overview: Configuring and Running the Reporter	12-93

13 Understanding Coherence Security

Objectives	13-2
Agenda	13-3
Cluster Connectivity (TCMP)	13-4
Member Identity	13-5
Authorized Hosts	13-6
TCMP Access Control	13-7
ClusterPermission	13-9
Extend Pluggable Identity: Architecture	13-10
Extend Pluggable Identity: Client-Side	13-11

Extend Pluggable Identity: Client-Side Code Example	13-12
Extend Pluggable Identity: Identity Transformer	13-13
Extend Pluggable Identity: Identity Transformer Code	13-14
Extend Pluggable Identity: Proxy-Side Identity Asserter	13-15
Extend Pluggable Identity: Identity Asserter Code	13-16
Extend Pluggable Identity: Identity Asserter Semantics	13-17
Extend Pluggable Identity: Identity Asserter Subject Scoping	13-18
Extend Access Control: Authorization Wrappers	13-19
Extend Access Control: Configuring Authorization Wrappers	13-20
Extend Access Control: EntitledCacheService	13-21
Extend Access Control: EntitledNamedCache	13-22
Extend Access Control: EntitledInvocationService	13-23
Extend Access Control: Authorization Example Implementation	13-24
Transport Layer Security: SSL	13-25
Transport Layer Security: SSL Recommendations	13-26
Transport Layer Security: Setting up SSL for the Cluster	13-27
Transport Layer Security: Setting up SSL for *Extend	13-28
Security Examples	13-29
Quiz	13-30
Summary	13-32

A UML Concepts Appendix

Objectives	A-2
What Is UML?	A-3
UML Diagrams	A-4
UML Components	A-5
Sequence Diagrams	A-6
Anatomy of a Sequence Diagram	A-7
Nested Sequence Diagrams	A-8
Structure Diagrams	A-9
Structure Diagrams and Inheritance	A-10
Structure Diagrams and Association	A-11
Crow's Foot Notation	A-12
Aggregation and Composition	A-13
Component Diagrams	A-14
Component Diagram Anatomy	A-15
Summary	A-16

B Coherence 3.7 New Features Overview

Objectives	B-2
Themes	B-3

New Features	B-4
Ease of Use New Features	B-5
Automatic Proxy Discovery	B-6
Dynamic Load Balancing for Coherence*Extend	B-7
XML Schemas for Validating Configuration	B-8
Portable Object Format (POF) Improvements	B-11
Integration New Features	B-12
Coherence*Web Support for GlassFish	B-13
F5 Load Balancing Integration	B-14
Representation State Transfer (REST)	B-15
RASP New Features	B-17
Query Optimization: Explain Plans	B-18
Query Optimization: Monitoring	B-20
Partition-Level Transactions	B-22
Innovation New Features	B-24
Elastic Data	B-25
Summary	B-28

C Coherence*Extend and Load Balancing

Objectives	C-2
Dynamic Load Balancing for Coherence*Extend	C-3
Connection Load Balancing	C-4
Proxy-Based Load Balancing	C-5
Client-Based Load Balancing: Per Proxy	C-6
Client-Based Load Balancing: Systemwide	C-7
Proxy-Based Load Balancing Algorithm	C-8
Custom Load Balancing	C-10
ProxyServiceLoadBalancer Interface	C-11
AbstractProxyServerLoadBalancer Class	C-12
ProxyServiceLoad Interface	C-13
DefaultProxyServiceLoadBalancer	C-14
Summary	C-17

D Representational State Transfer

Objectives	D-2
What Is REST?	D-3
Coherence REST	D-4
Coherence REST Requirements	D-5
Supported Representations	D-6
Annotations	D-7
Class Requirements (JAXB)	D-8

XmlAccessType.PUBLIC_MEMBER: Example D-9
XmlAccessType.PROPERTY: Example D-10
XmlAccessType.FIELD: Example D-11
XmlAttribute and XmlTransient D-12
Class Requirements (JSON) D-13
JSON Mapping: Example D-14
JSonProperty and JSonIgnore D-15
Key Converters D-16
KeyConverter: Example D-17
REST Configuration D-18
Registering Objects D-19
REST resources D-20
Serving Coherence REST Requests D-21
Configuring REST Proxies D-22
Configuration: Example D-23
Starting a REST Proxy D-24
REST Configuration Best Practices D-25
Custom REST Server D-26
Contract for AbstractHttpServer D-27
Start Method D-28
Stop Method D-29
Registering a Custom REST Server D-30
WebLogic Server Deployment D-31
REST Web Application Structure D-32
web.xml Deployment Descriptor D-33
weblogic.xml Deployment Descriptor D-34
Deploying the Web Application D-35
Single Object Operations D-36
Multiple Object Operations D-37
Partial Results D-38
Queries D-39
Summary D-40

E Partition-Level Transactions

Objectives E-2
Partition-Level Transactions E-3
Atomic Partition-Level Transactions E-4
Partitioned Cache: Review E-5
Entry Processor: Review E-6
Concurrent Processing and Entry Processors E-7
Partition-Level Operations Pre 3.7: Example E-8

Atomic-Level Operation Support	E-9
Partition-Level Operations 3.7 and Later: Example	E-10
Accessing Other Keys in the Same Cache	E-11
Summary	E-12

F Elastic Data

Objectives	F-2
What Is Elastic Data?	F-3
Benefits	F-4
Elastic Data and Journaling	F-5
Journaling and Garbage Collection	F-6
Configuring a Journaling Backing Map	F-7
Controlling Journaling Behavior	F-8
Considerations for Elastic Data	F-11
Summary	F-12

G Portable Object Format Annotations

Objectives	G-2
Annotations	G-3
Portable Object Format: Annotations	G-4
Specifying Portable Object Annotations	G-5
POF Annotations: Example	G-6
POF Automatic Indexing	G-7
POF Serializer Configuration	G-8
Summary	G-9

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and ORACLE CORPORATION use only

Performing In-place Processing of Data with Entry Processors



ORACLE®

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Manage concurrent access to data
- Implement Entry Processors to process data “in-place” where it is stored in the cache
- Configuring and implementing invocable agents



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Scalability and Parallel Processing

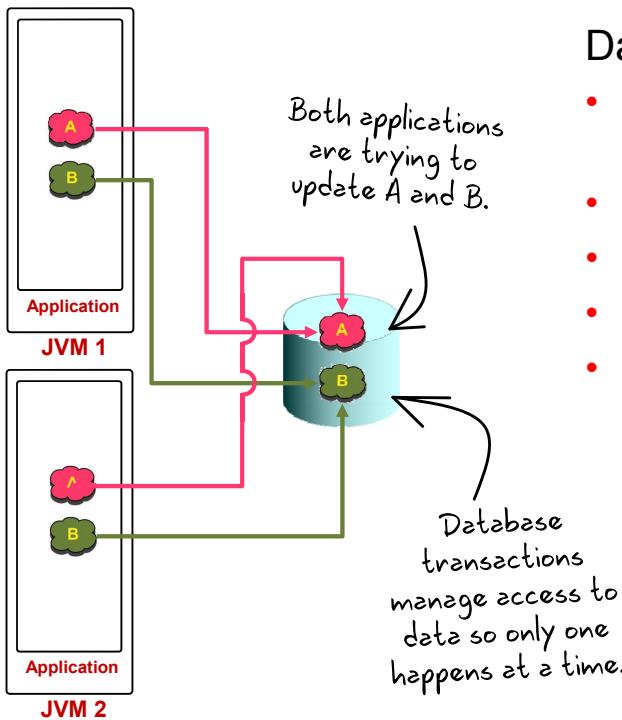
- Managing Concurrent Access to Data
- Entry Processor Execution
- Out-of-the-Box Entry Processors
- Developing Entry Processors

Invocation Service



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Managing Data Consistency and Transactions: Databases



Database Data Consistency:

- Transactions: Unit of work that wholly succeeds or fails
- Maintains ACID properties
- Ensures data consistency
- Persisted to disk
- Can be coordinated with other data sources using XA with a resource manager

ORACLE

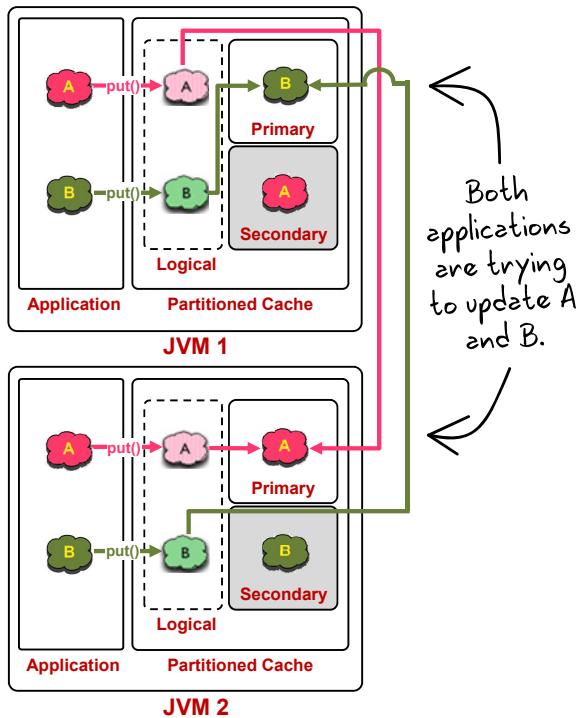
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Managing Data Consistency and Transactions: Databases

A database transaction is a unit of work that is performed against a database management system or a similar system that is treated in a coherent and reliable way independent of other transactions. A database transaction must be Atomic, Consistent, Isolated, and Durable (ACID). Transactions provide an "all-or-nothing" proposition, stating that work units performed in a database must be completed in their entirety, or take no effect whatsoever. Further, transactions must be isolated from other transactions, results must conform to existing constraints in the database, and transactions that complete successfully must be committed to durable storage.

Databases, and other data stores in which the integrity of data is important, often have the ability to handle transactions to ensure that the integrity of data is maintained. A single transaction is composed of one or more independent units of work, each reading and/or writing information to a database or other data store. In such a situation, it is important to ensure that the database or data store is in a consistent state.

Managing Data Consistency and Transactions: Coherence



Coherence Data Consistency:

- `ConcurrentMap`
- `TransactionScheme` and `Transaction Framework API`
- JCA Adapter for joining managed XA transactions
- `EntryProcessor`

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Managing Data Consistency and Transactions: Coherence

Just as it is important to ensure that a database is in a consistent state, it is also very important to ensure that data managed by Coherence is in a consistent state. Because there are multiple clients that can potentially have different copies of the cached data, it is important that the value of the data from the read until it is updated is accurate, and that some other process that has a copy of the data does not update it in the middle.

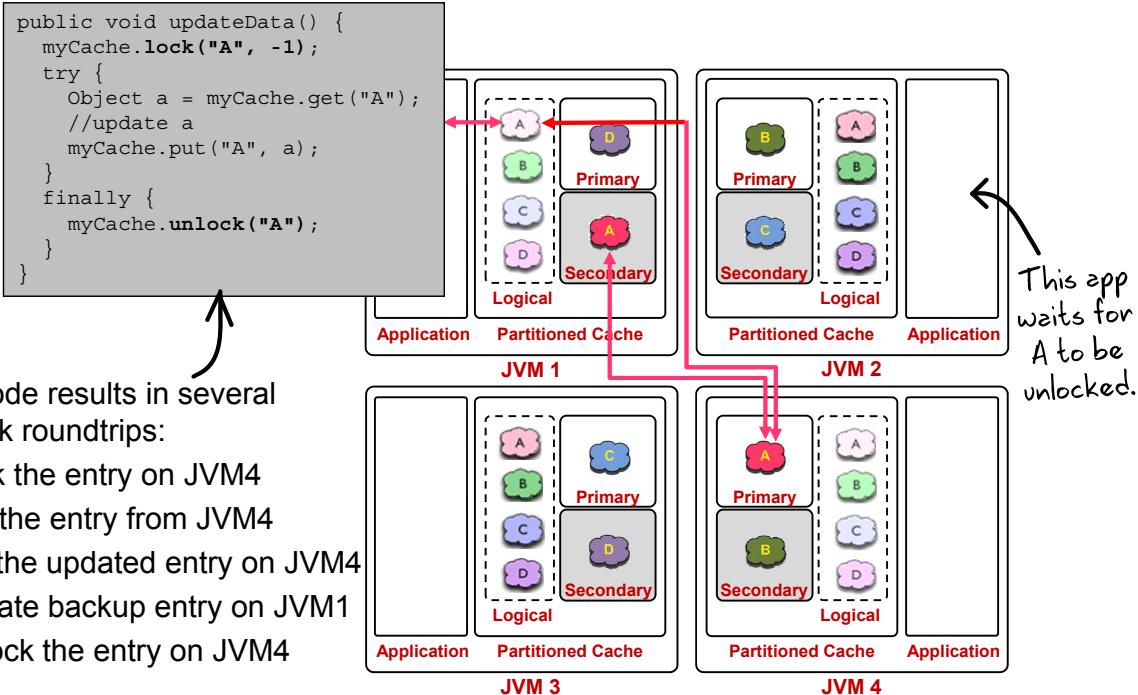
Coherence provides several options for managing concurrent access to data. This includes:

- **Explicit locking:** The `ConcurrentMap` interface (part of the `NamedCache` interface) supports explicit locking operations. Locks only block calls to `lock()`, much like Java `synchronized`. A lock can only be released when the locking client departs or by the same thread or another thread in the cluster node calling `unlock()`. This is determined by setting the `<lease-granularity>` element on the cache configuration to `thread` or `member`.

Managing Data Consistency and Transactions: Coherence (continued)

- **Transactions:**
 - Transaction features are new with version 3.6. The `TransactionScheme` and `Transaction` Framework API builds on top of explicit locking operations to support ACID-style transactions. It provides a connection-based approach to joining a Coherence cluster, and provides true transaction capabilities, including full XA support. Transactions are covered in more detail in the lesson titled “Transactions and Coherence.”
 - Container Integration: For transaction management in a Java EE container, Coherence provides a Java EE Connector Architecture (JCA) resource adapter to allow transactions to be managed via the Java Transaction API (JTA). Using the adapter, Coherence can participate in XA transactions that are managed by a Java EE container's transaction manager.
- **EntryProcessors:** Coherence also supports a lock-free programming model through the `EntryProcessor` API. For many transaction types, this minimizes contention and latency, and improves system throughput, without compromising the fault-tolerance of data operations.

ConcurrentMap



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ConcurrentMap

Explicit locking can be very expensive. The simplest of use cases involves several network hops for a single update. And while the entry is locked (see the code example), applications that also use locking are blocked until object A is unlocked.

ConcurrentMap

- ConcurrentMap:
 - Provides lock() / unlock() methods for keys (similar to Java synchronization)
 - Threads must coordinate reads/writes through locking
 - Locks are unaffected by server failure
 - Locks are released when client disconnects
 - get() and put() operations are allowed while key is locked



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ConcurrentMap (continued)

The NamedCache interface extends the ConcurrentMap interface which provides methods for locking and unlocking entries related to specific keys within a cache. Coherence lock functionality is similar to the Java synchronized keyword: locks only block locks. The difference is that synchronized is not distributed, whereas locks are distributed, as seen in the previous diagram. Threads must all use locking to coordinate access to data. If a key is locked, another thread can read the data without locking. If a server fails, the lock persists by failing over to the backup server. Conversely, if a client crashes and holds a lock to a key, the key will automatically be released immediately. Threads and clients have to coordinate locking/unlocking. ConcurrentMap locks are distributed globally, meaning they affect all cache clients, not just on one JVM. ConcurrentMap does not enforce that a lock is held prior to allowing a get operation. It would be up to the application to enforce that.

ConcurrentMap: Example

```
NamedCache cache = CacheFactory.getCache("Airports");  
  
Object key = "SFO"; Lock the SFO key.  
cache.lock(key, -1);  
try {  
    Object value = cache.get(key); Get SFO data from cache.  
    // application logic Do something...  
    cache.put(key, value); Update SFO data in cache.  
} finally {  
    cache.unlock(key); Always unlock in a finally block to ensure that uncaught exceptions don't leave data locked.  
}
```

Application.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ConcurrentMap: Example

Here is an example of the ConcurrentMap API used to lock and unlock data. The code accesses the Airports cache, and uses the lock method to lock the SFO key. The second argument passed is a -1 which specifies that the application is willing to wait indefinitely for the lock. Once the lock is obtained, the application gets the SFO data from the cache, does some processing, then puts the data into the cache to update the cache. The unlock method is then called as part of the try/catch/finally block to release the lock so other applications can potentially work with the SFO object.

Issues with Locking

- Locking can cause scalability problems:
 - Applications must wait for locks to be released before accessing data.
 - Locking increases latency via the number of network round trips.
- EntryProcessors provide scalable, lock-free updates of data.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Issues with Locking

Both the ConcurrentMap and Transaction Framework APIs provide locking mechanisms to control concurrent access to data, which is quite similar to how transactions are done in relational databases. The transaction API also provides for commits and rollbacks. You can update one or more objects and commit the transaction as one atomic transaction. If something goes wrong, you can roll back the transaction and undo the updates that occurred.

When you lock an object in an application, the lock prevents other applications and other Coherence users from updating that object or accessing that object. As a result, locking can cause scalability problems. If you lock an object, other requests from other users to access the object are affected. Typically, the application waits for that object to be unlocked (perhaps indefinitely). This can cause performance and scalability issues. While that transaction occurs, other users, who are waiting, see the hourglass icon on their screens.

Issues with Locking (continued)

A very important drawback to `lock()`/`unlock()` paradigm is the number of network roundtrips that are required. In the previous example, there are several network round trips that are required (`lock`, `get`, `put`, `unlock`), which greatly increases the client latency, as well as decreases scalability because of increased contention.

Though it is possible to perform locking and execute transactions in Coherence through the Coherence `ConcurrentMap` and Transaction Framework APIs, there is something that is unique to Coherence – a feature known as `EntryProcessors`. `EntryProcessors` enable you to update data on the system without locking and without the cost of losing scalability and performance.

Updating Data in Coherence

- So far you performed only single inserts or updates by key for updating data in Coherence:
`myCache.put(key, object);`
- What if you want to update data without the key? What if the key is unknown?
In SQL, this would be something like:
`UPDATE orders SET priority = 1
WHERE order_amount > 1000`
- What if you want to update a lot of data simultaneously in Coherence?



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Updating Data in Coherence

So far you performed single inserts and updates by key. You used the `put()` method of `NamedCache`. However, there can be situations where you might be required to update the data without knowing the value of the key. For example, you might need to execute statements that are similar to SQL, such as, `"UPDATE orders SET priority=1 WHERE order_amount > 1000."`

Because Coherence does not support SQL, you must perform these operations in Coherence without using SQL. However, the CohQL query feature, covered in lesson titled “Querying and Aggregating Data in the Cache,” can be used to set a query filter that is SQL-like. In Coherence, you have the capability of performing mass updates such as the one discussed in the slide. Not only can you perform a mass update, updating a lot of data simultaneously, you can do it without locking and without causing scalability issues.

What is an EntryProcessor?

- An EntryProcessor is an agent that performs processing against entries where they are managed:
 - Requests are sent directly to owners to do the work.
 - Requests are queued, so locks are not necessary.
- This is equivalent to "agents" executing services in parallel on the data in the cluster.
- Entry processing:
 - May mutate cache entries, including creating, updating, or removing them
 - May perform simple calculations or any other type of processing



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is an EntryProcessor?

Coherence supports a lock-free programming model through the `EntryProcessor` API. This minimizes contention and latency, and improves system throughput, without compromising the fault tolerance of data operations.

Every `NamedCache` implements the `InvocableMap` interface. The `EntryProcessor` interface (contained within the `InvocableMap` interface) is the agent that performs processing against the entries directly where they are managed. So, on the client, method of an `EntryProcessor` can be invoked remotely. This method is sent directly to the storage JVM where the data is located, and directly to the owners that actually do the work.

The requests are queued, so locks are not necessary. If there are multiple requests to update the same object, Coherence automatically queues them and performs the updates one after the other. This results in the ability to perform updates on the system without locks.

`EntryProcessors` are equivalent to agents executing services in parallel on the data in the cluster. For example, if a request to update many objects in the cluster is executed, Coherence automatically performs the update in parallel on all the storage JVMs that own that data. This is done automatically, and the client does not have to launch the agents or manage any agents.

What is an EntryProcessor? (continued)

The processing may mutate NamedCache entries, including creating, updating, or removing them. The processing can also perform calculations, aggregations, or any other function because you can also write your own `EntryProcessors` and invoke them from the client.

Coherence includes several `EntryProcessor` implementations for common use cases.

The `InvocableMap` superinterface of `NamedCache` allows for concurrent lock-free execution of processing code within a cache. This processing is performed by an `EntryProcessor`. In exchange for reduced flexibility compared to the more general purpose Transaction Framework and `ConcurrentMap` explicit-locking APIs, `EntryProcessors` provide the highest levels of efficiency without compromising data reliability.

Because `EntryProcessors` place an implicit low-level lock on the entries they process, the end user can place processing code in an `EntryProcessor` without having to worry about concurrency control. Note that this is not the same as the explicit lock(key) functionality provided by `ConcurrentMap`.

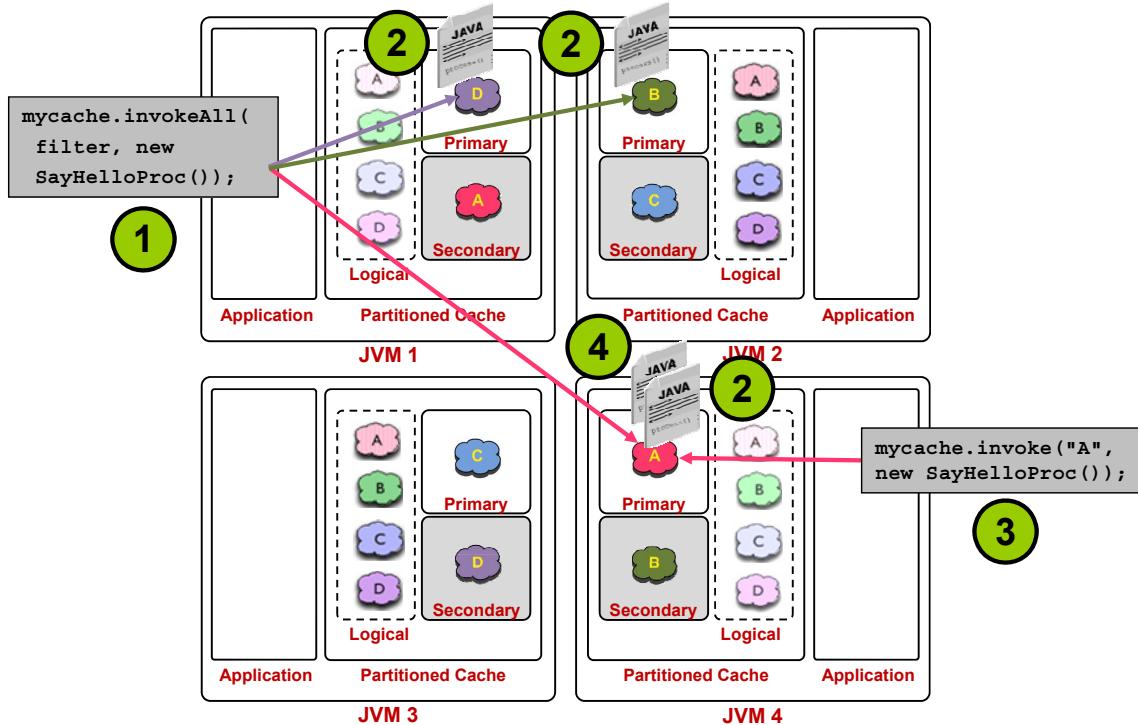
In a replicated cache or a partitioned cache that runs under Caching Edition, execution happens locally on the initiating client. In partitioned caches that run under Enterprise Edition or higher, execution occurs on the node that is responsible for the primary storage of data.

In partitioned caches that run under Enterprise Edition or higher, `EntryProcessors` are executed in parallel across the cluster (on the nodes that own the individual entries). This provides a significant advantage over having a client lock all affected keys, pull all required data from the cache, process the data, place the data back in the cache, and unlock the keys. The processing by `EntryProcessors` occur in parallel across multiple machines (as opposed to serially on one machine) and the network overhead of obtaining and releasing locks is eliminated.

`EntryProcessors` are individually executed atomically. However, multiple `EntryProcessor` invocations via `InvocableMap.invokeAll()` will not be executed as one atomic unit. As soon as an individual `EntryProcessor` has completed, any updates made to the cache will be immediately visible while the other `EntryProcessors` execute. Furthermore, an uncaught exception in an `EntryProcessor` will not prevent the others from executing. If the primary node for an entry fails while executing an `EntryProcessor`, the backup node will perform the execution instead. However, if the node fails after the completion of an `EntryProcessor`, the `EntryProcessor` is not invoked on the backup.

Note that in general, `EntryProcessors` should be short-lived. Applications with longer running `EntryProcessors` should increase the size of the distributed service thread pool so that other operations performed by the distributed service are not blocked by the long running `EntryProcessor`.

Entry Processor Execution Diagram



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Entry Processor Execution Diagram

When an `EntryProcessor` is invoked, the processor class that is passed as the argument to the call is serialized and sent to the JVM where each entry that matches the keyset exists. The processor is deserialized at each location across the cluster, the entry in question is passed as an argument to the processor, and the entries are processed in parallel.

This diagram shows two kinds of Entry Processor method invocations, one that uses a filter, and one that uses a single key directly. Here is what happens when these methods are invoked:

1. A filter has been constructed either using the Coherence filtering mechanism or the CohQL feature, and for the sake of argument only objects A, B, and D meet the filter specifications when the query is executed. The application calls `invokeAll(filter, new SayHelloProc())`, where the filter is the filter described previously, and `SayHelloProc` is a serializable Java class that implements a method called `process`.
2. The newly instantiated `SayHelloProc` object is serialized and sent to the JVMs where the primary entries for keys A, B, and D exist. Coherence deserializes the `SayHelloProc` object and passes the A, B, and D entries to its `process` method. The `process` method can execute methods on the entry object, or do whatever else it wants

Entry Processor Execution Diagram (continued)

to do. In this case, just assume that the `SayHelloProc`'s process method simply executes a method on the entries called, `sayHello()`, which only prints a "Hello" message out to the screen. That "Hello" message will appear on the stdout of whatever program is running Coherence and contains that key's entry. Notice that the #2 step in this process is listed in the diagram three times. This is because the entries are all getting processed in parallel.

3. While step #2 is processing, another application performs a direct invocation on the entry for key "A" by calling `invoke("A", new SayHelloProc())`. In this case, the previous Entry Processor that was running is still processing the `invokeAll()` call described previously. The `SayHelloProc()` object is still passed to the JVM where A exists, but...
4. Coherence queues the invoke operation while the first Entry Processor is executing. When it is done processing, then the second invocation request is executed. Any subsequent requests for A, or any other keys that are busy processing, are queued and processed in order.

Out-of-the-Box EntryProcessors

- There are a number of provided EntryProcessors:
 - AbstractProcessor, CompositeProcessor, ConditionalProcessor, ConditionalPut, ConditionalPutAll, ConditionalRemove, ExtractorProcessor, NumberIncrementor, NumberMultiplier, PreloadRequest, PropertyProcessor, UpdaterProcessor, VersionedPut, and VersionedPutAll

See docs
for more
details!

```
//put BZE into the cache if it's not already there
String key = "BZE";
Airport airport = new Airport(key, "Goldson International");
airportCache.invoke(key, new ConditionalPut(
    new NotFilter(PresentFilter.INSTANCE), airport));
```

Updates only if
not already
present in
cache.

Application.java

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Out-of-the-Box EntryProcessors

There are a number of default EntryProcessors that are provided by Coherence. Typically, you can also write your own, but there are some EntryProcessors, such as ConditionalProcessor or ConditionalPut, that enable you to update the data only if it meets a certain value or if the value exists.

- NumberIncrementor: An EntryProcessor that is similar to a sequence in Oracle Database where you can automatically increment a number
- CompositeProcessor: CompositeProcessor represents a collection of EntryProcessor objects that are invoked sequentially against the same entry.

An agent implements the EntryProcessor interface, typically by extending the AbstractProcessor class. A number of agents are included with Coherence, including:

- AbstractProcessor: An abstract base class for building an EntryProcessor
- ExtractorProcessor: Extracts and returns a specific value (such as a property value) from an object that is stored in an InvocableMap
- ConditionalProcessor: Conditionally invokes an EntryProcessor if a filter against the entry-to-process evaluates to TRUE

Out-of-the-Box EntryProcessors (continued)

- PropertyProcessor: An abstract base class for EntryProcessor implementations that depend on a PropertyManipulator
- NumberIncrementor: Pre- or postincrements any property of a primitive integral type, as well as Byte, Short, Integer, Long, Float, Double, BigInteger, and BigDecimal
- NumberMultiplier: Multiplies any property of a primitive integral type, as well as Byte, Short, Integer, Long, Float, Double, BigInteger, and BigDecimal, and returns either the previous or a new value

For additional details, refer to Coherence APIs Java Doc.

In this code example:

- A key is created as a string with the value "BZE" which stands for the airport code.
- An Airport object is created using the BZE key, and the name of the airport to add to the cache.
- Next, the invoke method is performed against the "BZE" key using a conditional put that uses a NotFilter that takes PresentFilter.Instance as parameter. This means that the object is placed into the cache only if it does not already exist. This code will not overwrite any object that already exists in the cache. The entire process of inserting the object occurs without downloading any data to the client, and the processor is invoked directly on the server. The server looks up the object to see if it is there. If it is not there, it does the insert (put).

The other way of doing this is to do a get to download the data to the client, see whether the data actually exists, and then perform a put. However, this involves a number of network hops. By using an EntryProcessor, tasks are performed in parallel directly on the server JVMs where the data is located. This involves fewer network hops and you do not have to download the data to the client. So EntryProcessors are faster and more efficient than performing updates on the client.

Note: The Coherence Incubator project also contains a number of complimentary EntryProcessors.

Developing EntryProcessors

Overview:

- Implement the `EntryProcessor` interface. Use the `InvocableMap.Entry` interface to interact with the passed-in cache entry.
- Call one of the `InvocableMap` methods to invoke the `EntryProcessor` on matching entries



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Developing EntryProcessors

The steps to develop an `EntryProcessor` include:

- Implementing the `EntryProcessor` interface with code that interacts with cache entries.
- The entries can be cast into their original class type, and the class' methods can be invoked directly. The `Entry` is passed in to the `EntryProcessor` method as an `InvocableMap.Entry` type. This interface provides methods to interact with the entry.
- Determine which `InvocableMap` method to use, and pass the `EntryProcessor` object to use as the second parameter. When the method is called, the `EntryProcessor` is passed to the JVMs of all matching entries for execution against each entry.

The next few slides cover each of these interfaces in more detail.

InvokableMap Interface

```

package com.tangosol.util;

public interface InvokableMap {
    public Object invoke(Object oKey,
                         InvokableMap.EntryProcessor processor);
    An informative subset of  
InvokableMap methods.  
See docs for complete list!
    Invokes the passed EntryProcessor against the entry specified by the passed key,  

    returning the result of the invocation.

    public Map invokeAll(Collection keys,
                         InvokableMap.EntryProcessor processor);
    Invokes the passed EntryProcessor against the entries specified by the passed keys,  

    returning the result of the invocation for each entry.

    public Map invokeAll(Filter filter,
                         InvokableMap.EntryProcessor processor);
    Invokes the passed EntryProcessor against the set of entries that are selected by the  

    given filter, returning the result of the invocation for each entry

    ...
}

```

InvokableMap.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

InvokableMap Interface

An InvokableMap is a map against which both entry-targeted processing and aggregating operations can be invoked. Though a traditional model for working with a map is to have an operation access and mutate the map directly through its API, InvokableMap allows that model of operation to be inverted such that the operations against the map contents are executed by (and thus within the localized context of) a map. This is particularly useful in a distributed environment, because it enables the processing to be moved to the location at which the entries-to-be-processed are managed, thus providing efficiency by localization of processing. The methods of the InvokableMap interface include:

- `invoke()`: Invokes the passed EntryProcessor against the entry specified by the passed key, returning the result of the invocation. The InvokableMap interface on the client can call invoke. A particular key is passed to update a single object, and the EntryProcessor object to execute is passed. The EntryProcessor is passed to the JVM where the specified entry exists, and passes that Entry to the EntryProcessor.
- `invokeAll(keys)`: Invokes the passed EntryProcessor against the entries specified by the passed keys, returning the result of the invocation for each entry. When invokeAll is executed, a collection of keys is passed if the keys for the entries to update are known.

InvocableMap Interface (continued)

- `invokeAll(filter)`: Invokes the passed `EntryProcessor` against the set of entries that are selected by the given filter, returning the result of the invocation for each entry.

Usually `invokeAll` is used with a filter, because the collection of keys to process against is unknown. When a filter is passed to the `invokeAll` method, Coherence queries the cache using the filter to figure out the entries to use for the call, sends the `EntryProcessor` code to each JVM where the entries exist, and passes each `Entry` to the `EntryProcessor` code for execution.

All processing is done in parallel on the server. Only the `invokeAll` method is performed on the client side. Applications scale well because processing is executed in parallel across the CPUs that comprise the Coherence cluster. The processing power of the `EntryProcessor` is directly proportional to the number of machines that are in the cluster.

InvocableMap.EntryProcessor Interface

```

package com.tangosol.util;

public static interface InvocableMap.EntryProcessor
    extends Serializable {
    public Object process(InvocableMap.Entry entry);
        Processes a Map.Entry object.

    public Map processAll(Set setEntries);
        Processes a set of InvocableMap.Entry objects (implementation typically provided by a
        super-class).
}

```

If `processAll()` is not implemented, Coherence executes `process()` repeatedly.

InvocableMap.EntryProcessor.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

InvocableMap.EntryProcessor Interface

Aside from the out-of-the-box EntryProcessors, Coherence allows implementing custom EntryProcessors to execute on the server. The `EntryProcessor` interface methods include:

- `process()`: The `process` method processes a single `InvocableMap.Entry` object at a time. As seen in the earlier example, an entry is what is passed into the `process` method. This is done automatically by Coherence. The server passes each entry that matches the associated filter or keys.
- `processAll()`: The `processAll` method processes a set of `InvocableMap.Entry` objects. It is optional to implement the `processAll` method (if extending the `AbstractProcessor`), which provides bulk processing for a large number of entries. The `processAll` method is not required for handling multiple entries, as the `EntryProcessor`'s `process` method is called repeatedly when the `processAll` method is not implemented.

InvocableMap.Entry Interface

```
package com.tangosol.util;                                         InvocableMap.Entry.java

public static interface InvocableMap.Entry extends QueryMap.Entry {

    public Object getKey();                                         Returns the key corresponding to this entry.

    public Object getValue();                                       Returns the value corresponding to this entry.

    public boolean isPresent();                                     Determines whether the entry exists in the map.

    public void remove(boolean isSynthetic);                         Removes this entry from the map if present.

    public Object setValue(Object value);                           Stores the value corresponding to this entry.

    public void setValue(Object value, boolean isSynthetic);       Stores the value corresponding to this entry.

}
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

InvocableMap.Entry Interface

An InvocableMap.Entry contains additional information and exposes additional operations that the basic Map.Entry does not. It allows nonexistent entries to be represented, thus allowing their optional creation. It allows the existing entries to be removed from the map. It supports several optimizations that can ultimately be mapped through to indexes and other data structures of the underlying map. Some of the methods that are available in EntryProcessor are:

- `getKey()`: Returns the key corresponding to this entry. This gets the key of the entry that is passed into the EntryProcessor.
- `isPresent()`: Determines whether the entry exists in the map.
- `remove()`: Removes an entry from the cache. A boolean value specifying if this entry is synthetic is passed as the parameter to this method. This parameter tells the system whether it is a synthetic entry or not.

In Coherence, there are *synthetic* and *nonsynthetic* updates. A synthetic update is something that Coherence does. For example, when the server JVM is taken off the cluster, or it crashes, Coherence automatically redistributes data as discussed earlier. This is an event or an update to the cache. Such an event is referred to as synthetic because it is Coherence that initiates it.

InvocableMap.Entry Interface (continued)

A nonsynthetic event is something that an application initiates, such as an actual update. This is discussed in the section on event notifications.

On the client, when events that are occurring on the grid (such as inserts, updates, and deletes), the `isSynthetic` flag is used to filter out any Coherence-generated, or synthetic, events.

- `setValue()`: This method stores the value corresponding to the entry, which mutates the entry. If `setValue` is not called, the entry is not mutated.

EntryProcessor Semantics

- EntryProcessors invoked against the same key are locally queued. This means responsibility-free lock (high throughput) processing.
- EntryProcessors can return any "serializable" value. This includes null if a result is not required.
- An EntryProcessor can be invoked against entries that do not yet exist. The `Entry.isPresent()` method determines if an entry exists.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

EntryProcessor Semantics

If multiple clients are executing an EntryProcessor against the same entry, each local storage JVM will automatically queue those EntryProcessors. This is done without locking or invoking any code on the client. Each entry executes one after the other, in the order they are queued. EntryProcessors can return values, and that value can be any serializable object. If a return value is not required, the EntryProcessor returns null. An EntryProcessor can perform aggregations or actions that return results to the client. An EntryProcessor can be invoked against entries that do not currently exist. Within the EntryProcessor, there is the `isPresent()` method to determine whether the entry exists or not.

EntryProcessor Behavior

- EntryProcessors are synchronous. The client waits until the `invoke()` or `invokeAll()` executes.
- `invoke()` is an atomic operation. It either succeeds entirely or fails entirely.
- `invokeAll()` is *not* an atomic operation.
- EntryProcessors must be idempotent. If the client fails:
 - An `invoke()` that returns successfully is executed
 - An `invokeAll()` may be partially executed. Some operations may succeed and others may not be executed.
- If the server (storage JVM) fails, the EntryProcessor is guaranteed to be executed on the surviving storage JVMs.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

EntryProcessor Behavior

It is important to understand the behavior of EntryProcessors, and that it is quite different from transactions in a database.

EntryProcessors are synchronous. The client waits until `invoke()` or `invokeAll()` executes. For example, when `invokeAll()` is performed, the client blocks until all the storage JVMs have executed `invokeAll()` before continuing to the next line of code.

`Invoke()` is an atomic operation. It either succeeds entirely or fails entirely. If it fails, Coherence performs a rollback. However, `invokeAll()` is *not* an atomic operation. If the client executes an `invokeAll()`, and it fails, any EntryProcessor that returns successfully is executed. An `invokeAll()` could be partially executed, that is, some operations may succeed whereas others may not be executed.

If the server JVM fails in the middle of an EntryProcessor, the EntryProcessor is guaranteed to be executed among the surviving storage JVMs. The queues are backed up as data. If the primary JVM goes down, Coherence guarantees that the EntryProcessor is executed. So, when the JVM goes down, there is a backup of that queue on another JVM.

EntryProcessor Behavior (continued)

Coherence internally keeps track of whether that EntryProcessor (on the failed JVM) was executed and executes that EntryProcessor automatically on another JVM after it gets redistributed.

In Coherence 3.3, there was the possibility of an EntryProcessor being executed twice. If an EntryProcessor was executed, and then the JVM went down before informing Coherence that the EntryProcessor had been executed, Coherence executed that EntryProcessor on another JVM. However, in Coherence 3.4 and later, Coherence not only guarantees that the EntryProcessor is executed, but that it is executed only once.

Contract for Writing EntryProcessors

- Exceptions thrown within EntryProcessors are wrapped and rethrown to the application-calling thread.
- Failure to call `entry.setValue(...)` or `entry.remove()` on a value means that no cache entry mutation will occur.
- If a filter is used, the EntryProcessor should not affect the filter evaluation.

Example:

```
invokeAll(New GreaterEqualsFilter("getAmount", 100), UpdateOrderAmount());
```

- EntryProcessors essentially have READ COMMITTED isolation. They read the most recently updated copy of the entry.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Contract for Writing EntryProcessors

Exceptions thrown within EntryProcessors are wrapped and rethrown to the calling thread. EntryProcessors do not automatically update entries. Updates only occur when `setValues()` or `remove()` is called. If a filter is used, the EntryProcessor should not affect the filter evaluation. An example of an EntryProcessor's filter affecting filter evaluation is when `invoke()` is called with a greater than or equals filter of `getamount=100`, and the EntryProcessor updates the order amount of each entry. This creates a circular dependency. In this example, the EntryProcessor is executed on entries where the amount is greater than 100, but also updates the order amount in the same EntryProcessor. This can have unpredictable results.

EntryProcessors, in database terminology, essentially have read-committed isolation. That is, EntryProcessors execute the most recently updated entries. If there are transactions occurring in Coherence, any uncommitted data is not going to be available to EntryProcessors. There are no dirty read isolations. That is, with EntryProcessors, you can be assured that they will execute only on entries that are actually committed into the Coherence grid.

Example: Creating a Custom EntryProcessor

- Example ChangeAirportCountryProcessor is as follows:

```
class ChangeAirportCountryProcessor extends AbstractProcessor {
    ...
    Object process(Entry entry) {
        Airport airport = (Airport)entry.getValue();
        airport.setCountry(this.country);
        entry.setValue(airport);
        return null;
    }
}
```

ChangeAirportCountryProcessor.java

- Now run this EntryProcessor on all entries:

```
// now run the EntryProcessor within the client application
airportCache.invokeAll(AlwaysFilter.INSTANCE,
    new ChangeAirportCountryProcessor("USA"));
```

Application.java

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Example: Creating a Custom EntryProcessor

This example demonstrates the following:

- An implementation of EntryProcessor called ChangeAirportCountryProcessor that does a bulk update on the system to change the country of all airports to USA.
- The EntryProcessor class extends AbstractProcessor, which is in the com.tangosol.util.processors package.
- The EntryProcessor class implements the process() method, which takes an Entry as the parameter. Within the process() method, the code gets the passed in Entry object, invokes its getValue() method to obtain the wrapped Airport object, and casts it into an actual Airport object.
- The code then invokes the Airport object's setCountry() method, using the value that was set in the constructor (not shown) when new was called on the ChangeAirportCountryProcessor object.
- After the value is changed in the Airport object, the new Airport object is updated within the Entry by calling the entry's setValue() method with the updated Airport object.

Example: Creating a Custom EntryProcessor (continued)

- The code does not return any values to the calling client, so the `process()` method returns `null`.
- The `ChangeAirportCountryProcessor` code is invoked when the application executes the `invokeAll()` method.

This EntryProcessor changes all Airport countries to "USA" because it uses the `AlwaysFilter.INSTANCE` filter which targets all entries in the cache. If there were 100,000 entries in the cache, this code would change every single entry's country to "USA". The EntryProcessor code is the code that actually runs on the server.

Quiz

Which of the following Coherence APIs manage data concurrency using locks? (Select all that apply)

- a. TransactionMap
- b. InvocableMap
- c. ConcurrentMap
- d. Transaction Framework



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: c, d

This is a bit of a trick question because technically "a" is also correct; however, the TransactionMap API is deprecated as of release 3.6. The InvocableMap is part of a framework that can update data without using locks.

Quiz

Which of the following provides scalable data updates without using locks?

- a. QueryMap
- b. ConcurrentMap
- c. EntryProcessor
- d. Transaction Framework



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: c

An EntryProcessor can update data without using locks. QueryMap does not update or lock data. ConcurrentMap and Transaction Framework both use locks for data concurrency.

Quiz

How are EntryProcessors able to perform reliable updates without using locks?

- a. The keys exist on different machines.
- b. Keys are synchronized on the server side.
- c. Transactions are used on the server side.
- d. Requests for the same key are queued.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: d

EntryProcessors queue requests to access the same key in the cluster. As a result, each processor is executed in order, and one at a time; thereby avoiding the need for complicated and expensive locking algorithms.

Practice.07.01 Overview: In-Place Processing of Data

This practice covers the following topics:

- Implementing a custom `EntryProcessor`
- Running a client that invokes the `EntryProcessor` to perform in-place processing of data
- Analyzing the results to determine `EntryProcessor` behavior



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.07.01 Overview: In-Place Processing of Data

This practice demonstrates implementing a custom `EntryProcessor`, coding a client application to execute the `EntryProcessor` across the cluster to perform in-place processing of data, and analyzing the results to see how `EntryProcessors` work.

Agenda

Scalability and Parallel Processing

Invocation Service

- What is an Invocation Service?
- Configuring an Invocation Service
- Implementing Invocation Service Agents



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is an Invocation Service?

An Invocation Service:

- Allows you to execute an agent on cluster members
- Does not execute against data entries in the cache
- Is suitable for cluster-wide management tasks
- Can be invoked synchronously or asynchronously
- Provides an “at most once” QoS:
 - Idempotent tasks
 - Non-critical tasks



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is an Invocation Service?

An invocation service is similar to an `EntryProcessor`, except that instead of executing against data entries in the cache, it executes against one or more members of the cluster. The concept is the same in that the invocation agent, which is conceptually like the `EntryProcessor` agent, is serialized and sent over to the Coherence servers for execution. Because they do not execute against data entries in the cache, they are suited more for management tasks in the cluster. Unlike `EntryProcessors` that can only be invoked synchronously, invocable agents can be executed either synchronously or asynchronously. Also unlike `EntryProcessors` that ensure that each entry is processed once and only once, the invocation service only guarantees that it will attempt to run the code once. Any failures are the responsibility of the developer to capture events and retry execution of the agent. There is no way to tell for sure if the code ran successfully or not, so invocation services are suitable only for non-critical or idempotent tasks.

Configuring an Invocation Service

Add the `<invocation-scheme>` element to the `<caching-schemes>` section of the `coherence-cache-config.xml` file:

```
<caching-schemes>
  ...
  <invocation-scheme>
    <scheme-name>MyInvocationServiceScheme</scheme-name>
    <service-name>InvocationService</service-name>
    <thread-count>5</thread-count>
    <autostart>true</autostart>
  </invocation-scheme>
  ...
</caching-schemes>
```



coherence-cache-config.xml

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuring an Invocation Service

To configure an invocation service in Coherence, edit the `coherence-cache-config.xml` file and add the `<invocation-scheme>` element and related child elements as shown in the example. Specifying a thread count provides a thread pool for running multiple invocation agents in parallel; otherwise, the main invocation service thread performs the execution.

Implementing the Invocable Interface

Invocable:

```
package com.tangosol.net;

public interface Invocable extends Runnable, Serializable {

    public void init(InvocationService service);
        Allows for initializing the agent.

    public void run();
        The method to execute on the cluster members.

    public Object getResult();
        Returns the result of the agent execution after the run method completes.
}
```

Invocable.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Implementing the Invocable Interface

The Invocable interface extends the Runnable interface by adding the `init()` and `getResult()` methods. Note that the class must be serializable in order to be passed to the cluster members for execution.

The interface methods work as follows:

- The `init()` method provides a hook for initializing the agent.
- The `run()` method is the method that contains the code to execute on the cluster members.
- The `getResult()` method is the method that is called to get the results after the execution of the `run` method.

Implementing the AbstractInvocable Interface

AbstractInvocable:

```
package com.tangosol.net;

public abstract class AbstractInvocable implements
    Invocable, Serializable {

    public void init(InvocationService service) { m_service = service; }

    Already implemented.

    public void run(); // Does not implement run!

    Intentionally left unimplemented. Use setResult() to set result for calls to getResult().

    public Object getResult() { return m_oResult; }

    Already implemented.

    protected void setResult(Object oResult) { m_oResult = oResult; }

    Already implemented.
}
```

AbstractInvocable.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Implementing the AbstractInvocable Interface

The AbstractInvocable interface is really an abstract class that extends the Invocable interface and provides a default implementation for the `init()`, `getResult()`, and `protected setResult()` methods. Writing a class that extends AbstractInvocable allows the developer the ease of only implementing the `run()` method. When using this approach, the `run` method should call the `setResult()` method prior to returning.

Example: AbstractInvocable Implementation

Here is an example of a class that extends AbstractInvocable:

```
public class CheckNumberOfCPUsAgent extends AbstractInvocable {  
    public void run() {  
        setResult(Runtime.getRuntime().availableProcessors());  
    }  
}
```

CheckNumberOfCPUsAgent.java

Returns the number
of CPUs available on
the machine where
the code executes.

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ORACLE

Example: AbstractInvocable Implementation

Here is an example of using the AbstractInvocable class to create an invocable agent. This is a class called CheckNumberOfCPUsAgent that simply implements the run method. The execution of this agent on a target cluster member obtains the number of available processors on the machine where the member exists, and sets the value using setResult(). A subsequent call to the agent's getResult() method would return the value to the calling client.

Examining the InvocationService Interface

InvocationService:

```
package com.tangosol.net;

public interface InvocationService extends Service {
    public abstract Map query(Invocable invocable, Set set);
    Synchronous execution method. Return value is a Map of results keyed by the member.

    public abstract void execute(Invocable invocable, Set set,
                                InvocationObserver invocationobserver);
    Asynchronous execution method. Results are returned as notifications to the
    InvocationObserver object.
}
```

InvocationService.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Examining the InvocationService Interface

After an Invocable agent is implemented, and an InvocationService is configured, the next step is to write the code that causes the agent to execute against the cluster. This is done by invoking one of the methods of the InvocationService interface:

- The `query()` method is for invoking the agent synchronously. The `Invocable` agent class is passed as the first parameter, and the set of target cluster members is passed as the second parameter. If `null` is passed for the member set, then the agent is executed on all cluster members. This method returns a Map of results that are keyed by member.
- The `execute()` method is for invoking the agent asynchronously. It takes the same parameters as the `query()` method, with the addition of an implementation of the `InvocationObserver` interface as a third parameter, which provides the callbacks for results and other notifications related to the execution of the agent. The `InvocationObserver` interface is shown on a later slide.

Executing an Invocation Agent Synchronously

Synchronous Execution:

The diagram shows a code snippet in a light gray box with handwritten annotations. At the top, an arrow points to the line `InvocationService service = (InvocationService) CacheFactory.getService("InvocationService");` with the annotation "Get invocation service.". Below it, another arrow points to the line `Map<Member, Integer> numCPUMap = service.query(new CheckNumberOfCPUsAgent(), null);` with the annotation "Invoke synchronously.". A third arrow points from the bottom left to the line `for (Map.Entry<Member, Integer> numCPU : numCPUMap.entrySet()) {` with the annotation "Iterate through returned results." A yellow box at the bottom right contains the file name `Application.java`.

```
InvocationService service = (InvocationService)
    CacheFactory.getService("InvocationService");

Map<Member, Integer> numCPUMap =
    service.query(new CheckNumberOfCPUsAgent(), null);

for (Map.Entry<Member, Integer> numCPU : numCPUMap.entrySet()) {
    System.out.println("Member: " + numCPU.getKey() +
        " Number of CPUs: " + numCPU.getValue());
}
```

Application.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Executing an Invocation Agent Synchronously

This is an example of application code that executes the `CheckNumberOfCPUsAgent` invocable agent synchronously using the `InvocationService query()` method. The code performs the following steps:

1. Obtains a reference to the `InvocationService` named "InvocationService" using the `CacheFactory.getService()` method. Note that this is the name specified in the example configuration shown previously.
2. Uses the returned service reference to call the `query()` method, passing in a new `CheckNumberOfCPUsAgent` class as the first parameter, and `null` as the second parameter. This causes the `CheckNumberOfCPUsAgent` class to get executed on all members of the cluster synchronously. Results of the execution are returned and captured in the `numCPUMap` variable.
3. Iterates through the `numCPUMap` result set, and prints out the results.

Executing an Invocation Agent Asynchronously

Invocation Observer Class:

```
private static class CPUInvocationObserver implements  
    InvocationObserver {  
  
    public void memberCompleted(Member member, Object result) {  
        System.out.println("Member: " + member + " Num CPUs: " + result);  
    }  
        Called when each member completes asynchronous agent execution.  
  
    public void memberFailed(Member member, Throwable throwable) {}  
        Called when a member fails during asynchronous agent execution.  
  
    public void memberLeft(Member member) {}  
        Called when a member leaves the cluster during asynchronous agent execution.  
  
    public void invocationCompleted() {}  
        Called when all members have completed asynchronous agent execution.  
}
```

CPUInvocationObserver.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Executing an Invocation Agent Asynchronously

In order to invoke an agent asynchronously, a class that implements the `InvocationObserver` interface must be written. This class provides the callback hooks when notifications related to the execution of the agent occur, including the return of results from each cluster member as well as when all cluster members have completed execution. The next slide shows how this class is used to invoke the agent asynchronously. Keep in mind that this class must be implemented as a thread safe class. Additionally, a member listener can be used to keep the application informed of cluster member state to make the use of the Invocation Service as efficient as possible.

Executing an Invocation Agent Asynchronously

Asynchronous Execution:

```
InvocationService service = (InvocationService)
    CacheFactory.getService("InvocationService");

service.execute(new CheckNumberOfCPUsAgent(), null,
    new CPUInvocationObserver());
}
```

Get invocation service.

Application.java

execute() is called with the agent and invocation observer.

The diagram shows a code snippet in a box labeled 'Application.java'. It starts with 'InvocationService service = ...', followed by 'CacheFactory.getService("InvocationService");'. An annotation 'Get invocation service.' points to this line with a curved arrow. Below it is 'service.execute(...);' with an annotation 'execute() is called with the agent and invocation observer.' pointing to it. A large curved arrow originates from the end of the 'Application.java' box and points to the 'execute()' call in the code.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Executing an Invocation Agent Asynchronously (continued)

This is an example of application code that executes the `CheckNumberOfCPUsAgent` invocable agent asynchronously using the `InvocationService execute()` method. The code performs the following steps:

1. The code obtains a reference to the `InvocationService` named "InvocationService" using the `CacheFactory.getService()` method.
2. Uses the returned service reference to call the `execute()` method, passing in a new `CheckNumberOfCPUsAgent` class as the first parameter, `null` as the second parameter, and a new `CPUInvocationObserver` object shown on the previous slide for receiving notifications. This causes the `CheckNumberOfCPUsAgent` class to get executed on all members of the cluster asynchronously. Results of the execution are returned by each cluster member by calling the `CPUInvocationObserver`'s `memberCompleted()` method.
3. Each member calls the `memberCompleted()` method which prints out the results.

Quiz

Implementing what interface defines the code to be executed using an Invocation Service?

- a. EntryProcessor
- b. InvocationService
- c. InvocableMap
- d. InvocableAgent



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: d

The InvocableAgent interface provides the `run()` method which contains the code to execute using an Invocation Service.

Quiz

How is the Invocation Service different from an EntryProcessor? (Select all that apply)

- a. Invocation Services do not require serialization.
- b. Invocation Services are not guaranteed to run.
- c. Invocation Services do not execute against data entries.
- d. Invocation Services queue requests for data entries.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: b, c

Because Invocation Services are sent over the network to other JVM processes, they must be serializable. Unlike EntryProcessors, which are guaranteed to run even in the event of a server failure, Invocation Services do not provide such a guarantee. Invocation Services do not execute against data entries; instead they execute non-data-related code on targeted cluster members. Because Invocation Services do not execute against data entries, they do not queue requests for data entries.

Quiz

Invocation Services can be invoked:

- a. Synchronously only
- b. Asynchronous only
- c. Both synchronously and asynchronously
- d. Persisted asynchronously



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: c

Invocation Services can be invoked *synchronously* using the `InvocationService query()` method, and *asynchronously* using the `InvocationService execute()` method. InvocationServices do not offer any form of persisted messaging.

Practice.07.02 Overview: Implementing Invocable Agents

This practice covers the following topics:

- Implementing a custom invocable agent
- Implementing a thread safe `InvocationObserver`
- Writing code that executes the invocable agent synchronously and asynchronously
- Examining the Invocation Service configuration
- Running the client and analyzing the results to determine behavior



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.07.02 Overview: Implementing Invocable Agents

This practice demonstrates implementing a custom Invocable Agent, coding a client application to execute the agent across the cluster to execute the agent on each cluster member, and analyzing the results to see how the Invocation Service works.

Summary

In this lesson, you should have learned how to:

- Manage concurrent access to data
- Update data in Coherence without locking
- Invoke methods of the `InvocableMap` interface
- Implement a custom `EntryProcessor`
- Implement a custom `Invocable Agent`



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and ORACLE CORPORATION use only

8

Transactions and Coherence

ORACLE®

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the basic concepts of transactions
- Implement a Coherence application that uses a transactional cache
- Implement a Coherence application that can participate in an XA transaction



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Overview of Transactions

- What are Transactions?
 - ACID Properties
 - Transaction Isolation Concepts

Coherence Cache Transactions

What are Distributed Transactions



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is a Transaction?

A Transaction defines a unit of work that is composed of a set of one or more state-changing operations, to be performed against a transactional resource in an all-or-nothing manner.

Examples of transactional resources:

- Databases
- Messaging Systems
- Transactional Caches



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Transaction Benefits

There are two main benefits of Transactions for applications.

Transactions:

- Ensure that a set of operations are performed as though they were a single operation
- Coordinate between multiple system components/processes/threads that are all attempting to concurrently modify the same information



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Transaction Benefits

- Should any one of the individual operations fail, for whatever reason, none of the operations in the transaction are performed, thus leaving the underlying resource unchanged.
- Transactions ensure that information remains consistent at all times in the resource.

ACID Properties

The term ACID is commonly used to describe the coherency of Transactions when using a specific Transactional Resource

Property	Description
Atomicity	Are operations all-or-nothing, that is, atomic?
Consistency	Are only valid operations allowed to be performed, that is, operations should not violate any consistency rules for the resource or information being changed?
Isolation	Are multiple transactions occurring concurrently isolated from impacting each other's execution?
Durability	Are committed transactions guaranteed not to be lost?



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ACID Properties

- **Atomicity:** Modifications must follow an “all-or-nothing” rule. Each transaction is said to be “atomic.” If one part of the transaction fails, the entire transaction fails. It is critical that the resource maintain the atomic nature of transactions in spite of any failure, including by operating system or hardware failure.
- **Consistency:** Only valid data will be written to the resource. If, for some reason, a transaction is executed that violates the consistency rules, the entire transaction will be rolled back and the resource will be restored to a state consistent with those rules. On the other hand, if a transaction successfully executes, it will take the resource from one state that is consistent with the rules to another state that is also consistent with the rules.

ACID Properties (continued)

- **Isolation:** Requires that multiple transactions occurring at the same time not impact each other's execution. For example, if Joe issues a transaction against a resource at the same time that Mary issues a different transaction, both transactions should operate on the resource in an isolated manner. The resource should either perform Joe's entire transaction before executing Mary's or vice versa. This prevents Joe's transaction from reading intermediate data produced as a side effect of part of Mary's transaction that will not eventually be committed to the resource. Note that the isolation property does not ensure which transaction will execute first, merely that they will not interfere with each other.
- **Durability:** Ensures that any transaction committed to the resource will not be lost. Durability is often ensured through the use of resource backups and transaction logs that facilitate the restoration of committed transactions in spite of any subsequent software or hardware failures.

Agenda

Overview of Transactions

Coherence Cache and Transactions

- Using the Transaction Framework API
 - Overview
 - Configuring Transactional Caches
 - Using a Transactional Cache
 - NamedCache API
 - Connection API
 - OptimisticNamedCache API
- Using the Coherence Resource Adapter
- Coherence Isolation Levels

What are Distributed Transactions



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Transactions Overview

- Introduced in Coherence 3.6
- Transactional caches are:
 - Specialized distributed caches that provide transactional guarantees
 - Managed by a Recovery Manager
- Recovery Manager is responsible for:
 - Recovering in-flight transactions when a member leaves the cluster
 - Acting as local resource manager for transactions that are either active or in the process of being committed/rolled-back
- There are two methods for using Transactional Caches:
 - Using the traditional NamedCache interface
 - Using the Connection-based API

*a part of a transaction resource
manager for Coherence
transactional caches.*



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Transactions Overview

The transactional framework uses a Recovery Manager to recover transactions when members that are acting as a transaction coordinator leave the service. If a transaction picked up from a departed member was partially committed then the Recovery Manager will drive that transaction to completion. Any transactions being recovered that are not partially committed will be rolled back.

Note: The previously provided transaction support using TransactionMaps is deprecated as of Coherence 3.6.

Configuring Transactional Caches

Transactional caches are defined within a cache configuration file using a <transactional-scheme> element.

```
<cache-config>
. . .
<cache-mapping>
    <cache-name>tx-*</cache-name>
    <scheme-name>transactional</scheme-name>
</cache-mapping>
. . .
<transactional-scheme>
    <scheme-name>transactional</scheme-name>
    <service-name>TxCacheService</service-name>
    <autostart>true</autostart>
    <request-timeout>30000</request-timeout>
</transactional-scheme>
. . .
</cache-config>
```



coherence-cache-config.xml

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

NamedCache API and Transactions

- Applications can obtain a transactional NamedCache instance through the standard Coherence CacheFactory APIs.

```
NamedCache cache = CacheFactory.getCache("tx-cache1");
```

- The transactional NamedCache instance can be used to implicitly perform cache operations within a single statement autocommit transaction.

```
Map map = new HashMap();
map.put(key1, "value1");
map.put(key2, "value2");
map.put(key3, "value3");
cache.putAll(map);
```

Transactions on the cache are atomic!



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

NamedCache API and transactions

The NamedCache API can be used to perform cache operations implicitly within the context of a transaction. However, this approach does not allow an application to change default transaction behavior. For example, transactions are in autocommit mode when using the NamedCacheAPI approach. This means each operation is immediately committed when it successfully completes. Multiple operations cannot be scoped into a single transaction. Applications that require more control over transactional behavior must use the Connection API.

The NamedCacheAPI approach is ideally suited for ensuring atomicity guarantees when performing single operations such as putAll. The following example demonstrates a simple client that creates a NamedCache instance and uses the CacheFactory.getCache() method to get a transactional cache. The example uses the transactional cache. The client performs a putAll operation that is only committed if all the put operations succeed. The transaction is automatically rolled back if any put operation fails.

Note that without a transaction, one of the three puts in the map could fail (for example, if denied by a map listener), but the other two would complete. With transactions, the failure of any put causes all puts to fail.

Connection API

The Connection API provides a logical connection to a Cluster to allow:

- Transaction boundary demarcation; for example, autocommit mode, explicit commit/rollback
- Specify Isolation levels; for example, read committed, statement consistent read, statement, monotonic consistent read, transaction consistent read
- Integration with transaction managers as a Resource Adapter (RA)



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Connection API

The connection factory deployed as part of the Coherence resource adapter will return Connection instances, and is different than non-RA use.

Using the Connection API

- Applications obtain a Connection instance through a transactional ConnectionFactory.

```
ConnectionFactory factory = new DefaultConnectionFactory();
Connection con = factory.createConnection("TxCacheService");
```

- Transactional Connections provide full programmatic control over transaction attributes, such as autocommit.

```
con.setAutoCommit(false);
con.setEager(true);
con.setIsolationLevel(STMT_CONSISTENT_READ);
OptimisticNamedCache c1 = con.getNamedCache("tx-cache1");
c1.insert(key1, value1);
con.commit();
con.close();
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Using the Connection API

Eager mode allows an application to control when cache operations are performed on the cluster. If eager mode is enabled, cache operations are immediately performed on the cluster. If eager mode is disabled, cache operations are deferred, if possible, and queued to be performed as a batch operation. Typically, an operation can only be queued if it does not return a value. An application may be able to increase performance by disabling eager mode. By default, eager mode is enabled and cache operations are immediately performed on the cluster.

Multiple Caches and Transactions

- The transactional Connection allows multiple caches to participate in a transaction.

```
...
OptimisticNamedCache c1 = con.getNamedCache("tx-cache1");
OptimisticNamedCache c2 = con.getNamedCache("tx-cache2");
...

con.setAutoCommit(false);
try {
    c1.insert(key1, value1);
    c2.insert(key2, value2);
    ...
    con.commit();
} catch (Exception e) {
    con.rollback();
}
...
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

OptimisticNamedCache API

- The OptimisticNamedCache interface is an extension to the NamedCache interface and is the base for all transactional caches.
- Each OptimisticNamedCache operation has an associated predicate in the form of a Filter that is used to validate the given value. The insert predicate is implicit.

```
package com.tangosol.coherence.transaction;
public interface OptimisticNamedCache extends NamedCache {

    public void insert(Object key, Object value);
    public update(Object key, Object value, Filter filter);
    public void delete(Object obj, Filter filter);
}
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

OptimisticNamedCache API

OptimisticNamedCache does not extend any operations from the ConcurrentMap interface since it uses its own locking strategy.

Note: All Transactional Caches provide by Coherence are optimistic in nature, hence the name OptimisticNamedCache.

In order to ensure correct updates/deletions are made during the prepare and commit phase, predicates, represented as filters, are checked prior to commit. Should a predicate fail, the transaction is rolled back.

There is generally no requirement to have pessimistic transactions support. This is also why Coherence does not support *serializable* level isolation and concurrency control

Example: OptimisticNamedCache

With OptimisticNamedCache, predicates:

- Allow puts to be “qualified”
- Are examined first, and then the put executed

```
Connection con = factory.createConnection("TxCacheService");
OptimisticNamedCache cache = con.getNamedCache("MyTxCache1");
. . .
try {
    //update only if the value is > 0
    cache.update(key, value, new GreaterFilter("balance", 0));
} catch (PredicateFailedException e) {
    // predicate failed, update not performed!
}
. . .
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Example: OptimisticNamedCache

The example shows an update only occurs if the predicate filter returns successfully.

Using the Coherence Resource Adapter

- The Coherence resource adapter allows applications to use a ConnectionFactory to create managed connections to a Coherence cluster.
- Coherence Transaction-aware ConnectionFactories may be obtained using a JNDI lookup. For example:

```
Context ic = new InitialContext();
ConnectionFactory cf =
    (ConnectionFactory)
        ic.lookup("java:comp/env/eis/CoherenceTxCF");
```

The underlying Coherence resource adapter leverages the Connection API of the Coherence Transaction Framework.

```
Connection conn= cf.createConnection("TxCacheService");
OptimisticNamedCache c1 = conn.getNamedCache("tx-cache");
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Using the Coherence Resource Adapter

The Coherence Resource Adapter allows caches to be combined into Web applications via an application server and treat caches as resources in the same way a JMS queue or a database connection is treated as a resource.

In this example a Web application can use an initial context to access Coherence and obtain a connection to a cache via JNDI.

Note that the initial context is created without arguments, indicating the object is being created within a Web application.

Coherence Resource Adapter and User-Managed Transactions

To use the Coherence Resource Adapter in user-managed transactions:

```
UserTransaction ut =
    (UserTransaction) ic.lookup("java:comp/UserTransaction");

OptimisticNamedCache cache = cohCon.getNamedCache("tx-cache");

ut.begin();
try {
    cache.insert(key1, "value1");
    cache.insert(key2, "value2");
    ut.commit();
    .
    .
} catch (Exception e) {
    ut.rollback();
    .
}
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Resource Adapter and User-Managed Transactions

Cache operations can be performed in the context of a user transaction as well as within the context of a container-managed transaction.

As the slide shows, an initial context is used to obtain a user transaction from the container, and then that transaction is used to encapsulate a set of `put` operations. Since both `puts` are within the same transaction, they happen atomically.

Multiple Resource Adapters

Multiple resources, such as JMS, datasources, and caches, can be combined into a single transaction.

```
.
.
.
OptimisticNamedCache cache = cohCon.getNamedCache("tx-cache");
DataSource ds = (DataSource) ic.lookup("jdbc/OracleDS");
java.sql.Connection dsCon = ds.getConnection();
java.sql.Statement statement = dsCon.createStatement();
.

ut.begin();
try {
    cache.insert(key1, "value1");
    statement.executeUpdate("INSERT INTO ... ");
    ut.commit();
} catch (Exception e) {
    ut.rollback();
}
.
```

Both in same TX



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Multiple Resource Adapters

Transactions are not limited to only Coherence caches, but can be combined with other transaction-aware elements, such as queues, databases, and any transaction-aware resource. In this example, a cache insert and a database statement are combined together in the same transaction, and complete or fail as a unit.

Transaction Isolation

- Transaction isolation (the “I” in “ACID”) describes what happens when two transactions operate on the same data concurrently.
- Transaction isolation can profoundly affect system performance as it typically involves:
 - Calculating isolation conditions
 - Managing partition, server, or cluster-level locking
 - Maintaining multiple versions of information for each transaction
- The *transaction isolation level* controls the behavior of a transaction by balancing:
 - Data integrity
 - Concurrent access speed



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Transaction Isolation

Two aspects of transactions can be controlled:

- The integrity of the data: Making sure the data is up-to-date and exact
- The concurrent access: Accelerating the access to data from multiple clients

Remember the inverse relationship: the higher the integrity of the data, the slower the access speed. These two variables can be controlled by the transaction isolation level.

Currently the transaction framework only uses row-level locking. Regardless of the isolation level being used, multiple versions of data are always maintained.

Transaction Phenomena

Three common conditions can occur in the middle of an uncommitted transaction:

- **Dirty read:** Reading an entry that was modified by another uncommitted transaction. That is, if the first transaction rolls back, the data you read may be stale.
- **Non-repeatable read:** Rereading an entry that was changed and committed by another transaction. That is, the entry changed in the middle of your transaction.
- **Phantom read:** Reperforming a query after another transaction has committed new insert entries in the result. That is, the same query produces a different set of entries.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Transaction Phenomena

The ANSI SQL-92 specification defines isolation conditions this way:

The isolation level specifies the kind of phenomena that can occur during the execution of concurrent transactions. The following phenomena are possible:

- P1 (“Dirty read”): Transaction T1 modifies an entry. Transaction T2 then reads that entry before T1 performs a COMMIT. If T1 then performs a ROLLBACK, T2 will have read a entry that was never committed and that may thus be considered to have never existed.
- P2 (“Non-repeatable read”): Transaction T1 reads an entry. Transaction T2 then modifies or deletes that entry and performs a COMMIT. If T1 then attempts to reread the entry, it may receive the modified value or discover that the entry has been deleted.
- P3 (“Phantom”): Transaction T1 reads the set of entries N that satisfy some condition. Transaction T2 then COMMITs operations that generate one or more entries that satisfy the condition used by Transaction T1. If Transaction T1 then repeats the initial query with the same condition, it obtains a different collection of rows.

For more details about isolation conditions, see the ANSI SQL-92 specification. A draft is available at: <http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt>.

Coherence Transaction Isolation Levels

Coherence supports five transaction isolation levels, a subset of which are shown.

Isolation Level	Dirty?	Non-Repeatable?	Phantom Read?	Data Version Used	Data Integrity	Performance
READ COMMITTED (default)	Not Possible	Possible	Possible	Latest	Low	Higher
STATEMENT CONSISTENT READ*	Not Possible	Possible	Possible	When Statement Execution Began	High	Middle
TRANSACTION CONSISTENT READ*	Not Possible	Not Possible	Not Possible	When Transaction Began	High	Lower

* Consistent read isolation levels may lag slightly behind the most current data in the cache.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Transaction Isolation Levels

Coherence supports the following transaction isolation levels:

- **READ_COMMITTED:** This is the default isolation level if no level is specified. This isolation level guarantees that only committed data is visible and doesn't provide any consistency guarantees. This is the weakest of the isolation levels, and generally provides the best performance at the cost of read consistency.
- **STMT_CONSISTENT_READ:** This isolation level provides statement-scoped read consistency, which guarantees that a single operation only reads data for the consistent read version that was available at the time the statement began. The version may or may not be the most current data in the cache.
- **STMT_MONOTONIC_CONSISTENT_READ:** This isolation level provides the same guarantees as STMT_CONSISTENT_READ, but reads are also guaranteed to be monotonic. This means that a read is guaranteed to return a version that is equal to or greater than any version that was previously encountered while using the connection. Due to the monotonic read guarantee, reads with this isolation may block until the necessary versions are available.

Coherence Transaction Isolation Levels (continued)

Coherence supports the following transaction isolation levels:

- TX_CONSISTENT_READ: This isolation level provides transaction-scoped read consistency which guarantees that all operations performed in a given transaction read data for the same consistent read version that was available at the time the transaction began. The version may or may not be the most current data in the cache.
- TX_MONOTONIC_CONSISTENT_READ: This isolation level provides the same guarantees as TX_CONSISTENT_READ, but reads are also guaranteed to be monotonic. This means that a read is guaranteed to return a version that is equal to or greater than any version that was previously encountered while using the connection. Due to the monotonic read guarantee, the initial read in a transaction with this isolation may block until the necessary versions are available

Note: * (asterisk) means that if a transaction writes and commits a value, then immediately reads the same value in the next transaction with one of the consistent read isolation levels, the updated value may not be immediately visible. If reading the most recent value is critical, then the READ_COMMITTED isolation level should be used.

Quiz

What does *eager mode* do? (Select all that apply)

- a. Preloads information required for a transaction into a local cache
- b. Automatically commits parts of a transaction when a timeout is reached
- c. Allows applications to control when cache operations are performed. Either immediately or in a batched manner is possible
- d. Is typically only useful when a transaction does not return a value



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: c,d

Quiz

Which is true of distributed transactions? (Select all that apply)

- a. They use resource managers to control access to resources such as database connections, caches, EJBs, JMS operations, and others
- b. They use transaction managers to coordinate resource managers
- c. They require a distinct second phase to all transactions where resource managers are asked if they are ready to commit changes



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: a, b

Quiz

The Connection API: (Select all that apply)

- a. May only be used by TCMP cluster members
- b. Is only supported in Java
- c. Is supported by .NET and C++ *Extend Clients
- d. Does not support InvocableMap



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: a,b

Quiz

True or False? Coherence Transactions may only be applied to a single cache at a time.

- a. True
- b. False



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: b

False, Using the Connection API, multiple transactional caches may be used in a single transaction.

Practice.08.01 Overview: Configuring, Running, and Reviewing Transactional Clients

This practice covers the following topics:

- Configuring a Coherence transactional scheme
- Writing a transactional Coherence client using the NamedCache API
- Writing a transactional Coherence client using the Connection API



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Overview of Transactions

Coherence Cache and Transactions

What are Distributed Transactions?

- 2PC and XA
- Transaction Managers



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What are Distributed Transactions?

- Coherence can participate in what is formally known as a *Distributed Transaction*.
- In Distributed Transactions:
 - A resource, such as data, a Coherence cache, a topic, a queue, or EJB is controlled through software known as a resource Manager (RM).
 - Transaction managers coordinate resource managers for resource sets.

Local transactions deal with a single resource manager



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What are Distributed Transactions?

A transaction originator initiates the transaction. The transaction originator can be a user application, Coherence, an Enterprise JavaBean (EJB), or a Java Message Service (JMS) client. A transaction is often referred to as a transaction context.

A transaction manager manages transactions on behalf of application programs. A transaction manager coordinates commands from application programs to start and complete transactions by communicating with all resource managers that are participating in those transactions. When resource managers fail during transactions, transaction managers help resource managers decide whether to commit or roll back pending transactions. The communication between the transaction manager and a specific resource manager is called a transaction branch.

A recoverable resource provides the ability to storage in a manner that is recoverable in the event of failure. Most recoverable resources use persistent storage to achieve this. Coherence uses other members of the cluster.

What are Distributed Transactions? (continued)

A resource manager provides access to a collection of information and processes. Transaction-aware JDBC drivers are common resource managers. Resource managers provide transaction capabilities and permanence of actions; they are entities accessed and controlled within a distributed transaction. Resource managers are software tools that are typically provided by the vendor of the storage mechanism, such as Oracle, Sybase, DB2, and so forth. Some resource managers are available from third-party vendors. There is nothing to prevent you from writing your own resource managers.

Two-Phase Commit (2PC) Protocol

- The 2PC protocol uses two steps or phases to commit changes within a distributed transaction.
 - Phase 1 asks all RMs to *prepare* to make the changes, and if any of them cannot, the transaction is aborted.
 - Phase 2 asks all RMs to actually commit and make the changes permanent.
- A *global transaction ID* (XID) is used to track all changes associated with a distributed transaction.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Two-Phase Commit (2PC) Protocol

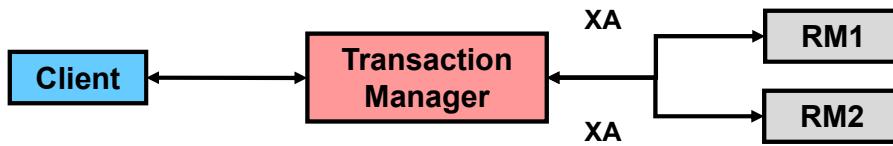
A distributed transaction is a transaction that updates multiple resource managers (such as databases) in a coordinated manner.

The two-phase commit protocol is a method of coordinating a single transaction across two or more resource managers. It guarantees data integrity by ensuring that transactional updates are committed in all the participating resources, or are fully rolled back out of all the resources, reverting to the state prior to the start of the transaction. In other words, either all the participating resources are updated or none of them are updated.

Many software packages support 2PC protocol. For example, WebLogic Server supports distributed transactions and the two-phase commit protocol for enterprise applications.

Open Group XA (Extended Architecture) Interface

- The Open Group's XA standard is the interface used between the WLS TM and resource managers.
- The XA standard:
 - Enables the 2PC protocol
 - Allows programs to control RMs that are involved in distributed transactions



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Open Group XA (Extended Architecture) Interface

WebLogic Server supports the Open Group XA interface.

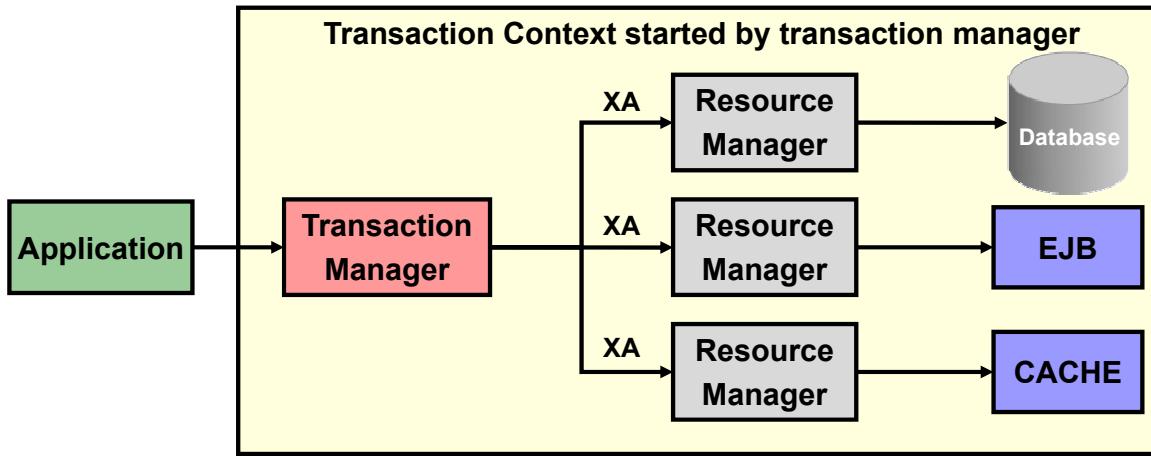
The Open Group Resource Managers are resource managers that can be involved in a distributed transaction by allowing their two-phase commit protocol to be controlled via The Open Group XA interface. The WebLogic Server supports interaction with The Open Group Resource Managers.

For a quick overview of how XA, 2PC, and Java EE work together, see Mike Spille's XA Exposed series of articles (I, II, and III), available at:

http://jroller.com/page/pyrasun/?anchor=xa_exposed

Transactions and Resource Managers

A single transaction manager typically coordinates multiple resource managers.

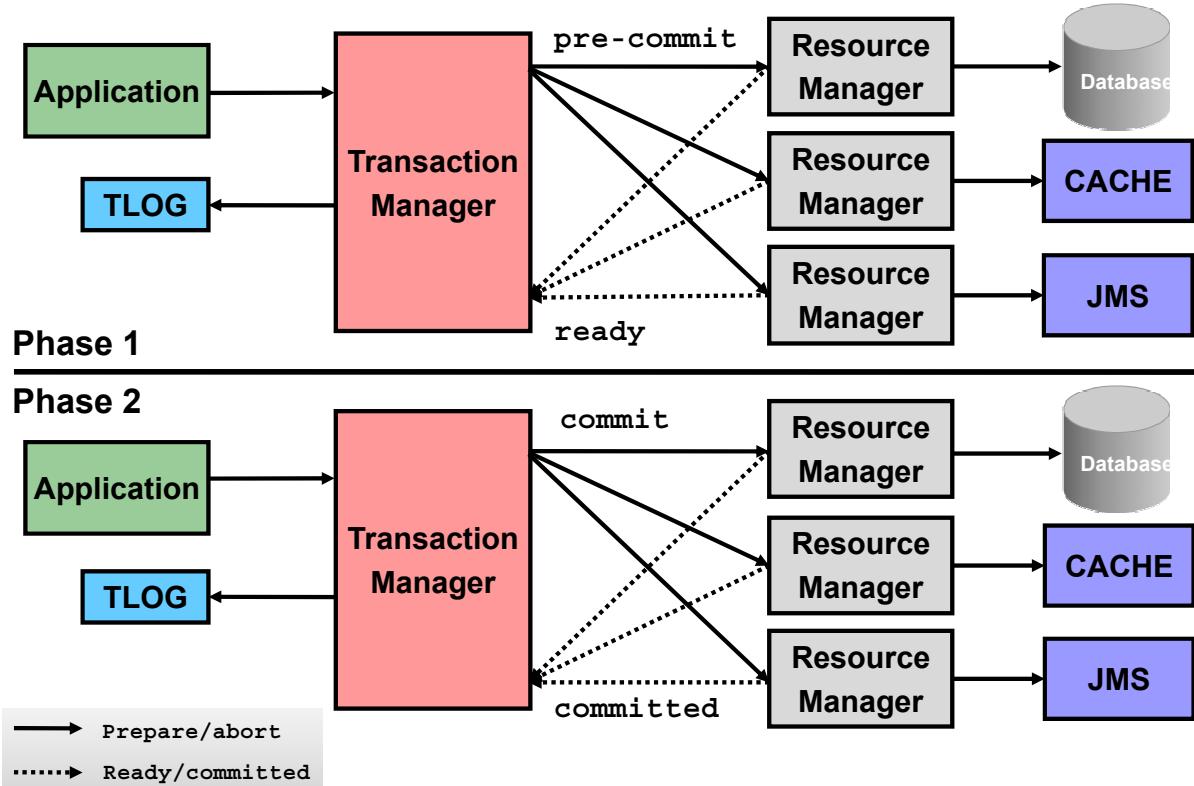


The two-phase commit protocol is used to coordinate the transaction.

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

A Successfully Committed 2PC



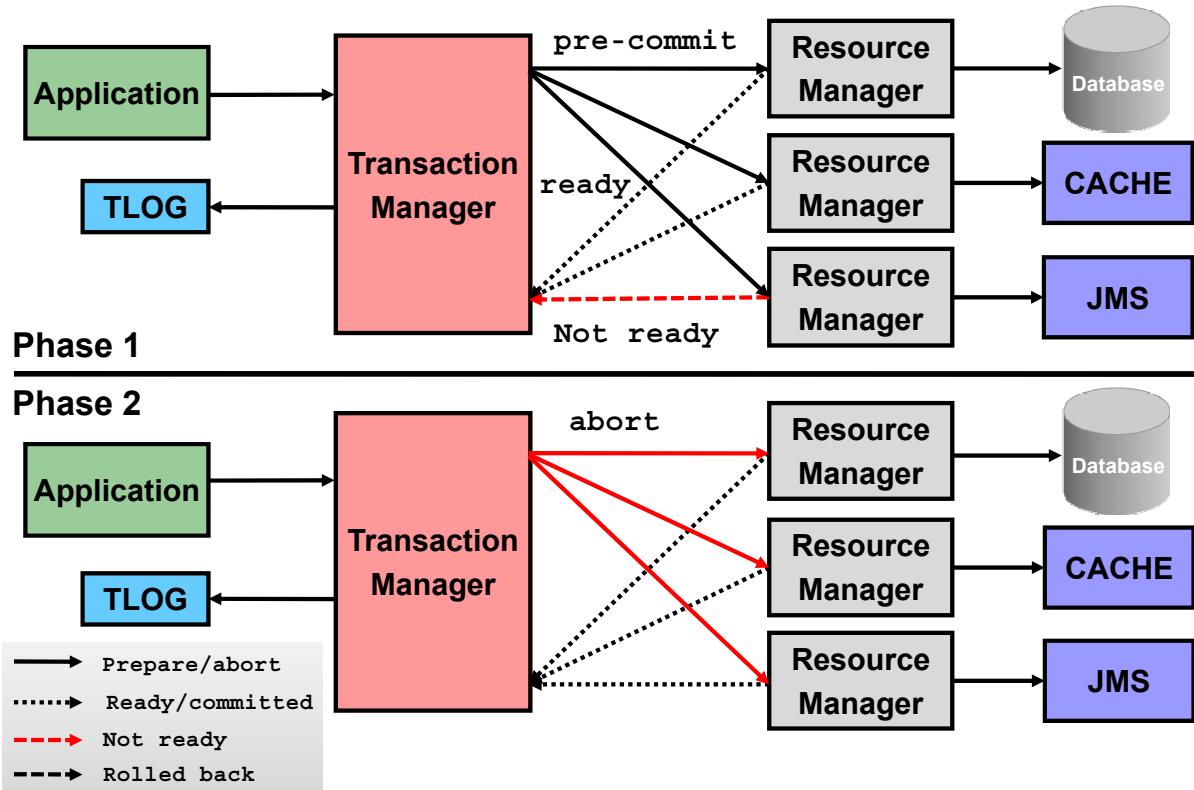
ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

A Successfully Committed 2PC

The first phase of the two-phase commit protocol is called the prepare phase. The required updates are recorded in a transaction log file, and the resource must indicate through a resource manager that it is ready to make the changes. Resources can either vote to commit the updates, or to roll back to the previous state.

A Successfully Aborted 2PC



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

A Successfully Committed 2PC

The first phase of the two-phase commit protocol is called the prepare phase. The required updates are recorded in a transaction log file, and the resource must indicate through a resource manager that it is ready to make the changes. Resources can either vote to commit the updates, or to roll back to the previous state.

Summary

In this lesson, you should have learned how to:

- Describe the basic concepts of transactions
- Implement a Coherence application that uses a transactional cache
- Implement a Coherence application that can participate in an XA transaction



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and ORACLE CORPORATION use only

Integrating a Data Source with Coherence

9

ORACLE®

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Identify and describe the configuration required to integrate a data source with Coherence
- Implement a CacheLoader and a CacheStore
- Implement a cache that is backed by JPA to a persistent store



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Persisting Data to Storage

- Overview
- ORM Integration
- Caching with Data Source Patterns

Data Source Integration Implementation

Coherence and JPA



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Persisting Data to Storage

- Coherence does not persist or load data to disk or database by default.
- Coherence can load data from, or load and persist data to, any source by implementing the CacheLoader or CacheStore interfaces.
- Coherence does not include its own Object Relational Mapping (ORM).
- Coherence supports transparent read-write caching of any data source, including databases, web services, packaged applications and file systems.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Persisting Data to Storage

In earlier lessons, you discussed writing data only to the memory of the Coherence JVMs. The data is persistent because every time you write an object into Coherence, it is backed up not only to another JVM, but also to another JVM on another physical server. Data is not persistently stored to disk because Coherence does not use the disk by default. However, you can integrate Coherence with the database, and save that data, and retrieve the data from a database. This is done by using the read-only CacheLoader if you only want to read data from the database. It can also be done by using a Coherence CacheStore that includes both reads and writes.

When you update the database, using `put()` or `putAll()`, or `EntryProcessors`, you can persist the data to a database. When you read from the cache using a `get()` (by a key), you can obtain data from a database if the object is not in the cache. You can write your own code to integrate Coherence with any back-end database, or even a mainframe.

Object Relational Mapping (O/RM) refers to the object being translated into relational database form. Coherence does not include O/RM by default. There are many solutions on the market to take care of O/RM, such as Oracle TopLink and Hibernate. You can integrate those solutions, or you can write your own Java Database Connectivity (JDBC) code.

Persisting Data to Storage (continued)

As of Oracle TopLink 11g, TopLink Essentials has been replaced with EclipseLink JPA. EclipseLink JPA implements JPA 1.0, and is responsible for delivering the persistence framework. Information on EclipseLink in TopLink 11g can be found at:
http://www.oracle.com/technology/products/ias/toplink/technical/tl11g_fov.htm#CHDIJJJJ.

Coherence supports transparent read-write caching of any data source, including databases, Web services, packaged applications, and file systems. However, databases is the most common use case. Effective caches must support both intensive read-only and read-write operations, and in the case of read-write operations, the cache and database must be kept fully synchronized. To accomplish this, Coherence supports read-through, write-through, refresh-ahead, and write-behind caching.

A CacheStore is an application-specific adapter used to connect a cache to an underlying data source. The CacheStore implementation accesses the data source via a data access mechanism such as Hibernate, JPA (Java Persistence Architecture), Java Data Objects (JDO), JDBC, and so on. A CacheStore understands how to build a Java object using data retrieved from the data source, map and write an object to the data source, and erase an object from the data source.

Both the data source connection strategy and the data-source-to-application-object-mapping information are specific to the data source schema, application class layout, and operating environment. Therefore, this mapping information must be provided by the application developer in the form of a CacheStore implementation.

O/RM Integration

- Coherence supports O/RM integrations:
 - Java Persistence Architecture (JPA)
 - Oracle TopLink 11g Grid
 - EclipseLink: JPA reference implementation
 - Hibernate
- Custom integrations are supported
 - Provides more control than O/RMs, such as:
 - Toplink
 - Hibernate
 - File system



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

O/RM Integration

Out-of-the-box O/RM integration includes the following:

- **Java Persistence Architecture (JPA):** JPA is an API for creating, removing, and querying across lightweight Java objects, and can be used both within a compliant EJB 3.0 Container or a standard Java SE 5 environment such as Coherence. JPA is a framework for mapping an object-oriented domain model to a traditional relational database. JPA solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions.
- **TopLink Grid:** Oracle's TopLink Grid 11g is an O/RM package for Java developers. It provides a powerful and flexible framework for storing Java objects in a relational database, or for converting Java objects to XML documents. It introduces seamless integration with Oracle Coherence, flexible Object-XML binding, and provides compatibility with Oracle WebLogic Server and Java frameworks, including Oracle Application Development Framework (ADF) 11g.

Oracle TopLink is often described as one of the most flexible and scalable O/RM libraries, and performs particularly well for read-intensive applications. EclipseLink offers all the essential functionality for O/R mapping through JPA. It also provides a number of other custom and advanced features for more sophisticated application usage.

O/RM Integration (continued)

- **EclipseLink** (also known as the Eclipse Persistence Service Project) is the JPA 2.0–reference implementation. EclipseLink is based on a previous version of Oracle TopLink. Oracle TopLink 11g enhances the industry-standard Java Persistence API (JPA) with advanced object-relational features for flexibility, performance, clustering, and scalability required for enterprise application deployments.
- JPA-enabled Data Grid provides seamless integration with Oracle Coherence, allowing applications to leverage in-memory data grid capabilities through the standard JPA programming model, and developers to leverage extreme performance without having to learn a new programming model.
- **Hibernate**: A popular, open source, object-relational persistence and query service for Java. It is commonly used by J2EE developers for generating tables for a database application, adding data to the tables, and retrieving, updating, and deleting table data.

The type of integration that you choose depends on your organizational needs and requirements. If your needs are fairly straightforward in terms of passing your objects to the database, using Hibernate or TopLink will probably work. You need not write any custom code. However, if you require a special algorithm, or have some special needs, you may need to write your own code.

Data Integration Patterns

There are five typically data integration patterns:

- **Cache-Aside:** The developer manages cache content manually, typically using AOP, or some crosscutting to cache data when first requested.
- **Read-Through***: get () operations first check the cache, and then “read-through” to a data source if not found. Note queries are only performed on cached data.
- **Refresh-Ahead***: Data is automatically and asynchronously refreshed for recently accessed cache data that hasn't expired.
- **Write-Through***: Data is written to the cache, and then synchronous writes to the backing data source.
- **Write-Behind**: Data is written to the cache, and then asynchronously persisted later.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Data Integration Patterns

There are five typical data integration/caching patterns:

- **Cache-Aside:** In cache-aside, it is the developers responsibility to manage the content of a cache. Typically a check is made on data requests, and the data manually loaded on cache misses. Often the logic is implemented using some sort of crosscutting behavior or Aspect-Oriented Programming (AOP).
- **Read-Through:** If the object is not found in the cache, the object is automatically retrieved from the back-end database using read-through. So, if a client application requests an object with a `get ()`, Coherence first looks for it in its near cache. If it is not there, it looks for it in the distributed cache. If it is not there, it goes to the database, gets it, puts it into the cache, and then returns it to the client. This works only with `get ()` by object ID. This does not work with queries. The querying that you did earlier in the course is done only to data that is already in the cache.
- **Refresh-Ahead:** If expiry is enabled, the data will have some common time to live, for example, the data expires after one hour. Coherence automatically reads the object from the database before it expires. This lessens the I/O to the database when you actually get the object.

Data Integration Patterns (continued)

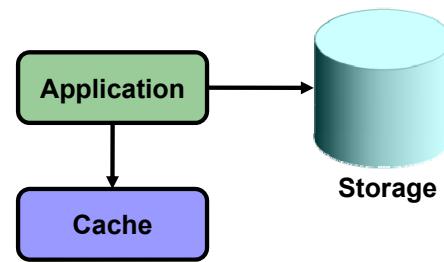
- **Write-Through:** This is a synchronous write. Every time you write to NamedCache, either with a `put()` or with an `EntryProcessor`, Coherence writes to the database synchronously. Coherence waits for the database write to succeed before returning control to the client. It is a guarantee that every time you write an object into the cache, the data will be written into the database. Write-through caching will be just as persistent as any other solution. However, it is not a solution for performance scalability because it depends on the database. The system will be only as fast as the database.
- **Write-Behind:** Is a very powerful feature of Coherence. In write-behind caching, persistence to database is asynchronous. When you write objects into the cache, the objects are persisted in Coherence because of Coherence backups. Later, there is a queue of transactions that then gets flushed asynchronously to the database. You can configure the time interval when the queue of transactions gets inserted into the database.

Cache-Aside Pattern

The Cache-Aside pattern requires the developer to manage the contents of the cache manually:

- Check the cache before reading from data source
- Put data into cache after reading from data source
- Evict or update cache when updating data source

The Cache-Aside pattern can be written as a core part of application logic, or it can be implemented using some form of crosscutting behavior such as AOP.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Cache-Aside Pattern

In the cache aside pattern, a traditional Java map is used to manage data. However, all the work of the data management is the responsibility of the developer. In cache-aside, a request for data happens via a data access object (DAO). If the data is not in the cache, some external mechanism is used to load it into a cache. Typically the cache aside pattern works via some sort of crosscutting behavior such as provided by AOP, EJB, or something similar. A cache miss is intercepted, the cache loaded, and the loaded data returned.

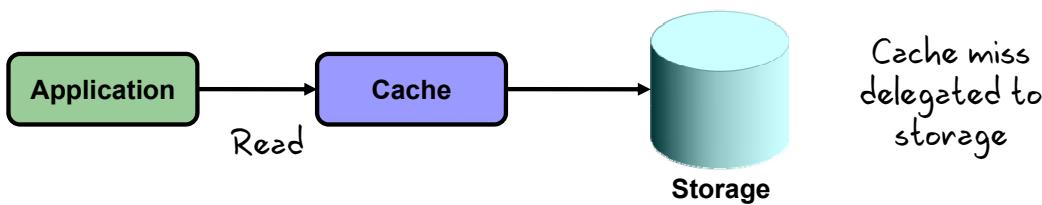
There are a variety of issues and concerns with cache-aside: support for threading, multiple database readers, if multiple instances of the same cache or data object is used, database concurrency, and so on.

Coherence doesn't support cache-aside directly because the pattern is almost exclusively written by the developer.

A concrete example of the AOP approach is TopLink Grid, which injects proxies into entities, so that relationships can be resolved by querying Coherence.

Read-Through Pattern

- Read-Through Pattern:
 - All data reads occur through cache
 - On cache miss, the cache loads data from a backing data source automatically, and no thread blocks on refreshes
- Coherence Read-Through:
 - If an object queried by key is not found:
 - Call the CacheLoader.load() to load data
 - Add the object to the distributed cache
 - Add the object to the local near cache, if it exists



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

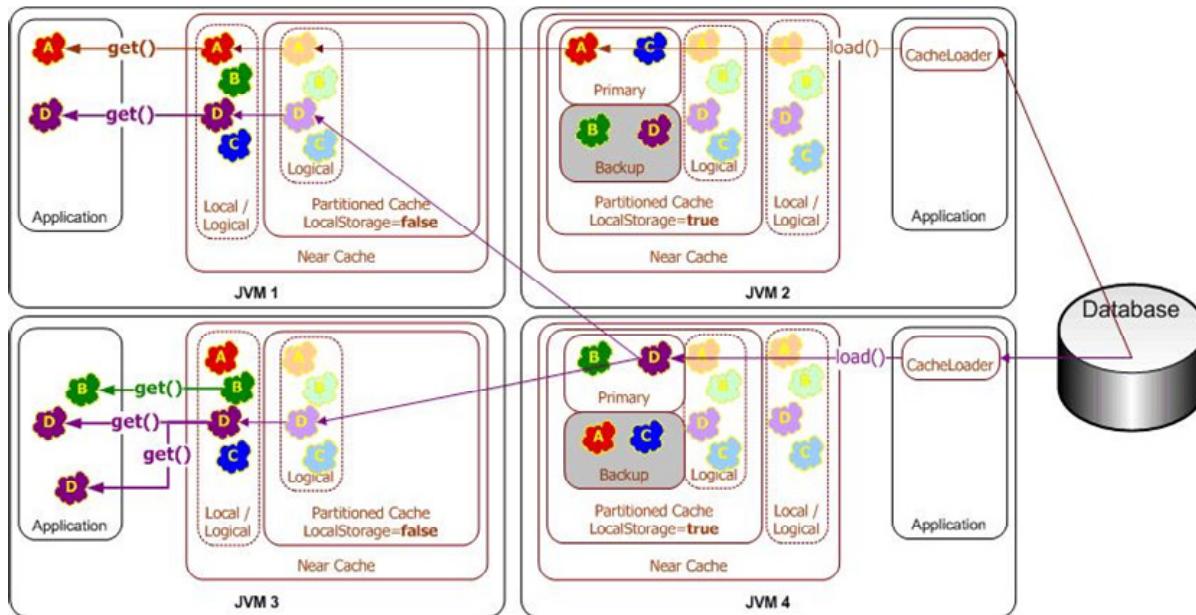
Read-Through Pattern

With the read-through pattern, when an application requests an entry from the cache, and the entry is not in the cache, the cache itself loads the state.

With Coherence, data requests are delegated to CacheLoader, which loads data from the underlying data source. If the data exists, the CacheLoader loads it, inserts it into the Coherence cache, and returns it to the requestor. This process is called read-through caching.

After data is in the cache, it can be queried by the query API, or QueryMap. Objects can expire with a time-based algorithm, or they can be evicted if the size limit is exceeded. If the object expires or is evicted, Coherence gets it from the database with read-through the next time it is retrieved by object ID.

Read-Through: Example



ORACLE®

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

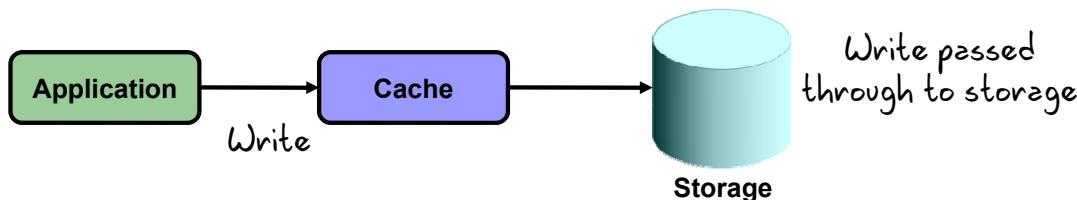
Read-Through Example

The diagram in the slide shows the workings of read-through caching.

- Start with object A in JMV1.
- When the client invokes a `get()`, Coherence first looks at the near cache if there is one configured.
- If the object is not in the near cache, Coherence goes to the distributed cache and retrieves the object from the distributed cache. If the object is not in the distributed cache, Coherence calls the `load()` method of CacheLoader. CacheLoader or CacheStore has a method called `load()` that is automatically invoked if you use the Hibernate or TopLink integration. This can also be a customized `load()` method. The `load()` method calls Hibernate or TopLink, or it calls your own JDBC code, for example, a `SELECT` statement in JDBC that retrieves the object from the database. Coherence places the object into the distributed cache, then into a near cache (if it is configured), and then the object is returned to the client.
- For object D, the `get()` method is called multiple times. The first `get()` retrieves the object from the database, because it is not in the cache yet. Subsequent gets retrieve the object either from the near cache or the distributed cache. This illustrates that after the object is in the cache, Coherence gets the object from the cache for the client and does not go to the database each time.

Write-Through Pattern

- Write-Through Pattern:
 - All data writes occur through cache
 - Updates to the data source are written
- Coherence Write-Through:
 - All the behaviors of read-through are supported
 - On update Coherence, via put () or an entry processor
 - Store the update in cache as required
 - **Synchronously** call CacheStore.store () to update the persistent store



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Through Pattern

The write-through pattern is the writer equivalent of the read-through pattern. On writes, the object is stored into the cache, then persisted to the backing storage.

With Coherence, writes requests are delegated to CacheStore, which extends the CacheLoader. CacheStore stores data synchronously into the underlying data source. This process is called write-through caching. Write-through has benefits and drawbacks.

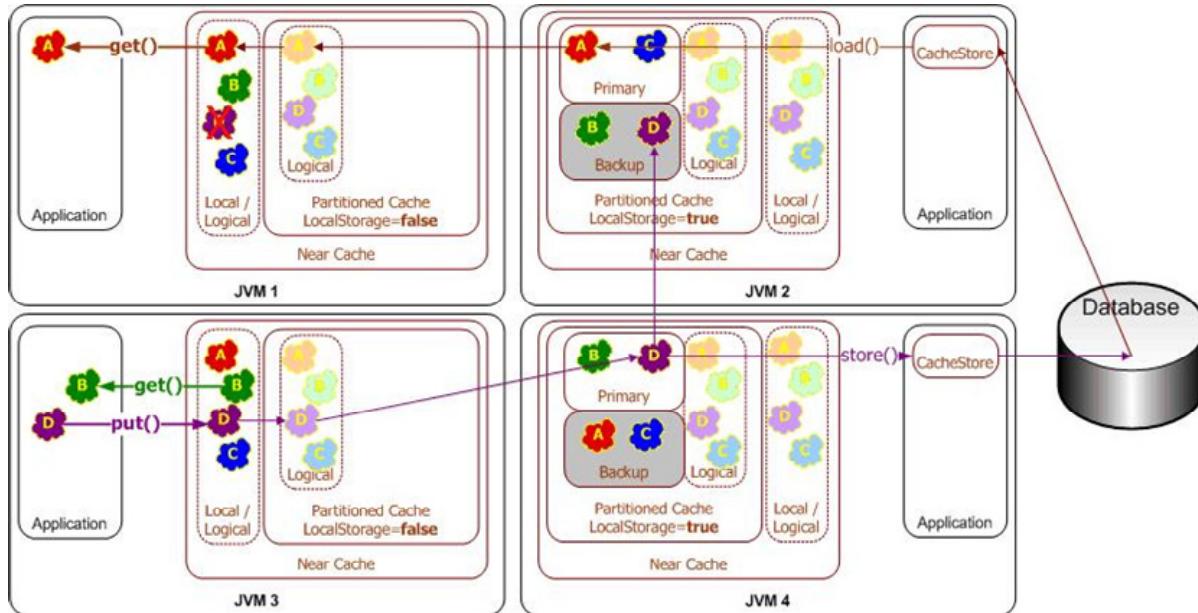
Advantages:

- A single point of access simplifies concurrent access.
- Write to cache can be rolled back upon data store error.
- The synchronous nature allows write-to-storage failures to be returned to the caller.

Disadvantages:

- Write performance can only scale as far as the database will allow.

Write-Through: Example



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

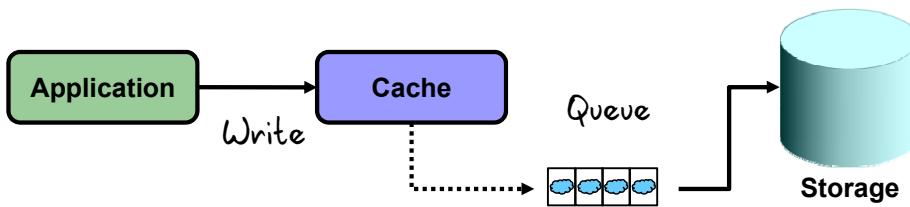
Write-Through: Example

The slide displays a simple diagram of how write-through works.

- At the bottom left, (JVM3) is object D that is represented in purple.
- When you insert the object into the cache, Coherence first updates the near cache, if there is one, and then it updates the distributed cache.
- A backup is taken, just as Coherence does normally, then Coherence calls the `store()` method of CacheStore, which actually writes the object into the database, usually with an `INSERT` or `UPDATE` in JDBC.
- Coherence waits for the `store()` method to return, and then the control is passed back to the client. This is *synchronous* and the client *waits* for the write to the database to complete before continuing.

Write-Behind Pattern

- Write-Behind Pattern:
 - All data writes occur through cache
 - Writes are queued for processing to a storage and persisted some time later, **asynchronously**
- Coherence Write-Behind:
 - Cache write are queued, and delegated to a CacheStore based on a configurable time out.
 - Multiple writes may be coalesced into a single batch write



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind Pattern

When working with applications with a high write frequency, most Coherence developers choose the write-behind strategy. With write-behind, writes are written to the cache normally, but backing writes are queued for delivery based on a write delay. Write-behind has a number of benefits, and some drawbacks.

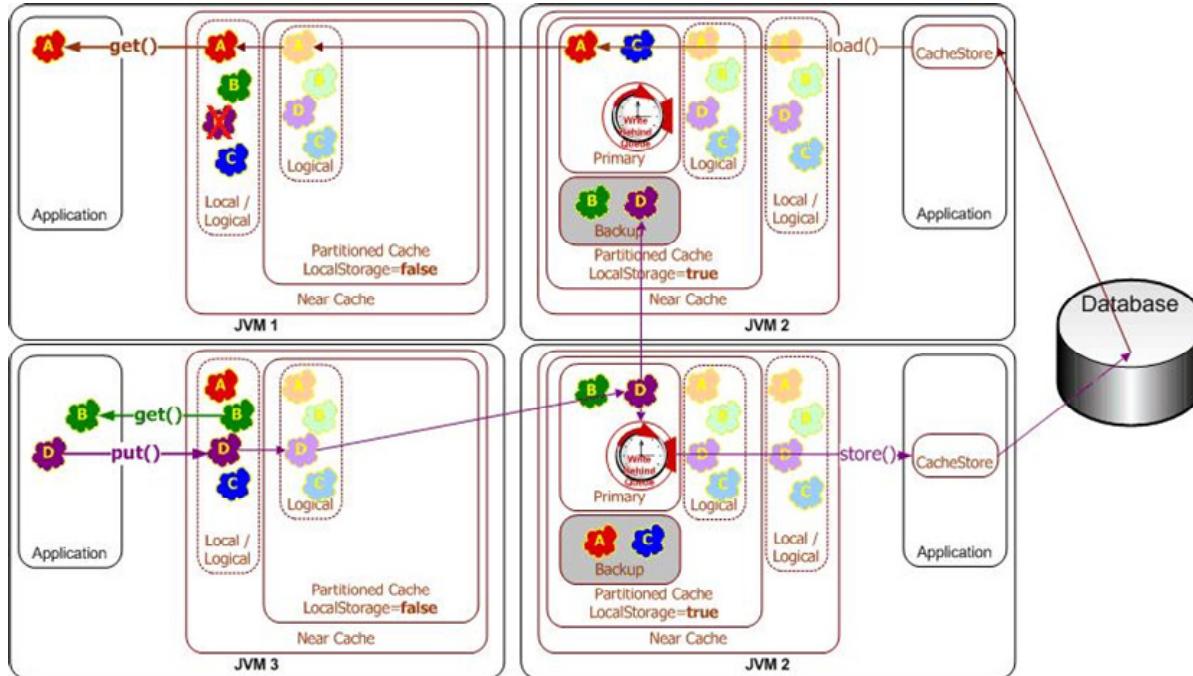
Advantages:

- Application response time and scalability are decoupled from data store, which is ideal for write-heavy applications.
- Multiple updates to the same cache entry are coalesced, reducing the load on the data store.
- Applications can continue to function upon data store failure.
- Loss of a single physical server will result in no data loss.

Disadvantages:

- In the event of multiple simultaneous server failures, some data in the write-behind queue may be lost.

Write-Behind: Example



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind: Example

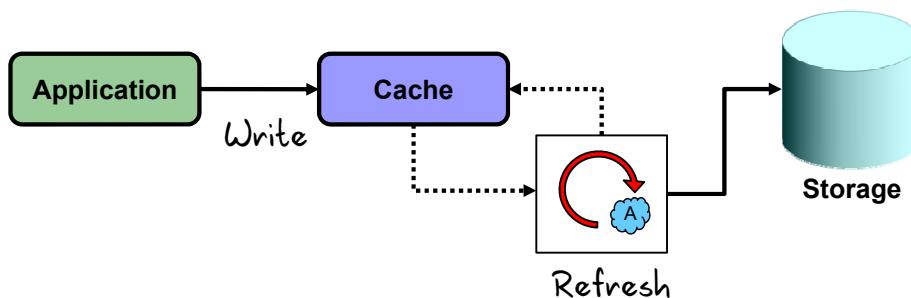
The slide depicts the way write-behind works. Look at the lower left corner of the slide (JVM3). When you execute a `put()` of object D into the cache, it updates the near cache, if there is one. It then updates the distributed cache, and it makes a backup. This is a synchronous operation as always in Coherence, and then on a time interval, the object is written into the database using the `store()` method of CacheStore.

After the object is written to the grid, control is immediately returned to the client. The write to the database is done at a later time within the time interval that you specify.

A CacheStore has both a `load()` and a `store()` method. If there is a `get()`, it calls the `load()` method of CacheStore. If there is a `put()`, it calls the `store()` method.

Refresh-Ahead Pattern

- Refresh-Ahead Pattern:
 - Often read, but frequently changing data can be automatically and asynchronously reloaded
 - Cache misses behave the same as read-through
- Coherence Refresh-Ahead:
 - Data is configured with an expiration and read interval
 - Data read close to expiration is refreshed



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Refresh-Ahead Pattern

In the refresh-ahead scenario, Coherence allows a developer to configure a cache to automatically and asynchronously reload (refresh) any recently accessed cache entry from the cache loader prior to its expiration. The result is that once a frequently accessed entry has entered the cache, the application will not feel the impact of a read against a potentially slow cache store when the entry is reloaded due to expiration. The asynchronous refresh is only triggered when an object that is sufficiently close to its expiration time is accessed. If the object is accessed after its expiration time, Coherence will perform a synchronous read from the cache store to refresh its value.

Agenda

Persisting Data to Storage

Data Source Integration Implementation

- CacheLoader and CacheStore
- Read-Through Caching
- Refresh-Ahead Caching
- Write-Through Caching
- Write-Behind Caching

Coherence and JPA



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

The CacheLoader Interface

The CacheLoader Interface is used for read-only solutions:

```
package com.tangosol.net.cache;
public interface CacheLoader {

    public abstract Object load(Object key);
    Implementations called to return the requested object based on the provided key.

    public abstract Map loadAll(Collection keys);
    Implementations called to return a Map of objects based on the provided collection of keys
    There must be a 1:1 mapping of keys to values even if the underlying lookup returns null.
}
```

CacheLoader.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

The CacheLoader Interface

CacheLoader and CacheStore, often referred to generically as a CacheStore, are generic interfaces that form the basis of data storage solutions.

There are no restrictions as to the type of data store that can be used, for example databases, Web services, mainframes, file systems, remote clusters, or any other interface that can be accessed from a Java class, can be used as a data source.

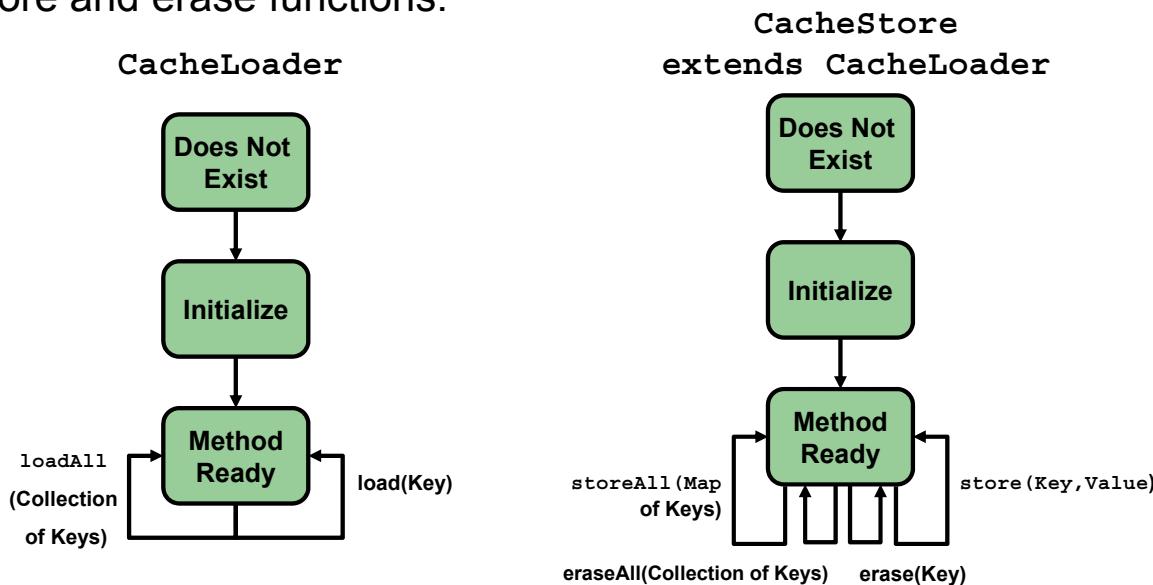
The CacheLoader interface defines two methods:

- **load:** Called to return an object for the specified key. Note that there is an expectation that the cache-loader understands the format of the provided key object.
- **loadAll:** Called to return a Map, often a `HashMap`, of the providing collection. Note that even if there is no object for a given key, a key-to-null mapping must be maintained.

CacheLoader and CacheStore Lifecycle

CacheLoader has an initialize, then process lifecycle.

CacheStore extends the CacheLoader lifecycle, and adds store and erase functions.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

CacheLoader and CacheStore Lifecycle

Both CacheLoader and CacheStore-based classes have a simple, well-defined lifecycle.

As the diagram shows, both go from Does Not Exist, through Initialization, to Method Ready. The core point of the Initialize state is that an implementation may perform custom initialization, most often taking arguments and data from the configuration.

CacheLoader**s** and Initialization

CacheLoader**s** constructors can take parameters using the `init-params` element.

```
<class-name> . . . </class-name>
<init-params>
    <init-param>
        <param-type>java.lang.Type</param-type>
        <param-value>corresponding value</param-value>
    </init-param>
</init-params>

<class-name> . . . CustomCacheLoader</class-name>
<init-params>
    <init-param>
        <param-type>java.lang.String</param-type>
        <param-value>value</param-value>
    </init-param>
</init-params>

public class CustomerCacheLoader implements CacheLoader {
    public CustomerCacheLoader (String initialValue) { . . . }
    .
}
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

CacheLoader**s** and Initialization

CacheLoaders can be initialized by creating a set of one or more `init-param` elements within an `init-params` element. Each individual `init-param` corresponds to a parameter in the constructor of the class. In the example shown, a single `init-param` is defined of type `String`, which corresponds in type and order to the constructor of the custom cache loader.

Arguments can be of type `java.lang.[Boolean, Float, Double, Long, String]`.

CacheLoader: Example

```
public class CustomerCacheLoader implements CacheLoader {
    private Connection m_conn = null;
    ...
    protected void configureConnection() {
        m_conn =
            DriverManager.getConnection(...);
        stmt = m_conn.createStatement(
            "SELECT col1, col2, col3 FROM mytable WHERE id = ?");
    }

    public Object load (Object okey) {
        if ( m_conn == null) configureConnection();
        stmt.setString(1, oKey.toString());
        ResultSet rs = stmt.executeQuery();
        ...
        return new Customer(. . .);
    }
    public Map loadAll(Collection collection) {
        ...
    }
}
```

CustomerCacheLoader.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

CacheLoader: Example

The slide gives you an example of what the code looks like if you write your own JDBC CacheLoader. This is not complete code, but it presents the bulk of the required code.

- Custom cache loaders must implement CacheLoader or CacheStore. CacheStore requires both the `load` and `store` methods. This example connects an Oracle Database, the specifics of which are beyond the scope of this course.
- The `load` implementation is provided a single argument representing a key object, which is used for a JDBC lookup and represents the original `get()` request parameter.

Configuring CacheLoaders

CacheLoaders are configured in the cache-store-scheme element of a backing map scheme.

```
<distributed-scheme>
  <scheme-name>example-distributed-with-cacheloader</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <read-write-backing-map-scheme>
      <cachestore-scheme>
        <class-scheme>
          <class-name>
            . . . CustomerCacheLoader</class-name>
          </class-scheme>
        </cachestore-scheme>
      </read-write-backing-map-scheme>
    </backing-map-scheme>
  </distributed-scheme>
```

coherence-cache-config.xml



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuring CacheLoaders

The slide shows how you can configure a cache store or a cache loader in the coherence-cache-config.xml file in the near distributed scheme. You have a backing-map-scheme for each cache.

You have an attribute called <backing-map-scheme>, under which there is a <read-write-backing-map-scheme>. Under this there is a <cachestore-scheme>, under which is <class-scheme>, where you specify the class name of your CacheLoader. This can either be your own JDBC code as you saw in the earlier slide or, JPA, TopLink JPA, or Hibernate. In this example, you use com.coherence.myCacheLoader, but you can put in JPA, TopLink JPA, or Hibernate under the class scheme.

The CacheStore Interface

The CacheStore Interface is used for read/write solutions.

```
package com.tangosol.net.cache;
public interface CacheStore extends CacheLoader {
    public abstract void store(Object key, Object value);
    Implementations called to store the provided value into the persistent store with the given key
    public abstract void storeAll(Map map);
    Implementations called to store the provided Map of key/value pairs into the persistent store
    public abstract void erase(Object key);
    Remove the object with the provided key from the persistent store if present
    public abstract void eraseAll(Collection collection);
    Remove the collection of keys from the persistent store if present
}
```

CacheStore.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

The CacheStore Interface

The CacheStore interface defines four additional methods beyond those inherited from CacheLoader:

- **store:** Store (create new) or update the provided key value pair.

- **storeAll:** Store the specified values under the specified keys in the underlying store. This method is intended to support both key/value creation and value update for the specified keys.

By default, you should implement this, and just call `store()` for all the objects that are passed into `storeAll()`. `storeAll()` takes a collection of objects and you can call `store` repeatedly, or you can write the bulkload code in JDBC. It is more efficient to do bulk loading than individual loading into the database or individual storing.

- **erase:** Remove the specified key from the underlying store, if present.

- **eraseAll:** Remove the specified keys from the underlying store, if present

CacheStore: Store Example

CacheStore.store:

- Takes key and value objects
- Represents insert or update

```
public class CustomerCacheStore implements CacheStore {
    ...
    public void store(Object oKey, Object oValue) {
        Connection con = getConnection();
        String sSQL;
        if (load(oKey) != null)
            sSQL = "UPDATE TABLE+ " SET . . . where id = ?";
        } else {
            sSQL = "INSERT INTO TABLE (id, . . . ) VALUES (?,?)";
        }
        try {
            ...
        } catch (SQLException e) {
            throw ensureRuntimeException(e, "Store failed: key=" + oKey);
        }
    }
}
```

CustomerCacheStore.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

CacheStore: Store Example

The cache store.store method performs two specific functions: load or update. In the example shown, the load method is used to determine if the object exists, and then does an update. If the object does not exist, then an insert is performed instead.

Note that the use of load can be eliminated completely by using a statement such as:

```
MERGE INTO accounts a USING
  (SELECT :accountId account_id, :name name,
   :balance balance FROM dual) a_new
  ON (a.account_id = a_new.account_id)
WHEN matched THEN
  UPDATE
  SET a.name = a_new.name,
      a.balance = a_new.balance
WHEN NOT matched THEN
  INSERT (a.account_id, a.name, a.balance)
  VALUES (a_new.account_id, a_new.name, a_new.balance);
```

CacheStore: Load Example

`CacheStore.load:`

- Takes a key object
- Returns null or a populated result

```
public class CustomCacheStore implements CacheStore {

    public Object load(Object key) {
        String select = "select ID, . . . from TABLE"+
                       " WHERE ID =" + key.toString();
        Statement stmt = getStatement();
        ResultSet rs = stmt.executeQuery(select);
        CustomObject customObject = . . . ← Find the object in
                                      backing store
        if (rs.next()) {
            int id = rs.getInt("ID"); ← Construct an
            . . . appropriate return
            customObject.setXXX(value); instance
        }
        return customObject; ← Return the instance,
                             or null if not found
    }
}
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

CacheStore: Load Example

`CacheStore.load` implementations are relatively simple. This JDBC-like example shows loading a custom object. Note that the details of JDBC are beyond the scope of this class and are mostly ignored.

The load example must:

- Examine the provided key, potentially decomposing into its core parts if complex, find the object using the backing store, and construct a representation.
- On a lookup miss, the method should return `null`.

CacheStore: *All Example

```

public class CustomerCacheStore implements CacheStore {

    . . .

    public void storeAll(Map mapEntries) {
        Iterator it = mapEntries.entrySet().iterator();
        while (it.hasNext()) {
            Map.Entry pair = (Map.Entry)it.next();
            store(pair.getKey(),pair.getValue());
        }
    }
    . . .

    public Map loadAll(Collection colKeys) {
        Map results = new HashMap();
        for (Object key: colKeys ) {
            Object value = load(key);
            if ( value != null) results.put(key,value);
        }
        return (results.isEmpty()? null: results);
    }
}

```

Simple implementation, iterate over all elements and call store

Iterate over all keys, calling load for each, and adding into map on hit



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

CacheStore: *All Example

The `storeAll` and `loadAll` methods of a cache store can be implemented via iterators. In the example provided:

- `storeAll`: This method is called with a set of map entries, each a key/value pair. In the example, the code iterates over the set of pairs, extracts each, and calls the `store` method to perform the actual work. Note that depending on the implementation of `store`, this may result in multiple database operations, which could be batched together.
- `loadAll`: This is similar to `storeAll`, using the underlying `load` method. However, the method must return something which implements the `Map` interface. Each key is passed the `load` method, then added to the map, if found. If the map is not empty, it is returned, else `null` is reported.

CacheStore Architecture

- All storage JVMs that store data for the NamedCache will instantiate instances of the CacheStore for that NamedCache.
- Data is written to the database from the storage JVM that owns the entry.
- The number of database connections depends on how the CacheStore is written.
 - If each CacheStore opens a database connection, the number of connections is equal to the number of storage JVMs.
 - A CacheStore can use a connection pool from an application server.
- Ordering of transactions with write-behind is not guaranteed, however execution of transactions is.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

CacheStore Architecture

The cache store is architected in such a way that all the storage JVMs that store data from the Namedcache instantiate instances of the CacheStore for that NamedCache. So, if you have 100 storage JVMs, there are 100 instances of that CacheStore running on each of the storage JVMs. Data is written to the database from the storage JVM that owns the entry.

The number of database connections depends on how the CacheStore is written. If each CacheStore opens a database connection, the number of connections to the database is equal to the number of storage JVMs. In your CacheStore, you can use connection pooling, either from JDBC, or from your application server, or connection pooling that you wrote yourself.

Ordering of transactions is not guaranteed with write-behind. If you put an object, for example, object ID 1 followed by a put of object ID 2, they may be executed out of order. Typically, these objects are stored in separate storage JVMs. In fact, one storage JVM may store the data before the other one. However, the actual execution of transactions is guaranteed. Coherence always executes the transaction even if the storage JVM or server fails.

Refresh-Ahead Caching

- Automatically obtains an object from the database before it expires
- Requires the cache must be configured to expire
- Refresh timing is controlled by `<refresh-ahead-factor>`, a percentage of the entry's expiry (between 0 and 1)

Example: Expiry is set to 60 seconds and `<refresh-ahead-factor>` is set to 0.75. If an object is more than 60 seconds old, Coherence executes a synchronous query to refresh the object.

If a query is executed on an object that is more than 45s (60×0.75) but less than 60 seconds old, the current value in the cache is returned, and then Coherence asynchronously refreshes the object from the database.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Refresh-Ahead Caching

Refresh-ahead automatically obtains an object from the database before it expires. However, this requires that you configure expiry. Therefore, you have to configure the cache to have an object to expire after some time. The other attribute is the `<refresh-ahead-factor>` that has a value between 0 and 1. `<refresh-ahead-factor>` is the percentage of the entry's expiry time. For example, if expiry is set to 60 seconds, and `<refresh-ahead-factor>` is set to 0.5, then Coherence will behave as follows:

- If an object is more than 60 seconds old, Coherence executes a synchronous query to refresh the object. That is, if the client invokes a `get()` for an object ID and that particular object is in the cache for more than 60 seconds, Coherence automatically has it expired. This means that the object is no longer in the cache and Coherence gets the object from storage.
- If the object is more than 30 seconds old, which is 60 second times the refresh ahead factor 0.5, but less than 60 seconds, the current value in the cache is returned, and then Coherence asynchronously gets the object from the database and puts the new copy into the cache. So the next time, you retrieve it, it is already in the cache.

Refresh-Ahead Caching (continued)

Refresh-ahead caching is very useful if the objects are being queried by a large number of users. This keeps the objects fresh in the cache with a fresh value. When users invoke a `get()`, they do not have to wait for the database I/O. The objects are already in the cache with a fresh copy. This improves the scaling and performance of your applications.

With refresh-ahead caching, Coherence allows a developer to configure the cache to automatically and asynchronously reload (refresh) any recently accessed cache entry from the cache loader before its expiration. The result is that after a frequently accessed entry has entered the cache, the application will not feel the impact of a read against a potentially slow cache store when the entry is reloaded due to expiration.

The refresh-ahead time is configured as a percentage of the entry's expiration time. For example, if specified as 0.75, an entry with a one-minute expiration time that is accessed within 15 seconds of its expiration will be scheduled for an asynchronous reload from the cache store.

Refresh-Ahead Configuration

```
<distributed-scheme>
  <scheme-name>example-jdbc-cacheloader</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <read-write-backing-map-scheme>
      <refresh-ahead-factor>0.75</refresh-ahead-factor>
      <cachestore-scheme>
        <class-scheme>
          <class name>... CustomerCacheLoader</class-name>
        </class-scheme>
      </cachestore-scheme>
      <internal-cache-scheme>
        <local-scheme>
          <scheme-ref>example-expiring-cache</scheme-ref>
        </local-scheme>
      </internal-cache-scheme>
    </read-write-backing-map-scheme>
  </backing-map-scheme>
  <local-scheme>
    <scheme-name>example-expiring-cache</scheme-name>
    <expiry-delay>60s</expiry-delay>
  </local-scheme>
```

coherence-cache-config.xml



The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red background.

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Refresh-Ahead Configuration

The slide shows how to configure refresh-ahead factor under the read-write-backing-map scheme.

You can configure a refresh-ahead factor. In this example, you set the refresh-ahead factor to 0.75. This factor has to be a value greater than 0 and less than 1. As in the earlier slide, you configure the name of your cacheloader as the name of the class.

At the bottom of the slide, you see how to configure expiry. Each distributed cache is actually backed by a local cache. Each storage JVM stores the data in the cache called local cache. In the local scheme, you can define expiry, and this is the expiry that is defined on each of the storage JVMs. You can also specify a scheme name.

You configure expiry delay, which, in this example, is 60s or 60 seconds. You can also define 60m for 60 minutes, 60h for 60 hours, 60d for 60 days or any time delay interval that you want. In this example, you specified an expiry of 60 seconds with a refresh-ahead factor of 0.75, so the object expires after being in the cache for 60 seconds.

You configure refresh-ahead parameters in the `coherence-cache-config.xml` file. Using the Java Management Extension (JMX) capability of Coherence, you can change the refresh-ahead factor and the expiry delay dynamically. You can configure the XML file, or you can also change the refresh-ahead parameter while the system is running.

Write-Through

Write-Through Caching:

- Synchronously call CacheStore.store [All] to update the persistent store as a result of a put [All] () or an EntryProcessor update.
- Performance is limited to the performance of the persistent store.
- Exceptions can optionally be thrown back to the calling application.

Write through is for applications that must know writes were successful.

Write though is NOT for applications with heavy write throughput requirements.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Through

If write-through is enabled, and an object is updated, Coherence automatically calls the CacheStore.store [All] () method defined in the backing map to persist the data.

Write-thought operations are synchronous, so the client blocks until the operation completes. The object can be updated through a put [All] () or through an EntryProcessor

The advantage of this is that every object that you write into the cache will be persisted to the backing store. However, the performance is limited to the performance of the backing store. If a large number of writes occur, then application performance may degrade to the performance of the backing store.

If something fails and an exception occurs, CacheStore throws an exception while persisting data to the database. If you write your own JDBC code, it should include exception handling. Optionally, you can throw the exception back to the client, so the client receives the exception as well, and knows that the write was not successful.

Write-Behind

- Each write to the cache is written to a database at a later time, so queues are in memory, but can optionally overflow to disk.
- The time interval is configurable (from 1 millisecond to hours).
- This significantly improves write-scaling and improves the response time to the user, so that the application does not wait for the database.
- If CacheStore throws an exception, the operation can be requeued and retried automatically.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind

Write-behind is a very powerful capability of Coherence that is used extensively. This provides for massive scaling and very high performance. Even though objects are being written to the database, users see very fast performance; it is as if the data is being written to memory. So, with write-behind, each write to the cache is written to the database at a later time.

Coherence maintains a queue of all the write transactions that execute in the grid, and it then flushes that queue to the database. The queues are in memory and also backed up. Even if a storage JVM fails, it is guaranteed that the write-behind transaction will be executed.

One of the things that you can do with write-behind is to allow for database outages. If the DBA wants to take the database down for a couple of hours, you can actually configure Coherence with an overflow to disk. The write transactions in Coherence write to the memory queue first. If it exceeds the size that you specified, you overflow that to disk files. This takes care of rather long database outages.

The time interval of when the queue gets flushed to the database is configurable, anywhere from 1 millisecond to hours. Thus you can have a write-behind after one minute, five minutes, or five hours. The write-behind time interval can be dynamically configured. Write-behind

Write-Behind (continued)

basically improves write-scaling and improves response time to the user. Applications need not wait for database I/O. One other powerful feature of write-behind is if CacheStore does throw an exception, and if the database is down, Coherence can automatically retry that transaction. Coherence automatically retries that transaction until the operation succeeds. The client does not have to include any exception handling code, exceptions are handled completely on the server.

Write-Behind

- Write-behind allows for Coherence to survive database outages, so that failed transactions can be requeued using <write-requeue-threshold>.
- Queues are backed up and survive Coherence server crashes, so that every write transaction is guaranteed to be executed.
- Each object update is written at most once.
 - If multiple updates to the same object occur during the write-behind interval, only the most recent update goes to the database.
 - This decreases the load on the database.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind (continued)

Write-behind allows for Coherence to survive database outages. Transactions are automatically queued and retried when the database is up again. Queues are backed up and survive Coherence server crashes. Every write transaction that goes into Coherence is guaranteed to be executed, even if the storage JVM or an entire server fails. This is a very important high-availability feature.

Each object is written only once. If there are multiple updates to the same object by different clients, Coherence writes that object only once to the database.

For example, you use write-behind for session management in a Web application, and your write interval is 10 minutes. The user has updated the session information five times during that 10 minutes. However, Coherence writes only the most recent copy of that session-related object to the database. This reduces the number of write I/Os to the database. The object is written only once after the time interval rather than being written five times.

Not only does write-behind reduce the response time to the user, but it also reduces the database I/O. It can enormously improve the scaling of your system. Database administrators often like this feature because it decreases the load on the database.

Write-Behind Configuration

Write-Behind Settings:

- Are sub elements of `<read-write-backing-map>`
- Include:
 - `<write-delay>`: Specifies the time interval for a write-behind queue to defer asynchronous writes.
 - A value of 0 (default) means synchronous with write-behind disabled.
 - Non-zero means write-behind enabled based on specified delay and units, for example, 1000ms or 10m.
 - `<write-requeue-threshold>` element: Specifies the maximum size of the write-behind queue for which failed write operations are requeued. If a value is set, and a CacheStore `store()` throws an exception, Coherence retries the transaction.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind Configuration

In the `cache config` file, when you configure write-behind, there is a `read-write-backing-map` section that gets updated in the Coherence `cache-config.xml` file. There is also a `write-delay` element set in the XML configuration. It is in this element that you set either write-through or write-behind. If you set the element to 0, write-through is used and the database write is synchronous.

If you set the element to other values, such as 60 seconds or 10m (10 minutes), write-behind is implemented. There is also a `write-requeue-threshold` element. This is the maximum number of transactions to requeue. For example, if the database is down, Coherence tries the transaction in write-behind mode. It gets an exception, because the database is down. It then automatically puts the transaction back into the queue and retries it.

Because the `write-requeue-threshold` element configures a maximum number of transactions to put in the queue, you will not run out of memory, and you have a limit on the number of transactions that go into the write-behind queue.

Write-Behind Configuration

Write-Behind Settings:

- `<write-batch-factor>` element: Used to calculate the "soft-ripe" time for write-behind queue entries
 - Must be > 0 and < 1
 - Default is zero, no write-batch-factor
 - Sets "soft-ripe" time = $(1.0 - \text{writeBatchFactor}) \times \text{WriteDelay}$

Example: `write-delay = 60s, write-batch-factor = 0.4`

$$(1.0 - 0.4) \times 60 = \text{soft-ripe time of 36 seconds}$$

After 60 seconds, Coherence writes all ripe (60 seconds old) and soft-ripe (36 seconds old) entries from the queue to the backing store.

A queue entry is considered to be "ripe" for a write operation if it has been in the write-behind queue for no less than the write-delay interval. The "soft-ripe" time is the point in time prior to the actual ripe time after which an entry will be included in a batched asynchronous write operation.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind Configuration (continued)

The next element in the `cache-config.xml` file is the `<write-batch-factor>`. The `storeAll()` method stores a group of objects, and the batch factor or the size of the batch is actually sent to the `storeAll()` method. The `write-batch-factor` must be set to a value greater than 0 but less than 1. The soft-ripe time that Coherence uses has a formula: "Soft ripe time = $(1.0 - \text{writeBatchFactor}) \times \text{WriteDelay}$ ".

For example, if you have a write-delay of 60 seconds, and a write-batch-factor of 0.4, you have $(1.0 - 0.4)$ times 60, which is equal to 36. This means that your soft-ripe time is 36 seconds. This means that after 60 seconds, Coherence writes all ripe (60 second old transactions) and soft-ripe entries from the queue into the database. Any object that is over 30 seconds old, which is the write factor, is written to the database. If this element is not set, the default is 0, which means that there is no write-batch factor. So every transaction is individually written to the database using `store()`. That is not very efficient, so if you do have it set to 0, remember that the write performance of the database would not be as good as it could be.

Write-Behind Configuration

```
<distributed-scheme>
  <scheme-name>example-jdbc-cachestore</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <read-write-backing-map-scheme>
      <!-- 30 second write behind. Set to zero for write-through -->
      <write-delay>30s</write-delay>
      <write-requeue-threshold>1000000</write-requeue-threshold>
      <write-batch-factor>0.8</write-batch-factor>
    <cachestore-scheme>
      <class-scheme>
        <class-name>com.coherence.myCacheStore</class-name>
      </class-scheme>
    </cachestore-scheme>
  </read-write-backing-map-scheme>
</backing-map-scheme>
</distributed-scheme>
```

coherence-cache-config.xml



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind Configuration (continued)

The slide shows an example of the configuration in the `cache-config.xml` file. You set a `write-delay` of 30 seconds, which means a write-behind queue of 30 seconds. There is a `write-requeue-threshold` of 1 million transactions and a `write-batch-factor` of 0.8.

What Happens If a Write-Behind Transaction Fails?

- If `store()` or `storeAll()` throws an exception, the exception is logged and the transaction is requeued and retried, up to the `<write-requeue-threshold>`.
- The transaction is retried until it succeeds, or until the threshold is exceeded.
- Exception handling in your CacheStore code is very important. You may want to ignore certain exceptions, or handle them in a special way. Consider constraint violations (for example, duplicate primary key): you may want to do a `DELETE + INSERT`, rather than throwing an exception.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What Happens If a Write-Behind Transaction Fails?

Consider a scenario in which a write-behind transaction fails. For example, if an `INSERT` statement is sent to the database and it fails, `store()` or `storeAll()` throws an exception. The exception is logged in the Coherence logs, and the transaction is requeued and retried.

The maximum number of transactions that are put into the queue or requeued is determined by the write-requeue threshold. The transaction is retried until it either succeeds, or the threshold is exceeded. This happens constantly and there is no limit on the number of retries.

Exception handling in your CacheStore is very important. CacheStore should not throw an exception unless you want Coherence to retry the transactions. For example, you may want to ignore certain exceptions, or handle them in a special way. Suppose that you have a referential integrity violation. In the database, you may want to only log that error rather than throwing an exception back to Coherence because, in the case of a constraint violation, the transaction will *never* succeed, and Coherence will retry *forever*.

Write-Behind Caveats

- Write-behind is very powerful, but you must be careful.
- The database transaction should not fail, that is, no constraint violations, referential integrity errors, and so on.
- Customers typically have database tables dedicated to Coherence write-behind without referential integrity constraints.
- Transaction must be idempotent:
 - If a Coherence server fails, a transaction may be executed twice during the failover.
 - The transaction must be re-executable and produce the same results.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind Caveats

You need to be careful when you implement write-behind. The database transaction should not fail. The only reason for a transaction in CacheStore to fail is a database failure.

Remember that Coherence constantly retries those failed transactions. If it does fail, then you should have some kind of exception handling logic to handle that failure. For example, you typically have database tables dedicated to Coherence write-behind without referential integrity constraints.

Unless you have written exception handling code, you would not want the transaction to fail for reasons other than the database being down.

The transaction must be idempotent. If a Coherence server fails, a transaction might be executed twice during the failover. This potential duplicate is because during a JVM crash, Coherence does not know if the attempt at the transaction was done or not. Idempotent means that a transaction must be re-executable and must produce the same results.

Idempotency

- An example of an idempotent transaction:

```
UPDATE costs SET price = 99.05 WHERE id = 100;
```

- The above transaction always produces the same results, even if executed repeatedly.
- An example of a non-idempotent transaction:

```
UPDATE costs SET price = price * 1.10 WHERE id = 100;
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Idempotency

Idempotency means that you can execute the same transaction and have the same results. An example of an idempotent transaction is in the slide:

```
UPDATE costs SET price = 99.05 WHERE id = 100;
```

The transaction in the slide always produces the same results, even if it is executed repeatedly.

An example of a non-idempotent transaction is an update in the price, the price \times 1.10. Every time you execute the transaction, it raises the price by .10 or 10 %. You should not have this type of transactions in your cache store. If you do, and the Coherence JVM or the server fails, it is possible that the transaction may be executed multiple times, in which case you will not have the desired value in the database.

All CacheStore operations should be designed to be idempotent (that is, repeatable without unwanted side-effects). For write-through and write-behind caches, this allows Coherence to provide low-cost fault tolerance for partial updates by retrying the database portion of a cache update during failover processing. For write-behind caching, idempotency also allows Coherence to combine multiple cache updates into a single CacheStore invocation without affecting data integrity.

Idempotency (continued)

Applications that have a requirement for write-behind caching but must avoid write-combining (that is, for auditing reasons) should create a “versioned” cache key (for example, by combining the natural primary key with a sequence ID).

Write-Behind Factor Can Be Set Dynamically

```
// Change the write-behind factor to 120 seconds.  
// This code is executed by an InvocationService.  
// A complete example is available on wiki.tangosol.com  
// You CANNOT dynamically change from Write-Behind to Write-Through  
  
String m_sCacheName = "myCustomerCache";  
m_newWriteBehindFactor = 120;  
  
NamedCache cache = CacheFactory.getCache(m_sCacheName);  
CacheService service = cache.getCacheService();  
  
ReadWriteBackingMap mapBack = (ReadWriteBackingMap)  
    ((DefaultConfigurableCacheFactory.Manager)  
        service.getBackingMapManager()).getBackingMap(m_sCacheName);  
  
mapBack.setWriteBehindSeconds(m_iWriteBehindSeconds);
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind Factor Can Be Set Dynamically

Here is sample code to set write behind factor dynamically. This can also be set in JMX.

If you want to change your write-behind factor from 60 seconds to 120 seconds dynamically, while the Coherence server is running, you can do it with this code example, or with JMX.

You cannot, however, change from write-behind to write-through. You cannot change the value dynamically from a nonzero value to 0. Coherence at startup is in either write-behind mode or it is in write-through mode.

Practice.09.01 Overview: Integration using a CacheStore

This practice covers the following topics:

- Creating a class which extends CacheStore and uses JDBC to load data
- Registering and testing the CacheStore based class



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Persisting Data to a Storage Data Source Integration Implementation

Coherence and JPA

- Introduction to JPA
- JPA and Coherence
- Configuring the JPACacheStore

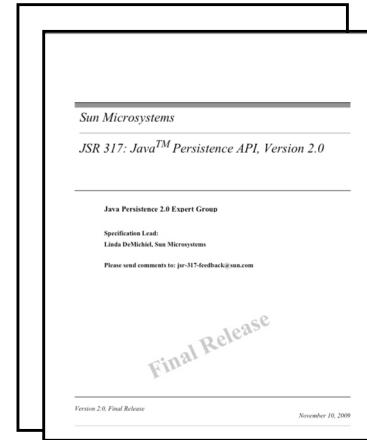


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

The Java Persistence Architecture

Java Persistence API (JPA):

- Is based on Java Specification Request (JSR) 220
- 2.0 is based on JSR 317
- Defines an external mechanism for solving the O/RM problem
- Is only a specification
- Requires a backing implementation such as:
 - Oracle TopLink Grid
 - Eclipse EclipseLink
 - Hibernate
 - Others



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

The Java Persistence Architecture

The JPA specification was originally part of JSR 220, but updated to JPA 2.0, in JSR 317. In either case, these are just specifications, and there is no accompanying software (apart from the reference implementation). You need a commercial or open source implementation to use JPA.

Apache OpenJPA is just such an open source implementation, built from code donated originally by BEA, now Oracle. OpenJPA fully implements the JPA specification, and extends it with some value-add functionality. It works as a stand-alone persistence engine, or integrates into any EJB 3.0-compliant container, as well as many lightweight frameworks. Oracle WebLogic Server is just such a fully-fledged Java EE application server (and more), and includes a JPA implementation fully integrated into its Java EE container services (and still provides standalone JPA functionality where that is required).

JPA and Persistence Requirements

A persistence solution should provide:

- Automatic state tracking and persistence
- Freedom from the “O/R Impedance Mismatch”
- The simplest configuration mechanism
- Freedom from proprietary solutions

A persistence solution should avoid:

- Homegrown attempts to embed JDBC in your code
- Homegrown attempts to write a DAO persistence layer
- Flaws in the EJB 2.x component model

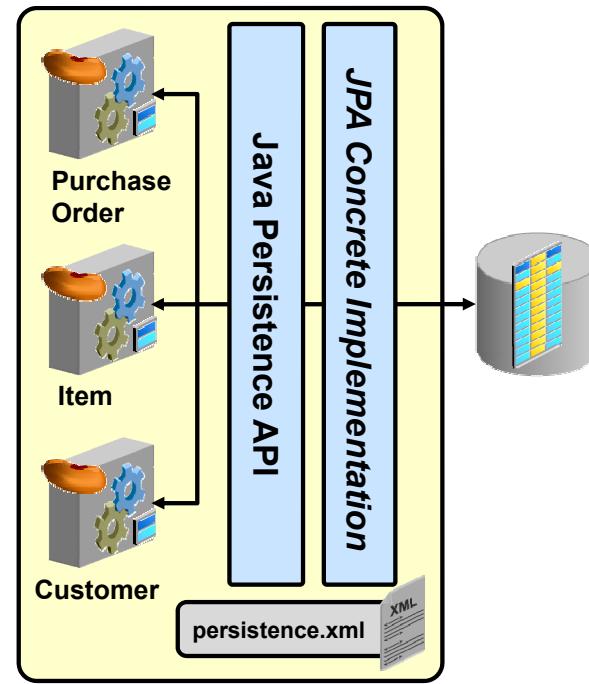


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

The JPA Approach

The JPA approach: Coherence Domain Objects!

- Entities are annotated Java Beans Can also be mapped via XML!
 - Appropriate defaults
 - Well-defined queries and O/R mapping
- Automatic state tracking via Entity Managers and bytecode weaving
- JPA Implementations are based on standards
- Minimal configuration via `persistence.xml`



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

The JPA Approach

As has been shown, good persistence solutions must meet a number of criteria. JPA provides an excellent solution to the general persistence problem.

- Annotations: Annotation solve a number of issues by allowing us to use traditional Java elements, in this case JavaBeans, and work with them in our applications normally. However, the annotations themselves are only part of the solution. Using bytecode weaving on a compiled Java class, the JPA persistence API can then automatically track changes to entity state, minimizing, and in many cases eliminating, the need for client application code to load and unload fields. Note that it is also popular to map entities via XML, especially when object source code is not available.
- Standards: Since JPA is a standard, a number of conformant implementations exist. Additionally the standardized abstract schema model allows developers to write JPA queries which execute against a number of databases. JPA Implementations vary widely in terms of performance and other optimizations.
- O/R Mapping: Beyond the JPA query language, JPA provides a variety of mechanisms to implement standard JavaBeans, yet map them easily and intelligently to their supporting tables.

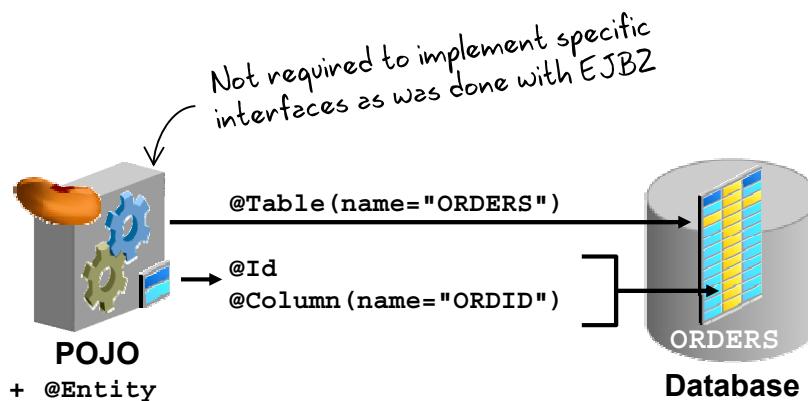
The JPA Approach (continued)

- Configuration: The set of well-thought-out defaults provided by JPA make it quick and easy to develop entities by exception, only changing behavior when required, as opposed to other solutions which require defining every behavior.

What Are JPA Entities?

A JPA entity is:

- A lightweight object that manages persistent data
- A Plain Old Java Object (POJO) marked with the `@Entity` annotation
- Mapped to a database by using annotations



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What Are JPA Entities?

The Java Persistence API (JPA) is a part of the Java EE 5/EJB 3.0 specification that simplifies Java persistence to a database. It provides an object-relational mapping (O/RM) approach that enables you to declaratively define how to map Java objects to relational database tables. The JPA works both inside a Java EE 5 application server, and outside an EJB container, in a Java Standard Edition 5 (Java SE 5) application.

A JPA entity, or simply, “entity”:

- Manages persistence data
- Has fields that are mapped to columns in a relational database table by using annotations
- Does not require that any interfaces be defined

Using JPA, you can designate any POJO class as a JPA entity by marking the POJO with the `@Entity` annotation. An entity is not required to implement any interfaces. In an entity, all fields are considered persistent, unless annotated with the `@Transient` annotation.

Note: You need to implement the `java.io.Serializable` interface if the entity needs to be passed by value through a remote interface.

What Are JPA Entities? (continued)

The following annotations are used to map a POJO to a relational data construct:

- `@Table` maps the object to a table.
- `@Column` maps a field to a column (required if the field and column names are different).
- `@Id` identifies primary key fields.

Entity Class Requirements

An entity class must:

- Be annotated with `@Entity`
- Have a public or protected no-arg constructor
- Not be declared `final`, nor have persistence methods, nor variables marked `final`
- Have a declared primary key, identified with `@Id`, `IdClass` or `EmbeddedID`
- Be identified in a persistence unit
- Only have persistence fields of supported types, and should be private or protected scope
- *Should be* serializable, if passed by reference to outside consumers



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Entity Class Requirements

Entity classes are derived from simple JavaBean POJO objects, and must follow a set of minimum requirements to be used by a persistence provider.

Minimum requirements include:

- Must be identified by the Entity annotation. However, as you shall see later, the entity may alternatively be based on an embeddable class or derived from another class.
- Must have a no-argument constructor. Because the persistence provider creates instances of objects and then populates them with data, every entity must have a no-argument constructor.
- Must not be final nor have final variables. Final causes a “lockdown” of the class and its variables. Final variables can be set only once. Final classes are not subclassable. Both cause problems with JPA.
- Must have a primary key. Because all entities are ultimately persisted to and loaded from a database, an identifier is required.
- Persistence fields must be from the known list of supported persistence types.
- Additionally, it is beneficial if entity classes are serializable, primarily for the purpose of sending detached instances across the network.

JPA and CacheStores

Coherence supports two JPA implementations via CacheStores:

- **JPACacheStore**: A generic JPA implementation based on CacheLoader/CacheStore set which can be used with any JPA integration
- **TopLink Grid CacheStore**: A targeted JPA implementation optimized for use with EclipseLink, which provides some configuration automation

```
package com.com.tangosol.coherence.jpa;
public class JpaCacheStore
    extends JpaCacheLoader implements CacheStore {
    .
    .
}
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

JPA and CacheStores

The generic JPA support is provided via two classes:

- **JPACacheLoader**: A load-only implementation that uses any JPA implementation to load entities from a data store.
- **JPACacheStore**: A full load-and-store implementation that uses any JPA implementation to load and store entities to, and from, a data store.

In both cases, entities must be mapped to the data store, and a JPA persistence unit configuration must exist. The persistence unit is assumed to be set to use RESOURCE_LOCAL transactions.

Integrating JPA and Coherence

To integrate using JPA:

- Obtain a JPA Implementation: The `JpaCacheStore` implementation works with any JPA provider, but no provider is shipped with Coherence.
- Map entities: Mapping entities to JPA is done via annotations to a Java class, or via a mapping configuration such as `orm.xml`.
- Configure JPA: JPA is configured using `persistence.xml`.
- Configure Coherence to use JPA: Coherence uses a combination of a caching scheme defined around the `JPACacheStore`, and a set of cache mappings.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Integrating JPA and Coherence

To use JPA with Coherence, four specific things are required:

1. Obtain a JPA provider implementation. JPA is a generic standard, and supported by a variety of vendors. Coherence does not come with a JPA provider, however they are easily obtained from the Internet. EclipseLink is the JPA reference implementation and freely available.
2. Objects or entities must have their fields mapped to a table row and column. This is normally done by annotating the Java class, but can also be done using a mapping file known as `orm.xml`.
3. JPA must be configured to know what database connection information to use. `persistence.xml` is normally used to provide this information.
4. The final step is to configure a Coherence caching-scheme and a cache mapping to support the `JpaCacheStore`.

Obtaining a JPA Implementation

JPA Implementations are available from a variety of sources, including:



Oracle TopLink Grid: Specifically tailored to Coherence, TopLink Grid provides simple configuration, support for complex object storage, support for eager and lazy relationships and more



EclipseLink: A standards-based Object-Relational persistence solution with additional support for many advanced features. Available from <http://www.eclipse.org/>



Hibernate: One of the earlier O/RM mapping projects. Hibernate provides an open source, but well-supported, JPA implementation.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Obtaining a JPA Implementation

JPA is based on JSR 220, and is a specification rather than an implementation. It is the responsibility of the developer to choose and configure an JPA implementation. Any number of JPA providers exist, several of which are listed on the slide. One of the benefits of JPA is its ability to easily replace a provider with another as requirements or performance needs change.

- Oracle TopLink Grid: A premier JPA provider, designed to work directly with Coherence and providing a number of specialized features and optimizations to mesh seamlessly with coherence implementations, providing the same scaling and reliability expected of Coherence itself.
- EclipseLink: EclipseLink 2.0 is the reference implementation of JPA and is one of the easiest providers to develop and test against.
- Hibernate: A open source O/RM mapping, Hibernate provided one of the first O/RM mapping tools and has become a well-supported, well-respected and high-performing JPA implementation.

Mapping Entities

There are two ways to map entities to tables:

- `orm.xml`: A configuration-only solution

```
<?xml version="1.0" encoding="UTF-8"?>
. . .
<entity class="Customer">
  <table schema="MYAPPLICATION" name="CUSTOMER" />
  <attributes>
    <id name="id"><column name="ID" /></id>
    <basic name="lastName"><column name="LASTNAME" /></basic>
  . . .
```

- Java JPA Annotations: Maps classes to tables

```
@Entity
public class Customer {
  @Id long id;
  @Basic String lastName= null;
  @Basic String firstName = null;
}
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Mapping Entities

One of the most important aspects of JPA is its ability to map entity objects to their database counterparts. With JPA this is done in one of two ways:

- Annotations: Assuming access to the source code, a Java class can be annotated to mark it an entity, mark its fields as persistent, note its relationships to other fields, and so on. A side effect of annotating the source is byte-code enhancement. Once the code has been compiled, a byte-code enhancement step is typically run to further extend to code to include functionality defined by the annotations.
- `orm.xml`: Assuming you do not have access to the source code, a mapping XML file, typically named `orm.xml`, can be used. The mapping file has all the same capabilities as the annotations, except it is a configuration-only solution, and can be used when source code is not available.

Configuring JPA

JPA is configured via `persistence.xml`, which contains:

- Lists of persisted entities, and optional mapping files
- Database connection information
- Optional mapping files and more

```
<?xml version="1.0"?>
<persistence . . . />
  <persistence-unit name="Sample" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
      <class>com.acme.entities.PurchaseOrder</class> . .
      <properties>
        <property name="eclipselink.jdbc.driver"
                  value="oracle.jdbc.driver.OracleDriver"/>
        <property name="eclipselink.jdbc.url"
                  value="jdbc:oracle:thin:@localhost:1521:XE"/>
        <property name="eclipselink.jdbc.user"      value="uname" /> . .
      </properties>
    </persistence-unit>
  </persistence>
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuring JPA

JPA is configured to access a database, as well as other aspects of entity management, via a `persistence.xml` file. `persistence.xml` contains a number of configuration settings.

- Persistence unit: Each persistence file can have one or more persistence units, which define a transaction scope, as well as a name.
- Provider class: Since JPA can use a number of persistence providers, the name of a bootstrap class is defined in the provider element.
- One or more class elements defining the fully qualified class path to persistable classes
- A set of properties, typically used to define database connection information

Coherence JPA Configuration

Coherence is configured to use JPA, similarly to any CacheStore solution.

- Define a caching-scheme which uses a JPACacheStore.
- Define a cache-mapping for each entity being managed based on the scheme.

```
<cache-config>
  ...
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>jpa-distributed</scheme-name>
      <service-name>JpaDistributedCache</service-name>
      ...
      <cachestore-scheme> . . . </cachestore-scheme>
    </distributed-scheme>
  </caching-schemes>
  ...
</cache-config>
```



coherence-cache-config.xml

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Cachestore-scheme

The `cachestore-scheme` element defines how the JPA mapping to a persistence unit and entity occurs.

```
...<cachestore-scheme>
    <class-scheme>
        <class-name>
            com.tangosol.coherence.jpa.JpaCacheStore
        </class-name>
        <init-params>
            <init-param>
                <param-type>java.lang.String</param-type>
                <param-value>{cache-name}</param-value>
            </init-param>
            <init-param>
                <param-type>java.lang.String</param-type>
                <param-value>com.acme.{cache-name}</param-value>
            </init-param>
            <init-param>
                <param-type>java.lang.String</param-type>
                <param-value>Sample</param-value>
            </init-param>
        </init-params>
    </class-scheme>
</cachestore-scheme>
```

coherence-cache-config.xml



ORACLE

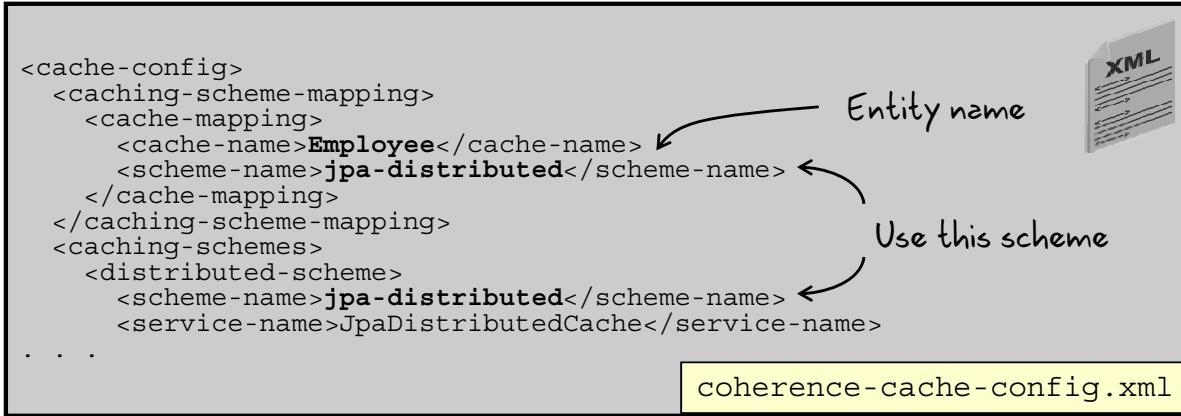
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Cachestore-scheme

The `cachestore-scheme` element of the backing map defines how JPA maps a `JpaCacheStore` to an entity and a persistence unit. The example shows three `init` parameter elements defined that map directly to the three-argument constructor of the `JpaCacheStore`. There must be a one-to-one mapping of cache to entity, so the cache name variable is used to provide the name of the entity.

Cache Mapping

A cache-mapping is defined, which uses a predefined caching scheme to create an instance of a `JpaCacheStore` to complete the JPA mapping.



The diagram shows a snippet of XML configuration code with handwritten annotations:

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>Employee</cache-name> ← Entity name
      <scheme-name>jpa-distributed</scheme-name> ← Use this scheme
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>jpa-distributed</scheme-name> ←
      <service-name>JpaDistributedCache</service-name>
    <...>
  </caching-schemes>
</cache-config>
```

The file is labeled `coherence-cache-config.xml`.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Provide a brief overview of data source integration
- Explain the process of persisting data to a database
- Implement data management features:
 - Read-through
 - Write-through
 - Refresh-ahead
 - Write-behind caching
- Explain and configure Coherence for JPA



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and ORACLE CORPORATION use only

10

Understanding Typical Caching Architectures

ORACLE®

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Explain the different caching architectures that are typical of Coherence applications
- Describe how to distinguish each caching strategy
- Explain the purpose and function of each caching strategy



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Evolution of Data Grid Design Patterns

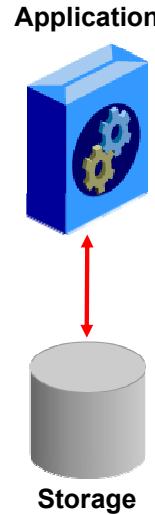
- Single Application Instances
- Multiple Application Instances
- Local Caching Pattern
- Distributed Coherent Caching Pattern
- Cache-Aside and Read-Through Patterns
- Write-Through and Write-Behind Patterns
- External Update Patterns
- Near-Caching and Client-Side Processing Patterns
- Server-Side Processing and Distributed Computing Patterns



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Single Application Instances

In the beginning, applications were simple, and communicated directly with the database.



ORACLE

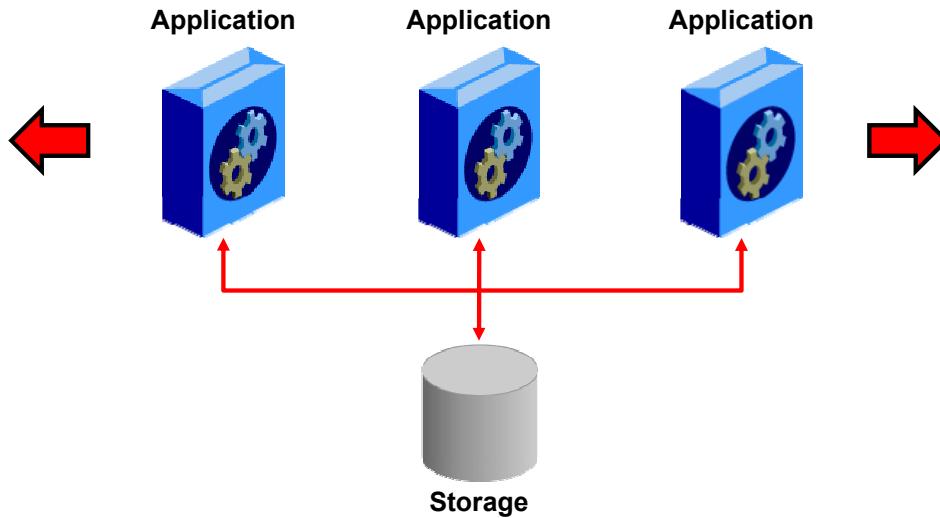
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Single Application Instances

Applications often are written without any type of caching. These applications work directly with the database, and do not involve multiple instances of programs that work together and involve many users. This type of application is very easy to write and maintain, but is very basic and is not the basis for enterprise computing. As the number of user requests grow, the application design needs to change to accommodate that growth.

Multiple Application Instances

Then things started to scale, because customer requests needed more capacity.



ORACLE

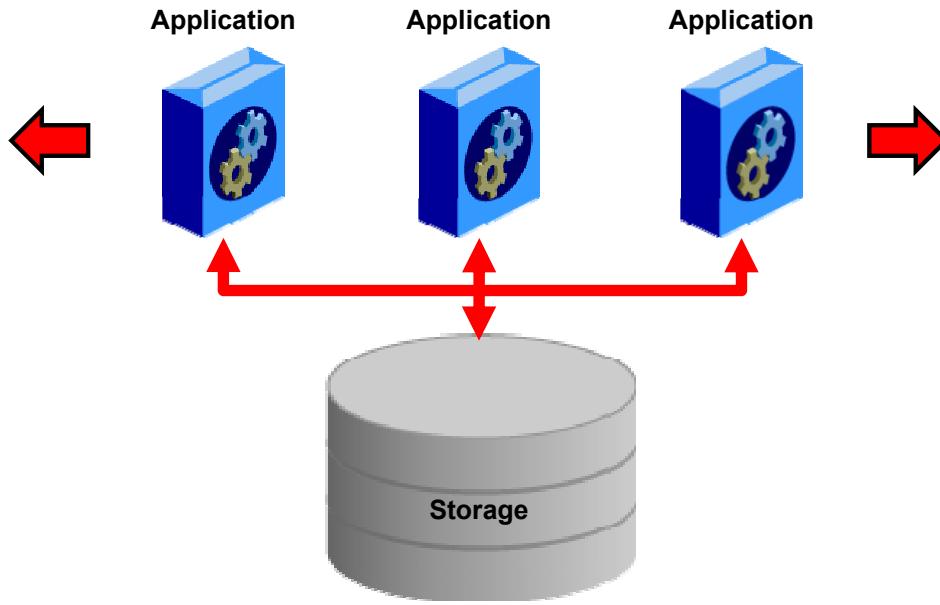
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Multiple Application Instances

The first step in increasing the capacity of an application is to increase the number of instances of the application that can handle user requests. Often this is done without taking into consideration the capacity of the back-end data source. All application instances continue to use the same database.

Multiple Application Instances

Storage and I/O was determined to be a bottleneck, so those resources were scaled up.



ORACLE

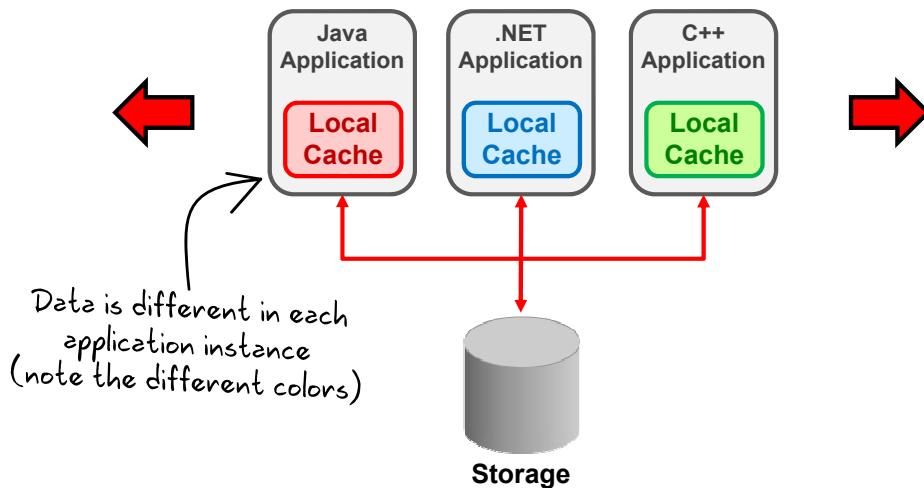
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Multiple Application Instances (continued)

As the user requests flow into the application instances, the back-end data source is not able to keep up with the number of requests being made by all of the application instances. This is often the time when a company will upgrade their hardware and software resources for the back-end to enable it to handle the load. Eventually, the data source can only be scaled up so much until the process becomes very cost-prohibitive. If the data source is a database, then tactics, such as database partitioning, are employed to reduce the load by spreading it across numerous databases that work together to form a unified data store. This can be a maintenance issue.

Local Caching Pattern

But that was not enough, so applications started using the "Local Caching Pattern."



ORACLE

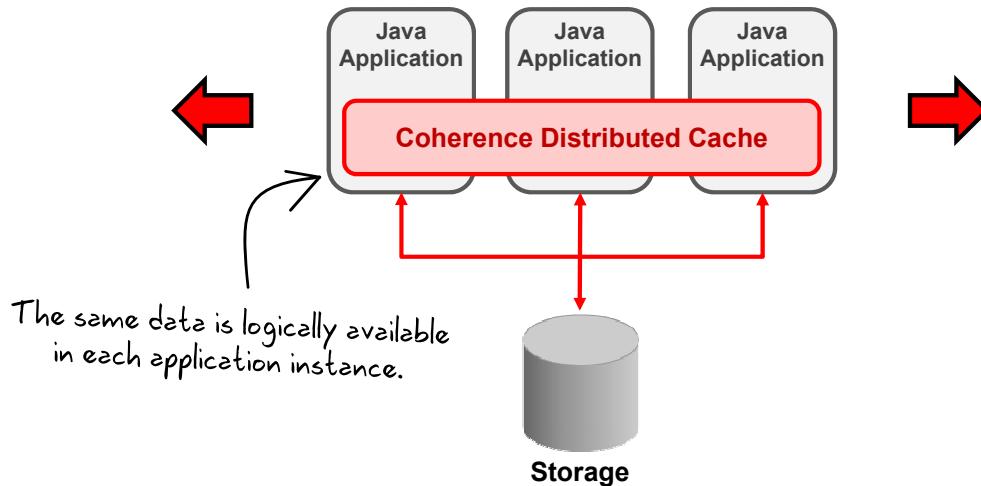
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Local Caching Pattern

It becomes apparent at some point that read-intensive applications are overtaxing the data source, so a caching strategy is introduced to lower the number of requests (and thereby the load) on the data source by keeping frequently-read data close to the application that is using the data. The data is reused, and the application benefits from lower latency because the data source was not involved, and the data source benefits from a lower load because it is handling fewer requests. However, each application instance contains a cache that is not necessarily in sync with the other application instances, and each instance is unaware of the data in the other.

Distributed Coherent Caching Pattern

With *local* caching, information was not consistent across servers, so caching was distributed.



ORACLE®

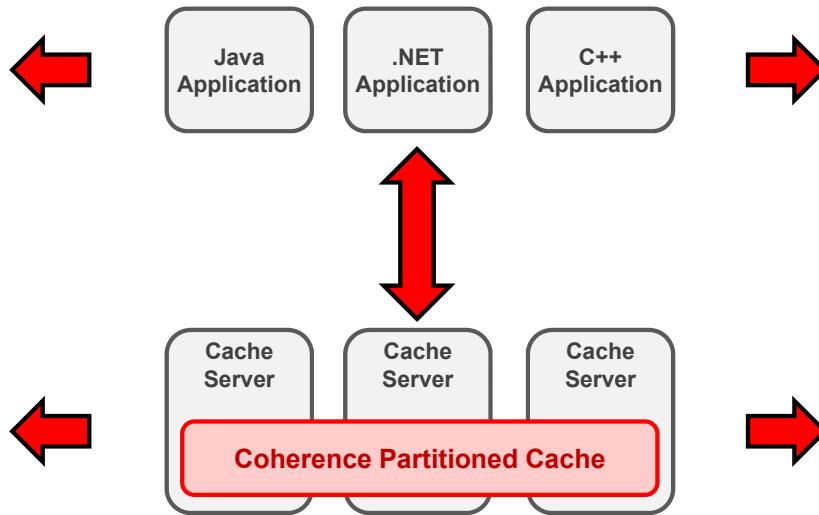
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Distributed Coherent Caching Pattern

In order to keep data consistent across application instances, a distributed caching service is introduced that is aware of the data in each instance. This provides the ability to cache data in the applications, and to keep the data consistent across each instance. In this case, it could be a replicated cache or a partitioned cache. The rest of the lesson assumes a partitioned cache.

Introducing the Caching Infrastructure Layer

Caching became a distinct *resource*, so that resource became a new layer.



ORACLE

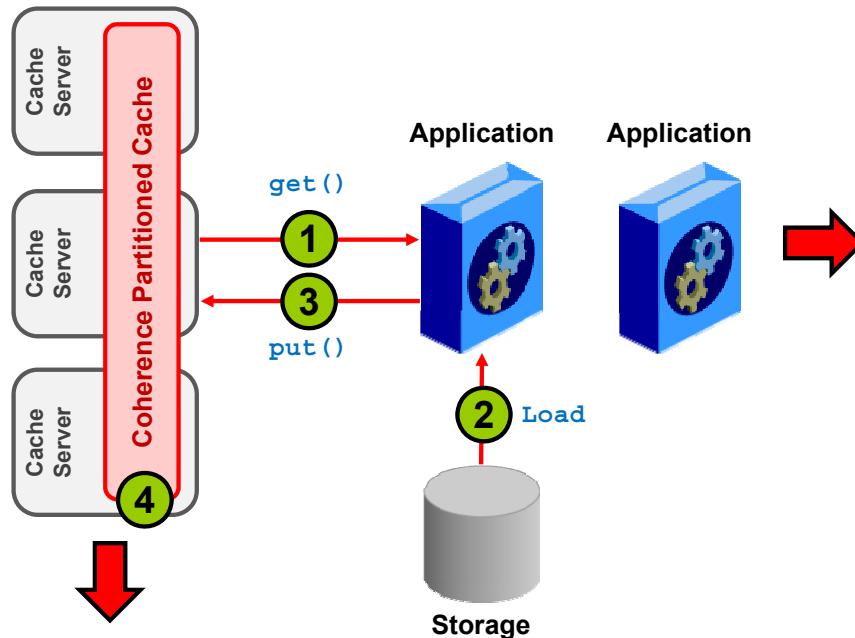
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Introducing the Caching Infrastructure Layer

Because there was a service that was cache-aware, it made sense to create a proper infrastructure around the technology that could potentially be used by applications written in different languages. With this infrastructure in place, more robust caching patterns could be realized as well.

Cache-Aside Patterns

Cache as a separate resource



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

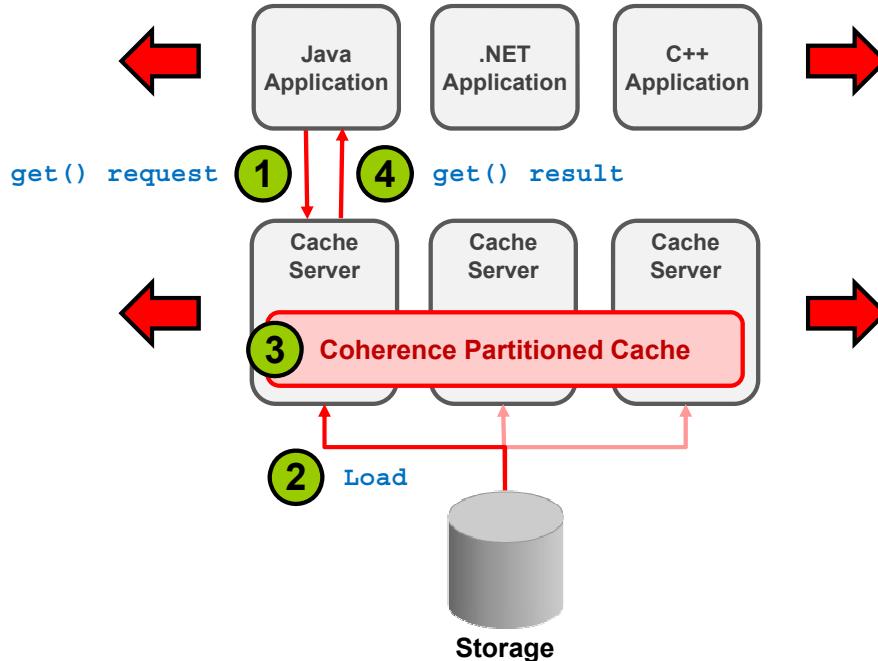
Cache-Aside Pattern

A typical caching pattern is to manually cache data:

1. An application requests data from the cache. If the data is in the cache, then the application has the data it needs and continues processing.
2. If the data requested is not in the cache, then the application loads the data from the data source directly.
3. After loading the data from the data source, the application places the data in the cache for future requests.
4. The distributed caching service manages the cache, and where to put the data.

Read-Through Pattern

The cache was integrated with back-end data sources.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

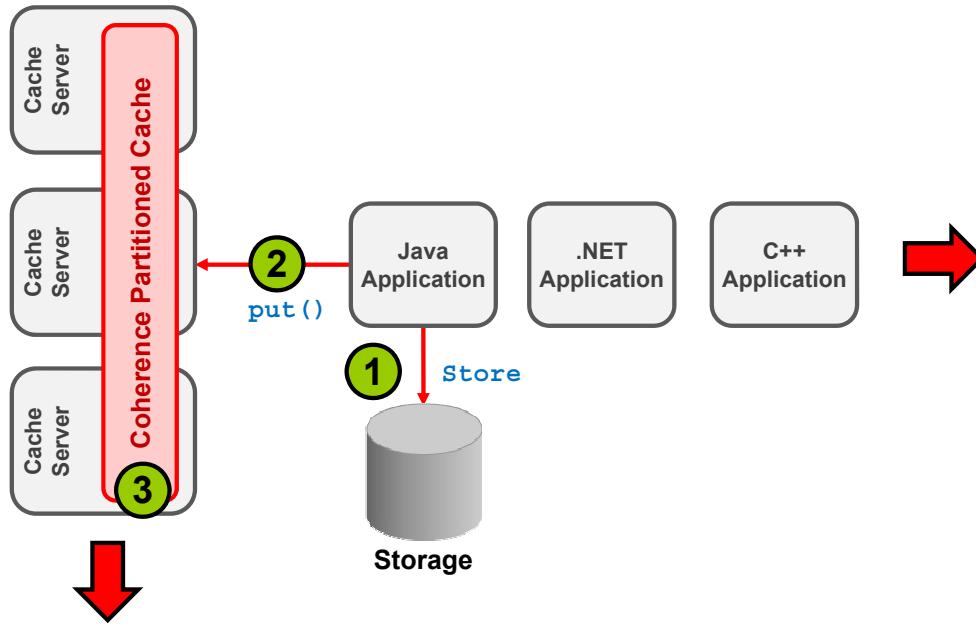
Read-Through Pattern

The read-through pattern involves using a `CacheLoader` implementation that knows how to work with the back-end data source. This abstracts the data source from the application, which works directly with the cache. The cache then manages getting data from the data source on behalf of the application:

1. The application requests data from the cache. If the data is there, then the application has the data and continues processing.
2. If the data is not in the cache, then the caching service uses a configured `CacheLoader` to load the data from the back-end data source.
3. The caching service places the data in the cache.
4. The data is returned to the application.

Write-Aside Pattern

Updates required the ability to write to the cache.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Aside Pattern

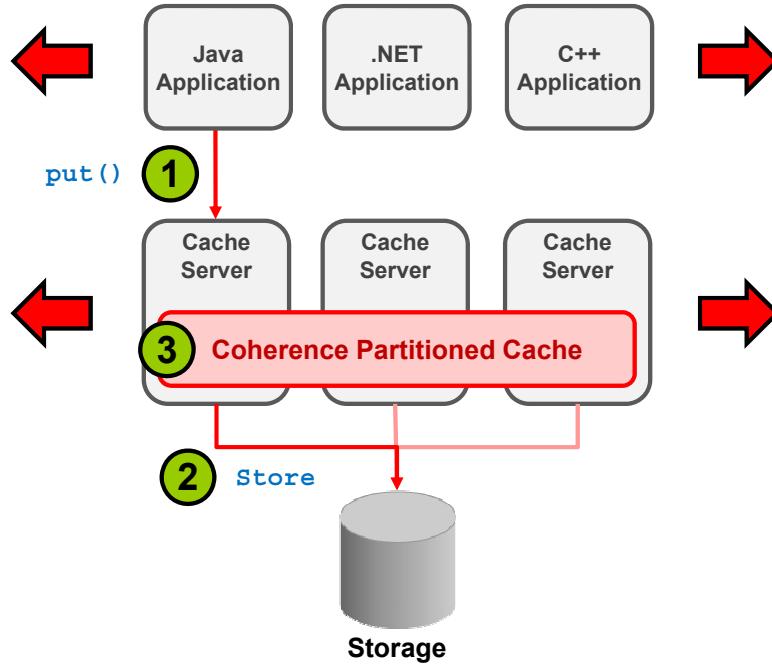
At some point, an application needs to update data that exists in the cache. A typical pattern for this is called the write-aside pattern:

1. The application stores the data directly in the database.
2. The application then separately updates the cache with the data.
3. The caching service places the data in the cache.

One potential issue for this pattern is that in the event of a failure of the application performing this task, it is possible that the data is inconsistent between the data source and the cache.

Write-Through Pattern

Writing to the cache can automatically update the database.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

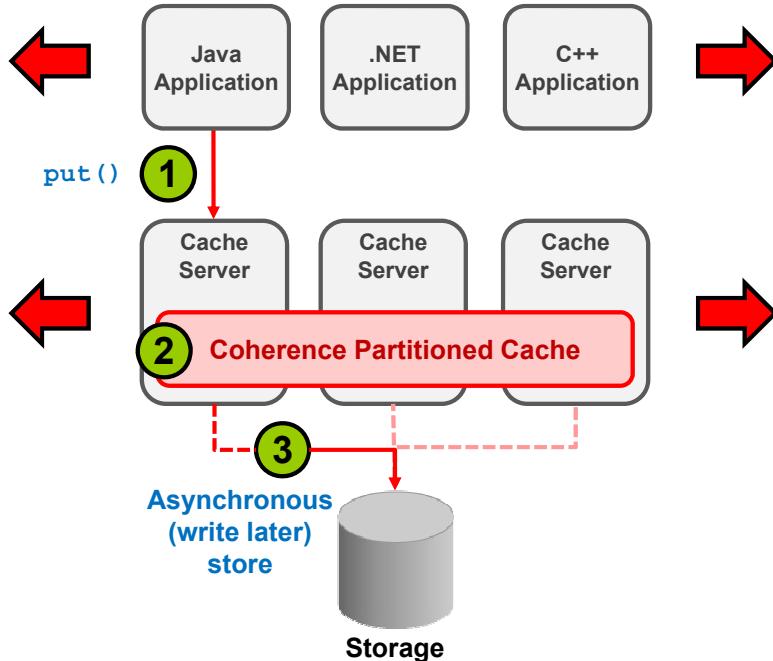
Write-Through Pattern

The write-through pattern is very similar to the read-through pattern, except it is for storing data in the data source. This pattern involves using a `CacheStore` implementation that knows how to work with the back-end data source. This abstracts the data source from the application, which works directly with the cache. The cache then manages putting data in the data source on behalf of the application:

1. The application puts data in the cache.
2. The caching service uses a configured `CacheStore` to store the data in the back-end data source.
3. The caching service places, or updates, the data in the cache.

Write-Behind Pattern

The cache can be the system of record for updates.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Write-Behind Pattern

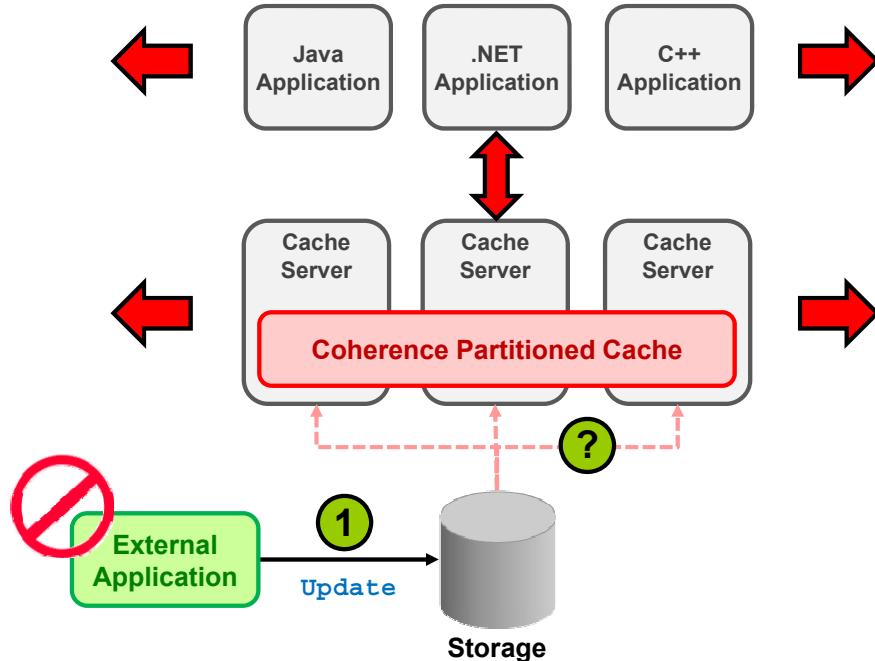
The write-behind pattern takes things a step further. This pattern also involves using a `CacheStore` implementation that knows how to work with the back-end data source. This abstracts the data source from the application, which works directly with the cache. The cache then manages putting data in the data source on behalf of the application. And in the case of the write-behind strategy, the caching service becomes the system of record for the data instead of the database. The write-behind strategy involves:

1. The application puts data in the cache.
2. The caching service places, or updates, the data in the cache.
3. The caching service uses a configured `CacheStore` to store the data in the back-end data source. All updates are queued so that they are processed in order. There are no guarantees that the update occurs immediately, as Coherence will manage the back-end update asynchronously.

Keep in mind that multiple updates on a single entry result in a single update to the database.

External Update Anti-Pattern

External applications can disrupt cache data consistency:



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

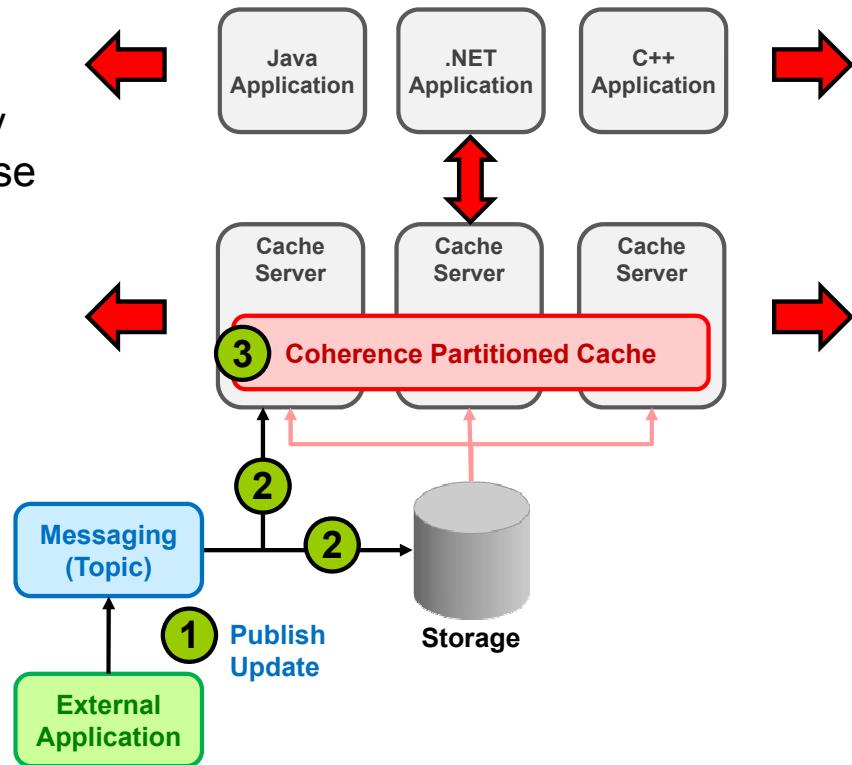
External Update Anti-Pattern

One of the potential issues with using the write-behind strategy (as well as other strategies) is when an application that is external to the caching application updates the back-end data source directly, bypassing the cache. This causes the data in the cache to be stale, and usually there is no mechanism to notify the cache that the data is invalid:

1. An external application updates the data source directly.
- ? The caching service has no idea that anything has occurred with the data source.

External Update Pattern Using Messaging

Messaging can help manage consistency between the database and the cache.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

External Update Pattern Using Messaging

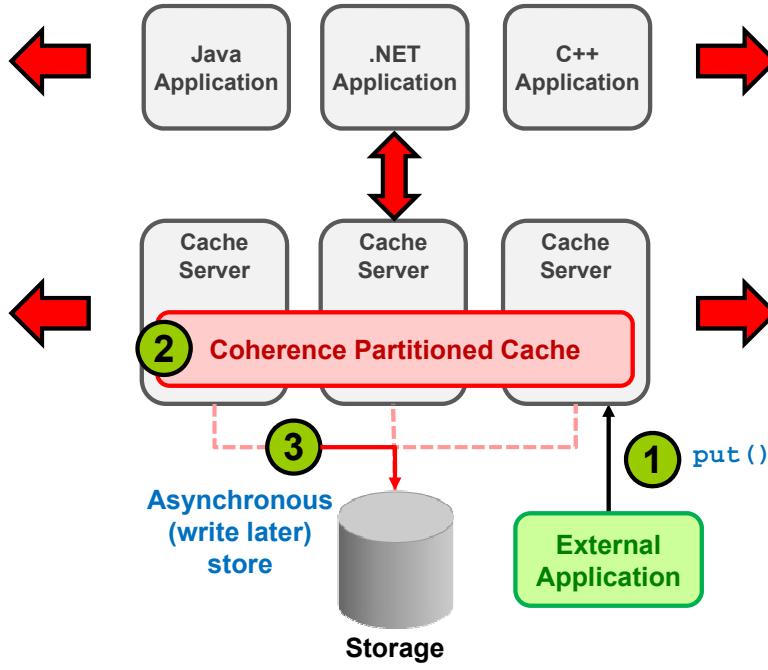
An external application can manage keeping the data in-sync between the cache and the data source by using a messaging topic:

1. External application publishes an update.
2. The messaging topic subscribers receive the update, update the data in the data source, and sends the update to the caching service.
3. The caching service updates the data in the cache.

JMS is deprecated as of Coherence version 3.6, and a messaging topic or queue would have to be via a Java EE application server such as WebLogic Server.

External Update with Direct Access

External applications can update the database using the cache.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

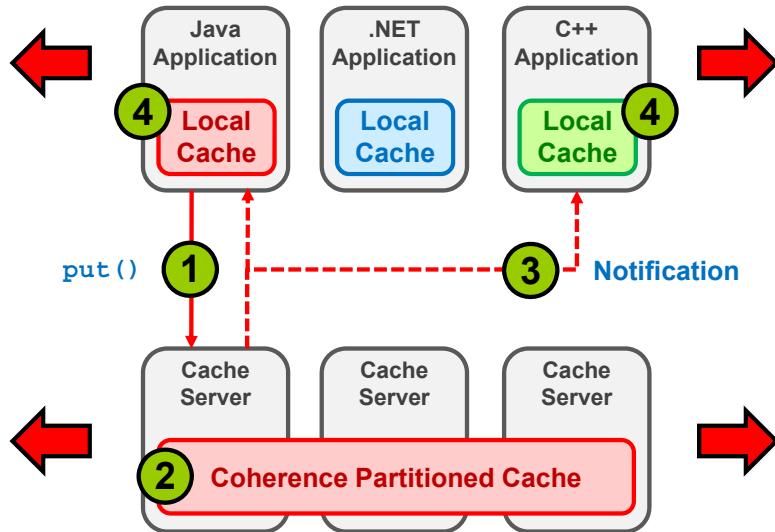
External Update with Direct Access

An external application can also manage keeping the data in-sync between the cache and the data source by using the caching service directly, as in this example that uses write-behind caching:

1. External application sends an update directly to the caching service.
2. The caching service updates the data in the cache.
3. The write-behind mechanism asynchronously manages writing the data to the data source.

Near Caching Pattern

Near caching provides fast local read performance for a larger back-end cache, and uses events to invalidate data when it is updated in the back cache.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Near Caching Pattern

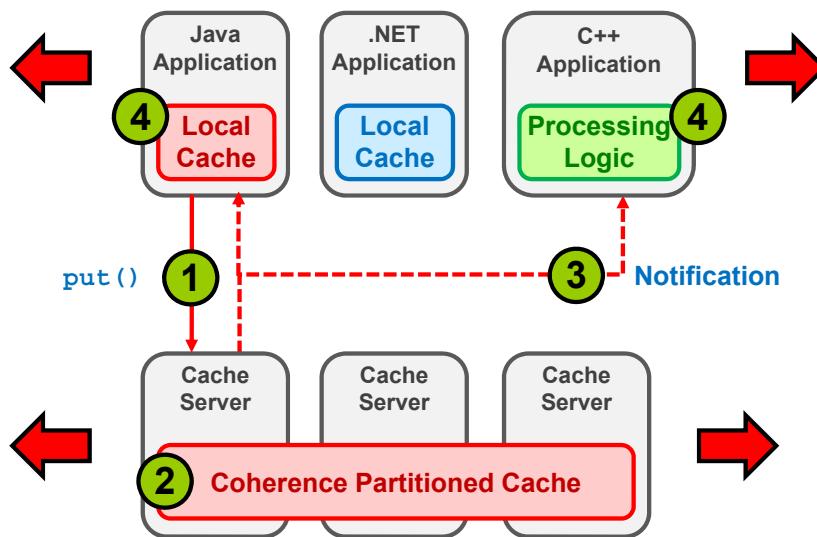
Coherence partitioned caching is able to handle very large data sets, up to the terabyte range. This cache is very scalable and provides predictable performance, but read performance is slower than local caches. The near caching pattern provides a size-limited local cache, which is colocated with the application, and backed by the larger data set. The local cache is a cached subset of the back cache, and provides very fast read performance for frequently-read data. The near caching pattern also uses event notification to invalidate locally cached data entries as they are updated in the back cache.

The near caching pattern:

1. The application updates the data entry using the Coherence `put ()` method, and the request is sent to the partitioned caching service for the back cache.
2. The data entry is updated in the back cache.
3. An event is published that the data entry has been updated.
4. The data entry exists in the local cache of another application instance. The local entry and that entry receive the event that the back cache entry was updated, and invalidate themselves.

Client-Side Event Processing Pattern

Clients can react to event notifications in real-time.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Client-Side Event Processing Pattern

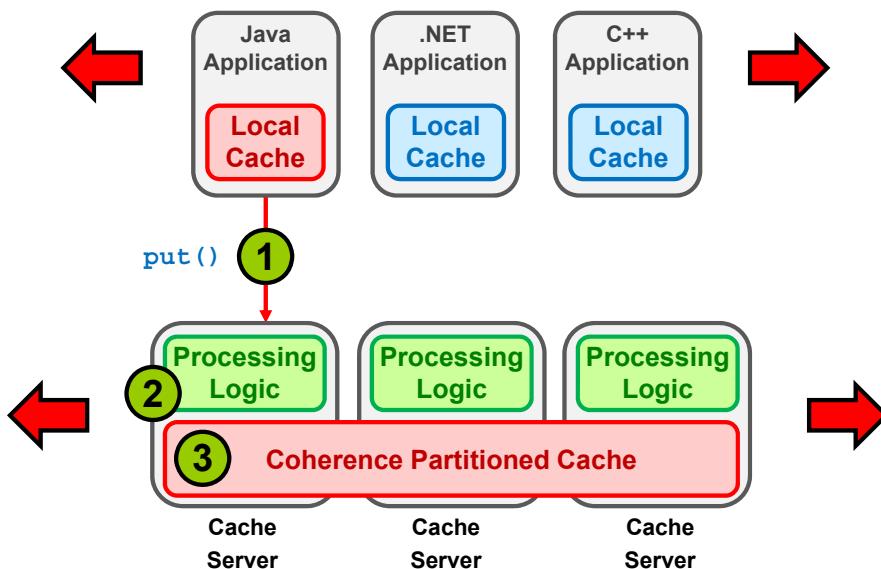
Client processes can register map listeners that can subscribe to certain events when data entries are inserted, updated, or deleted. This was discussed in detail in the lesson titled “Observing Data Grid Events.”

The process works as follows:

1. The application updates data in the cache by calling `put()`, and the data is sent to the caching service.
2. The caching service updates the cache.
3. An event is sent to all subscribing listeners containing information about what data has changed.
4. The subscribers receive and handle the event programmatically.

Server-Side Event Processing Pattern

Servers can react to events in real time.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Server-Side Event Processing Pattern

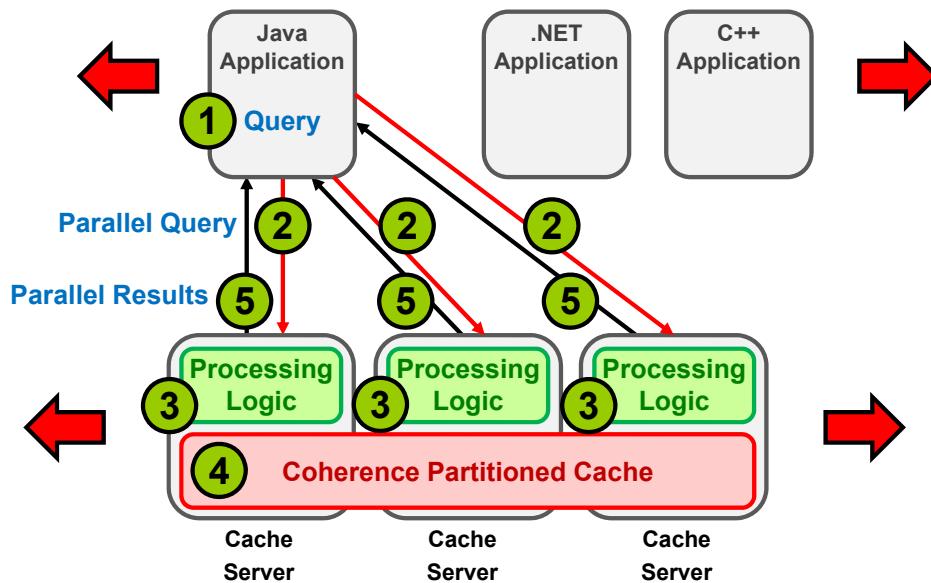
Server-side processes can register map triggers that can subscribe to certain events when data entries are inserted, updated, or deleted. This was discussed in detail in the lesson titled "Observing Data Grid Events."

The process works as follows:

1. The application updates data in the cache by calling `put()`, and the data is sent to the caching service.
2. An event is triggered before the cache is modified, and the server side executes some code that performs any type of processing prior to the data getting placed into the cache.
3. The caching service updates the cache.

Server-Side Processing Pattern: Queries, Map Reduction, Computation...

The cache can be queried, and process data in parallel.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

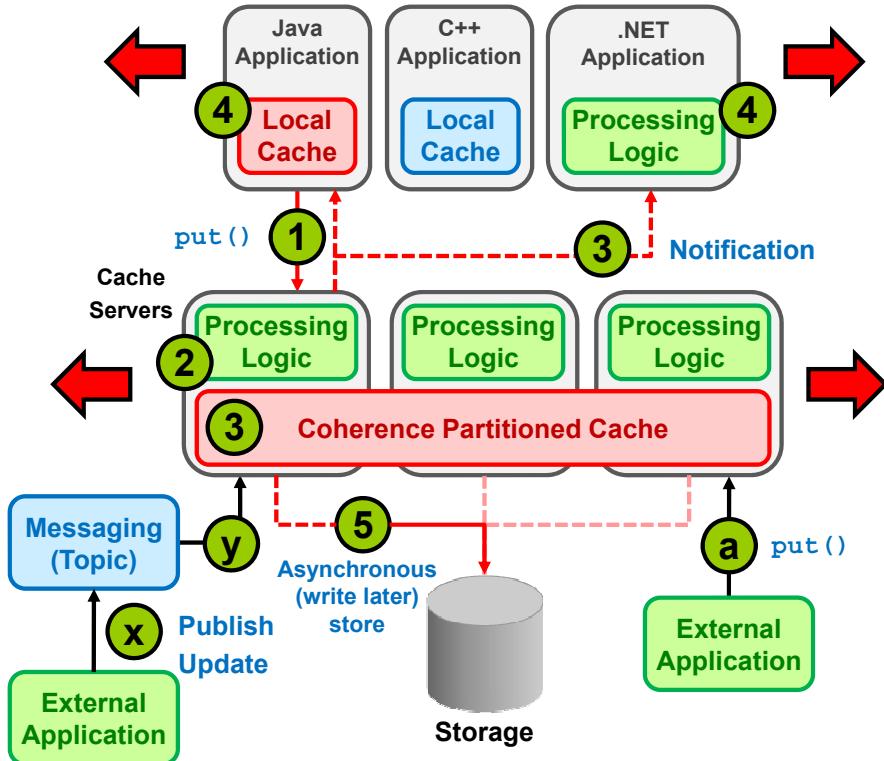
Server-Side Processing Pattern: Queries, Map Reduction, Computation...

Using the query, EntryProcessor, and server-side event mechanisms, a Coherence cluster can process data entries across the grid in parallel; and if the processing involves inserting, updating, or deleting a cache entry, the server-side can process the event without having to send data over the network. All processing is spread across the cluster where the cache entries exist, which maximizes CPU utilization and reduces latency.

Combined = Scalable Platform!

Combining these features provides the ability to:

- Process data in parallel
- Integrate with a data source
- Handle external data source updates
- Handle events in real time



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Combined = Scalable Platform!

Combining the various strategies described in this lesson provides a highly scalable architecture that can handle any caching needs:

1. The application updates the cache by calling `put()`, and the data is sent to the caching service.
2. Prior to the cache getting modified, the server-side triggers an event and performs some processing on the data.
3. The cache is updated with the new data, and an event is sent to a subscribing client.
4. The subscribing clients receive and handle the event.
5. At some configured interval, the write-behind strategy in use asynchronously updates the back-end data store.
 - a. An external application uses the cache directly to update data in the data source via the write-behind strategy used by the cache.
 - x. Another external application updates the cache using a messaging strategy by publishing the update to a JMS topic.
 - y. The topic subscriber receives the update, and updates the cache directly using the `put()` method.

Quiz

What is the main problem with applications that do not use caching as they grow?

- a. They do not scale well because user requests can overrun the database.
- b. They do not scale well because user requests can overrun network bandwidth.
- c. They are difficult to maintain when techniques such as database partitioning are used.
- d. All of the above.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: d

As an application grows a larger user base, the number of requests that enter the application become too much for the application to handle. In most cases, a finely-tuned application will still experience some performance issues around the database, which cannot keep up with demand for so many requests. By caching the data next to the application, the number of requests to the database are reduced drastically for data that is frequently used. This also decreases the amount of data that is sent over the network. Additionally, techniques such as database partitioning are not required as the load is reduced on the database; thus eliminating the need to partition the data.

Quiz

What is the main difference between a local cache and a replicated cache?

- a. Data is stored in serialized form in a local cache.
- b. Data is stored in serialized form in a replicated cache.
- c. Data is consistent across a cluster of local caches.
- d. Data is consistent across a cluster of replicated caches.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: d

Both local and replicated caches store data in nonserialized form because reads will always be local. Data across a cluster of local caches will most likely be different across a cluster of local caches. Data is guaranteed to be consistent across all the cluster nodes in a replicated cache.

Quiz

What is the difference between the Write-Through and Write-Behind strategies?

- a. Write-Behind writes to the database asynchronously.
- b. Write-Through writes to the database asynchronously.
- c. Write-Through requires that Coherence is the system of record.
- d. Write-Behind handles external updates.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: a

Write-behind specifically writes data to the database asynchronously, while write-through is synchronous. Write-behind requires that Coherence is the system of record, while write-through assumes that the data source is the system of record. Either strategy can be used for an external update pattern.

Quiz

What strategy involves checking the cache for data, loading the data from the data source if it is not in the cache, and placing the loaded data into the cache?

- a. Write-Aside
- b. Cache-Aside
- c. Write-Through
- d. Read-Through



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

- Describe the different caching strategies that are available with Coherence
- Distinguish each caching strategy
- Explain the purpose and function of each caching strategy



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.10.01 Overview: Examining Sample Topologies

This practice covers the following topics:

- Examine a set of predefined application domain scenarios.
- Review each in terms of a proposed Coherence architecture.
- Critically review, compare, and contrast proposed architectures in terms of performance, reliability, data lifecycle, and other criteria.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

11

Coherence*Extend

ORACLE®

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe, configure, and deploy a Coherence*Extend
- Describe, configure, deploy, and use Coherence*Web HTTP sessions with WebLogic Server



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Concepts of Coherence*Extend

- What is Coherence*Extend?
- Coherence*Extend Architecture
- Capabilities
- Advantages and Disadvantages
- When to use Coherence*Extend
- Coherence*Extend Clients
- Data Replication

Configure and use Coherence*Extend

Coherence*Web



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What Is Coherence*Extend?

- Allows applications to harness the power of the Coherence data grid without taking on the responsibilities of being a "good citizen" of the grid
- With Coherence*Extend, there is:
 - No in-process transport
 - No cluster and service membership
 - No data ownership

Local caches are not
considered to have
data ownership.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What Is Coherence*Extend?

Coherence*Extend extends the reach of Coherence to applications that are not well-suited to run within the Coherence cluster. It provides a rich client API to the grid without any of the overhead associated with being a cluster member. Coherence*Extend clients do not need to concern themselves with partitioning, data ownership, or cluster membership. Typical uses of Coherence*Extend include:

- Providing desktop applications with access to Coherence caches (including support for near cache and continuous query)
- Coherence cluster "bridges" that link together multiple Coherence clusters that are connected via a high-latency, unreliable WAN

Coherence*Extend Architecture

- Native Client (Java, .NET, C++) runs outside the cluster. All requests are routed to a Proxy Service.
- Proxy Service runs inside the cluster. It delegates requests to an actual clustered service (for example, Partitioned Cache Service).
- TCP/IP is used as a transport binding.
- Coherence*Extend has language-independent serialization (PIF/POF).



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

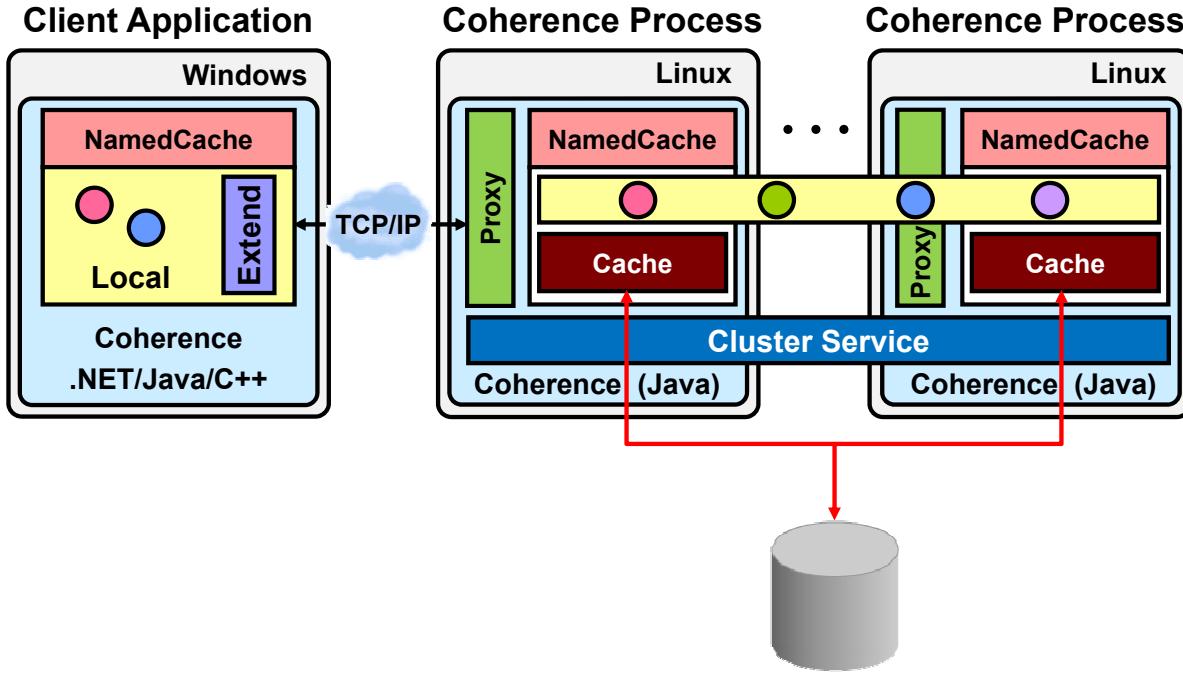
Coherence*Extend Architecture

Architecturally there are two main components to Coherence*Extend:

- A native client
- Proxy Service

The native client routes all requests through a supported transport binding to the Proxy Service, which in turn delegates the requests to a clustered service. Coherence*Extend supports a TCP/IP transport binding. Communication can also be secured using SSL, which is covered in the lesson titled “Coherence Administration.” There are Java, C++, and .NET clients available for Coherence*Extend. As such, Coherence*Extend clients use Portable Invocation Format (PIF) for remote invocation, and Portable Object Format (POF) for serialization because they are a language-independent solution. Java clients may also use Java (or `ExternalizableLite`) serialization. Obviously, these clients will not be able to share data with .NET and/or C++ clients, but `ExternalizableLite` is an option in a Java-only deployment.

Coherence*Extend Architecture



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence*Extend Architecture (continued)

The image on the slide illustrates Coherence*Extend's architecture: The Coherence*Extend .Net, Java, or C++ client communicates with the Proxy Service running inside the Coherence cluster, using TCP/IP to access cached data inside clustered NamedCache objects.

Coherence*Extend Capabilities

Support for most facilities available to a TCMP client node:

- Lifecycle (MemberListener) *
- Caching (CacheMap)
- Query (QueryMap)
- Events (MapListener)
- InvocationService *
- Aggregation and Processing (InvocableMap)
- Near Caching (NearCache)
- Cache Views (ContinuousQueryCache)
- SSL

* *Caveats!*



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence*Extend Capabilities

Coherence*Extend client supports most of the capabilities offered by clustered clients.

Caveats

- Lifecycle (MemberListener): Coherence*Extend clients cannot witness (get callbacks) from Cluster Member events, except for the proxy to which they are connected. They can only register a listener on the proxy server, but not other members in the TCMP cluster.
- The InvocationService works normally when using Coherence*Extend. The caveat is that only the synchronous invocation model is available when using Coherence*Extend.
- SSL is a new feature for Coherence that allows encrypted communication between Coherence*Extend clients and the cluster. Encryption Filters are going to be deprecated.

Transactions are not yet supported with Coherence*Extend.

TCMP stands for Tangosol Cluster Management Protocol, but that name was coined when the product was part of the Tangosol company, before the merger with Oracle.

Coherence*Extend Advantages

Advantages of using Coherence*Extend include:

- Direct access to the data grid from non-Java applications
- Access to multiple data grids from a single application
- Lightweight native client library
- Designed with transient applications in mind
- Designed with end-user machine environments in mind
- Relaxed memory and CPU requirements
- Relaxed network requirements
- Firewall-friendly transport bindings
- Support for numerous client applications



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence*Extend Advantages

The slide lists some of the advantages of using Coherence*Extend. In particular, note that Coherence*Extend supports many clients connecting to a cluster. Coherence*Extend clients do not become a part of the cluster's TCMP communication, and therefore do not cause the extra overhead to the cluster as a whole. This allows Coherence to support tens of thousands of connected clients easily.

Coherence*Extend Disadvantages

Disadvantages of using Coherence*Extend include:

- No direct view of cluster and service membership
- No direct knowledge of data ownership
- Limited direct invocation support
- No asynchronous invocation support
- Limited clustered locking facilities
- No support for client-side transactions
- Extra network hop
- Connection-based design



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence*Extend Disadvantages

The slide lists some of the disadvantages of using Coherence*Extend. It would be possible to bypass some Coherence*Extend limitations by invoking an invocation service on the Proxy Server with an agent that could perform work against the caches of the cluster. When the agent is executed on the Proxy Server, that code has the full visibility of the cluster that a cluster member has and is able to harness the full power of the cluster. For example, normally a Coherence*Extend-based invocation service can only invoke a service on the Proxy Server, because that is the only cluster member it can see. However, if the Coherence*Extend client were to invoke an agent on the Proxy Server that then executed another invocation service of its own, it could then invoke the service on any Coherence node in the cluster.

When to Use Coherence*Extend?

- "Fat" client applications deployed on desktop machines
- Non-Java access to the data grid
- Access to the data grid over a high-latency, unreliable, heterogeneous, restrictive network (across switches, through firewalls, WAN, mixed bandwidth, and so on)
- Coherence Incubator project patterns. For example: Cluster connectors (hot/warm and hot/hot disaster recovery strategies, regional data access)



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

When to Use Coherence*Extend?

Coherence*Extend extends the reach of the core Coherence TCMP cluster to a wider range of consumers, including desktops, remote servers, and machines located across WAN connections. Typical uses of Coherence*Extend include:

- Providing desktop applications with access to Coherence caches (including support for near cache and continuous query)
- Coherence cluster "bridges" that link together multiple Coherence clusters that are connected via a high-latency, unreliable WAN

Coherence*Extend is the means to connect an application to the cluster. It supports "fat client" real-time applications, such as trading desks, as well as other server tiers. It provides almost all the features that a full cluster member can provide, including near-caching, event listeners, and the ability to execute `EntryProcessors`. The connection is over TCP/IP. It also provides the capability of continuous query, which is similar to a database materialized view. Continuous query is a query that is updated in real time.

Coherence*Extend Clients

There are three types of clients for Coherence*Extend, allowing cluster access from other languages over TCP/IP:

- Java
 - Included with main Coherence product
 - Uses same Java API as clustered clients
- C++ 
- .Net 

Separate download
& similar API. No
Java required.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence*Extend Clients

Java: The Coherence*Extend Java client is included with the main Coherence product. It uses the same Java API that the clustered Java client programs use, but all calls are implemented using TCP/IP instead of TCMP.

C++: The Coherence*Extend C++ client is available as a separate download. The client API follows the interface and concepts of the Java client, so users familiar with the Java API find it easy to code C++ clients. A `NamedCache` instance is retrieved by using the `CacheFactory::getCache(...)` API call. Once it is obtained, a client accesses the `NamedCache` in the same way as it would if it were part of the Coherence cluster.

.Net: The Coherence*Extend .Net client is available as a separate download. The client API is also easy for Java developers to understand. An `INamedCache` instance is retrieved by using the `CacheFactory.GetCache(...)` API call. Once it is obtained, a client accesses the `INamedCache` in the same way as it would if it were part of the Coherence cluster. The .Net product also includes a session state store provider implementation for storing ASP .NET HTTP session objects in the data grid.

Data Replication

- Coherence*Extend can replicate one cluster to another.
 - Can be used for disaster recovery sites
 - Requires high-network bandwidth, depending on transaction load
 - Minimize the data to be replicated to what is absolutely necessary
- For more information on replicated clusters, see *Push Replication* on the Coherence Incubator Project site
<http://coherence.oracle.com/display/INCUBATOR/Home>
(Note: the URL is case-sensitive)



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Data Replication

Coherence*Extend can also replicate data from one cluster to another over a WAN connection. Keep in mind that Coherence is so fast, it can actually saturate a gigabit ethernet network. A network between sites, particularly if the sites are in different cities, has very limited bandwidth. If the customer wishes to replicate data between clusters, they should minimize the data to be replicated to that which is absolutely necessary. Otherwise, the cost of a network that could supply the required bandwidth would be prohibitively expensive.

The data replication link on the referenced page points to samples of Coherence*Extend data replication.

The Push Replication link on the referenced page points to the Coherence Incubator project, where Coherence engineers are experimenting with different ways to use Coherence that may end up as new features in the Coherence product. Oracle recommends using the Push Replication pattern to replicate data between Coherence clusters. URL Links are subject to change. The current Incubator Project page is
<http://coherence.oracle.com/display/INCUBATOR/Home>.

Agenda

Concepts of Coherence*Extend

Configure and Run Coherence*Extend

- Configuring Coherence*Extend
- Client-Side Cache Configuration Descriptor
- Cluster-Side Configuration Descriptor
- Launching an Extend-enabled DefaultCacheServer Process and Java Client Application
- Tuning Coherence*Extend
- Coherence*Web



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuring Coherence*Extend

- Create a client-side Coherence cache configuration descriptor that includes one or more <remote-cache-scheme> and/or <remote-invocation-scheme> configuration elements.
- Create a cluster-side Coherence cache configuration descriptor that includes one or more <proxy-scheme> configuration elements.
- Launch one or more DefaultCacheServer processes.
- Create a client application that uses one or more Coherence*Extend services.
- Launch the client application.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuring Coherence*Extend

The <remote-cache-scheme> element is for accessing caches over Coherence*Extend.

The <remote-invocation-scheme> element is for executing code in the cluster via Coherence*Extend.

The <proxy-scheme> element is for configuring a ProxyService on the cluster side that will accept Coherence*Extend client TCP connections.

The next few pages describe each of these steps in detail.

Client-side Cache Configuration Descriptor

```
<remote-cache-scheme>
  <scheme-name>extend-dist</scheme-name>
  <service-name>ExtendTcpCacheService</service-name>
  <initiator-config>
    <tcp-initiator>
      <remote-addresses>
        <socket-address>
          <address>MyProxyHostname1</address>
          <port>9099</port>
        </socket-address>
        <socket-address>
          <address>MyProxyHostname2</address>
          <port>9099</port>
        </socket-address>
      </remote-addresses>
      <connect-timeout>10s</connect-timeout>
    </tcp-initiator>
  </initiator-config>
</remote-cache-scheme>
```



Host and port of
the cluster Proxy
Server

client-cache-config.xml

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Client-side Cache Configuration Descriptor

Coherence*Extend clients are configured via the `remote-cache-scheme` element. A `<remote-cache-scheme>` signifies that it is a Coherence*Extend client. The `<tcp-initiator>` must be configured to point at the address and port of a Proxy Service (proxy-scheme) running within the Coherence cluster. It is recommended that POF be used as the serialization method for Coherence*Extend clients. When the client application retrieves a NamedCache via the CacheFactory using, for example, the name "dist-extend", the Coherence*Extend adapter library connects to the Coherence cluster via TCP/IP (using the address "MyProxyHostname1" and port 9099) and returns a NamedCache implementation that routes requests to the NamedCache with the same name running within the remote cluster. Note that the `<remote-addresses>` configuration element can contain multiple `<socket-address>` child elements. The Coherence*Extend adapter library attempts to connect to the addresses in a random order, until either the list is exhausted, or a TCP/IP connection is established.

Cluster-side Cache Configuration Descriptor

```

<proxy-scheme>
  <service-name>ProxyService1</service-name>
  <thread-count>2</thread-count>
  <acceptor-config>
    <tcp-acceptor>
      <local-address>
        <address>MyProxyHostname1</address>
        <port>9099</port>
        <reusable>true</reusable>
      </local-address>
    </tcp-acceptor>
  </acceptor-config>
  <autostart>true</autostart>
</proxy-scheme>
<proxy-scheme>
  <service-name>ProxyService2</service-name> . . .
  <acceptor-config>
    <tcp-acceptor>
      <local-address>
        <address>MyProxyHostname2</address>
        <port>9099</port>
      </local-address>
    </tcp-acceptor>
  </acceptor-config>
  <autostart>true</autostart>
</proxy-scheme>
. . .

```

Host and port match
the settings in the
client-side
configuration

coherence-cache-config.xml



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Cluster-side Cache Configuration Descriptor

Cluster nodes that need to support Coherence*Extend clients must be configured with a `proxy-scheme`. A `proxy-scheme` configures a Proxy Service to run within a cluster member. `Proxy-scheme's` must be configured with a network address and port to listen for requests. As mentioned in the previous slide, POF is the recommended serialization method for Coherence*Extend applications.

For a Coherence*Extend-TCP client to connect to a Coherence cluster, one or more `DefaultCacheServer` processes must be running that use a Coherence cache configuration descriptor, which includes a `<proxy-scheme>` element with a child `<tcp-acceptor>` element containing various TCP/IP-specific configuration information.

This cache configuration descriptor defines a Coherence*Extend TCP Proxy scheme that allows remote Coherence*Extend TCP clients to connect to the Coherence cluster. The `<proxy-scheme>` element has a `<tcp-acceptor>` child element, which includes all TCP/IP-specific information needed to accept client connection requests over TCP/IP. Because this descriptor is used by a `DefaultCacheServer`, it is important that the `<autostart>` configuration element for each service is set to `true` so that the clustered services are automatically restarted upon termination.

Cluster-side Cache Configuration Descriptor (continued)

The Coherence*Extend clustered service listens to a TCP/IP ServerSocket (bound to address "MyProxyHostname1" and port 9099) for connection requests. When a client attempts to connect to a Coherence NamedCache called "dist-extend-direct", the Coherence*Extend clustered service proxies subsequent requests to the NamedCache with the same name which, in this case, is a partitioned cache.

The slide shows you how to configure Coherence*Extend in the XML configuration file. First, you must configure it on the server:

- You must have a Proxy scheme defined.
- You specify the names of the Coherence servers that you are running, which in this example are `ProxyService1` and `ProxyService2`.
- You allocate a thread count and a typical starting point as 1 thread per 5 to 10 clients. For example, if you want to connect 50 clients to Coherence, you probably should start with approximately 10 threads. This is something that you can configure.
- You also specify the host name and port that the Coherence proxy will use. A Proxy Service will be started that listens for TCP/IP connections from the network, and also maintains those connections. The default port is 9099. However, you can also use any port. Multiple proxies can run on the same physical server. A highly-available configuration requires different ports and multiple proxies. Coherence automatically fails over to another proxy if the first proxy fails.
- When `<autostart>` is set to `true`, Coherence automatically starts the proxy when the Coherence node is started.

Launching a Coherence*Extend-Enabled DefaultCacheServer Process and Java Client Application

```
java -cp coherence.jar:<classpath to shared jars>
      -Dtangosol.coherence.cacheconfig=file://<path to the
      server-side cache configuration descriptor>
      com.tangosol.net.DefaultCacheServer
```

Server

```
java -cp coherence.jar:<classpath to client application>
      -Dtangosol.coherence.cacheconfig=file://<path to the
      client-side cache configuration descriptor>
      <client application Class name>
```

Client



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Launching a Coherence*Extend-Enabled DefaultCacheServer Process and Client Application

To start a DefaultCacheServer that uses the cluster-side Coherence cache configuration discussed earlier to allow Coherence*Extend-TCP clients to connect to the Coherence cluster via TCP/IP, you must perform the following:

- Change the current directory to the Coherence library directory (%COHERENCE_HOME%\lib on Windows and \$COHERENCE_HOME/lib on Unix).
- Make sure that the paths are configured so that the Java command will run.
- Start the DefaultCacheServer command line application with the -Dtangosol.coherence.cacheconfig system property set to the location of the cluster-side Coherence cache configuration descriptor that was discussed earlier.

The client steps are almost identical. To start a client application that uses Coherence*Extend-TCP to connect to a remote Coherence cluster via TCP/IP, you must perform the following:

Launching an Extend-Enabled DefaultCacheServer Process and Client Application (continued)

- Change the current directory to the Coherence library directory (%COHERENCE_HOME%\lib on Windows and \$COHERENCE_HOME/lib on Unix).
- Make sure that the paths are configured so that the Java command will run.
- Start your client application with the -Dtangosol.coherence.cacheconfig system property set to the location of the client-side Coherence cache configuration descriptor that was discussed earlier.

Quiz

A Coherence Proxy Service cluster configuration provides:

- a. A TCP/IP bridge for Coherence cluster nodes
- b. A TCP/IP address for Coherence*Extend clients to connect
- c. Connectivity for C++ TCMP Coherence*Extend clients
- d. A TCMP bridge for Coherence*Extend client connectivity



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: b

Agenda

Concepts of Coherence*Extend

Configure and use Coherence*Extend

Coherence*Web

- What is Coherence*Web?
- Coherence*Web Architecture
- When to use Coherence*Web
- Configuring and Running Coherence*Web



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is Coherence*Web?

- Coherence*Web:
 - Is an HTTP session state-management framework for any J2EE application server
 - Is certified with WebLogic, GlassFish, WebSphere, JBoss, Tomcat, SunOne, and so on
 - Replaces the existing HTTP session state replication
- The value of Coherence*Web for application server includes:
 - More sophisticated state management that is policy based
 - Optional offloading of state management to an independent tier from the application server
 - Advanced options for controlling storage of HTTP sessions



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is Coherence*Web?

Within Coherence, there is a high-performance session management implementation called Coherence*Web, which seamlessly adds high-scale and highly reliable clustering of HTTP session data to the Java EE application servers.

Coherence*Web delivers linear scale, even up to hundreds of servers, and provides instant and transparent failover, while ensuring that HTTP session data is never lost. Coherence can be used to manage HTTP session information. This is provided out-of-the-box. This does not require any coding, and the functionality is called Coherence*Web. If your application server currently uses the HTTP session object, you can just plug Coherence into your existing architecture without any coding. Coherence then manages those session objects with all its features and capabilities.

Coherence*Web is certified with WebLogic, GlassFish, Oracle OC4J, JBoss, IBM WebSphere, Apache Tomcat, and SunOne.

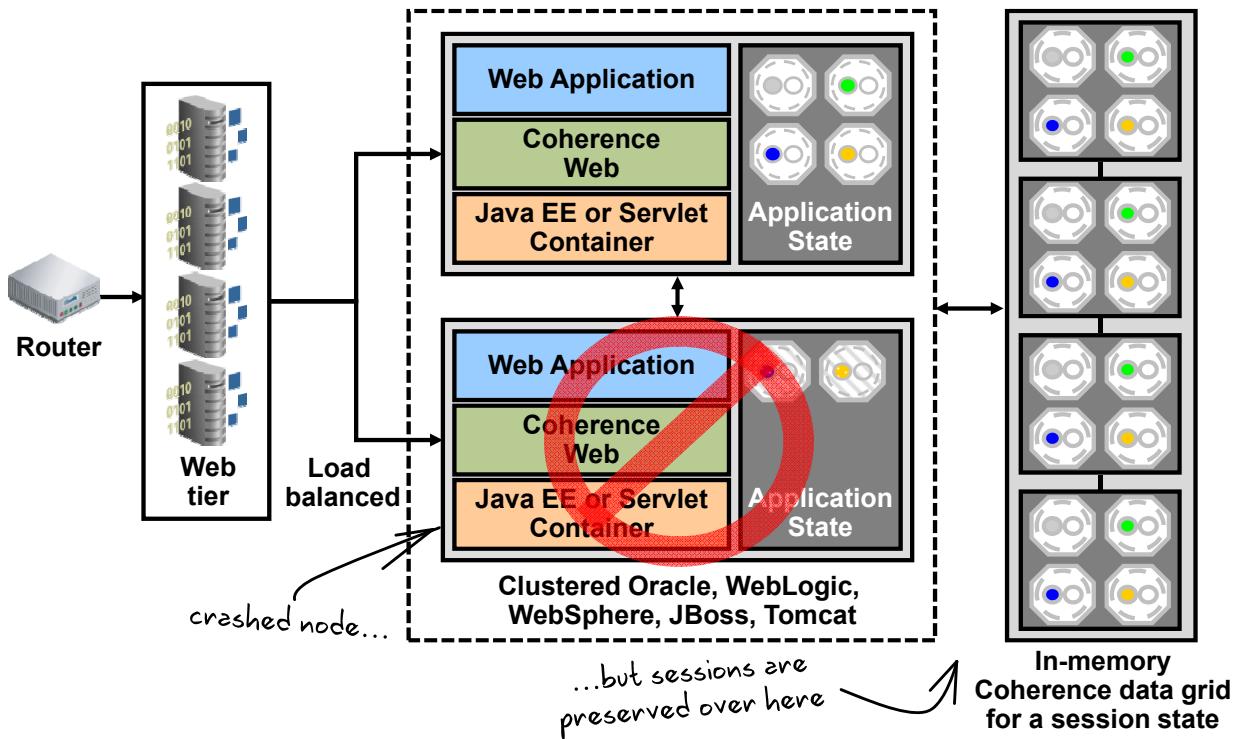
What is Coherence*Web? (continued)

It plugs directly into Oracle WebLogic Server. Note that this capability is only to manage HTTP session objects. However, with your own application code, you can use the full capability of Coherence. If you want to have stateful EJB replication, you can use the existing application server infrastructure. The value that Coherence*Web provides for application servers is more sophisticated, more scalable, and a much more highly-available session management. Thus, with Coherence*Web, even when an application server instance fails, the HTTP session state is still fault-tolerant.

Coherence*Web manages your HTTP Session objects so you can easily bounce application servers without disrupting currently connected clients. Coherence automatically repartitions the session objects across the grid, or if the application servers are running with local storage disabled, then all sessions are intact by default.

Coherence*Web allows you to configure how a session is actually stored in the data grid. One example of this is the Split Session strategy in which the session metadata, and the “small attributes” associated with that session, are stored in one cache. The “large” attributes associated with that same session are stored in a separate cache. This allows for quick access to the most commonly requested data, while still maintaining accessibility to the entire session state.

Coherence*Web Architecture



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence*Web Architecture

The diagram in the slide illustrates that if a server were to fail, if you lose the entire application server or a JVM, Coherence still maintains the data. Therefore, you do not lose any session state information. Using your existing infrastructure, the user session fails over to one of the other application servers, or the one that is surviving. Therefore, no data is lost. The end user may not even notice that there is an outage. The user session continues without the user even noticing that anything went wrong because Coherence manages the data.

One common use case for Coherence clustering is to manage user sessions (conversational state) in the cluster. This capability is provided by the Coherence*Web module, which is a built-in feature of Oracle Coherence. Coherence*Web provides linear scalability for HTTP Session Management in clusters of hundreds of production servers. It can achieve this linear scalability because, at its core, it is built on Oracle Coherence dynamic partitioning.

Session management highlights the scalability problem that typifies shared data sources: If an application cannot share data across the servers, it would have to delegate that data management entirely to the shared store, which is typically the application's database. If the HTTP session is stored in the database, each HTTP request (in the absence of sticky load-balancing) would require a read from the database, causing the desired reads-per-second from the database to increase linearly with the size of the server cluster.

Coherence*Web Architecture (continued)

Further, each HTTP request causes an update of its corresponding HTTP session. Therefore, regardless of sticky load balancing, to ensure that HTTP session data is not lost when a server fails, the desired writes-per-second to the database also increases linearly with the size of the server cluster. In both cases, the actual reads and writes per second that a database is capable of does not scale in relation to the number of servers requesting those reads and writes, and the database quickly becomes a bottleneck, forcing availability, reliability (for example, asynchronous writes), and performance compromises. Additionally, each read from a database has an associated latency that is related to performance, and that latency increases dramatically as the database experiences increasing load.

Coherence*Web, on the other hand, has the same latency in a 2-server cluster as it has in a 200-server cluster, because all the HTTP session read operations that cannot be handled locally are spread out evenly across the rest of the cluster, and all update operations (which must be handled remotely to ensure survival of the HTTP sessions) are, likewise, spread out evenly across the remaining cluster. The result is linear scalability with constant latency, regardless of the size of the cluster.

Lastly, partitioning supports linear scalability of both data capacity and throughput. It accomplishes the scalability of data capacity by evenly balancing the data across all servers, so four servers can naturally manage two times as much data as two servers.

Scalability of throughput is also a direct result of load-balancing data across all servers, because as servers are added, each server is able to utilize its full processing power to manage a smaller and smaller percentage of the overall data set. For example, in a 10-server cluster, each server has to manage 10% of the data operations, and—because Oracle Coherence uses a peer-to-peer architecture—10% of those operations come from each server. With 10 times that many servers (that is, 100 servers), each server manages only 1% of the data operations, and only 1% of those operations come from each server. But there are 10 times as many servers, so the cluster is accomplishing 10 times the total number of operations.

In the 10-server example, if each of the 10 servers issues 100 operations per second, they would each send 10 of those operations to each of the other servers. The result would be each server receiving 100 operations (10×10) that it is responsible for processing.

In the 100-server example, each server would still issue 100 operations per second, but each would send only one operation to each of the other servers. So the result would be each server receiving 100 operations (100×1) that it is responsible for processing.

This linear scalability is made possible by modern switched network architectures that provide backplanes that scale linearly to the number of ports on the switch, providing each port with dedicated, fully duplexed (upstream and downstream) bandwidth. Because each server sends and receives only 100 operations (in both the 10-server and 100-server examples), the network bandwidth utilization is roughly constant per port regardless of the number of servers in the cluster.

Configuring and Running Coherence*Web

Oracle by Example (OBE) online tutorials:

- Fusion Middleware: Search for Coherence
 - Setting up an Eclipse Development Environment
 - Using Oracle Coherence*Web: Part 1 – Installation
 - Using Oracle Coherence*Web: Part 2 – Application Development

<http://otn.oracle.com/obe>



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuring and Running Coherence*Web

There are Oracle by Example (OBE) online tutorials that provide hands-on instructions for configuring and running Coherence*Web using Eclipse. The OBEs were written for Coherence 3.4 and 3.5, but will work with Coherence 3.6.

Quiz

What is the primary purpose of Coherence*Web?

- a. To use Coherence caching for application data
- b. To integrate Coherence with application servers
- c. To use Coherence caching for HTTP Sessions
- d. To use Coherence caching for client connections



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: c

Practice.11.01 Overview: Configuring and running Coherence*Extend

This practice covers the following topics:

- Configuring a Proxy Service
- Running a Proxy Service



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.11.01 Overview: Configuring and running a Coherence*Extend

This practice has you configuring Coherence*Extend Proxy Service configuration and running the Proxy Service as part of a Coherence cluster. This Proxy Service is then ready for Coherence*Extend client connections.

Practice.11.02 Overview: Writing a Coherence*Extend Java client

This practice covers the following topics:

- Writing a Coherence*Extend Java client
- Using the Java client to connect to a Coherence cluster via a Proxy Service



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.11.02 Overview: Writing a Coherence*Extend Java client

This practice covers writing a Java Coherence*Extend client that uses the previously configured Proxy Service from practice 11.01 to connect to a Coherence cache.

Practice.11.03 Overview: Writing a Coherence*Extend C++ Client

This practice covers the following topics:

- Examining the code for a C++ Coherence*Extend client that uses POF serialization
- Using the C++ client to connect to a Coherence cluster via a proxy server



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.11.03 Overview: Writing a Coherence*Extend C++ Client

This practice covers examining and running a C++ Coherence*Extend client that uses the previously configured proxy server from practice 11.01 to connect to a Coherence cache.

Summary

In this lesson, you should have learned how to:

- Describe what Coherence*Extend does, its architecture, its advantages and disadvantages, and when to use it
- List and explain the three types of Coherence*Extend clients and their uses
- Configure, run, and tune Coherence*Extend
- Explain the concepts of Coherence*Web
- Configure and run Coherence*Web



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and ORACLE CORPORATION use only

12

Coherence Administration

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Identify and describe the primary management capabilities of Coherence
- Describe the JMX Reporter, and use the Reporter's out-of-the-box (OOTB) reports to manage capacity and troubleshoot problems
- List the Oracle management tools available for managing Coherence clusters
- Size a Coherence data grid
- Identify and describe the basic tasks for performance tuning Coherence cache clusters



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Coherence JMX Management

- Configuring Coherence JMX
- Accessing the Coherence MBean
- System MBeans to Watch

Coherence Reporter

Cache and Cluster Management

Production Checklist



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Overview

Coherence includes facilities for managing and monitoring Coherence resources via JMX:

- Enterprise Edition and Grid Edition:
supports JMX statistics for the entire cluster from any member
- Standard Edition:
supports JMX statistics for the local node *only*
- With JMX, administrators can monitor Coherence Cluster with any JMX-compliant tool
 - JConsole
 - MC4J
 - and so on

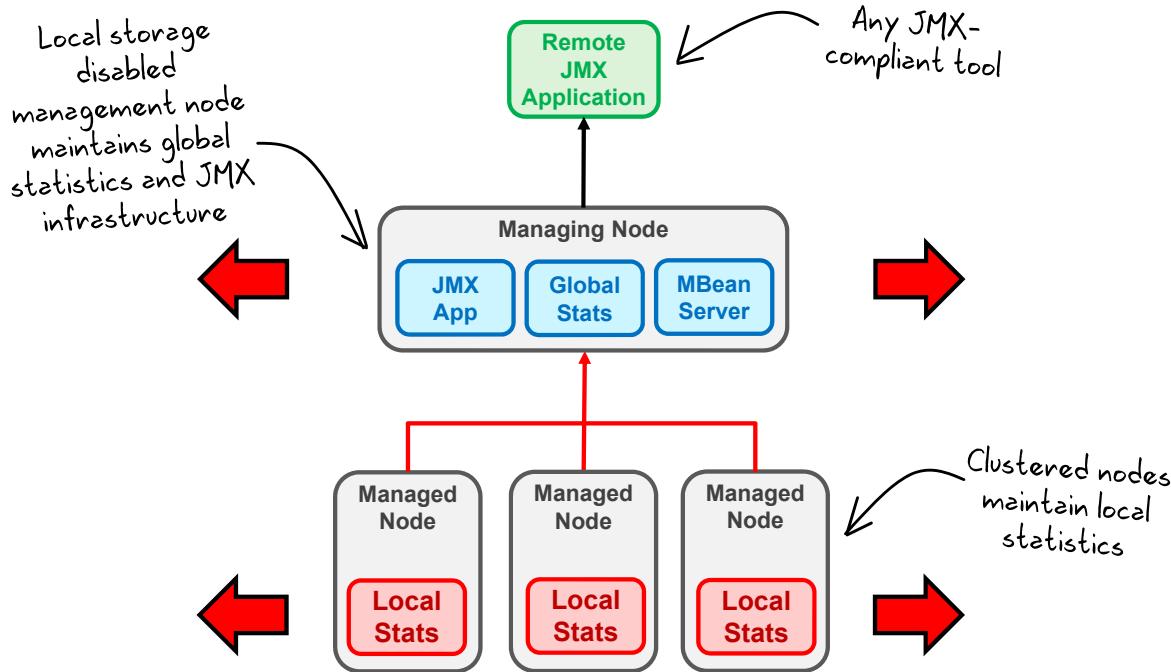


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Overview

Coherence offers a JMX management infrastructure that provides statistics about a running cluster. Any JMX-compliant tool can connect to a Coherence management node and monitor the statistics to analyze the system at runtime.

Management Architecture



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Management Architecture

Coherence provides a JMX-based infrastructure that exposes the statistics of a running cluster. It involves configuring a management node that is part of the cluster that gathers statistics from each of the cluster nodes so administrators can monitor how Coherence is performing. The management node provides a JMX connection that can be used by any JMX-compliant tool to read the MBean settings and statistics of the running cluster. It is a best practice to run the management node in storage-disabled mode.

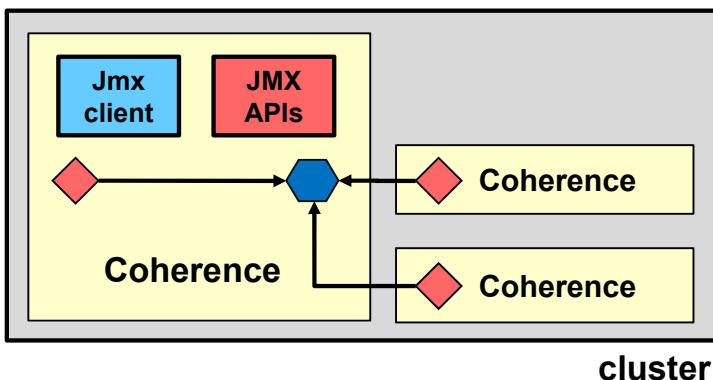
Configuring Coherence JMX

Coherence MBeanServer

-Dtangosol.coherence.management=all

Coherence System MBean

-Dtangosol.coherence.management.remote=true



Prior to J2SE 5.0, need to add JMX libraries into the classpath manually

Use of a dedicated JMX cluster node is a common pattern

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuring Coherence JMX

JMX management is enabled by configuring one or more nodes to host an MBean server, which manages the managed objects of all the other cluster nodes. The use of dedicated JMX cluster members is a common pattern because it avoids loading JMX software into every single cluster member, while still providing fault-tolerance in the event a single JMX member fails. JMX management in the cluster is disabled by default. To enable JMX management, add a `<managed-nodes>` element within the `<management-config>` element in an operational override file. The `<managed-nodes>` element specifies whether a cluster node's JVM has an in-process MBean server, and if so, whether the node allows management of other nodes' managed objects. Values for the `<managed-nodes>` and `<allow-remote-management>` elements can also be set using Java system properties shown on this slide.

The `<managed-nodes>` element supports the following values:

- `none`: No MBean server is instantiated.
- `local-only`: Manage only MBeans which are local to the cluster node.
- `remote-only`: Manage MBeans on other remotely manageable cluster nodes. See `<allowed-remote-management>` subelement.
- `all`: Manage both local and remotely manageable cluster nodes.

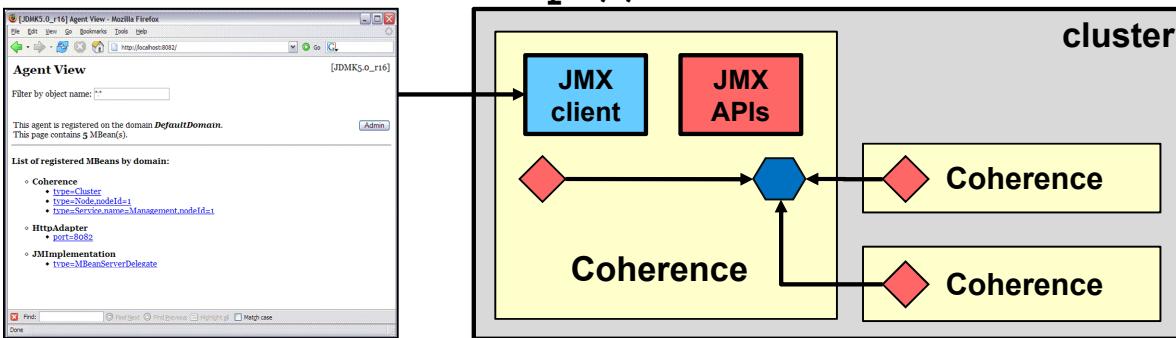
Accessing the Coherence MBean using HTTP and JMX RI

Use the JMX RI to access the Coherence MBean via HTTP:

- Run a Coherence instance with:

```
java -cp jmxri.jar;jmxtools.jar;coherence.jar
      -Dtangosol.coherence.management=all
      -Dtangosol.coherence.management.remote=true
      com.tangosol.net.CacheFactory
```

- Run the command "jmx 8082" from the Coherence console and access `http://host:8082`



ORACLE

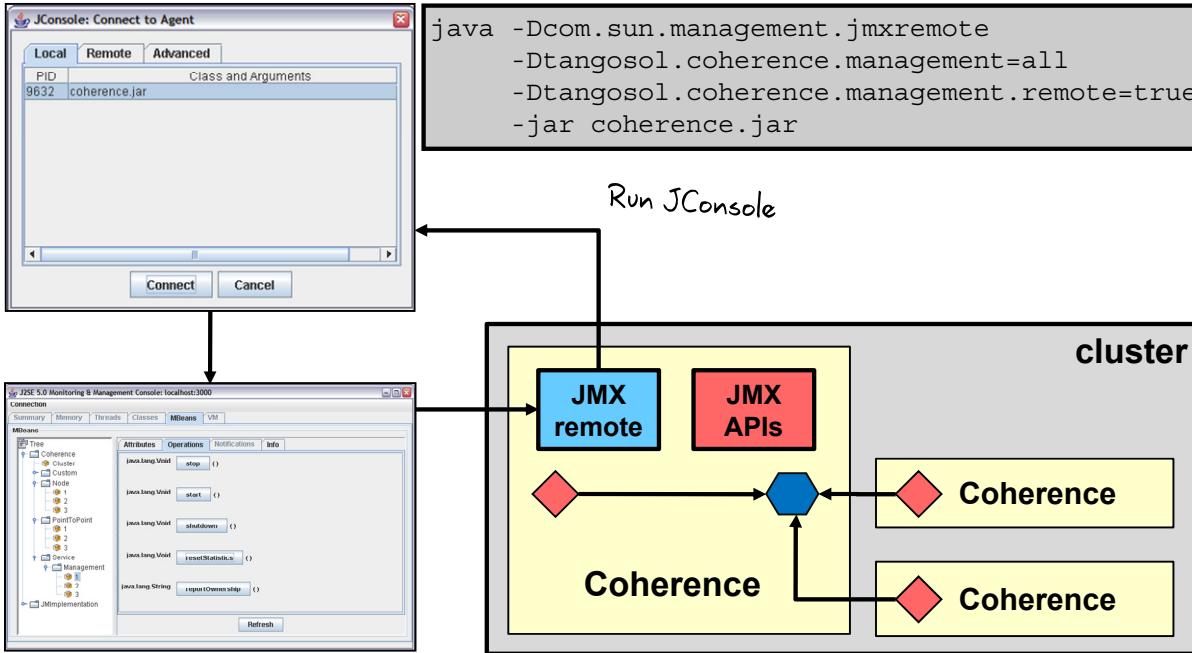
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Accessing the Coherence MBean using HTTP and JMX RI

The HTML Adapter Web Application uses the HTTP adapter (`HtmlAdaptorServer`) that is shipped as part of the JMX reference implementation (`jmxtools.jar`). To run the Web application, start a Coherence node as shown on the slide. After the Coherence command line application starts, enter `jmx 8082`. This starts an HTTP adaptor on `http://localhost:8082` in the cluster node's JVM, and makes the cluster node an MBeanServer host.

Accessing the Coherence MBean using Java bundled JMX Implementation

Run a Coherence instance with :



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Accessing the Coherence MBean using Java bundled JMX Implementation

The JConsole utility, located in the JDK /bin directory, can be used to view and manipulate Coherence MBeans. To do so, set the system properties when starting a Coherence node as shown in this slide. When the node has started, launch the JConsole utility, and open a new connection to the JVM process running the Coherence command line application.

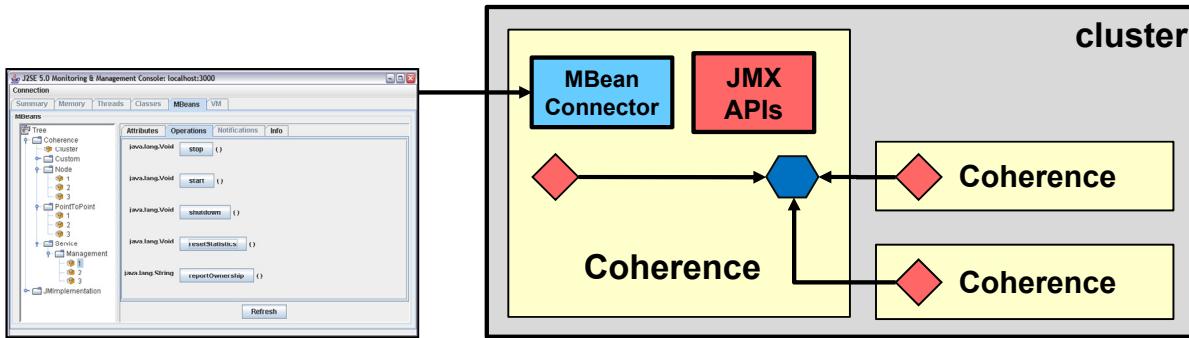
Accessing the Coherence MBean using the Coherence MBeanConnector

- Run a Coherence instance with: *No command prompt*

```
java -Dtangosol.coherence.management=all
      -cp coherence.jar com.tangosol.net.management.MBeanConnector -rmi
```

- Run JConsole from J2SE5.0 and access the URL:

```
service:jmx:rmi://host:3000/jndi/rmi://host:9000/server
```



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Accessing the Coherence MBean using the Coherence MBeanConnector

Coherence ships with a program to launch a cluster node as a dedicated MBean server host. This program provides access to Coherence MBeans by using the JMX Remote API using RMI, or the HTTP server provided by Sun's JMX RI. The RMI and HTTP ports can be configured, allowing for access through a firewall. The server is started using the command shown in the slide. To allow access by using JMX RMI, include the `-rmi` flag. To allow access by using HTTP and a Web browser, include the `-http` flag. Both flags may be included; however at least one must be present for the node to start.

System MBeans to Watch

Node MBean:

- Publisher/Receiver SuccessRate (should be .98 or higher)
- Publisher/Receiver Packet Utilization
- Weakest Node (Is this node having problems?)
- SendQueueSize

Point-to-Point MBean:

- Threshold
- Deferring
- PauseRate

Service MBean:

- Request/Task Average Duration

Cache MBean:

- Size, Units, HitProbability



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

System MBeans to Watch

- **Publisher SuccessRate:** The publisher success rate for this cluster node since the node statistics were last reset. Publisher success rate is a ratio of the number of packets successfully delivered in a first attempt to the total number of sent packets. A failure count is incremented when there is no ACK received within a timeout period. It could be caused by either very high network latency, or a high packet drop rate. This should be .98 or higher on a running cluster.
- **Receiver SuccessRate:** The receiver success rate for this cluster node since the node statistics were last reset. Receiver success rate is a ratio of the number of packets successfully acknowledged in a first attempt to the total number of received packets. A failure count is incremented when a re-delivery of previously received packet is detected. It could be caused by either very high inbound network latency, or lost ACK packets. This should be .98 or higher on a running cluster.

System MBeans to Watch (continued)

- **Publisher Packet Utilization:** The publisher packet utilization for this cluster node since the node socket was last reopened. This value is a ratio of the number of bytes sent to the number that would have been sent had all packets been full. A low utilization indicates that data is not being sent in large enough chunks to make efficient use of the network. In practice, this will not cause stability problems, but is something to be considered if looking for maximum performance.
- **Receiver Packet Utilization:** The receiver packet utilization for this cluster node since the socket was last reopened. This value is a ratio of the number of bytes received to the number that would have been received had all packets been full. A low utilization indicates that data is not being sent in large enough chunks to make efficient use of the network.
- **Point-to-Point Threshold:** The maximum number of outstanding packets for the viewed member that the viewing member is allowed to accumulate before initiating the deferral algorithm. Not useful by itself; Deferring is a more useful statistic.
- **Point-to-Point Deferring:** Indicates whether or not the viewing member is currently deferring packets to the viewed member.
- **PauseRate:** The percentage of time since the last time statistics were reset in which the viewing member considered the viewed member to be unresponsive. Under normal conditions, this value should be very close to 0.0. Values near 1.0 would indicate that the viewed node is nearly inoperable, likely due to extremely long GC pauses. This becomes less meaningful over time (just like any percentage based statistic); using the reporter to track deltas on this statistic is more useful.
- **Service MBean Request attributes:** There are a number of request-related attributes that can help monitor the health of a running cluster.
- **Service MBean TaskAverageDuration:** The average duration (in milliseconds) of an individual task execution.
- **Cache MBean Size:** The number of entries in the cache.
- **Cache MBean Units:** The size of the cache measured in units. This value needs to be adjusted by the UnitFactor.
- **Cache MBean HitProbability:** The rough probability ($0 \leq p \leq 1$) that the next invocation will be a hit, based on the statistics collected since the last time statistics were reset.

A complete description of these statistics can be found in the documentation:

http://download.oracle.com/otn_hosted_doc/coherence/350/com/tangosol/net/management/Registry.html

Quiz

Which of the following management tools does Coherence provide?

- a. SNMP MIB
- b. JMX MBeans
- c. Administration Console
- d. Scripting Tools



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: b

Coherence provides a JMX MBean infrastructure that contains runtime metric data for a running Coherence cluster. This infrastructure can be leveraged by JMX-capable tools for monitoring and managing Coherence applications.

Agenda

Coherence JMX Management

Coherence Reporter

- What is the Reporter?
- Configuring Basic Settings
- Managing Reporter MBean Attributes and Operations
- Finding Reporter Log Data
- Viewing Reporter Data
- Creating Custom Reports
- Running Reporter in a Distributed Environment

Cache and Cluster Management

Production Checklist



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is the Reporter?

The reporter is an out-of-the-box JMX snapshot and delta utility for troubleshooting development systems:

- Benefits
 - Easy to use
 - Easy to transmit the data
 - Zero setup
 - Configurable/Custom reports
- Limitations
 - No data management capability
 - No visualization
 - No subsetting of data

WARNING: The Reporter creates a large volume of information, and there must be a plan for archiving and removing Reporter log data BEFORE starting the Reporter!



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is the Reporter?

The Reporter is a Coherence management tool that provides canned reports for a cluster's management data. The Reporter is very easy to set up, and generates tab-delimited output files containing the management data. The Reporter also allows for the creation of custom reports that can be run along with prepackaged default reports. Combining the two can provide the full coverage needed for the application overall. Currently, there are no data management tools, so a comprehensive archiving or removal plan is required to keep the file system from filling up. There are also no graphical tools for visualizing the raw data produced by the Reporter, and no tools for dissecting the data for only the parts that are desired.

Configuring Basic Settings

Enabling the Reporter with basic content requires setting system properties:

- On the "Management" node:

```
-Dtangosol.coherence.management.report.autostart=true  
-Dtangosol.coherence.management=all  
-Dcom.sun.management.jmxremote
```

- On the "Managed" node:

```
-Dtangosol.coherence.management.remote=true
```

- To view all prepackaged reports (on "Management" node):

```
-Dtangosol.coherence.management.report.configuration=reports/report-all.xml
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuring Basic Settings

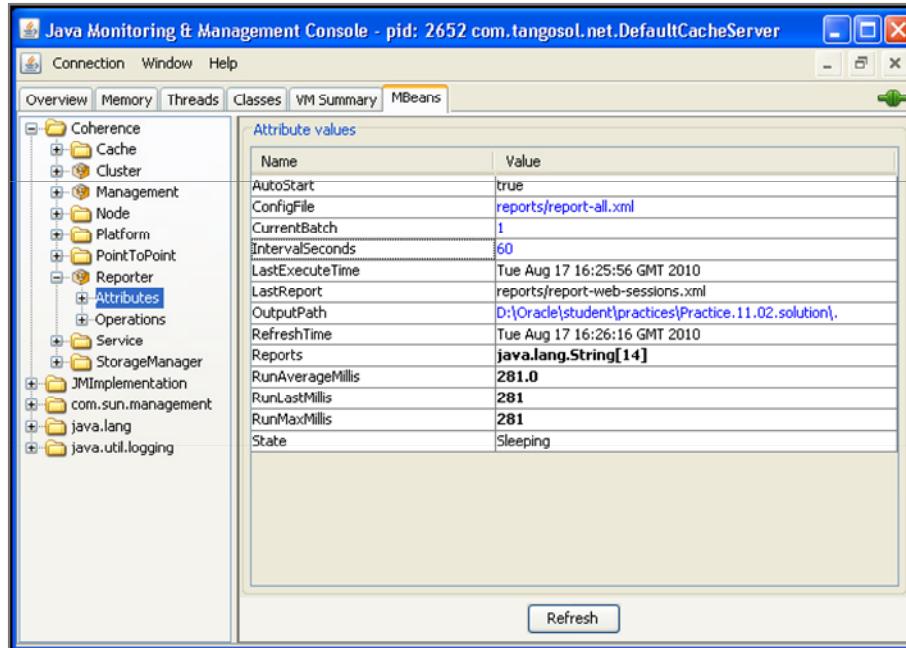
Configuring the Reporter involves the same settings for configuring JMX management, and also includes adding the

`-Dtangosol.coherence.management.report.autostart=true` system property for the management node. Alternatively, the Reporter can be started using a JMX MBean browser to run the Reporter's `start()` operation. Basic configuration will create a single Reporter node that will log the JMX statistics for all nodes in the cluster. The log files will be placed in the working directory of the application. By default, the report configuration file setting is `reports/report-group.xml`, which contains a subset of prepackaged reports offered by the Reporter. Setting the property

`tangosol.coherence.management.report.configuration=reports/reports-all.xml` configures the Reporter to run all of the prepackaged reports available with Coherence. All of the prepackaged reports, `report-group.xml`, and `report-all.xml` are contained in the `coherence.jar` file by default. This system property can also be used to point to a report configuration file outside of the `coherence.jar` file.

Managing Reporter MBean Attributes

Reporter
MBean
attributes as
shown in
JConsole



ORACLE®

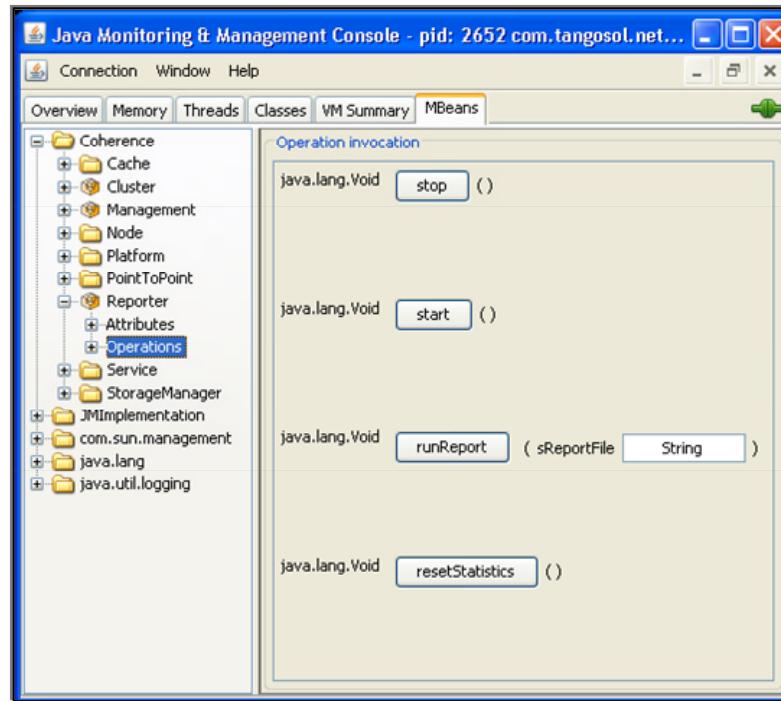
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Managing Reporter MBean Attributes

The JMX Reporter is managed through an MBean under the Coherence Domain. The Reporter MBean provides information related to the status and performance of the Reporter. The MBean also provides the capability to start and stop the service and run a report on demand. This slide illustrates the attributes available to the Reporter MBean. The JConsole is being used to view the MBean.

Managing Reporter MBean Operations

Reporter MBean operations as shown in JConsole



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Managing Reporter MBean Operations

This slide illustrates the operations available to the Reporter MBean:

- `stop()`: Stops the Reporter from generating reports.
- `start()`: Causes the Reporter to start generating reports.
- `runReport()`: Allows a report XML file as input for execution. For example, entering `reports/report-cache-size.xml` would result in the Reporter running that report.
- `resetStatistics()`: Causes the Reporter to continue generating reports with refreshed statistics that set back to their original values.

For a full description of the Reporter Attributes see the Reporter section of the javadoc.

Finding Reporter Log Data

Files generated by the Reporter:

File Name	Description
YYYYMMDDHH-memory-status.txt	Contains memory and garbage collection information about each node.
YYYYMMDDHH-network-health.txt	Contains the publisher success rates and receiver success rates for the entire grid.
YYYYMMDDHH-network-health-detail.txt	Contains the publisher success rates and receiver success rates for each node.
YYYYMMDDHH-node.txt	Contains the list of nodes that were members of the grid.
YYYYMMDDHH-service.txt	Contains Request and Task information for each service.
YYYYMMDDHH-proxy.txt	Contains utilization information about each proxy node in the grid.
YYYYMMDDHH-cache-usage.txt	Contains cache utilization (put, get, and so on) statistic for each cache.



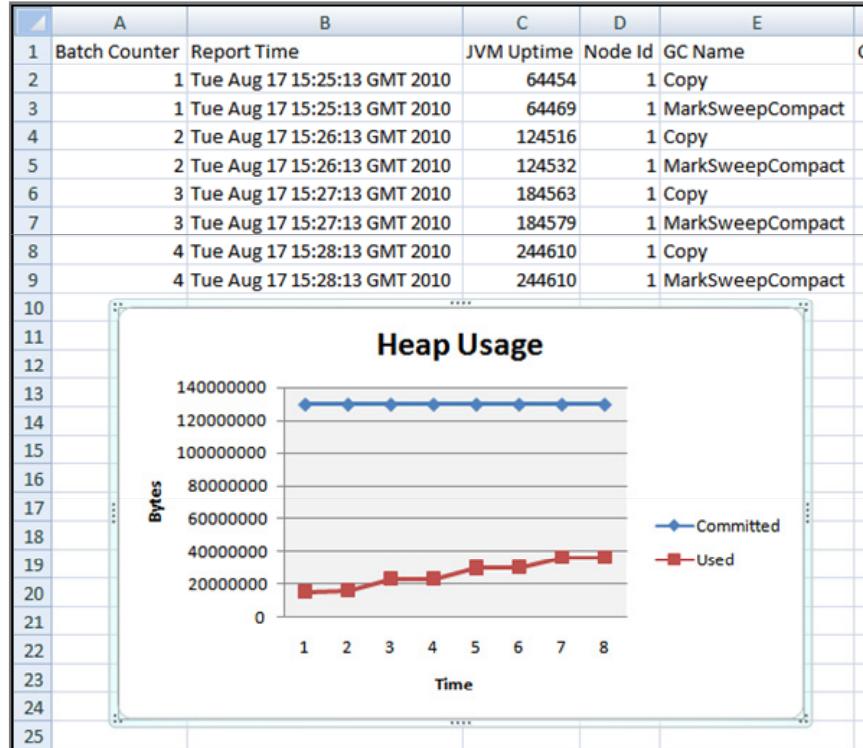
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Finding Reporter Log Data

Seven files are created each hour by the Reporter. Each file is prefixed with the date and hour the report was executed in a YYYYMMDDHH format. This allows for easy location and purging of unwanted information. The files generated are described in the table depicted on this slide. See the Coherence documentation for a complete description of the data contained in each file.

Viewing Reporter Data

Example
Microsoft Excel
chart of
Coherence
memory status



ORACLE

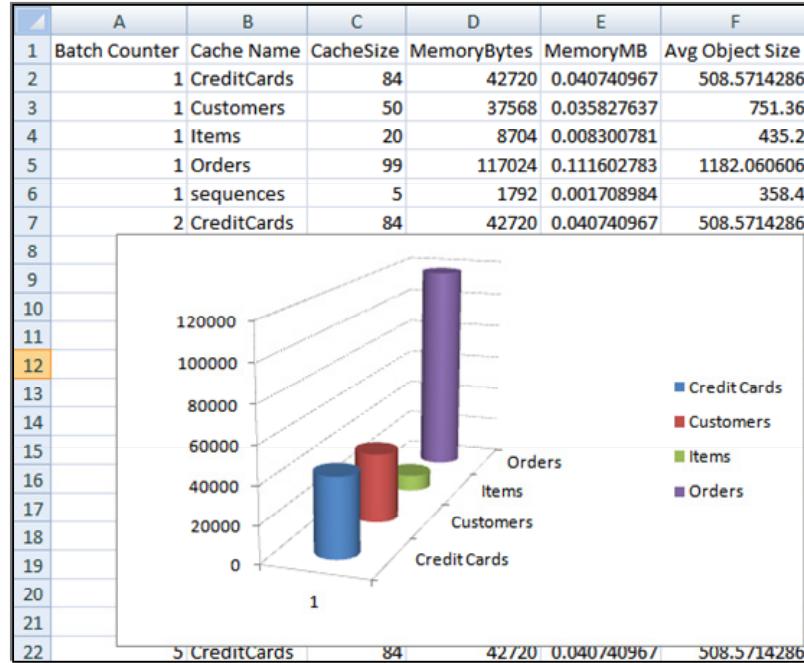
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Viewing Reporter Data

Here is an example of the memory status file generated by the Reporter. The file is opened using Microsoft Excel, and a chart is created using some of the data points to visualize the heap usage of the cluster. In this case, it is a single node cluster, but all nodes in the cluster that have enabled the Reporter would be included in the file.

Viewing Reporter Data

Example
Microsoft Excel
chart of
Coherence
cache sizes

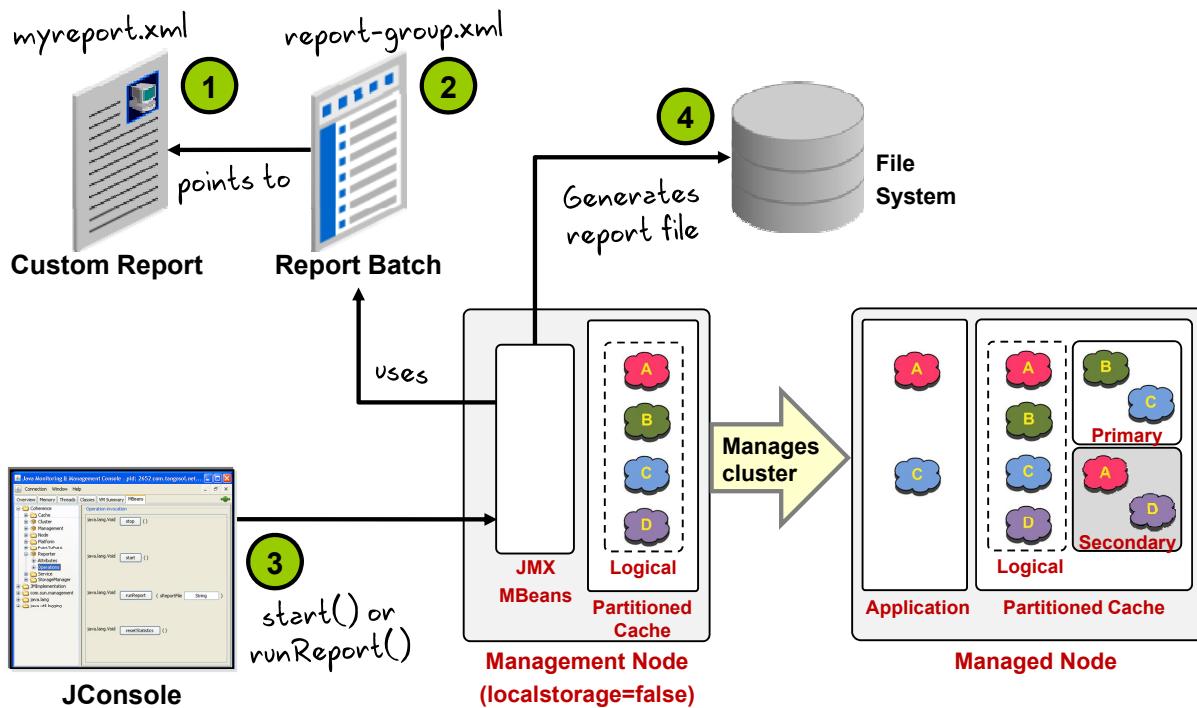


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Viewing Reporter Data (continued)

Here is an example of the cache size file generated by the Reporter. The file is opened using Microsoft Excel, and a chart is created using some of the data points to visualize the sizes of the various caches in the cluster.

Creating Custom Reports



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Creating Custom Reports

Coherence allows for the creation of custom reports. A report configuration file is used to specify what data to include in the custom report. This configuration includes several powerful ways to direct the Reporter to collect data from the cluster. Some of the features available for collecting report data are:

- Including an attribute obtained from a query pattern
- Information from composite attributes (a part of a complex attribute)
- Information from multiple MBeans
- Report macros
- Constant values
- JMX MBean queries
- Filters to limit returned data
- Functions (for example, add, subtract, multiply, and divide)
- Aggregates (for example, average, maximum, minimum, and sum)
- Delta functions (returns the difference between last run and the current run of the report)

Creating Custom Reports (continued)

The following steps are required to create a custom report:

1. Create the custom report configuration file. See the documentation for a complete list of how to create a report configuration file. After a custom report has been created, it can be included in a report batch, and executed on a specified time interval by the ReportControl MBean. For a complete description of the report configuration XML file, see the `report-config.dtd`, which is packaged in the `coherence.jar` file.
2. Update report batch to execute the report. This involves updating a `report-group.xml` file to include (or point to) the custom configuration file so it is included as part of the reports run by the Reporter.
3. Run on demand. This can be done using an MBean console to invoke the `start()` or `runReport()` operations on the Reporter MBean.
4. The Reporter runs the report and generates the report file.

Running Reporter in a Distributed Environment

- On the managing node:

```
-Dtangosol.coherence.management.report.autostart=false  
-Dtangosol.coherence.management.report.distributed=true  
-Dtangosol.coherence.management=all  
-Dcom.sun.management.jmxremote
```

- On the managed node:

```
-Dtangosol.coherence.management.report.autostart=true  
-Dtangosol.coherence.management.report.distributed=true  
-Dtangosol.coherence.management=local-only  
-Dtangosol.coherence.management.remote=true
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Running Reporter in a Distributed Environment

When running in distributed mode, each node logs local JMX statistics while allowing for centralized management of the Reporters. To enable this configuration, set the system properties as shown in this slide. A distributed configuration is only recommended in situations where grid stability is an issue. In this configuration, the distributed reporters will run independently, and the execution times will not align. Therefore, grid level analysis is extremely difficult, but node level analysis during periods when nodes may be leaving or joining the grid will still be available.

Quiz

What value does the Reporter add to the Coherence JMX infrastructure? (Select all that apply)

- a. Comprehensive reports based on collecting related metrics
- b. Historical data written to logs for troubleshooting issues
- c. Less configuration to set up than JMX management
- d. Runs on each cluster node so it is closer to the data



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: a,b

Practice.12.01 Overview: Configuring and Running the Reporter

This practice covers the following topics:

- Configuring a Coherence management node
- Configuring the Reporter to run on the management node
- Configuring a cluster node to run as a managed node
- Starting a cluster that includes the management node
- Connecting to the Coherence management server using JConsole to review MBean attributes and operations
- Running a report
- Viewing the reports generated by the Reporter



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.12.01 Overview: Configuring and Running the Reporter

This practice demonstrates configuring a JMX management server and the Reporter, changing some configuration settings to run all reports, analyzing Coherence and Reporter MBeans, and reviewing generated reports.

Agenda

Coherence JMX Management

Coherence Reporter

Cache and Cluster Management

- Key Challenges for Coherence Admins
- Coherence Management Pack for Oracle Enterprise Manager
- Third-Party Management tools

Production Checklist



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Understanding Coherence Administrator Needs

Coherence Administrators need the following:

- Visibility of clusters, members, and services
- Visibility into cache performance and throughput
- Visibility of health of other tiers
- Tools to propagate run-time configuration changes
- Tools to scale-out and provision Coherence clusters
- A way to maintain cluster stability in case of departing nodes

Follow ITIL practices when using Coherence applications:

- Event/Alert Management
- Config Management



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Understanding Coherence Administrator Needs

Although Coherence provides a management infrastructure, it does not provide any management tools itself to assist with operations and administration of a running cluster. It also does not provide a runtime platform to manage and provision the processes that run in the cluster. This means that there is no way to start an entire Coherence cluster from a single administration point. A Coherence cluster typically involves running with numerous JVM processes, and the operations team is left with operating system, homegrown, and external management and monitoring product approaches to manage the cluster as a whole. The most viable way to get the most out of a Coherence cluster is to use a management tool that is Coherence-aware. This tool should listen for certain types of Coherence events, monitor important Coherence metrics, start/stop/restart nodes, and automatically spawn new nodes based on its understanding of how the cluster is running.

ITIL: Information Technology Infrastructure Library, <http://www.itil-officialsite.com/>

Complete Coherence Management: Overview OEM 11gR1

- Simplify monitoring of multiple Coherence clusters using single console
- Proactive performance monitoring for caches, near caches, nodes, and so on, using alerts and notifications
- Default historical monitoring for trend analysis
- Manage cluster storage and nodes:
 - Start new nodes for more storage
 - Stop poor nodes
- Recycle the whole cluster:
 - Stop all cluster nodes
 - Start again, maintaining the topology
- Maintain cluster stability:
 - Automatically fire the same node upon detecting a node crash
- Complete cluster provisioning:
 - Create new cluster
 - Scale up by adding nodes
- Support for Coherence*Web and Coherence*Extend
- Configuration Management
- Diagnose JVM issues using deep JVM diagnostics:
 - Differential heap analysis
 - Cross tier analysis between Coherence and database



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Complete Coherence Management: Overview OEM 11gR1

Oracle Enterprise Manager (OEM) is the only solution for all aspects of Coherence monitoring and management, which includes complete cluster monitoring, automated provisioning, life-cycle management, and configuration management.

- Enterprise Manager provides a dashboard view of the health for the entire cluster, which helps administrators identify and monitor potential problem areas within the cluster. Key information, such as the number of weak nodes, nodes with minimum memory, and departed nodes, quickly tell you the overall health of the cluster and give you an idea of a potential problem. Caches with the lowest "hits-to-gets" ratio are highlighted to help administrators identify potential application issues.
- Continuously collects more than 300 metrics for Coherence. Aggregations generated from the raw metrics provide immense intelligence, allowing Administrators to perform deep diagnostics.
- Performance views shows a variety of charts and metrics, which Administrators can use to find out if the cache is evenly balanced on multiple nodes, and to isolate the worst performing nodes, based on key metrics.

Complete Coherence Management: Overview OEM 11gR1 (continued)

Performance views can be used to view real-time data for real-time analysis, or historical data for strategic analysis.

- View cache performance in context of the applications using it (support for Coherence*Web)
- Oracle Enterprise Manager also allows you to manage the life cycle of the entire cluster from a central console. Nodes can be started or stopped easily, giving complete control to the Administrator.
- Set monitoring policy in such a way that a node can be started on detecting a node crash on any machine, as long as the EM agent is running on that machine.
- EM also helps the Administrator to automate the tasks of provisioning, which are time-consuming and error-prone.

If you get WebLogic management Pack EE, you also get JVM Diagnostics (formerly known as AD4J). You can use this to diagnose any heap or cross tier issues.

Oracle Coherence Support

- One of the nodes is a management node with an MBeanServer
- All Coherence MBeans are registered in this MBeanServer
- Uses bulk management beans used by OEM, which makes collection metrics faster
- **Supported versions: 3.3 and greater**
- JDK 1.5 / 1.6



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Oracle Coherence Support

Coherence Management Pack (CMP) runs as a Management Agent that communicates with a management node that is running as part of the Coherence cluster. All Coherence MBeans are registered in the management server's MBeanServer, so CMP has access to all Coherence metrics at run time. The CMP Management Agent can be present on the same machine as the management node, or on a remote machine. The Management Agent communicates with the management node to collect metrics and propagate runtime configuration changes. CMP uses bulk management beans, which provides better performance for metrics collection.

Proactive Monitoring Using Alerts and Notification

Avoids down time by proactive monitoring

Metric	Comparison Operator	Warning Threshold	Critical Threshold
Average Duration Of Individual Task Execution(ms)	>		
Average No. Of Active Threads	>		
Average Time For Gets (ms)	>	600	800
Average Time For Hit(ms)	>		
Average Time For Miss(ms)	>	100	150
Average Time For Put Invocation(ms)	>		
Average Time Spent For Read Operation (ms)	>		
Average Time Spent For Write Operation (ms)	>		
Cache Hits	>	500	1000
Cache Hits Rate	>		
Cache Hits to Gets Ratio	<	40	20

Granularity at each component level e.g. set different thresholds per node, per cache, etc

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

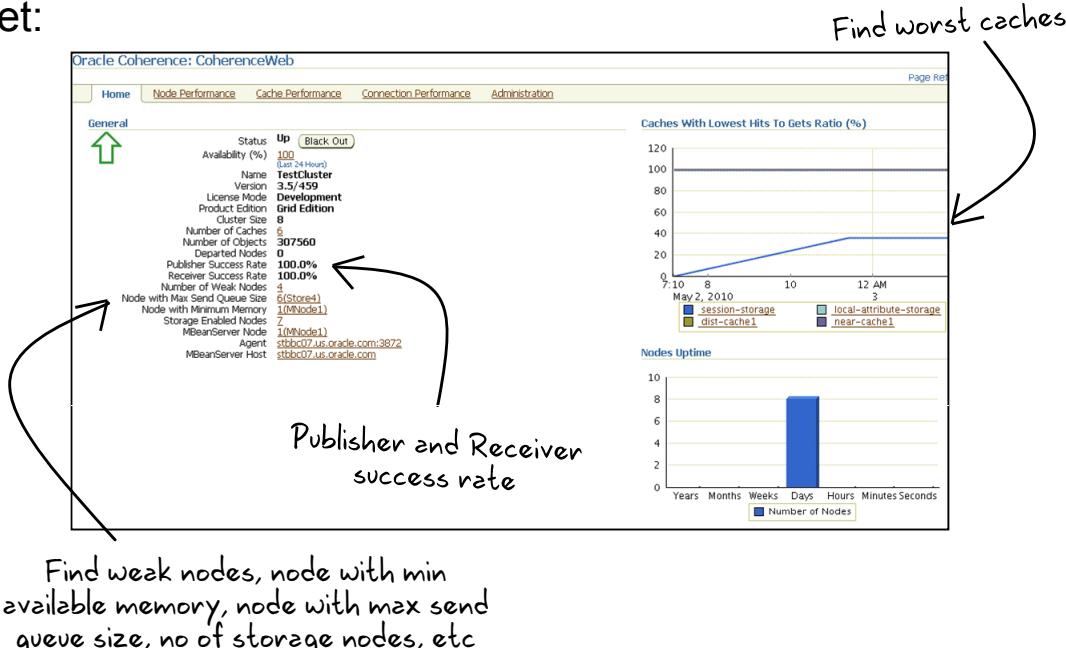
Proactive Monitoring Using Alerts and Notification

Enterprise Manager allows you to proactively monitor Oracle Coherence targets using various alerts. You can set critical and warning threshold values for a metric, and an alert will be generated to notify you of a potential problem in the system. You can view and change the threshold values using the **Metric and Policy Settings** link in the **Related Links** section. On the Metric Thresholds page, select **All Metrics** to view all the metrics for which you can define warning and critical thresholds.

Coherence Monitoring and Dashboard

Complete cluster visibility

Manage complexity by modeling Coherence cluster as a single target:



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Monitoring and Dashboard

Monitor multiple Coherence clusters from a single central console. Enterprise Manager reduces complexity by modeling a Coherence Cluster as a single target. Each Coherence cluster is shown with a dashboard view, which points to potential performance hotspots and possible application issues. The underperforming caches are potential performance bottlenecks.

The dashboard

- Shows the worst caches in the grid based on the “hits to gets” ratio
- Allows for drilling down to any cache shown in the display
- Shows vital grid statistics: Number of nodes, total objects in the grid, storage nodes, and so on
- Helps find nodes that need attention: Weak nodes, a node with minimum available memory, a node with maximum send queue size
- Helps find network issues of the grid by looking at publisher/receiver success rate

Coherence Monitoring and Dashboard

Complete cluster visibility

Quickly find performance hot spots:

- Services and their status
- Applications and corresponding caches
- Drill down to any object for further investigation

Cluster service dashboard

The diagram illustrates the monitoring hierarchy. It starts with 'Cluster hosts' pointing to the 'Select Host' dropdown in the Cluster Management interface. This leads to 'Cluster services', which points to the 'Services' section. From there, it moves to 'Monitor caches in context of the application', which points to the 'Applications' section. Finally, 'Proactive monitoring using alerts' points to the 'Metric Alerts' section.

Cluster Management					
Start New Nodes		Stop Nodes	Stop Cluster		
Select Host	Number of Nodes	CPU Used %	Memory Used %		
stbclu1.us.oracle.com	1.51	55.46			
Services					
Service Name	Service Type	Status	Number of Nodes	Enabled Nodes	Endangered Nodes
CustomerService	DistributedCache	NODE-SAFE	5	4	0
DistributedCache	DistributedCache	NODE-SAFE	5	4	1
DistributedSessions	DistributedCache	MACHINE-SAFE	7	4	0
LocalSessionCache	n/a	MACHINE-SAFE	7	6	1
Management	Invocation	MACHINE-SAFE	1	0	0
Proxy	Proxy	MACHINE-SAFE	1	0	0
ReplicatedSessionsMisc	ReplicatedCache	MACHINE-SAFE	1	0	0
SessionOwnership	Invocation	MACHINE-SAFE	1	0	0

Applications					
Application	Local Attribute Cache	Attributes	Local Session Cache	Sessions Name	Overflow Cache
Riddle	local-attribute-storage	0		session-overflow	0
Coherence testapp	local-attribute-storage	U		session-overflow	U

Metric Alerts				
Metric Name	Severity	Severity Message	Alert Trig	
Cache Hits for local-attribute-storage/LocalSessionCache	CRITICAL	Metric Value is 1,338	Apr 26, 2012	2
Cache Hits for near-cache1/DistributedSessions	CRITICAL	Metric Value is 1,015	Apr 26, 2012	2
Cache Hits for dist-cache1/DistributedCache	CRITICAL	Metric Value is 1,020	Apr 26, 2012	2
Cache Hits for session-storage/DistributedSessions	CRITICAL	Metric Value is 1,035	Apr 26, 2012	2

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Monitoring and Dashboard (continued)

Coherence provides effective caching using a variety of cluster services. Multiple caches are registered under each service. The status of the service directly reflects the cache performance. The status collectively indicates the health of the nodes working as cache servers for caches in the service. The status of all services is displayed on the dashboard, which allows for drilling down to each service, or the nodes in that service. The dashboard also displays all the hosts the grid is running on. Proactively monitor the grid by looking at alerts. This can help solve problems before they start affecting application service levels. You can monitor the HttpSession caches in the context of an application using these dashboard alerts.

Monitoring Cache

Quick resolution of cache performance issues

- View top caches in the cluster based in key metrics
- Locate a cache in a large environment
- Drill down views for each cache for monitoring and diagnostics
 - Find top nodes in a cache
 - Compare performance of multiple nodes side by side
 - Correlate metrics for diagnostics

View top caches per metric.
Compare load and performance.
Drill down to each cache.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Monitoring Cache

This page displays the cache-related performance over a specified period of time. You can view the performance of the top caches or all the caches. If you select All Caches, you can see the total and average metric values over the selected period of time. The image on this slide shows the performance of the top caches at the Cluster level. You can drill down into a specific cache to get more detailed information. “Top cache” means whichever cache has the highest statistic for the metric being measured. It is neither good or bad, it depends on the metric. If the metric is the most cache *hits*, then the top cache is the cache that is getting the most hits. If the metric is the most cache *misses*, then the top cache is the cache that is getting the most misses.

Monitoring Cache

Quick resolution of cache performance issues

- Support for near caches: Compare front versus back
- Storage manager statistics

Cache drill down performance view



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Monitoring Cache (continued)

This page displays the performance of a specific cache over a specific period of time. You can view charts showing the number of cache hits, misses, store reads, and store writes. You can also see the aggregated totals and average metric values over the selected period of time.

Monitoring Nodes

Node monitoring:

- Find top nodes by each metric:
 - Storage nodes
 - Weak nodes
 - More
- Monitor nodes in the context of a cache
- Search nodes
- Compare usage of different caches
- Real-time and historical



ORACLE®

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Monitoring Nodes

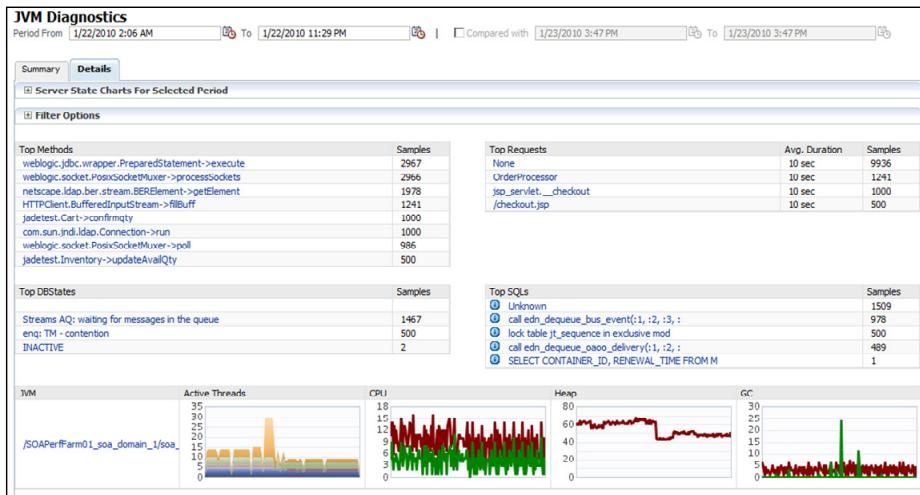
From the Cluster Level Home Page, you can see a global view of the cluster, which provides information about the top nodes that are weak and have communication and performance issues. This page provides a historical view of the metric data as it is stored in the repository. This page displays the performance of all the nodes in the cluster over a specified period of time. It provides charts showing the top nodes with lowest available memory, maximum send queue size, maximum puts, and maximum gets. By default, you can see the average performance metrics for the last 24 hours in all the Performance pages. If a target has been recently added, you can view real-time charts since the 24 hour performance metrics will not be available. To view the real-time charts, select one of the Real Time options in the View Data drop-down list in any of Performance pages. Using the View Data options, you can also view the average performance metrics for the last 7 or 31 days.

The Node Performance page tab shows the performance of all nodes in this cluster. If you click on a link that shows multiple nodes, like weak nodes, storage nodes, and so on, the performance of the selected nodes will be displayed on this page. You can toggle between the two modes to see the performance of the selected nodes or all the nodes.

Diagnosing JVM Issues

Diagnose JVM issues

- Quickly pinpoint what is causing the heap to leak
- Diagnose cross tier issues to database, and database to JVM
- Correlate user requests to Java threads and trace threads



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Diagnosing JVM Issues

OEM also provides other capabilities beyond Coherence management. It can manage and monitor the entire stack, including the JVM processes themselves.

Complete Coherence Grid Management

Drastically improve productivity and reliability

- Scale up capacity by starting new nodes dynamically:
 - Start new nodes
 - Clone an existing node
- Maintain grid stability:
 - Automatically detect departed node
 - Automatically start departed node
- Stop nodes:
 - Kill one node process
 - Kill all nodes
- Recycle whole cluster
 - Stop/start entire cluster
 - On restart, EM creates the same grid topology



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Complete Coherence Grid Management

CMP can provision new servers, which can be based on existing nodes. CMP also recognizes when a Coherence cluster node has left the cluster, and can automatically restart it. Additionally, CMP can start and stop any node in the cluster, including the entire cluster as a whole. This provides the true infrastructure needed to manage the numerous JVM processes that comprise the Coherence cluster across numerous machine boundaries.

Provisioning

Cut cost and reduce risk by automation

- Create gold-image of data grid:
 - Keep all Coherence binaries, node start scripts, application files, configuration XMLs, and so on, in EM repository
 - Keep a version of image as a template
- Create completely new coherence cluster:
 - Move gold-image files to multiple machines
 - Start multiple nodes on a machine
- Add new nodes to the cluster:
 - Move required files from gold-image to new machines
 - Start multiple nodes on a machine



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Provisioning

CMP provides the ability to create an image or template of a Coherence runtime environment that can easily be reused to provision new Coherence cluster nodes. This can be done for an entirely new cluster, or to add new machines and/or cluster nodes to an existing cluster.

Configuration Management

Reduces time to diagnose and repair issues

Compare Nodes		
Select two nodes to compare		
	First Node	Second Node
Compare Configuration		
Result	Attribute Name	Node 3 Node 1
=	Buffer Publish Size (Packets)	32 32
=	Multicast Enabled	true true
#	Unicast Port	8091 8089
=	Traffic Jam Count	8192 8192
=	Maximum Memory (MB)	511 511
=	Unicast Address	stakc11.us.oracle.com/140.87.26.37 stakc11.us.oracle.com/140.87.26.37
=	Multicast Threshold (%)	25 25
=	Nack Enabled	true true
#	Site Name	StoreSite Msite1
=	Multicast TTL (ms)	4 4
Compare Performance		
Result	Attribute Name	Node 3 Node 1
=	Packets Resent Early	0 0
#	Packets Sent	613 1349
#	Packets Received	854 1368
#	Total Gets	0 0
=	Packet Delivery Efficiency	0 0

Compare performance & configuration side by side - for two caches, nodes, services

Change Config on Node	
Batch Factor	0.0
ExpiryDelay	0
Flush Delay	0
High Units	2147483647
Low Units	
Queue Delay	
Refresh Factor	
Requeue Threshold	
<input type="button" value="Update"/> <input type="button" value="Update All Nodes"/>	
Home Performance Administration	

Change a single node or all nodes

Tune runtime parameters for nodes, caches and services

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Configuration Management

CMP provides configuration management tools that allow comparing configurations, and comparing the performance of nodes that use each configuration. The tool also allows applying configuration changes to a single or multiple nodes in the cluster.

Monitor Complete Infrastructure as a Single System

Single console drastically reduces total cost of ownership

- Monitor Coherence as part of complete system:
App Server + Coherence + DB
- Service modeling for applications based on WLS and Coherence
- Define and monitor SLA based on synthetic transactions
- System and service dashboards

Select Name	Type	Status
All Middleware		
EnterpriseManager0_stakc14.us.oracle.com	Oracle Application Server	(1)
stakc11.us.oracle.com_workshop_7001	Oracle WebLogic Server Domain	n/a
stakc11.us.oracle.com_workshop_7001.cgServer	Oracle WebLogic Managed Server	(1)
Cluster1_PST	oracle_coherence	(1)
TestCluster2	oracle_coherence	(1)

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ORACLE

Monitor Complete Infrastructure as a Single System

Because CMP is based as a management pack for OEM, the complete solution can monitor an entire platform within a single system. One example of this is being able to monitor multiple WebLogic Server (WLS) domains alongside multiple Coherence clusters, all from a single console.

Integrated Fusion Middleware Management



- Single solution for managing Fusion Middleware 11g
 - Oracle WebLogic Server
 - Oracle SOA Suite & Service Bus
 - Oracle Coherence
 - Oracle Identity Management Suite
 - Oracle Business Intelligence
 - Oracle WebCenter
 - Oracle Web Tier
- Diagnose production performance with zero overhead
- Automate operations for cloning and scaling up

ORACLE

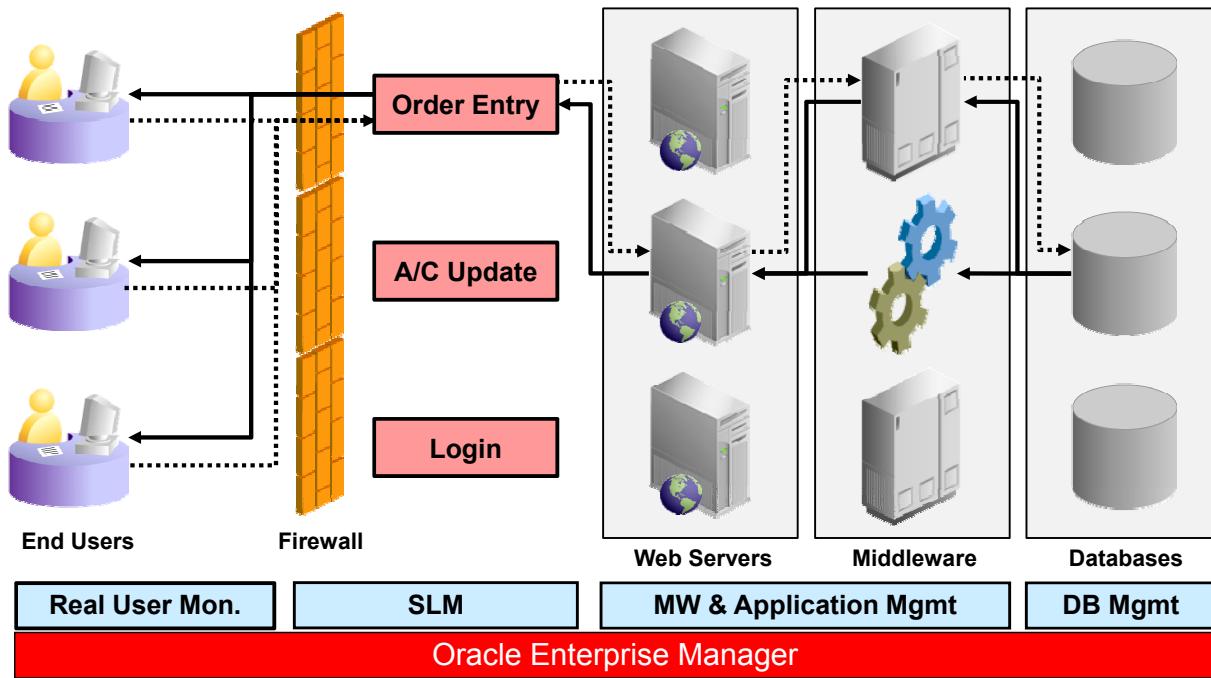
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Integrated Fusion Middleware Management

OEM manages the entire Oracle stack. It is the only out-of-the-box solution for the entire platform.

End-to-End Management

Manage performance and change across all tiers



- Transaction tracing across all tiers
- Patching and patch impact analysis on application and user



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

End-to-End Management

This slide shows an example of end-to-end management of multiple technology tiers in an enterprise level application, from the users all the way to the database.

Third-Party Management Tools

Partner: Product	Product Description	Image
Evident Software: ClearStone Live	A real-time monitoring and management application	
SL Corporation: RTView	A real-time monitoring and management application	
ITRS Group: ITRS Geneos	A real-time monitoring application	



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Third-Party Management Tools

Oracle has partnered with third-party companies that offer tools for monitoring and managing Coherence and other systems.

Quiz

Why is a tool like Coherence Management Pack for OEM useful for Coherence applications?

- a. Coherence does not provide any tools beyond a JMX infrastructure.
- b. Coherence does not provide an administrative platform for provisioning new cluster nodes.
- c. Coherence does not provide an administrative platform for starting, stopping, and restarting cluster nodes.
- d. All of the above.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: d

Agenda

Coherence JMX Management

Coherence Reporter

Cache and Cluster Management

Production Checklist

- Production versus Development Modes
- Hardware and JVM Recommendations
- Sizing a Coherence System
- Network Tuning and Troubleshooting
- Tuning Operating System and Coherence Settings
- Coherence Quorum
- Coherence Logging
- General Advice



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Production versus Development Modes

Before deploying Coherence, ensure that the following requirements are met:

- Validated Network connectivity and performance
- Established reasonable performance expectations
- Properly sized caches
- Set Coherence to “production” mode



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Production versus Development Modes

Coherence tends to be so simple to use in development that developers do not take the necessary planning steps and precautions when moving an application into production using Coherence. Before you run Coherence, make sure that you perform the following:

- Create a healthy appreciation for the complexities of deploying production software, particularly large-scale infrastructure software and enterprise applications.
- Enumerate areas that require planning when deploying Coherence.
- Define why production awareness should exist for each of these areas.
- Suggest or require specific approaches and solutions for each of these areas.
- Provide a checklist to minimize risk when deploying to production.

There are certain requirements on your system that you will have to meet before you install and run Coherence. You need to have:

- Network connectivity and performance have been validated. You will discuss later how to test the network to make sure that you are getting ample bandwidth.
- Set up reasonable expectations as to the cache performance you should see in your deployment environment.
- Limited the size of the caches to avoid out-of-memory conditions. Not size-limiting the caches can lead to cascading failures.

Production versus Development Modes (continued)

- Put Coherence into "production" mode, default mode is "development" mode.
- Reviewed the full production checklist available on the Coherence wiki.

For more detailed information, please refer to *Coherence Planning: From Proof of Concept to Production* document on OTN.

Hardware Recommendations

- Scale *out* (horizontal scaling) rather than *up*
 - 10 "small" machines will provide better performance and fault tolerance than one or two "big" machines
- Utilize commodity hardware:
 - 4-8 CPU cores
 - 4-16 GB RAM
 - Gigabit NIC
- The most popular platforms include:
 - Linux on Xeon/Opteron
 - Solaris on SPARC, or Xeon/Opteron
 - Other UNIX (HP/UX, IBM, and so on)
 - Windows on Xeon/Opteron



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Hardware Recommendations

As far as hardware is concerned, you do not need expensive hardware to run Coherence. Coherence is designed to run on inexpensive, commodity hardware, and is designed for scale out. That is, you can add inexpensive commodity servers to your grid when you want to increase the capacity of the system.

Consider the cumulative CPU, network, and memory available from 10 commodity boxes. This will far outperform a single "big" box. Also a total machine failure will impact a much smaller segment of the cache contents on 10 boxes.

You can run Coherence on large enterprise class Symmetric Multiprocessing (SMP) platforms, but this is not necessary.

The most popular platforms include:

- **Linux:** Most popular with either Xeon or Opteron.
- **Solaris:** Also popular with Opteron, Xeon, or SPARC
- Other UNIX platforms such as HP, UX, IBM AIX
- Windows as the deployment platform

For Coherence, there is no official list of supported platforms, unlike other Oracle products. Coherence runs on any platform that supports Java 1.4 or higher. Note Coherence for C++ currently supports Linux, Windows, and OS X, on x86 and x64 platforms.

Hardware Recommendations (continued)

It is advantageous to have multiple CPU and multicore CPUs. Performance advantages have been observed with the new Xeon Woodcrest and Clovertown chips over older Xeons. Because of Coherence's design, you can have multiple JVMs. Coherence is also designed to support multicore chips.

Most developers have relatively fast workstations. Combined with test cases that are typically non-clustered and tend to represent single-user access (that is, only the developer), the application may seem extraordinarily responsive.

- Include as a requirement that realistic load tests be built that can be run with simulated concurrent user load.
- Test routinely in a clustered configuration with the simulated concurrent user load.

Coherence is compatible with all common workstation hardware. Most developers use PC or Apple hardware, including notebooks, desktops, and workstations.

- Developer systems should have a significant amount of RAM to run a modern IDE, debugger, application server, database, and at least two cluster instances. Memory utilization varies widely, but to ensure productivity, the suggested minimum memory configuration for the developer systems is 2 GB. Desktop systems and workstations can often be configured with 4 GB for minimal additional cost.
- Developer systems should have two CPU cores or more. Although this will possibly make developers happier, the actual purpose is to increase the quality of code that is related to multithreading, because many bugs related to the concurrent execution of multiple threads will show up only on multi-CPU systems (systems that contain multiple processor sockets and/or CPU cores).

Coherence runs on virtually all major server hardware platforms. A majority of users use the "commodity x86" servers, with a significant number deploying on Sun SPARC (including Niagra) and IBM Power servers.

- The most cost-effective server hardware platform is "commodity x86," either Intel or AMD, with one to two processor sockets and two to four CPU cores per processor socket. If you select an AMD Opteron system, it is strongly recommended that you select a two-processor socket system, because memory capacity is usually halved in a single socket system. Intel Woodcrest and Clovertown Xeons are strongly recommended over the previous Intel Xeon CPUs due to significantly improved 64-bit support, much lower power consumption, much lower heat emission, and far better performance. These new Xeons are currently the fastest commodity x86 CPUs, and can support a large memory capacity per server, regardless of the processor socket count, by using fully buffered memory called "FB-DIMMs."
- It is strongly recommended that servers be configured with a minimum of 4 GB of RAM. For applications that plan to store massive amounts of data in memory, tens or hundreds of gigabytes, or more, it is recommended that you evaluate the cost-effectiveness of 16 GB or even 32 GB of RAM per server. Currently, commodity x86 server RAM is readily available in a density of 2 GB per DIMM, with higher densities available with only a few vendors, and carrying a large price premium. This means that a server with 8 memory slots only supports 16 GB in a cost-effective manner. Also note that a server with a very large amount of RAM is likely to need to run more Coherence nodes (JVMs) per server in order to utilize that much memory; so having a larger number of CPU cores helps.

Hardware Recommendations (continued)

- Applications that are "data heavy" will require a higher ratio of RAM to CPU, whereas applications that are "processing heavy" will require a lower ratio. For example, it may be sufficient to have two dual-core Xeon CPUs in a 32 GB server that is running 15 Coherence "Cache Server" nodes and performing mostly identity-based operations (cache accesses and updates). But if an application makes frequent use of Coherence features such as indexing, parallel queries, entry processors, and parallel aggregation, it would be more effective to have two quad-core Xeon CPUs in a 16 GB server—a 4:1 increase in the CPU:RAM ratio.
- A minimum of 1,000 Mbps for networking (for example, Gigabit Ethernet or better) is recommended. Network Interface Card (NICs) should be on a high bandwidth bus such as PCI-X or PCIe, and not on the standard PCI. In the case of PCI-X, having the NIC on an isolated, or otherwise lightly loaded, 133MHz bus may significantly improve performance.

Coherence is primarily a scale-out technology. While Coherence can effectively scale up on large servers by using multiple JVMs per server, the natural mode of operation is to span a number of small servers (for example, 2-socket or 4-socket commodity servers). Specifically, failover and fail back are more efficient in larger configurations. Therefore, the impact of a server failure is lessened in these configurations. As a rule of thumb, a cluster should contain at least five physical servers. In most wide area network (WAN) configurations, each data center will have independent clusters (usually interconnected by Extend-TCP). Coherence is quite often deployed on smaller clusters (one, two, or three physical servers) but this practice has increased risk if a server failure occurs. To easily maximize internode bandwidth, Coherence clusters are ideally confined to a single switch (for example, fewer than 96 physical servers). In some use cases, applications that are compute-bound or memory-bound (as opposed to network-bound) may run acceptably on larger clusters.

Also note that given a choice between a few large (multi gigabyte) JVMs and many small (1 GB) JVMs, the latter is the preferred option. There are several production environments of Coherence that span hundreds of JVMs. Some care is required to properly prepare for clusters of this size, but smaller clusters of dozens of JVMs are readily achieved. Note that disabling the UDP multicast (via WKA) or running Coherence on slower networks (for example, 100 Mbps Ethernet) reduces network efficiency, and makes scaling more difficult.

The following rules should be used to determine how many servers are required for a reliable high-availability configuration, and how to configure the number of storage-enabled JVMs.

- There must be at least three server machines. A grid with only two servers stops being machine-safe as soon as the number of JVMs on one server is not the same as the number of JVMs on the other server. So even if you start with two servers and an equal number of JVMs, losing one JVM forces the grid out of the machine-safe state. Five or more machines present provides for a very stable topology, but deploying on just three servers would work if the other rules are adhered to.
- For a server that has the largest number of JVMs in the cluster, the number of JVMs must not exceed the total number of JVMs on all the other servers in the cluster. A server with the smallest number of JVMs should run at least half the number of JVMs as a server with the largest number of JVMs; this rule is particularly important for smaller clusters.
- The margin of safety improves as the number of JVMs tends toward equality on all the machines in the cluster. This is more of a "rule of thumb" than the preceding rules.

JVM Recommendations

Use the server JVM:

- Sun JVM 1.6 is preferred.
- Sun JVM 1.5 is supported.
- Oracle JRockit offers deterministic garbage collection.
- Other JVMs (for example, IBM) can also be used.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

JVM Recommendations

Java 1.6 is preferred, but 1.5 is also supported. JRockit and HotSpot JVMs are both supported. JRockit requires more memory and CPU, and only represents about 25% of the Coherence production deployment base.

JVM Deployment Concerns

- Garbage collection (GC) pauses (rule of thumb)
 - J2SE 1.5: Limit heap to 1 GB
 - J2SE 1.6: Limit heap to 4-6 GB
 - Run with fixed size heap. Set `-Xms` and `-Xmx` to same value
 - Log garbage collection via `-Xloggc`
- JVM tuning: Utilize server mode JVM by specifying `-server`
- Swapping
 - Must be avoided!
 - Assume 1.2 GB of RAM per 1 GB JVM heap
 - Validate above assumption in your environment



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

JVM Deployment Concerns

The main issues related to using a different JVM in production (as compared to development) are:

- Logging and monitoring differences, which may mean that the tools used to analyze logs and monitor live JVMs during development testing may not be suitable for production
- Significant differences in optimal GC configuration and approaches to GC tuning (production systems see different usage patterns)
- Differing behaviors in thread scheduling, garbage collection behavior and performance, and the performance of running code

Make sure that regular testing is occurring on the JVM that will be used in production.

JVM Deployment Concerns (continued)

JVM configuration options vary over versions and between vendors, but the following are generally suggested:

- Using the server option results in substantially better performance.
- Using identical heap size values for both `-Xms` and `-Xmx` yields substantially better performance, as well as "fail fast" memory allocation.
- For naive tuning, a heap size of 1 GB is a good compromise that balances the per-JVM overhead and garbage collection performance when using J2SE 1.5. J2SE 1.6 has improved garbage collection, which allows for larger heap sizes of 4-6 GB.
- Larger heap sizes are allowed, and commonly used, but may require tuning to keep garbage collection pauses manageable. There is also better support for 64-bit JVMs.
 - Hotspot: `CompressedOops` option allows for compressing 64-bit pointers to 32-bit size for better performance.
 - JRockit: Same option called compressed references: `-XX:compressedRefs`

In terms of Oracle Coherence versions:

- Coherence 3.x versions are supported on the Sun JDK versions 1.4, 1.5, and JVMs corresponding to these versions of Sun JDK.
- Starting with Coherence 3.3, the 1.6 JVMs are also supported.

Often the choice of JVM is dictated by other software. For example:

- IBM only supports IBM WebSphere running on IBM JVMs.
- Oracle WebLogic typically includes a JVM that is intended to be used with it. On some platforms, this is the Oracle JRockit JVM.
- Apple Mac OS X, HP-UX, IBM AIX, and other operating systems have only one JVM vendor (Apple, HP, and IBM respectively).
- Certain software libraries and frameworks have minimum Java version requirements because they take advantage of relatively new Java features.
- On commodity x86 servers running Linux or Windows, the Sun JVM is recommended. As a best practice, you should test and deploy using the latest supported Sun JVM based on your platform and Coherence version.

Basically, at some point before going to production, a JVM vendor and version should be selected and well-tested. If no errors appear during testing and staging with this JVM, you should use this JVM when going to production.

Coherence is a pure Java software and can run in clusters composed of any combination of JVM vendors and versions. Note that it is possible for different JVMs to have different serialization formats for Java objects, meaning that it is possible for an incompatibility to exist when objects are serialized by one JVM, passed over the network, and a different JVM (vendor and/or version) attempts to deserialize it. It is highly recommended that you test mixed configurations for consistent serialization before deploying in a production environment.

JVM Heap Sizing

- For cache server JVMs, use a fixed-size heap:
 - Larger heap sizes present garbage collection problems.
 - `java -Xms4096m -Xmx4096m`
- Evaluate the actual physical RAM used by your JVM process.
- Never configure your JVMs to exceed the physical RAM:
 - This causes swapping and reduces performance.
 - Run "`swap -l`," "`top`," or "`vmstat`" to verify whether the system is not swapping and that RAM is available.
 - Allow ~400 MB for the operating system.
 - Take into account other software running on the system.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

JVM Heap Sizing

Heap sizing is very important for Coherence because it stores potentially a large number of objects within the Java heap space. The recommendation is to have a fixed heap size. This helps reduce GC performance. The recommendation for the heap size is 1 GB or less for version 1.5 JVMs, and 4-6 GB for version 1.6 JVMs.

Set up the heap size by passing `-Xms` on the Java command line, and setting the size value to the specified number of MB. In this example, the heap size is `4096m`, which is 4 GB. Alternatively, you can pass `4g` (notice no "b"). Running with a fixed-size heap saves your JVM from having to grow the heap on demand, and results in improved performance. To specify a fixed-size heap, use the `-Xms` and `-Xmx` JVM options, setting them to the same value.

Remember that a JVM with 1 GB heap uses more than 1 GB of physical RAM. So, when sizing the system, the physical size of the running process must be used. As with other applications, the system should never be configured to exceed the physical RAM. This severely impacts performance. This should be taken into consideration when determining the maximum number of JVMs that you will run on a machine. The actual allocated size can be monitored with tools such as `top`.

JVM Heap Sizing (continued)

What happens when your JVMs in total exceed the physical RAM on the system? In such a scenario, the system starts swapping the memory to disk, and this thrashing results in extremely bad performance.

There are various operating system utilities in Linux and UNIX that you can run, such as `swap -l`, `top`, and `vmstat`, to verify that the system is not swapping, and that you still have some available RAM on your system. Remember that you require around 400 MB for the operating system, maybe more if you run other applications. If you run other software on the system, you need to keep that in mind as well. You can also dedicate servers to Coherence using inexpensive commodity hardware. If you have a database installed, that database can take up a lot of memory. Take this into account when sizing the memory for the system.

For large data sets, partitioned or near caches are recommended. Because the scalability of the partitioned cache is linear for both reading and writing, increasing the number of Coherence JVMs (on a fixed set of hardware) will not significantly affect cache performance.

It is common to use multiple Coherence instances per physical machine. As a general rule of thumb, the current JVM technology works well up to 6 GB heap sizes. So, using a number of 6 GB Coherence instances provides good performance without a great deal of JVM configuration or tuning.

For performance-sensitive applications, experimentation may provide better tuning. When you consider heap size, it is important to find the right balance. The lower bound is determined by the per-JVM overhead (and also, the manageability of a potentially large number of JVMs). For example, if there is a fixed overhead of 100 MB for infrastructure software (for example, JMX agents, connection pools, and internal JVM structures), the use of JVMs with 256 MB heap sizes results in close to 40% overhead for noncache data. The upper bound on JVM heap size is governed by the memory management overhead, specifically the maximum duration of GC pauses, and the percentage of CPU allocated to GC (and other memory management tasks).

For JVMs running on commodity systems, the following rule generally holds true (with no JVM configuration tuning): With a heap size of 1 GB (1.5 JVM) or 4-6 GB (1.6 JVM), full GC pauses should not exceed one second.

It is important to note that GC tuning has an enormous impact on GC throughput and pauses. There are many variations that can substantially impact these numbers, including the machine architecture, CPU count, CPU speed, JVM configuration, object count (object size), and object access profile (short-lived versus long-lived objects).

For allocation-intensive code, garbage collection can theoretically consume close to 100% of CPU. For both cache server and client configurations, most CPU resources will typically be consumed by the application-specific code. It may be worthwhile to view verbose garbage collection statistics (for example, `-verbosegc`). Use the profiling features of the JVM to get profiling information, including CPU usage by GC (for example, `-Xprof`).

Tuning Garbage Collection

- Monitor garbage collection with either of the following JVM command-line options:
 - verbose : gc: Displays the garbage collection pause times to standard output
 - Xloggc : [filename]: Logs garbage collection pause times to a file
- Run JVM in server mode using `-server`
- Avoid heaps larger than the recommended sizes
- 64-bit JVMs are supported starting with J2SE 1.6:
 - HotSpot: `-XX : +UseCompressedOops`
 - JRockit: `-XXcompressedRefs`



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Tuning Garbage Collection

Frequent garbage collection pauses that are in the range of 100 milliseconds or more are likely to have a noticeable impact on network performance. During these pauses, a Java application is unable to send or receive packets, and in the case of receiving, the OS-buffered packets may be discarded and need to be retransmitted.

In Coherence, log messages are generated when one cluster node detects that another cluster node has been unresponsive for a period of time, generally indicating that a target cluster node was in a garbage collection cycle.

The PauseRate indicates the percentage of time for which a node was considered unresponsive since the statistics were last reset. Nodes reported as unresponsive for more than a small percentage of their lifetime may be worth investigating for garbage collection tuning.

It is important that you monitor garbage collection in Java. If you have performance problems in your application, you may want to be able to diagnose quickly whether that problem is due to garbage collection.

Tuning Garbage Collection (continued)

You can use either of the following on the JVM command line to monitor the frequency and duration of garbage collection pauses:

- `-verbose:gc` that displays verbose garbage collection on the Java command line
- `-Xloggc: [filename]` where `[filename]` is the name of the file to which you log the garbage collection pause time

If you have either of these options enabled, every time JVM garbage collects, it makes an entry in the log and also displays the number of milliseconds in which the garbage collection process occurred. Generally, you must restrict garbage collection time to be less than 100 milliseconds.

Run your JVMs in server mode by setting the `-server` option in your Java command line because server JVMs are more efficient. However, you would need a full JDK, and not just a JRE to be able to do that.

Avoid using heaps larger than the sizes discussed earlier.

As of J2SE 1.6, 64-bit JVMs are good for use with Coherence. Compressing 64-bit memory pointers into 32-bit forms helps improve performance by keeping these references smaller in memory. Use the respective options for HotSpot or JRockit to turn compressed references on. 64-bit JVMs are useful for running JVMs larger than 4 GB.

Sizing a Coherence System

- Because data is stored in memory, you must carefully size a Coherence cluster.
- By default, data is stored in the heap of the storage JVMs.
- Size limits are per-server, so adding additional servers increases the total without requiring reconfiguration.
- If you don't limit the size of your caches, you can trigger out-of-memory errors, which can lead to cascading cluster failures.
- Fear of eviction should not keep you from size limiting. A cascading failure puts *all* data at risk.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Sizing a Coherence System

Coherence is an in-memory data manager that stores data in memory. So you need to make sure that the data that is placed in the grid does not exceed the available memory. It is important that you do some planning. So you also need to do some more preliminary work in sizing than you would for disk-based software such as a disk-based database. By default, data is stored in the heap of the storage JVMs. You must plan for the number of storage JVMs that you have on your system.

However, it is not a problem if the sizing in your system turns out larger than you expected. As discussed earlier, you can dynamically add or drop storage JVMs. Coherence automatically resizes the cluster and repartitions the data when you add or drop storage JVMs. Coherence is dynamically configurable, and you can change the size and configuration of the system dynamically without having to shut anything down.

General Sizing Guidelines

- Allow extra space for overhead:
 - Every object has one full backup on another JVM on another machine.
 - If a JVM fails, Coherence automatically fails over, and creates new backups on other JVMs. This means that if a JVM fails, other JVMs would need to accommodate the backups of the objects.
 - Size limit, allowing for at least one or two machine failures.
- Each 1 GB JVM can store 350 MB of actual object data. This means a 16 GB machine supports about 4.5 GB of raw object data.
- The upper bound on storage-enabled JVMs is 500.
- The recommended minimum number of storage JVMs is 3.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

General Sizing Guidelines

Following are the general sizing guidelines:

- Allow extra space for overhead:
 - Each object has one full backup on another JVM on another machine. This means that you now need to have double the space to store the backup. If a JVM fails, you need extra space on the grid to store the new backups, when Coherence automatically creates new backups and the repartitioning event.
 - Every NamedCache that is created internally uses approximately 250 KB.
 - Every entry that is placed in the cache has an overhead of 144 bytes. This may not seem like a lot, but when millions of entries are placed into a cache it makes a big difference with a sizing exercise.
 - Every Coherence index needs to be calculated in bytes. Indexes are stored as native objects. To do the most accurate job of capacity planning, you need to actually measure index heap consumption with a heap analysis tool, such as VisualVM, Eclipse MAT, AD4j, or JRMC.

General Sizing Guidelines (continued)

- The rule of thumb is that a 1 GB JVM can store about 350 MB of actual object data. You must allow for a significant amount of overhead. With Java, you can have data that is about one-third the heap size. This would make the garbage collection algorithms more efficient and faster. You also want to allow for extra space if there is a sudden spike in demand and more objects are being put on the system than you anticipated. If a 1 GB JVM can store 350 MB of actual data, it means that a 16 GB machine will support about 4.5 GB of raw object data. On a 16 GB machine, you do not want to put 16 JVMs, because of the overhead of JVMs. For example,
 $16 / 1.2 \text{ GB} = 13.3 \text{ JVMs}$ (leave that last 0.3 GB for OS)
 $13 \text{ JVMs} * 350 \text{ MB} = 4.55 \text{ GB}$.
So you have $13 \times 350 \text{ MB} = 4.55 \text{ GB}$ of data.
- The maximum number of storage JVMs that is recommended is about 500. There is no hard-coded maximum size with the number of storage JVMs. However, based on the benchmarking results, the overhead of maintaining the cluster and TCMP becomes a scaling issue as your cluster approaches 500 JVMs. If there is going to be a large number of clients or applications accessing the cache, it is best to make them Coherence*Extend clients that can be multiplexed over a set of TCP/IP Proxy Service connections. This avoids the overhead of cluster communication and TCMP calls over a larger number of cluster nodes by limiting the overall cluster size.
- The minimum number of storage JVMs that is recommended as production configuration is three.

Size-Limiting Storage JVMs

- The `<local-scheme>` should be size-limited to prevent any JVMs from running out of memory.
- A local cache can be the:
 - Backing map for partitioned caches
 - Front scheme for near caches
- The unit or measure for sizing can be:
 - Number of objects
 - Number of bytes (not supported for near-cache)
- The limit is per cache per server



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Size-Limiting Storage JVMs

Remember that Coherence storage JVMs store data in a local cache. The local cache can be size-limited and you actually need to limit the size. By default, Coherence has no size limitation. That is, if you do not configure some sort of size limitation, chances are that an application can continuously put objects into the cache without deleting them. This eventually results in the `java.lang.OutOfMemory` error, which is an error that causes a JVM to crash. You will not lose any data, but overloading a cache with objects without deleting them will cause the JVM to crash when it runs out of memory.

You can size the system with either of the following:

- The number of objects
- The number of bytes

If you specify a limit on the number of objects in a local cache, you would need to calculate the average size of your objects. Coherence also provides a binary calculator that works with the distributed topology. This is used to size-limit the number of bytes.

Binary Calculator: Example

- This example from a `cache-config.xml` file limits a JVM to 350 MB for Coherence-managed objects (approximately 1/3rd of a 1 GB heap size).
- The setting is for each storage-enabled JVM.
- Binary calculator works only for distributed caches.

```
<backing-map-scheme>
  <local-scheme>
    <high-units>350M</high-units>
    <unit-calculator>BINARY</unit-calculator>
  </local-scheme>
</backing-map-scheme>
```



coherence-cache-config.xml

The Oracle logo, which consists of the word "ORACLE" in a white sans-serif font inside a red horizontal bar.

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Binary Calculator: Example

The slide shows an example of how to use the binary calculator. In this example, you limit the JVM to 350 MB of Coherence-managed objects. This is again based on the best performance of the garbage collection algorithm that is 1/3rd of a 1 GB heap size. The setting is for each storage-enabled JVM. You can set the size of the cache in the `cache-config.xml` file.

In the XML configuration file, there is a backing map scheme that indicates to Coherence to store the data in a Coherence local cache or local scheme. `<high-units>` is the configuration used to limit the size of the cache. In this example, the value is 350,000,000.

You also specify that `<unit-calculator>` is binary. If you do not specify `<unit-calculator>` as binary, this number will be the number of objects and not the number of bytes. You have set `<unit-calculator>` as binary with a setting of 350,000,000 in this example. So, when the local cache in each storage JVM is reaching 350,000,000, it begins to evict objects from the cache based on an eviction algorithm that you specify. This is typically a combination of least recently used, and least frequently used objects.

Network Configuration

- Nodes in your cluster should be connected with Gigabit Ethernet, ideally on the same switch.
- With multiple switches, interswitch bandwidth becomes a bottleneck:
 - Maximize interswitch bandwidth.
 - Balance nodes across switches (view each switch as a large node.).
- In large deployments, switch tuning may be necessary. For example, Cisco switches allow for tuning of packet buffers.
- Utilize built-in safe guards to prevent accidental cluster collisions:
 - Do not run production on default multicast address and port
 - Utilize cluster name
 - Limit TTL to minimum required to span network



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Network Configuration

Network configuration and network tuning is very important in Coherence because you often come across issues with the switch and network configuration. For all cluster members, it is recommended that you use Gigabit Ethernet. It is recommended that all storage-enabled members of a given cache service be on the same switch so that it provides for optimum performance.

You can have multiple switches on your cluster, but remember that interswitch bandwidth can become a bottleneck. In such a case, you view each switch as a node and ensure that one switch is not overloaded.

In large deployments, switch-tuning may be necessary. Therefore, you may need to get the help of a network administrator, who is an expert on configuring switches that you might be using. This is typically not necessary, but may be needed if multicast has not been enabled within the switches, or if the switch is unable to handle the high packet rates Coherence is capable of producing.

During development, a Coherence-enabled application on a developer's local machine can accidentally form a cluster with the application running on the machines of other developers.

Network Configuration (continued)

Developers often use and test Coherence locally on their workstations. There are several ways in which they may accomplish this: by setting the multicast time-to-live (TTL) to zero, by using the "loopback" interface, or by each developer using a multicast address and port that is different from all other developers. If one of these approaches is not used, multiple developers on the same network will find that Coherence has clustered across different developers' locally-running instances of the application.

Setting the TTL to zero on the command line is very simple: Add the following to the JVM startup parameters:

```
-Dtangosol.coherence.ttl=0
```

On some UNIX operating systems, including some versions of Linux and Mac OS X, setting the TTL to zero may not be enough to isolate a cluster to a single machine. To be safe, assign a different cluster name for each developer, for example, using the developer's email address as the cluster name. If the cluster communication does go across the network to other machines, the different cluster name will cause an error on the node that is attempting to start.

To ensure that the clusters are completely isolated, select a different multicast IP address and port for each developer.

Make sure that the application is being tested in a clustered configuration as development proceeds. There are several ways in which clustered testing can be a natural part of the development process. For example:

- Developers can test with a locally clustered configuration of at least two instances running on their own machine. This works well with the TTL=0 setting, because clustering on a single machine works with the TTL=0 setting.
- Unit and regression tests can be introduced that run in a test environment that is clustered. This may help automate certain types of clustered testing that an individual developer would not always remember (or have the time) to do.

Most production networks are based on Gigabit Ethernet, with a few still built on the slower 100 Mb Ethernet, or the faster 10 Gigabit Ethernet. It is important to understand the topology of the production network, and the full set of devices that will connect all the servers that will run Coherence. For example, if there are 10 switches being used to connect the servers, are they all of the same type (make and model) of switch? Are they all of the same speed? Do the servers support the network speeds that are available?

In general, all servers should share a reliable, fully switched network. This generally implies sharing a single switch (ideally, two parallel switches and two network cards per server for availability). There are two primary reasons for this:

- Using multiple switches almost always results in a reduction in effective network capacity.
- Multiswitch environments are more likely to have network "partitioning" events, where a partial network failure results in two or more disconnected sets of servers. Though partitioning events are rare, Coherence cache servers ideally should share a common switch.

Network Configuration (continued)

To demonstrate the impact of multiple switches on bandwidth, consider a number of servers plugged into a single switch. As additional servers are added, each server receives dedicated bandwidth from the switch backplane. For example, on a fully switched Gigabit backplane, each server receives a Gigabit of inbound bandwidth, and a Gigabit of outbound bandwidth for a total of 2 Gbps "full duplex" bandwidth. Four servers would have an aggregate of 8 Gbps bandwidth. Eight servers would have an aggregate of 16 Gbps, and so on, up to the limit of the switch (in practice, usually in the range of 160-192 Gbps for a Gigabit switch).

However, consider the case of two switches connected by a 4 Gbps (8 Gbps full duplex) link. In this case, as servers are added to each switch, the servers will have full "mesh" bandwidth of up to a limit of four servers on each switch (for example, all four servers on one switch can communicate at full speed with the four servers on the other switch). However, adding additional servers will create a bottleneck on the interswitch link. For example, if five servers on one switch send data to five servers on the other switch at 1 Gbps per server, the combined 5 Gbps will be restricted by the 4 Gbps link. Note that the actual limit may be much higher, depending on the traffic per server, and also the portion of traffic that actually needs to move across the link. Also note that other factors, such as network protocol overhead and uneven traffic patterns, may make the usable limit much lower from an application perspective.

Avoid mixing and matching network speeds. Make sure that all servers can, and do, connect to the network at the same speed, and that all the switches and routers between these servers are running at the same speed or faster.

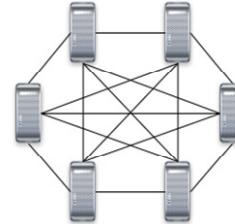
If the datagram and/or multicast tests have returned poor results, it is likely that there are configuration problems with the network devices in use. Even if the tests passed without incident, and the results were perfect, it is still possible that there are issues lurking with the configuration of network devices.

The Coherence cluster protocol is capable of detecting and handling a wide variety of connectivity failures. The clustered services can identify the connectivity issue and force the offending cluster node to leave and rejoin the cluster. Thus, the cluster ensures a consistent shared state amongst its members.

Whenever possible, specify PreferIPv4 to avoid using IPv6 networking. This can cause problems when running Coherence clusters.

Network Configuration

- Coherence can saturate a Gigabit Ethernet network.
- Avoid mixing network speeds:
 - All nodes in the cluster should be Gigabit Ethernet at a minimum.
 - If client platforms have 100BaseT Ethernet or worse, connect to the cluster via TCP/IP and Coherence*Extend.
- If possible, use PCI-Express or PCI-X NICs:
 - Some PCI NICs cannot reach full bandwidth due to bus limitations.
 - Validate NIC MTU, speed, and duplex.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Network Configuration (continued)

Coherence is very fast and is very scalable. It is well capable of saturating a Gigabit Ethernet network, based on client demand.

All the nodes in the cluster should be Gigabit Ethernet or better. If you have other network speeds, or clients connecting via wide area networks (WAN), connect those clients via TCP/IP. With Coherence*Extend, the client is not a full member of the cluster, and its slower network performance will not affect the rest of the cluster. So your full cluster members, which are either local storage-disabled or local storage-enabled, should be connected with at least Gigabit Ethernet.

For optimal gigabit performance, your network card should be either PCI-Express or PCI-X. Some of the older PCI buses are not fast enough and they do not support full Gigabit Ethernet speeds. Often, when you try testing Coherence on your PC (in a test or development environment), you notice that you get half the speeds that you normally get from Gigabit Ethernet.

With the help of your network administrator, make sure that the switch is tuned, and NIC MTU, speed, and full-duplex are validated.

Operating System Tuning

- UNIX:
 - Verify that UDP receives buffers that are at least 2 MB
 - OS X only: 1 MB buffers are preferred
- Linux:
 - Take care of the 2.4 kernel threading issues.
 - Set the TCP/IP send and receive buffer sizes higher.
- Windows: Run \coherence\bin\optimize.reg



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ORACLE

Operating System Tuning

You also need to tune the operating system. Following are a few parameters in the operating system that you need to set:

- **On UNIX:** Verify that UDP can receive buffers that are at least 2 MB. OS X only needs to be able to receive 1 MB buffers.
- **On Linux:** If you have the Linux 2.4 kernel, you will want to check the NPTL (Native Posix Thread Library) version that you are running because of threading issues in early version. If you are on a pre-1.0 version, it is recommended that you disable NPTL, or upgrade to a post-1.0 version, and preferably to the corresponding 2.6 kernel.
- On Windows, Coherence ships with a registry script `optimize.reg` in the `coherence\bin` directory. When you execute `optimize.reg`, it adds appropriate tuning parameters to the registry. The TCP/IP packet sizes that are configured for Windows by default are configured for very old networks. They are not optimized for Gigabit Ethernets. However, you can fix the problem by running this script.

Setting Buffer Sizes

On Linux, execute (as root):

```
sysctl -w net.core.rmem_max=2096304  
sysctl -w net.core.wmem_max=2096304
```

On Solaris, execute (as root):

```
ndd -set /dev/udp udp_max_buf 2096304
```

On Windows, execute:

```
\coherence\bin\optimize.reg  
reboot
```



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Setting Buffer Sizes

To help minimize packet loss, the OS socket buffers need to be large enough to handle the incoming network traffic while your Java application is paused during garbage collection. By default, Coherence attempts to allocate a socket buffer of 2 MB. If your OS is not configured to allow for large buffers, Coherence utilizes smaller buffers. Most versions of UNIX have a very low default buffer limit, which should be increased to at least 2 MB. You will receive the following warning if the OS fails to allocate the full size buffer:

"UnicastUdpSocket failed to set receive buffer size to 1428 packets (2096304 bytes); actual size is 89 packets (131071 bytes). Consult your OS documentation regarding increasing the maximum socket buffer size. Proceeding with the actual value may cause sub-optimal performance."

Though it is safe to operate with smaller buffers, it is recommended that you configure your OS to allow for larger buffers.

On Linux, execute (as root):

```
sysctl -w net.core.rmem_max=2096304  
sysctl -w net.core.wmem_max=2096304
```

Setting Buffer Sizes (continued)

On Solaris, execute (as root):

```
ndd -set /dev/udp udp_max_buf 2096304
```

Windows does not impose a buffer size restriction by default.

The slide shows the commands that you must execute on specific operating systems. On UNIX and Linux, you need to set the appropriate parameters and you can see the commands to execute on Linux in the slide. The slide also shows the command that you must execute on Solaris to set the packet buffer size. You need to execute `optimize.reg` on Windows only once.

Note: Check your operating system version. Some operating systems require a reboot after setting these kernel parameters.

If these parameters are not set properly, you see messages such as "Your Operating System buffer size parameters are not set properly" in the Coherence log. The message in the Coherence log then recommends looking at the installation guide.

Testing Multicast

- Run `/coherence/bin/multicast-test` on each machine of your cluster. This verifies that multicast is working.
- If multicast is not working (due to firewalls, routers, and so on), consider well-known addresses.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Testing Multicast

The term "multicast" refers to the ability to send a packet of information from one server and to have that packet delivered in parallel by the network to many servers. Coherence supports both multicast and multicast-free clustering. It is recommended that you use multicast when possible because it is an efficient option for many servers to communicate. However, there are several common reasons why multicast cannot be used:

- Some organizations disallow the use of multicast.
- Multicast cannot operate over certain types of network equipment. For example, many WAN routers disallow, or do not support, multicast traffic.
- Multicast is occasionally unavailable for technical reasons. For example, some switches do not support multicast traffic.

First, determine whether multicast will be used. Alternatively, determine whether the desired deployment configuration can use multicast. Before deploying an application that will use multicast, you must run the Multicast Test to verify that multicast is working, and to determine the correct (the minimum) TTL value for the production environment. Applications that cannot use multicast for deployment must use the Well-Known Addresses (WKA) configuration.

Coherence make judicious use of multicast and thereby avoids the negative downsides commonly associated with multicast, such as multicast storms.

Testing Multicast (continued)

When the new cluster member announces itself, it checks to see whether there are other cluster members running. If it finds them, it automatically joins the grid.

If you want to use multicast, you can test it by using the utility called `multicast-test` in the `/coherence/bin` directory. This tests to see whether multicast is enabled on your system.

Often, you may be required to disable multicast due to firewalls. Before using multicast, it is a good practice to test it first. If for some reason, multicast does not work, due to firewalls, or you simply want to disable it, you can set well-known addresses. This is very simple to implement.

You specify the host name or IP addresses where the Coherence server is running. After you set the well-known addresses, the well-known address setting automatically disables multicast. So, Coherence will not use multicast at all.

Note: Coherence does not require multicast.

Included with Coherence is a Multicast Test utility, which helps you to determine whether multicast is enabled between two or more computers. This is a connectivity test, not a load test. Each instance, by default, transmits only a single multicast packet once every two seconds.

To run the Multicast Test utility, use the following syntax from the command line:

```
java com.tangosol.net.MulticastTest <command value> <command value>
...
...
```

For example: `java com.tangosol.net.MulticastTest -group 237.0.0.1:9000`

For ease of use, the `multicast-test.sh` and `multicast-test.cmd` scripts are provided in the Coherence `bin` directory, and can be used to execute this test.

Test whether you can use multicast address 237.0.0.1 and port 9000 (the test's defaults) to send messages between two servers , for example, Server A with IP address 195.0.0.1 and Server B with IP address 195.0.0.2.

Starting with Server A, determine whether it has multicast address 237.0.0.1 and port 9000 available for 195.0.0.1 by first checking the machine or the interface by itself as follows:

- From a command session, enter:
`multicast-test.sh -ttl 0`
- Press Enter. You should see the Multicast Test utility showing you how it is sending sequential multicast packets and receiving them, as follows:

```
Starting test on ip=servera/195.0.0.1,
group=/237.0.0.1:9000,ttl=0
```

```
Configuring multicast socket...
```

```
Starting listener...
```

```
Tue Jan 17 15:59:51 EST 2006: Sent packet 1.
```

```
Tue Jan 17 15:59:51 EST 2006: Received test packet 1 from self.
```

```
...
```

- After you have seen a number of these packets sent and received successfully, press `Ctrl + C` to stop further testing.

Testing Multicast (continued)

If you do not see something similar to the preceding progress message, multicast is not working. Also, note if you specified a TTL of 0 to intentionally prevent the multicast packets from leaving Server A.

You can repeat the same test on Server B to assure that it too has multicast enabled for its port combination.

To test multicast communications between Server A and Server B, use a nonzero TTL, which allows the packets to leave their respective servers. By default, the test uses a TTL of 4. If you believe that there may be more network hops required to route packets between Server A and Server B, you may specify a higher TTL value.

- Start the test on Server A and Server B by entering the following command into the command windows and pressing Enter:
`multicast-test.sh`
- You can see that both Server A and Server B are issuing multicast packets, and seeing their own and each other's packets. This indicates that multicast is functioning properly between these servers using the default multicast address and port.

Datagram Test

- Tests network bandwidth and reliability.
- On machine 1:
 - cd [*coherence directory*]
 - bin/datagram-test.sh (on Linux/UNIX)
 - bin\datagram-test (on Windows)
- On machine 2 (3,4,5, and so on):
 - cd [*coherence directory*]
 - bin/datagram-test.sh *machine1_ip_address*
 - Example: bin/datagram-test.sh 192.168.1.107
- Look for > .99 success rate (1 is ideal).
- Look for 80 – 115 MB per second on Gigabit Ethernet:
 - If you are not getting this rate, network or switch may need tuning.
- Turn down rate (-txRate) to find 100% success rate point.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Datagram Test

Warning: Do not run this test on a production system as it can completely saturate the network! If possible, run during off hours.

Before deploying an application, you must run the Datagram test to test the actual network speed and determine its capability for pushing large amounts of data. Furthermore, the Datagram test must be run with an increasing number of publishers-to-consumers, because a network that appears fine with a single publisher and a single consumer may completely fall apart as the number of publishers increases.

It is very important that you run the Datagram test to make sure that your network is running at optimal speed. Make sure that you do this, right after you install Coherence. You may think that your network is fast, and that it is tuned appropriately, but usually it is not. The Datagram test is basically a very quick and easy way to determine the performance of your network. It simply sends datagrams to another server as fast as it can, and measures the speed.

To run this test, execute a script called `datagram-test.sh` that is also in the `coherence/bin` directory. It is available on Windows as `datagram-test.cmd`.

You run this utility on the first machine. Then on the other machines, you specify the IP address of machine 1. For example, the IP address of machine 1 is 192.168.1.107. You execute the Datagram test, and then pass this IP address to the other machines.

The screen displays that datagrams are being sent, and you get reports on the packet success rate and the megabytes per second.

Datagram Test (continued)

You should look for a packet success rate of greater than .99, or even 1.0, which is ideal. Often, you observe that the network is dropping packets and the packet success rate is quite low. If that is the case, Coherence and other network applications will perform poorly. So, at that point, you need to diagnose the reason for packet loss.

The recommended rate is about 80 to 115 megabytes per second on Gigabit Ethernet. If you do not get this rate, something needs to be tuned. You may have forgotten to execute the commands discussed earlier to tune your operating system, or the switch itself needs to be tuned.

Again, the older network cards, the older PCI buses, may be limited to 45 megabytes, due to bus bandwidth limitations.

The Datagram test has a command line option `-txRate` that you can use to lower transmission rate and find out at what point the network is saturated. You do not have to set this option on a well-tuned Gigabit Ethernet. This is one method using which you can lower the transmission rate to find the point where you do not get any packet drops and where you get optimal bandwidth.

Included with Coherence is a Datagram Test utility that can be used to test and tune network performance between two or more machines. The Datagram test operates in one of three modes, either as a packet publisher, a packet listener, or both. When run, a publisher transmits UDP packets to the listener that measures the throughput, success rate, and other statistics.

To achieve maximum performance, it is suggested that you tune your environment based on the results of these tests.

The Datagram test supports a large number of configuration options, though only a few are required for basic operation. To run the Datagram Test utility, use the following syntax from the command line:

```
java com.tangosol.net.DatagramTest <command value ...>
<addr:port ...>
```

Listener:

```
java -server com.tangosol.net.DatagramTest -local box1:9999
-packetSize 1468
```

Publisher:

```
java -server com.tangosol.net.DatagramTest -local box2:9999
-packetSize 1468 box1:9999
```

For ease of use, the `datagram-test.sh` and `datagram-test.cmd` scripts are provided in the Coherence `bin` directory, and can be used to execute this test.

Test network performance between two servers, for example, Server A with IP address 195.0.0.1 and Server B with IP address 195.0.0.2. One server acts as a packet publisher and the other as a packet listener. The publisher transmits packets as fast as possible and the listener measures and reports the performance statistics.

- First, start the listener on Server A with the command `datagram-test.sh`.
- After pressing Enter, you should see the Datagram Test utility showing you that it is ready to receive packets. As you can see, by default, the test tries to allocate a network receive buffer that is large enough to hold 1428 packets, or about 2 MB. If it is unable to allocate this buffer, it reports an error and exits. You can either decrease the requested buffer size using the `-rxBufferSize` parameter, or increase your OS network buffer settings.

Datagram Test (continued)

- For best performance, it is recommended that you increase the OS buffers.
- After the listener process is running, you may start the publisher on Server B, directing it to publish to Server A.

```
datagram-test.sh servera
```

- After pressing Enter, you should see the new Datagram test instance on Server B start both a listener and a publisher. Note that in this configuration, Server B's listener is not used. A series of "o" and "O" tick marks appear as data is (O)output on the network. Each "o" represents 1000 packets, with "O" indicators at every 10,000 packets.
- On Server A, you should see a corresponding set of "i" and "I" tick marks, representing network (I)nput. This indicates that the two test instances are communicating.

Service Configuration

- Set distributed cache `<thread-count>` to the number of CPU cores if you are I/O bound or compute-intensive.
- If using CacheStore, ideal `<thread-count>` will be based on estimate of cache hit-to-miss ratio. Ideally, there will be enough threads to handle all concurrent misses, plus at least one additional thread to handle hits.
- Set `<thread-count>` only for distributed topologies. Replicated topologies are single-threaded.
- ExtendTCPProxyService thread count:
 - Is necessary if you use TCP/IP (Coherence*Extend) clients
 - Must be increased if clients perform actions with high latency
 - Can be decreased if clients are event-driven
 - Typical ratio is 5 to 10 clients per proxy



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Service Configuration

Coherence has services that are launched by default, and these services are run automatically on a cluster. But, you can configure these services. For example, you can set a thread count for these services if your tasks are compute-intensive.

You can set the number of threads by setting the `<thread-count>` attribute in the XML configuration file. However, `<thread-count>` is only for distributed topologies. By default, the service thread runs as only one thread per JVM. But, you may want to increase the thread count if you have EntryProcessors or event listeners running. This is something that you should experiment with for optimal performance.

For Coherence*Extend, you need to set the ExtendTCPProxyService thread count. You can increase the number of threads if your clients perform actions with high latency, or decrease the number of threads if your clients are event-driven. The typical ratio is 5 to 10 clients per proxy. You run proxy services on storage-disabled JVMs. For Coherence*Extend, you need to run at least one proxy. It is recommended that you run multiple proxies with different port IDs. Thus, if one proxy fails, you have the other proxy as backup.

It is important to note that when multiple cache schemes are defined for the same cache service name, the first to be loaded dictates the service-level parameters. Specifically, partition count, backup count, and thread count are shared by all caches of the same service.

Setting Thread Count

- In coherence-cache-config.xml:

```
<distributed-scheme>
  <scheme-name>MyDistributedScheme</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>
    <local-scheme></local-scheme>
  </backing-map-scheme>
  <thread-count>4</thread-count>
  <autostart>true</autostart>
</distributed-scheme>
```

coherence-cache-config.xml



- On the Java command line when launching JVMs:

```
java -Dtangosol.coherence.distributed.threads=4
```

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Setting Thread Count

For caches that are backed by a cache store, it is recommended that the parent service should be configured with a thread pool because requests to the cache store may block on I/O. The pool is enabled via the thread-count element. For non-*CacheStore*-based caches, more threads are unlikely to improve performance and should be left disabled.

This is an example of how to set the thread count in the XML configuration file. In this example, for the distributed cache server, you execute four threads. You can set it either in the XML configuration file or on the Java command line as follows:

`-Dtangosol.coherence.distributed.threads=4` (where 4 is the number of threads that you want to run).

Partition Count

- Partition count is a required setting in coherence-cache-config.xml for distributed caches.
- Set *<partition-count>* to a prime number that ensures that a single partition is 50 MB or less.
- Keep in mind that very high partition counts (>16,000) can lead to performance issues.

```
<distributed-scheme>
  <scheme-name>default-distributed</scheme-name>
  <service-name>DistributedCache</service-name>
  <backing-map-scheme>...etc...</backing-map-scheme>
  <partition-count>1601</partition-count>
</distributed-scheme>
```



coherence-cache-*config*.xml



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Partition Count

The *<partition-count>* element specifies the number of partitions that a partitioned cache will be "chopped up" into. Each member running the partitioned cache service that has the *<local-storage>* subelement set to true will manage a balanced number of partitions. The number of partitions should be a prime number, and sufficiently large such that a given partition is expected to be no larger than 50 MB in size. A list of first 1,000 prime numbers can be found at <http://primes.utm.edu/lists/>.

The following are good defaults for sample service storage sizes:

service storage	partition-count
100M	257
1G	509
10g	2039
50G	4093
100G	8191

Tuning the OS for Coherence*Extend

On Windows:

- Run `optimize.reg`, and then reboot.
- Set the send and receive TCP/IP buffer sizes to at least 128KB

use the `<send-buffer-size/>`
and `<receive-buffer-size/>`
configuration elements

UNIX

- Configure the listen socket to be reusable to avoid `TIMED_WAIT` errors

use the `<reusable/>`
configuration element. Note
required as of version 3.6.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Tuning the OS for Coherence*Extend

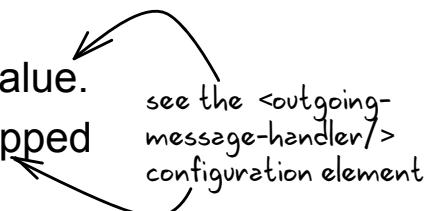
Windows

- There is a registry file that can be used to optimize network settings after its settings are merged with the Windows registry and the computer is rebooted.
- Set the send and receive TCP/IP buffer sizes to at least 128KB so that the send and receive buffers are large enough that packets are not lost by the operating system when under load. In some cases, the default buffer sizes for operating systems are set so low that it is impossible to either send or receive enough packets to use the bandwidth of a network connection.

UNIX:

- By default, many UNIX systems that close a networking socket will cause the socket to enter into a `TIMED_WAIT` state, a state in which a new connection cannot be established. Use the `<reusable/>` configuration element to make the listen socket reusable to avoid restart errors. Note that as of Coherence 3.6, this is no longer required on Windows or Linux as Coherence is automatically configured to be reusable.

Tuning the Coherence*Extend Client Side

- Set a reasonable request timeout value.
- Configure a heartbeat to detect dropped connections.

see the <outgoing-message-handler/> configuration element
- Always configure two or more remote addresses.

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Tuning the Coherence*Extend Client Side

The request-timeout element is used after the client has already connected, and is used to detect fatal errors during request processing or a non-responsive Proxy Service. Configure a heartbeat to help detect potentially dropped connections. Use the <outgoing-message-handler/> configuration elements to set timeout and heartbeat values. Always configure two or more remote addresses for both load balancing and failover of TCP/IP clients.

Tuning the Coherence*Extend Cluster Side

Run a separate tier of storage-disabled ProxyService JVMs and tune appropriately.

- Configure a ProxyService thread pool.
- Optionally, configure a heartbeat.
- Limit the size of the NIO ByteBuffer pools to throttle "greedy" clients:

use multiple
<proxy-scheme/>
configuration elements

```

<incoming-byte-buffer-pool>
    <buffer-size>100KB</buffer-size>
    <buffer-type>DIRECT</buffer-type>
    <capacity>1024</capacity>
</incoming-byte-buffer-pool>

<outgoing-byte-buffer-pool>
    <buffer-size>100KB</buffer-size>
    <buffer-type>DIRECT</buffer-type>
    <capacity>1024</capacity>
</outgoing-byte-buffer-pool>
```



coherence-cache-config.xml

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Tuning the Coherence*Extend Cluster Side

When tuning the cluster side of Coherence*Extend, it is a good practice to run a separate tier of proxy servers that are storage-disabled. This provides the following benefits:

- Proxy servers have different tuning needs than regular cluster members. Proxy servers manage numerous client connections and serialization processing. It is best to keep the actual data processing separate from the processing that occurs in a proxy server.
- Keeping proxy servers storage-disabled allows them to be started and stopped without regard for causing data repartitioning across the cluster.
- Running multiple proxy servers allows for high availability and scalability.

Configure a heartbeat to detect when connections are dropped to avoid long waits on the network for sending and receiving data.

Limit the size of the NIO ByteBuffer pools to ensure that certain clients don't use too much of the proxy server process' memory. NIO buffers are used for all I/O.

Cluster Quorum

Quorum:

- Ensures that a minimum number of members are kept in the cluster when terminating suspect members
- Is useful in networks with variable performance

```
<cluster-config>
  <member-identity>
    <role-name>Server</role-name>
  </member-identity>
  <cluster-quorum-policy-scheme>
    <timeout-survivor-quorum role="Server">5</timeout-survivor-quorum>
  </cluster-quorum-policy-scheme>
</cluster-config>
```

XML

coherence-cache-config.xml



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Cluster Quorum

A quorum, in Coherence, refers to the minimum number of service members that are required in a cluster before a service action is allowed or disallowed. Quorums are beneficial because they automatically provide assurances that a cluster behaves in an expected way when member thresholds are reached. Quorum thresholds policies are set using a cache configuration file.

The cluster quorum policy defines a single quorum (the timeout survivor quorum) for the cluster service. The timeout survivor quorum mandates the minimum number of cluster members that must remain in the cluster when the cluster service is terminating suspect members. A member is considered suspect if it has not responded to network communications, and is in imminent danger of being disconnected from the cluster. The quorum can be specified generically across all members, or constrained to members that have a specific role in the cluster, such as client or server members.

This quorum is typically used in environments where network performance varies. For example, intermittent network outages may cause a high number of cluster members to be removed from the cluster. Using this quorum, a certain number of members are maintained during the outage, and are available once the network recovers. This behavior also minimizes the manual intervention required to restart members. Naturally, requests that require cooperation by the nodes that are not responding will not be able to complete and will be either blocked for the duration of the outage, or will be timed out.

Partitioned Cache Quorum

- Mandates *how many service members* are required before partitioned cache service operations begin.
- Four kinds of configurable quorums:

```
<caching-schemes>
  <distributed-scheme>
    <scheme-name>partitioned-cache-with-quorum</scheme-name>
    <service-name>PartitionedCacheWithQuorum</service-name>
    <backing-map-scheme>
      <local-scheme/>
    </backing-map-scheme>

    <partitioned-quorum-policy-scheme>
      <restore-quorum>3</restore-quorum>
      <distribution-quorum>4</distribution-quorum>
      <read-quorum>3</read-quorum>
      <write-quorum>5</write-quorum>
    </partitioned-quorum-policy-scheme>

    <autostart>true</autostart>
  .
.
```

coherence-cache-config.xml



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Partitioned Cache Quorum

The partitioned cache quorum policy defines four quorums for the partitioned cache service (DistributedCache) that mandate how many service members are required before different partitioned cache service operations can be performed:

Restore Quorum: Mandates the minimum number of storage-enabled members of a partitioned cache service that must be present before the partitioned cache service is allowed to restore lost primary partitions from backup

Distribution Quorum: Mandates the minimum number of storage-enabled members of a partitioned cache service that must be present before the partitioned cache service is allowed to perform partition distribution

Read Quorum: Specifies the minimum number of storage-enabled members of a partitioned cache service that must be present in order to process read requests. A read request is any request that does not mutate the state or contents of a cache.

Write Quorum: Specifies the minimum number of storage-enabled members of a partitioned cache service that must be present in order to process write requests. A write request is any request that may mutate the state or contents of a cache.

Partitioned Cache Quorum (continued)

These quorums are typically used to indicate at what service member levels different service operations are best performed, given the intended use and requirements of a distributed cache. For example, a small distributed cache may only require three storage-enabled members to adequately store data and handle projected request volumes. A large distributed cache may require 10, or more, storage-enabled members to adequately store data and handle projected request volumes. Optimal member levels are tested during development, and then set accordingly, to ensure that the minimum service member levels are provisioned in a production environment.

If the number of storage-enabled nodes running the service drops below the configured level of read or write quorum, the corresponding client operation will be rejected by throwing the `com.tangosol.net.RequestPolicyException`. If the number of storage-enabled nodes drops below the configured level of distribution quorum, some data may become "endangered" (no backup) until the quorum is reached. Dropping below the restore quorum may cause some operation to be blocked until the quorum is reached, or the operation to be timed out.

Extend Proxy Quorum

- Mandates the *minimum number of proxy service members* that must be available before the proxy service can allow client connections.

```
<caching-schemes>
  <proxy-scheme>
    <scheme-name>proxy-with-quorum</scheme-name>
    <service-name>TCPProxyService</service-name>
    <acceptor-config>
      <tcp-acceptor>
        <local-address>
          <address>localhost</address>
          <port>32000</port>
        </local-address>
      </tcp-acceptor>
    </acceptor-config>
    <autostart>true</autostart>
    <partitioned-quorum-policy-scheme>
      <connect-quorum>3</connect-quorum>
    </partitioned-quorum-policy-scheme>
  . . .
```

coherence-cache-config.xml



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Proxy Quorum

The proxy quorum policy defines a single quorum (the connection quorum) for the proxy service. The connection quorum mandates the minimum number of proxy service members that must be available before the proxy service can allow client connections.

This quorum is typically used to ensure enough proxy service members are available to optimally support a given set of TCP clients. For example, a small number of clients may efficiently connect to a cluster using two proxy services. A large number of clients may require three or more proxy services to efficiently connect to a cluster. Optimal levels are tested during development, and then set accordingly, to ensure that the minimum service member levels are provisioned in a production environment.

Coherence Logs

- In addition to garbage collection logs (discussed earlier), Coherence logs should be saved for every JVM.
- The following setting saves the log and appends the process ID to the log file name:

```
java -Dtangosol.coherence.log=log/coherence-server$$.log [other options  
...]
```

- You can use pluggable Java logging libraries such as log4J.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Logs

By default, Coherence logs data to the standard error. However, the log files are not set up by default, therefore it is important that you set them up.

In addition to the garbage collection logs that you discussed earlier, Coherence logs should be saved for every JVM. This is very useful when you face an issue and request support. When you raise a service request, the support team usually requires the logs to diagnose the problem. Therefore, it is important to save the logs. You can also set up your own logging system by using log4j.

The example shows how to save the log files to a log in UNIX and the \$\$ sign adds the process ID to the log files. So, in this case, each server JVM has its own log file.

Performance Testing: General Advice

- Use storage-disabled JVMs for clients.
- Implement `ExternalizableLite` or POF for all custom objects.
- Maximize client threads for throughput tests.
- Minimize client threads for latency tests.
- Use bulk operations when possible:
 - `cache.putAll(map)` versus `cache.put(k, v)`
- Minimize application-introduced bottlenecks:
 - Use `java.util.concurrent` for your local collections that are accessed by multiple threads (do not use `synchronized Vector` or similar).
 - Avoid synchronized operations.
 - Monitor application bottlenecks with a Java profiler tool.
- Scale out both client and server tiers.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Performance Testing: General Advice

In summary:

- Use storage-disabled JVM for clients.
- Implement `ExternalizableLite` for all custom objects. `ExternalizableLite` greatly reduces the bandwidth and CPU usage for serialization.
- Maximize client threads for throughput tests. If you want to have high throughput or large number of transactions per second, the larger the number of clients the better.
- If you are testing latency, it is better to minimize the client threads. This is very important because you test the response time here.
- Whenever possible, use bulk operations in your application. For example, `putAll()` is much more efficient than `put()`.
- Minimize application-introduced bottlenecks. This is very important. Often, when you see scaling issues with Coherence, if you observe scaling that brings performance limitations, it is usually because the network is saturated, or because you introduced application bottlenecks. Some factors to limit scalability include using synchronized vectors or using the `synchronized` keyword. This is not necessary in Coherence

Performance Testing: General Advice (continued)

because Coherence has built-in concurrency. If you use synchronized vectors or synchronized keywords, this means that the threads have to wait for the locks to be released.

Please refer to "Evaluating Performance and Scalability" section on Coherence Knowledge Base (on Coherence wiki page).

- `java.util.concurrent` is available in JDK 1.5. If you use your own collections, it is a good idea to use `java.util.concurrent` rather than older collections like synchronized vector.
- Avoid synchronized operations where possible.
- As with any Java application, it is a good idea to monitor the applications using some kind of a Java profiler tool.
- Coherence is designed for horizontal scaling, so scale out both the client and server tiers. Add servers when you need capacity.
- Use `EntryProcessors` to modify entries on the server side to keep locking contention to a minimum.
- Use keys versus filters and queries whenever possible to avoid expensive operations on the cluster.
- Always use an index for queries.

Quiz

How many 1 GB JVMs can you run on a box with 16 GB RAM?

- a. 13
- b. 14
- c. 15
- d. 16



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz

How many 16 GB machines are required to support 20 GB of data in the grid (plus backups)?

- a. 2
- b. 5
- c. 10
- d. 20



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

- Configure and deploy a JMX management server
- Configure and run the Reporter
- Explain the tools available for use with Coherence
- Tune and configure JVM
- Size a Coherence system
- Configure a network
- Tune an operating system
- Set buffer sizes
- Test datagram and multicast
- Monitor Coherence



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.12.01 Overview: Configuring and Running the Reporter

This practice covers the following topics:

- Configuring a Coherence JMX management node and JMX console
- Configuring a cluster node to run as a managed node
- Configuring the Reporter to run on the management node
- Running Reporter and viewing the reports



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Practice.12.01 Overview: Configuring and Running the Reporter

At the end of this practice, you should be able to:

- Configure a Coherence JMX management node.
- Configure the Reporter to run on the management node.
- Configure a cluster node to run as a managed node.
- Change settings that control how the Reporter runs.
- Identify and write a Reporter report group batch file.
- Start a Coherence cluster that includes a management server.
- Connect a JMX console to a Coherence management server to analyze Coherence MBean attributes and operations.
- Start and stop the Reporter from a JMX console.
- Generate canned reports using the Reporter.
- View reports generated by the Reporter

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and ORACLE CORPORATION use only

13

Understanding Coherence Security

ORACLE®

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to describe the security options available with Coherence.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Agenda

Security Overview

- Cluster Connectivity (TCMP)
- TCMP Access Control
- Extend
- Transport Layer Security



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Cluster Connectivity (TCMP)

- Problem:
 - Coherence cluster membership is automatic (that is both good news and bad news)
 - Easy for malicious clients to connect and access data
- Solutions:
 - Coherence cluster configuration settings
 - Programmatic and declarative access control
 - Encrypted network connections (two-way SSL)



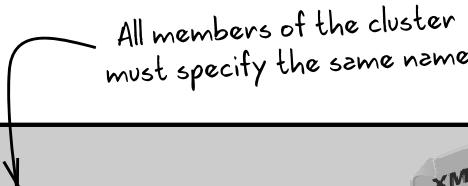
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Cluster Connectivity (TCMP)

The default Coherence client connects to any running Coherence cluster automatically. At first, this appears to make Coherence an easy target for unauthorized users to exploit to access data that is stored in the cache. Coherence provides a number of security mechanisms that ensure that unauthorized users cannot access the cluster itself, or the data that is stored in the cache. This lesson covers the configuration settings, security frameworks, and network encryption capabilities offered by Coherence.

Member Identity

Use member-identity to restrict access to the cluster:



```
<member-identity>
  <cluster-name system-property="tangosol.coherence.cluster">
    MySharedClusterName</cluster-name>
  <site-name    system-property="tangosol.coherence.site"></site-name>
  <rack-name   system-property="tangosol.coherence.rack"></rack-name>
  <machine-name system-property="tangosol.coherence.machine"></machine-name>
  <process-name system-property="tangosol.coherence.process"></process-name>
  <member-name  system-property="tangosol.coherence.member"></member-name>
  <role-name    system-property="tangosol.coherence.role"></role-name>
  <priorities   system-property="tangosol.coherence.priority"></priorities>
</member-identity>
```

tangosol-coherence.xml



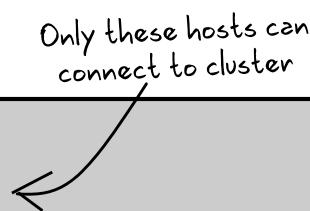
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Member Identity

Coherence cache, proxy, management, and client processes all become "joined" with the cluster at startup. Coherence has the ability to autodiscover other cluster members. Due to this nature of Coherence, it is very important to ensure that rogue Coherence processes are not able to join the production cluster. Coherence provides features that control which processes can join the cluster, such as Member Identity. The `tangosol-coherence.xml` file has a `<member-identity>` element in the cluster configuration file. When set, only cache services running with the same `member-identity` set can join the cluster.

Authorized Hosts

Use authorized-hosts to restrict which hosts can join the cluster:



Only these hosts can connect to cluster

```
<authorized-hosts>
    <host-address>hostname1</host-address>
    <host-address>hostname2</host-address>
    <host-range>
        <from-address>10.1.1.1</from-address>
        <to-address>10.1.1.100</to-address>
    </host-range>
</authorized-hosts>
```

tangosol-coherence.xml



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Authorized Hosts

Coherence also provides a feature to allow only certain hosts to access the cluster. The tangosol-coherence.xml file has an `<authorized-hosts>` element in the cluster configuration. When set, only cache services running on these hosts can join the cluster.

TCMP Access Control

AccessController security framework:

- Works with JAAS authenticated user
- Allows Coherence to:
 - Authorize access to clustered resources
 - Encrypt and decrypt keys for authentication
- Default JAAS keystore-based implementation

```
import com.tangosol.net.security.AccessController;
public interface AccessController
{
    void checkPermission(ClusterPermission perm, Subject subject);
    Object decrypt(SignedObject so, Subject subjEncryptor);
    SignedObject encrypt(Object o, Subject subjEncryptor);
}
```

AccessController.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

TCMP Access Control

In addition to joining the cluster, users of Coherence applications need to be granted access to the data within the cache. Coherence provides a security framework that leverages the standard JAAS security model to authenticate a user, and to authorize access to the cluster and its cache data. Coherence provides an Access Controller framework that allows you to implement your own security mechanisms. Coherence provides a DefaultController implementation that uses a Java keystore-based LoginModule and permission.xml file to provide authentication and authorization services. Other LoginModule types may be implemented to integrate with existing authentication and authorization systems, such as an LDAP LoginModule, and Oracle Entitlements Server (OES), a centralized fine-grained entitlement engine.

The Access Controller framework provides hooks for authentication and authorization. This includes methods for encrypting and decrypting identity keys for authentication purposes. For example, if the public key is used to successfully decrypt the identity, then it must have been encrypted with the associated private key.

TCMP Access Control (continued)

Authorization is provided for operations on clustered resources, such as:

- Creating a new clustered cache or service
- Joining an existing clustered cache or service
- Destroying an existing clustered cache

ClusterPermission

Represents access to a clustered resource such as a Service or NamedCache:

- Passed into checkPermission ()
- Permissions: ALL, CREATE, DESTROY, JOIN, NONE
- Consists of a target name and a set of actions valid for that target

```
import com.tangosol.net.ClusterPermission;
public final class ClusterPermission {
    String getName();
    String getActions();
    String getServiceName();
    boolean implies(Permission permission);
    ...
}
```

ClusterPermission.java



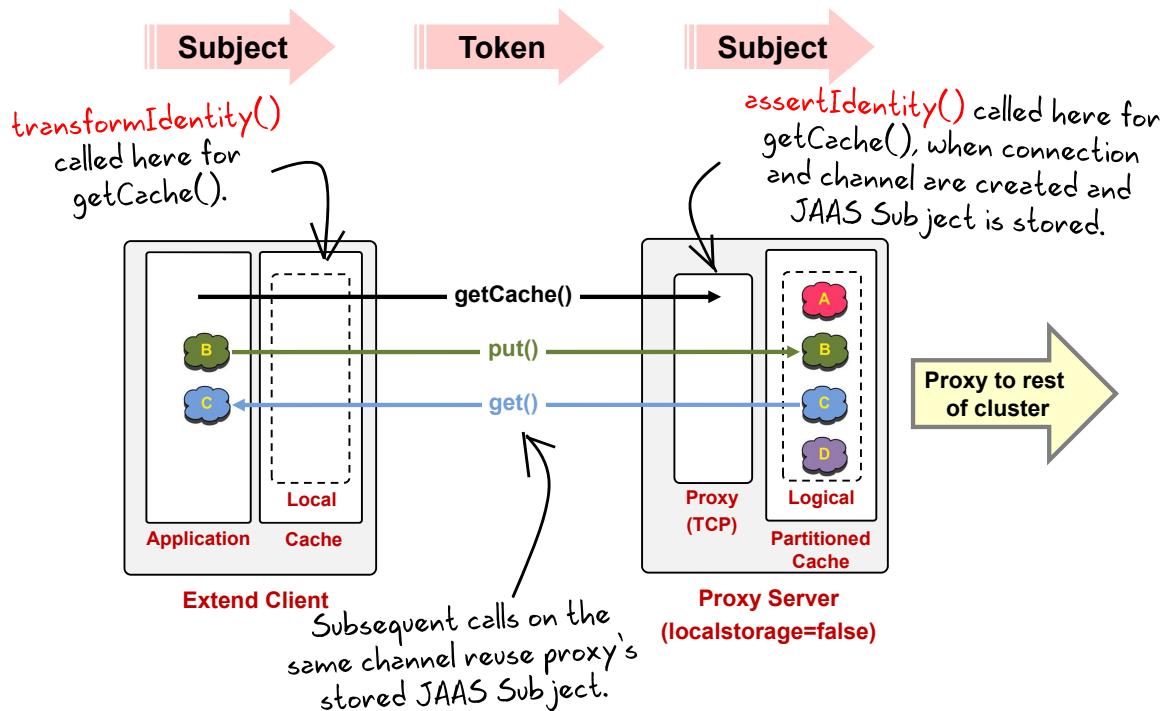
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ClusterPermission

The AccessController's checkPermission() method uses a ClusterPermission class as the first parameter to pass the criteria used to make the authorization decision for the request. The ClusterPermission class specifies which resource is being requested (the target), and the privileged actions that are valid for that resource.

- getName () : Contains the name of the cache or service that needs to be protected
- getActions () : Return the actions (privileges) as a String in a canonical form. This represents the actions the current request is trying to attempt.
- getServiceName () : Return the service name for this permission object or null if the permission applies to any service
- implies () : Check if the specified permission's actions are "implied by" this object's actions

Extend Pluggable Identity: Architecture



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Architecture

Extend Pluggable Identity answers the question, “How do I get an identity from the Extend client to the Coherence proxy server?” Coherence*Extend provides the ability to convert a JAAS Subject to a corporate standard security token type, such as SAML, or a custom token type, such as shown in the examples of this presentation, to support single-sign-on and authorization checks for clients connecting to a Coherence proxy server. The client side calls a `transformIdentity()` hook that can be custom implemented, which is responsible for converting a JAAS Subject into the token used for identity assertion on the proxy side. The proxy server calls the `assertIdentity()` hook when a new user is connecting to the server, and is responsible for converting the token into a usable JAAS Subject, which is stored and associated with the connection, and multiplexed channel for that user. Subsequent calls on a NamedCache or Service that are requested on that channel, for that user, reuse the stored Subject to avoid the expensive operation of asserting the identity repeatedly. There is no notion of a "session" in Coherence. However, when the CacheService's `releaseCache()` method is called, the associated channel is closed and any subsequent requests by that subject will trigger the identity assertion process again. It is up to the `identityAsserter()` code to check to see if the subject's token is still valid.

Extend Pluggable Identity: Client-Side

- Authenticate using company-specific standard such as Kerberos, SAML, or others
- Make calls in context of the identity using `Subject.doAs()`
- No knowledge of security token needed on client
- Cache reference permanently associated with that `Subject` (More details later about cache reference)



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Client-Side

Authentication occurs either within an application that has embedded Coherence*Extend as part of its architecture to work with a Coherence clustered cache, or is a standalone Coherence*Extend client. In either case, the authentication mechanism used by the client application is entirely up to the application. This could include standard JAAS authentication provided by Coherence, a Kerberos solution, a SAML solution, or some other solution. The end result is that whatever mechanism is used, it must be represented as a JAAS `Subject` in order to perform a `Subject.doAs()` operation that wraps a call to either `CacheFactory.getCache()` or `getService()`, which will get the `Subject` associated with a channel for all user requests for that cache or service.

Extend Pluggable Identity: Client-Side Code Example

Bootstrapping the Subject:

```

Subject subject = MySecurityHelper.login(user);           Application handles login

try {
    NamedCache cache = (NamedCache) Subject.doAs(
        subject, new PrivilegedExceptionAction() {
            public Object run() throws Exception {
                return CacheFactory.getCache("airports");
            }
        });
} catch (Exception e) {
    //Get exception if the password is invalid
    System.out.println("Unable to connect to proxy");
    e.printStackTrace();
}

```

Subject.doAs() sets Subject context

CacheFactory.getCache()
method propagates Subject to proxy server

MyExtendClient.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Client-Side Code Example

Here is a code example of a client invoking `Subject.doAs()` with a `PrivilegedExceptionAction` that calls `CacheFactory.getCache()`. This initiates communication with the proxy server over TCP/IP. Here is what happens behind the code:

- If this is the first connection, then the connection is established, then a channel is created.
- The Extend client invokes the `transformIdentity()` method to convert the `Subject` into whatever token is required for identity assertion on the proxy side.
- The token form of the identity is transferred to the proxy server.
- The `Subject` is associated with the channel for subsequent calls.

The proxy side of the equation is discussed in the identity assertion slides.

Extend Pluggable Identity: Identity Transformer

- Coherence*Extend client produces token:
 - IdentityTransformer Interface
 - Object transformIdentity(Subject subject)
 - Subject from current security context (may be null)
 - Subject is transformed into a security token
 - Token can be any **Serializable** type
 - Token will be passed to proxy only once when:
 - A new connection opened
 - A new channel for an existing connection
 - DefaultIdentityTransformer returns a Serializable Subject as the token



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Identity Transformer

The `transformIdentity()` method can be implemented with custom code that converts the `Subject` to a token to work with any corporate security standard. The `transformIdentity()` method is called, and the security token is passed to the proxy server up to three times for an authenticated user:

- If it is the first TCP connection, then the `transformIdentity()` method is called and the token passed to the proxy server when the connection is created.
- The connection is multiplexed, using a Coherence mechanism called channels. Each new `Subject` that connects to Coherence gets its own channel and:
 - When the channel is *created*, the `transformIdentity()` method is called, and the token is passed.
 - When the channel is *accepted*, the `transformIdentity()` method is called, and the token is passed.

This is required to avoid security vulnerabilities.

The out of the box `DefaultIdentityTransformer` returns a `Serializable` version of a `Subject`.

Extend Pluggable Identity: Identity Transformer Code

Example of creating a token from a Subject:

```
public Object transformIdentity(Subject subject) throws SecurityException {
    Set setPrincipals = subject.getPrincipals();
    String[] asPrincipalName = new String[setPrincipals.size() + 1];
    int i = 0;
    ↗ Security token is an array of strings
    asPrincipalName[i++] =
        System.getProperty("coherence.password", "secret-password"); ↗ First element
                                                               is password

    for (Iterator iter = setPrincipals.iterator(); iter.hasNext();) { ↗ Second element
        asPrincipalName[i++] = ((Principal)iter.next()).getName(); ↗ is username, and
    }                                                               subsequent
                                                               elements are
                                                               roles, if any

    return asPrincipalName;
}
```

MyIdentityTransformer.java

WARNING: These examples are simplified! Need better password management!

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Identity Transformer Code

This is a code example of the `transformIdentity()` method that converts a Subject into an array of Strings, which is returned as the security token:

- First element of array is password
- Second element is user name
- Subsequent elements are role names

This is a simplified example! A real world version would require:

- A better mechanism for getting the password other than the clear text system property shown here
- Signing and/or encryption of token

Extend Pluggable Identity: Proxy-Side Identity Asserter

Coherence*Extend proxy deserializes token:

- IdentityAssertor Interface
 - Subject assertIdentity(Object oToken)
 - Security token from client (may be null)
 - Token is asserted (validated)
 - Subject is produced from token
- Code will be executed in the Subject's context
- DefaultIdentityAssertor returns Subject to the proxy



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Proxy-Side Identity Asserter

The assertIdentity() method can be implemented with custom code that converts the token created by transformIdentity() back into a Subject that the proxy server can use. The assertIdentity() method is called for an authenticated user as the connections and channels are created per each Subject. The assertIdentity() code is responsible for validating the token. Once the proxy server has validated and associated the Subject, all subsequent calls on that channel are executed in the context of that Subject. The DefaultIdentityAssertor supplied with Coherence returns a Subject to the proxy.

Extend Pluggable Identity: Identity Asserter Code

Example of creating a Subject from a token:

```
public Subject assertIdentity(Object oToken) throws SecurityException {
    String sPassword = System.getProperty("coherence.password", "secret-password");
    Set setPrincipalUser = new HashSet();
    Object[] asName = (Object[])oToken;
    //First name must be password
    if (((String)asName[0]).equals(sPassword)) {
        for (int i = 1, len = asName.length; i < len; i++) {
            setPrincipalUser.add(new PofPrincipal((String)asName[i]));
        }
    }
    return new Subject(true, setPrincipalUser, new HashSet(), new HashSet());
} catch (SecurityException e) {
    throw new SecurityException("Access denied");
}
```

MyIdentityAsserter.java

Annotations on the code:

- Compare first element against password
- Second element is username
- Returns Subject representation for token



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Identity Asserter Code

This sample code shows an example of converting the passed token created by the `transformIdentity()` method into a JAAS Subject. This code does not actually validate the token.

Just as with the `transformIdentity()` code, this example is oversimplified! A real world version would require:

- A better mechanism for getting the password, instead of using a clear text system property
- Decrypting and validating the token

The `PofPrincipal()` method creates a generic principal implementation that can be used to represent the identity of any remote entity.

Extend Pluggable Identity: Identity Asserter Semantics

Validate:

- Solves trust issue of token received from the network
- Re-authenticate if necessary or desired

Fast Performance:

- Called once per connection open and channel open
- Channel permanently associated with Subject



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Identity Asserter Semantics

Security solutions such as SAML generate and assert tokens based on various types of secure mechanisms, such as encrypted keys and digitally signed tokens, that ensure the sender is who the token says they are. The token is asserted (or validated) within the `assertIdentity()` method to ensure the validity of the token. Usually, this validation process is an expensive operation, due to the processing needs of encryption and decryption algorithms. To alleviate this performance hit, Coherence associates the `Subject` with the channel that is created for the user (`Subject`), and subsequent calls on that `Subject`'s behalf execute on the proxy without performing the assertion process.

Extend Pluggable Identity: Identity Asserter Subject Scoping

Subject scoping ties a Subject's requests to a channel:

- Get reference in Subject context (`Subject.doAs()`)
 - `cacheFactory.getCache()`
 - `cacheFactory.getService("InvocationService")`
- Any methods called on those references are automatically associated with the Subject that created them
- Allows access control based on Subject

WARNING: Subject scoping is OFF by default! This causes all cache requests by all users to use the same channel!

ADVICE: If Subject scoping is turned ON, reuse references to NamedCache to avoid expensive operations!



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Pluggable Identity: Identity Asserter Subject Scoping

The concept of associating a Subject with a proxy channel to keep requests from one user separate from requests of another user is called Subject scoping. By default, this option is turned off as an optimization in Coherence. This causes the first Subject that establishes the TCP connection to validate across the network. All subsequent calls from all users, even if they perform a `CacheFactory.getCache()` operation within a `Subject.doAs()`, will bypass the identity assertion process, and all users will receive a reference to the same cache that the first user was validated to use.

When Subject scoping is turned on, the best practice to follow is to store and reuse references to any NamedCache in the code. This is to avoid expensive channel creation and authentication processing in the event that the NamedCache is called by a different thread that does not have the Subject in its stack.

Extend Access Control: Authorization Wrappers

- EntitledNamedCache: Wraps all NamedCache methods on proxy
- EntitledCacheService: Wraps all CacheService methods on proxy
- EntitledInvocationService: Wraps InvocationService methods on proxy



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Access Control: Authorization Wrappers

Coherence*Extend provides wrapper classes that allow for extending the NamedCache, CacheService, and InvocationService classes. These wrapper classes provide the ability to intercept method calls to these classes in order to perform security authorization checks prior to granting the caller access to executing the method.

Extend Access Control: Configuring Authorization Wrappers

Set class names to use in configuration file:

```
<proxy-config>
  <cache-service-proxy>
    <class-name>com.tangosol.examples.security.EntitledCacheService
    </class-name>
  </cache-service-proxy>

  <invocation-service-proxy>
    <class-name>com.tangosol.examples.security.EntitledInvocationService
    </class-name>
  </invocation-service-proxy>
</proxy-config>
```

coherence-cache-config.xml



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Access Control: Configuring Authorization Wrappers

Coherence*Extend authorization wrappers are configured in the `<proxy-config>` element of the `coherence-cache-config.xml` file. Specify the fully qualified class name of the wrapper, and make sure the class is in the classpath of the proxy server.

Extend Access Control: EntitledCacheService

Intercept CacheService methods to authorize access to cache:

```
public NamedCache ensureCache(String sName, ClassLoader loader) {
    MySecurityHelper.checkAccess(MySecurityHelper.READER_ROLE); ←
    return new EntitledNamedCache(super.ensureCache(sName, loader));
} ← EntitledNamedCache created here
public void releaseCache(NamedCache map) {
    if (map instanceof EntitledNamedCache) {
        EntitledNamedCache cache = (EntitledNamedCache) map;
        MySecurityHelper.checkAccess(MySecurityHelper.READER_ROLE); ←
        map = cache.getNamedCache();
    }
    super.releaseCache(map);
} ← Channel is closed but Subject still exists. Next request will trigger identity assertion again.
```

Authorization check prior to cache access

EntitledCacheService.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Access Control: EntitledCacheService

Wrapping the CacheService enables the ability to authorize access to a cache in the first place. In this example, the `ensureCache()` method uses a helper class to check if the current Subject has access to the cache by checking to see if the Subject contains the `READER_ROLE` role or greater. If the Subject has access, then a new `EntitledNamedCache()` class is created and initialized with the underlying super class' `ensureCache()` method, and then returned to the calling client. The `releaseCache()` method checks to see if the `NamedCache` that was passed in is an instance of an `EntitledNamedCache`. If it is, then `releaseCache` checks to see if the Subject has access to release the cache prior to executing the release on the super class. The `releaseCache()` method is analogous to logging out the principal. When it is called, the channel is closed, but the Subject still remains in the system. When the next request from this principal enters the system, the identity assertion mechanism will be called again, and it is up to the code within to validate if the "session" or token is still valid.

Not shown here is the `destroyCache()` method.

Note that the `checkAccess()` method throws an exception on error, so there is no error handling code here.

Extend Access Control: EntitledNamedCache

Intercept NamedCache methods to authorize requests:

```
public Object get(Object oKey) {  
    MySecurityHelper.checkAccess(MySecurityHelper.READER_ROLE); ←  
    return super.get(oKey);  
}  
  
public Object put(Object oKey, Object oValue, long cMillis) {  
    MySecurityHelper.checkAccess(MySecurityHelper.WRITER_ROLE); ←  
    return super.put(oKey, oValue, cMillis);  
}
```

Authorization check prior to cache access

EntitledNamedCache.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Access Control: EntitledNamedCache

The EntitledNamedCache class wraps the methods of the NamedCache interface to intercept the calls for security authorization. This slide shows examples of the get() and put() methods. The get() method checks to see if the caller has READER_ROLE privileges, and the put() method checks to see if the caller has WRITER_ROLE privileges. After the security check grants access, the get() and put() methods of the NamedCache super class are invoked, which performs the actual work.

Extend Access Control: **EntitledInvocationService**

Intercept InvocationService methods to authorize requests:

```
public Map query(Invocable task, Set setMembers) {  
    MySecurityHelper.checkAccess(MySecurityHelper.WRITER_ROLE); ↗  
    return super.query(task, setMembers);  
}
```

Authorization check
prior to service access

EntitledInvocationService.java



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Access Control: **EntitledInvocationService**

The EntitledInvocationService class wraps the methods of the InvocationService in the same way. The InvocationService only has two methods, query() for synchronous service invocation, and execute() for asynchronous invocation. Note that the execute() is not available for Coherence*Extend clients, and therefore is not shown here because it is not available as part of the EntitledInvocationService class.

Extend Access Control: Authorization Example Implementation

Authorization code called by interceptors:

```
public static void checkAccess(String sRoleRequired) {  
    Subject subject = MySecurityHelper.getCurrentSubject();  
    Map mapRoleToId = s_mapRoleToId;  
    Integer nRoleRequired = (Integer)mapRoleToId.get(sRoleRequired);  
  
    for (Iterator iter = subject.getPrincipals().iterator(); iter.hasNext();) {  
        Principal principal = (Principal)iter.next();  
        String sName = principal.getName();  
  
        if (sName.endsWith("_ROLE")) {  
            Integer nRolePrincipal = (Integer)mapRoleToId.get(sName);  
            if (nRolePrincipal == null) {  
                // invalid role  
                break;  
            }  
  
            if (nRolePrincipal.intValue() >= nRoleRequired.intValue()) { return; }  
        }  
    }  
  
    throw new SecurityException("Access denied, insufficient privileges");  
}
```

MySecurityHelper.java

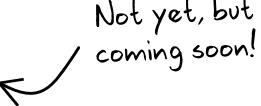


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Extend Access Control: Authorization Example Implementation

This is an example of the `checkAccess()` method that was shown in the previous slides. The code has access to the current `Subject` context, and can compare the passed-in role requirement against the `Subject`'s assigned roles, which are stored from its third element to the last element. This code uses an integer to represent the roles, which in this example is used determine if a user has the intended role, or a higher integer value indicating that the role is assumed.

Transport Layer Security: SSL

- Extend and/or Cluster (TCMP)
- One-way or two-way (client authentication)
- Configurable Trust Manager
 - PeerX509 (default), SunX509, other
- TCMP
 - Requires well-known addresses (WKA) since no multicast SSL
- Deprecate encryption filters
- Extend Support
 - Java
 - .NET
 - C++ 

Not yet, but
coming soon!

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Transport Layer Security: SSL

Coherence provides SSL transport layer security for TCP/IP protocols for Coherence clustered nodes and Coherence*Extend nodes. Currently, there are two included trust managers: PeerX509 which is the default selection, and SunX509. The Coherence framework allows for the integration of any trust manager. When using SSL for clustered (TCMP) nodes, the use of well-known addresses (WKA) TCP/IP connections is required because there is no SSL support for the multicast networking protocol.

The encryption filters that were previously part of the Coherence product are deprecated as of the 3.6 release, and replaced by the SSL feature.

SSL support is available for Coherence*Extend nodes, but the C++ binding is still under development.

Transport Layer Security: SSL Recommendations

- Enable two-way, especially for TCMP
- Test performance
- Hardware acceleration
- Scale out to regain performance



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Transport Layer Security: SSL Recommendations

Oracle recommends using two-way SSL, especially for TCMP clustered nodes, because there is no notion of a client in the cluster, and it does not make sense for one clustered node to provide a trust certificate to another node so it can be trusted, while not receiving some form of trusted artifact in return from the other cluster member. Naturally, with any SSL or encryption-intensive processing, the application should be tested to see if it performs well, and standard performance improving techniques should be employed when needed, such as the use of hardware encryption/decryption accelerators, which can nearly eliminate processing latency, or scaling out the cluster to provide more CPU processing power to handle the load.

Transport Layer Security: Setting up SSL for the Cluster

Simplest possible usage (for testing TCMP)

- Create keystore (JKS) with keypair:

```
keytool -genkeypair -alias admin -keypass password -keystore  
keystore.jks -storepass password
```

Command line

- Specify SSL, password, and WKA using system properties:

```
-Dtangosol.coherence.socketprovider=ssl  
-Dtangosol.coherence.security.password=password  
-Dtangosol.coherence.wka=localhost
```

Command line

- keystore.jks in current directory

- All SSL options also available through configuration



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Transport Layer Security: Setting up SSL for the Cluster

This slide shows the simplest way to set up SSL for clustered nodes:

- A Java Key Store (JKS) keystore is created or used to store a certificate public and private keypair.
- SSL, password, and WKA settings are configured using system properties from the command line. Note in this case that the password is then in clear text, and is used for both the keystore password and the private key password. The keystore.jks file, which holds the keypair, must be in the current directory where the JVM is executed. The alias does not need to be specified because internally, the keystore API being used by Coherence assumes that only a single private key exists in the keystore, but perhaps exists in multiple public certs.
- All the SSL settings are also available through the operational tangosol-coherence.xml configuration override file.

Transport Layer Security: Setting up SSL for *Extend

Simplest possible usage (for testing *Extend):

- Create keystore (JKS) with keypair:

```
keytool -genkeypair -alias admin -keypass password -keystore
keystore.jks -storepass password
```

Command line

- Specify password using system property:

```
-Dtangosol.coherence.security.password=password
```

Command line

- Use <defaults> in client and proxy config:

```
<cache-config>
  <defaults>
    <socket-provider>ssl</socket-provider>
  </defaults>
```



coherence-cache-config.xml

Used for keystore
and private key

- All SSL options also available through configuration.

ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Transport Layer Security: Setting up SSL for *Extend

This slide shows the simplest way to set up SSL for *Extend nodes:

- A Java Key Store (JKS) keystore is created or used to store a certificate public and private keypair.
- Configure the password using the system property from the command line. Note in this case that the password is then in clear text, and is used for both the keystore password and the private key password.
- The <defaults> element in the client cache configuration file and the <proxy-scheme> configuration in the coherence-cache-config.xml file are used to specify SSL as the transport protocol.
- All the SSL settings are also available through the operational tangosol-coherence.xml configuration override file.

Security Examples

- <http://coherence.oracle.com/display/EXAMPLES/Home>
 - PasswordExample
 - AccessControlExample
 - PasswordIdentityTransformer
 - PasswordIdentityAsserter
 - EntitledNamedCache
 - EntitledCacheService
 - EntitledInvocationService
 - Complete configuration
- Examples of everything in this presentation except SSL



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Security Examples

There are security examples available to try these features out for yourself. These examples are not part of the course, but can help get you started with some hands-on materials.

Quiz

What security features can hinder a rogue Coherence process from joining a cluster? (Select all that apply)

- a. AccessController
- b. Authorized Hosts
- c. IdentityAssertion
- d. Member Identity



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: b,d

Authorized hosts allows only those hosts that are configured access to join the cluster, and member identity sets a specific name that must be configured for a cluster node to join the cluster. Random rogue Coherence processes that are not on the authorized hosts list, or have the improperly configured member identity setting, will be denied access to the cluster. The AccessController is for authorizing access to caches that are already running in a cluster. IdentityAssertion is for validating the authentication of a user from a trusted source via a security token of some sort. It does not establish whether or not the cluster node can join the cluster in the first place.

Quiz

What is the purpose of the pluggable identity feature?

- a. To integrate with standard security standards to authenticate users using Coherence*Extend
- b. To provide fine-grained entitlements for authorizing users to caches and cached data based on user properties
- c. To allow integration with standard security protocols to pass trusted Subjects from Coherence*Extend to the cluster
- d. None of the above.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Answer: c

The primary purpose of the pluggable identity feature is to securely pass a user Subject from Coherence*Extend to the Proxy Service in order to do work on behalf of the user. The pluggable identity feature provides hooks for integrating with security standard solutions such as SAML and Kerberos.

Summary

In this lesson, you should have learned how to:

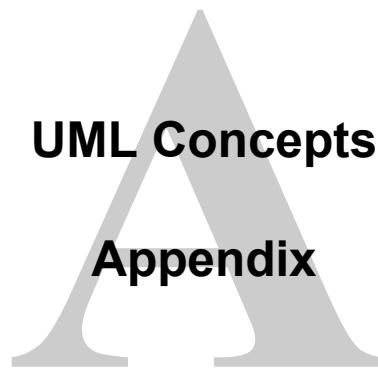
- Explain how to protect a Coherence cluster from rogue processes
- Explain how the AccessController and ClusterPermission objects provide access control from TCMP cluster nodes
- Describe how the Coherence pluggable security token feature works
- Describe how to configure SSL for TCMP and Extend Coherence processes



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you learned about the various security features available in Coherence that protect application data from external attacks. You should now be able to explain these features, and how they work to others.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to read:

- Sequence diagrams
- Structure diagrams
- Component diagrams



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What Is UML?

UML, or the Unified Modeling Language:

- Is a standardized graphical “modeling language”
- Is used to develop implementation-independent software design models
- Allows specification of:
 - User and system interaction
 - Responsibility partitioning (often class diagrams)
 - Data flow and dependency relations
 - Operation orderings (flowcharts and algorithms)

UML is not process, and does not describe how to do things, but rather what should be done.



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

UML Diagrams

There are 14 traditional types of UML diagrams, representing:

- Use cases
- Class diagrams
- Sequence diagrams
- Component Diagrams
- State diagrams
- Activity diagrams
- Packaging diagrams
- And others

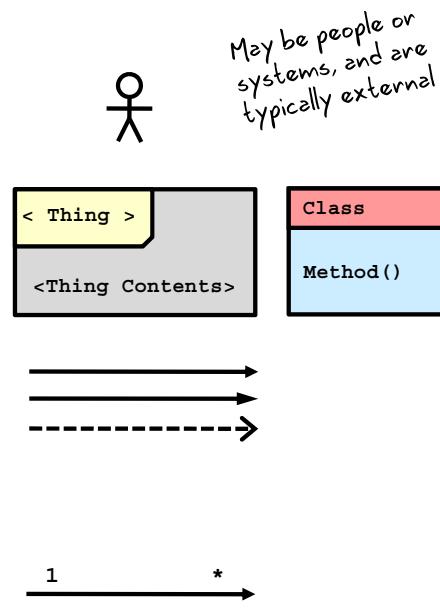


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

UML Components

UML uses a variety of components:

- **Actors:** Indicate points of interaction with a system
- **Boxes:** Represent discrete elements, groupings, and containment
- **Arrows:** Typically indicate flow, dependency, association, or generalization
- **Cardinality:** Is applied to arrows to show relationships between elements, such as 1:1, 1:Many, Many:Many, Many:1, and minimum/maxima



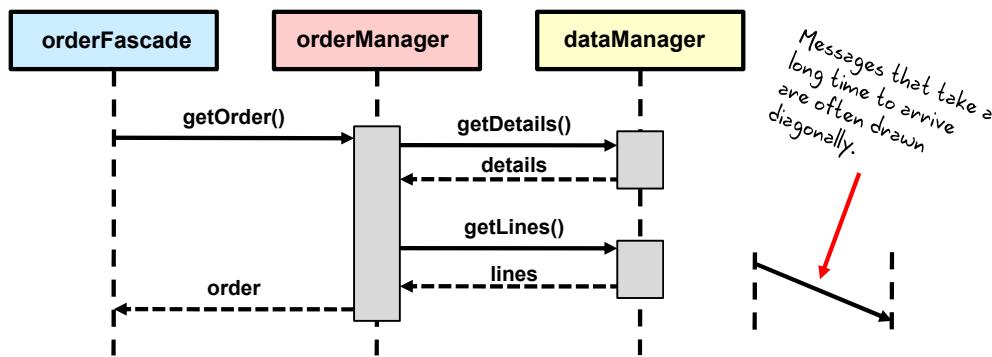
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ORACLE

Sequence Diagrams

A sequence diagram:

- Is used to define an event sequence
- Is focused on message order and outcome
- Shows time from top to bottom
- Shows order of events from left to right



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

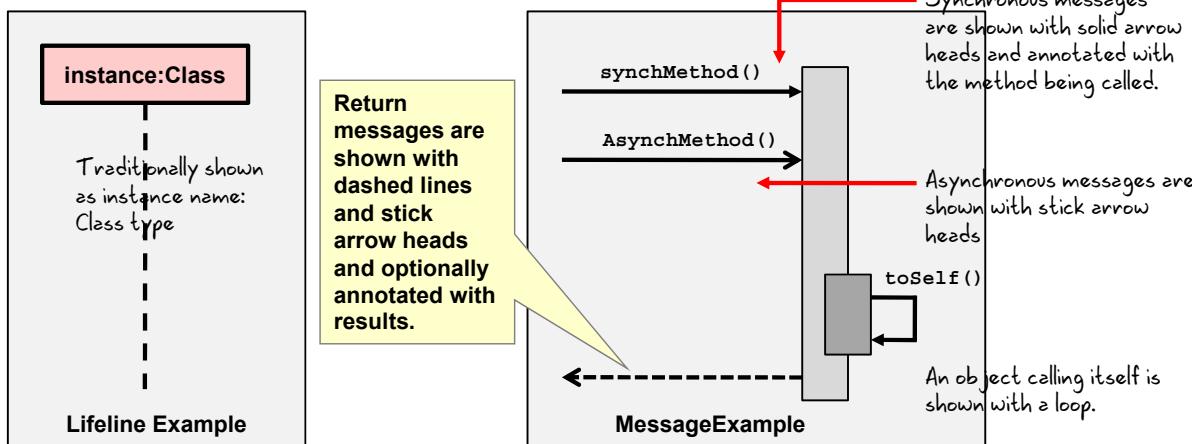
Sequence Diagrams

Sequence diagrams are designed to define the order of events in some sort of sequential activity. The focus is less on the messages and more on the order of events. Typically, sequence diagrams communicate what methods are called, and the order in which they are called. Sequence diagrams then present this information along the horizontal and vertical dimensions: the vertical dimension shows, from top to bottom, the time sequence of calls as they occur, and the horizontal dimension shows, from left to right, the object instances that the messages are sent to.

Anatomy of a Sequence Diagram

Sequence diagrams are composed of:

- Lifelines: Placed across the top of the diagram, representing either roles or objects being modeled
- Messages: Denoted by various arrow types and shown from top to bottom for each synchronous or asynchronous call



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Anatomy of a Sequence Diagram

Sequence diagrams are composed of two core elements:

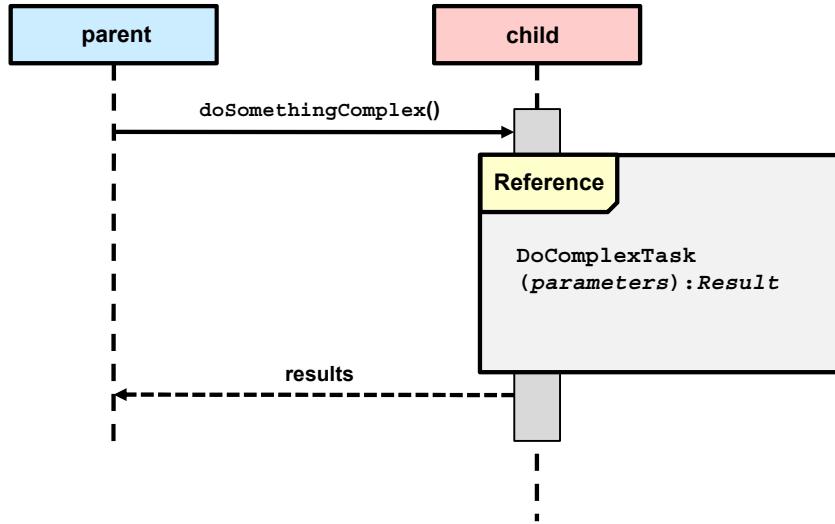
- **Lifelines:** Representing classes and often shown as `instance : Type`, which represents the classes being modeled
- **Messages:** Representing the calls, methods, or messages being sent, each with the name of the method being called. Arrows from left to right indicate calling into an object. Arrows from right to left represent returns from methods. A solid arrow head on a call from left to right represents a synchronous call; a stick arrow head represents an asynchronous call. Return messages are represented by a dashed line from right to left, with a stick arrow head and may be optionally annotated with a result. Return methods are not always modeled depending on the level of abstraction or detail required. In general, messages represent the methods that must be implemented on the class being modeled. Messages from an object to itself, often omitted, are shown as a loop.

Additionally, guards can be provided on method calls, usually in the form of `[boolean expression]` and represent preconditions applied before a method can be invoked. Other forms of sequence diagrams also exist, such as alternative and optional sequences, representing replacement or continuation of a sequence (assuming that some condition was met).

Other sequences, such as loops, are denoted with box around a sub-sequence with a boolean condition showing the requirement to continue looping.

Nested Sequence Diagrams

Sequence diagrams can be nested.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Nested Sequence Diagrams

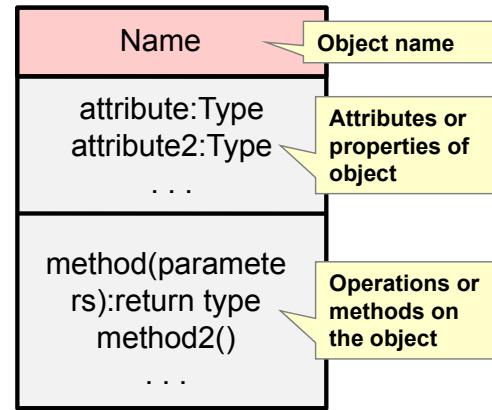
Sequence diagrams can reference other sequence diagrams via nesting. In the example shown, the child class uses another sequence diagram named DoComplexTask, which takes some set of parameters and returns some result.

Structure Diagrams

Structure diagrams are used to model types of objects including classes, interfaces, data types, and components.

Structure diagrams are composed of:

- Class Name
- Attribute List
- Methods
- Inheritance or association relationships



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Structure Diagrams

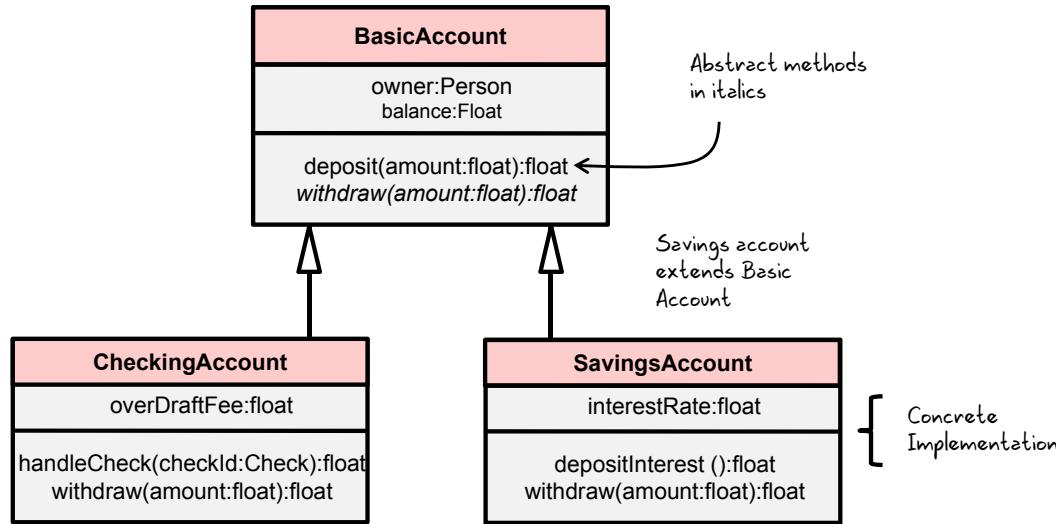
Structure diagrams, sometimes referred to as class diagrams, show the static structure of a system being modeled, and focus on the elements of a system that are unchanged over time. Structure diagrams show the types and instances, as well as relationships, and are typically used by different members of a team or cooperating teams to understand, design, or document a system.

Structure diagrams are composed of:

- **Name:** This is the name of an element or class. The name does not represent an instance, but rather a class if you are modeling a Java object.
- **Attributes:** This includes a set of both internal and external attributes. Depending on how the model is being used, the internal attribute may not be modeled.
- **Methods:** For those objects that have behavior, methods may be described along with their parameters and any returned values.
- **Relationships:** Inheritance relationships are shown in class diagrams via arrows with derived classes typically shown below their ancestor classes.

Structure Diagrams and Inheritance

Structure diagrams can be used to show both inheritance and associations.



ORACLE

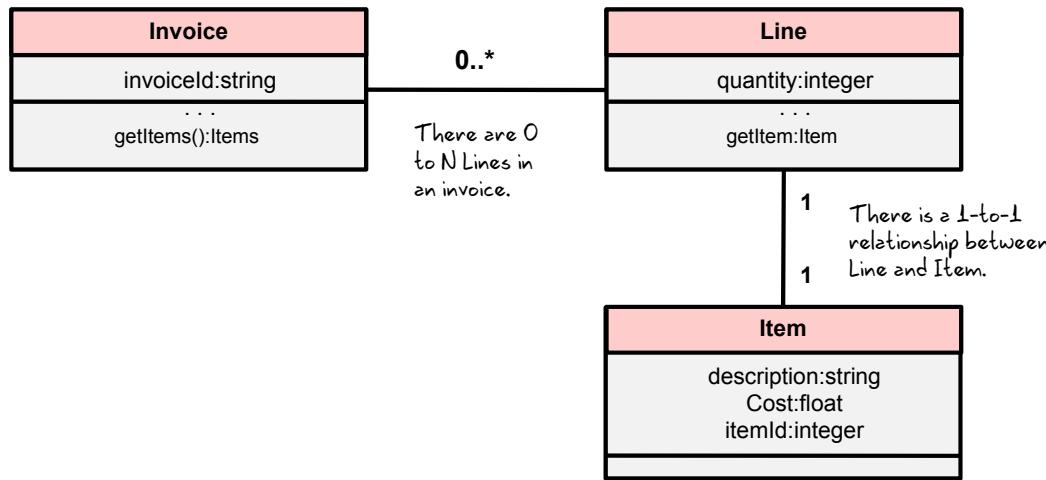
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Structure Diagrams and Inheritance

Structure diagrams show inheritance via an arrow with an open head pointing to the parent or ancestor class. Both data and behavior can be inherited. In the example shown, **SavingsAccount** is a parent class, which is abstract, based on its withdrawal method, which has no implementation. A **SavingsAccount** is then a **BasicAccount** and has all its properties. In addition, **SavingsAccount** has an interest rate property as well as a `depositInterest` method and a `withdraw` implementation. Likewise, **CheckingAccount** is a **BasicAccount** but has its specific properties and behaviors.

Structure Diagrams and Association

Structure diagrams can also show associations, ownership, and cardinality with other structure diagrams.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

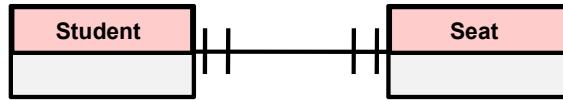
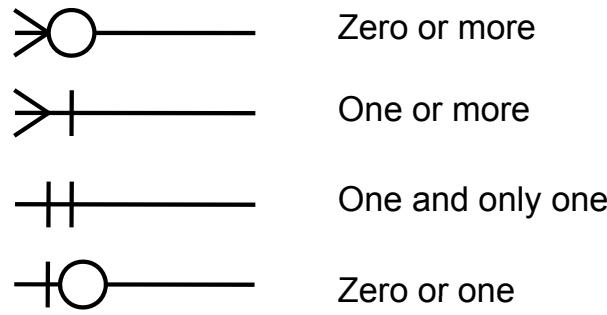
Structure Diagrams and Association

Structure diagrams can show association as well as inheritance. When two structure diagrams are related, there are a number of concerns, specifically:

- **Cardinality:** How are the two elements related? A line connecting two elements represents A's relationship with B. In the example shown, Invoices contain 0 or more Lines. Lines contain 1 and only 1 Item.
- **Direction:** Is the relationship between elements one direction or bidirectional? Elements may be related to each one way or unidirectional. Items exist as part of Lines and Lines are associated with Items, but Lines have no existence without their containing Invoice.
- **Aggregation:** Is there an owner-owned relationship?

Crow's Foot Notation

Crow's foot notation describes the relationship between two elements and is used interchangeably with numeric notation.



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

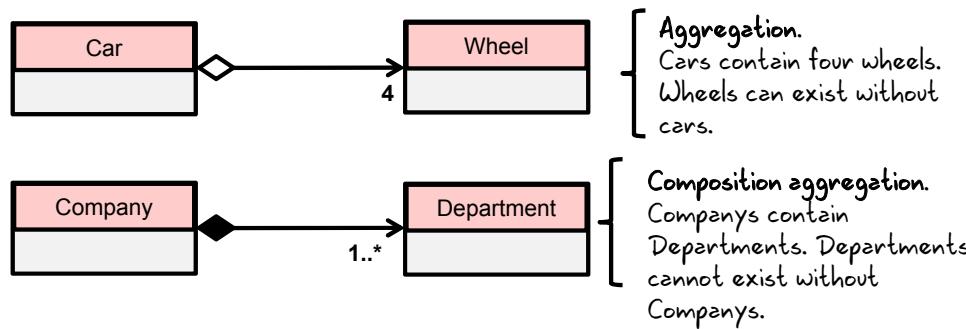
Crow's Foot Notation

It's not uncommon to see the use of Crow's foot notation in structure diagrams. This slide shows the relationship annotations used to show the various relationships. If the relationship is unidirectional, the notation can be used on both ends of the line to describe the cardinality from each element's perspective.

Aggregation and Composition

Aggregation is used to describe ownership, where one element is owned by another, but can exist on its own (comprises).

Composition is used to describe relationships, where one object is owned by another but cannot live on its own (has).



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

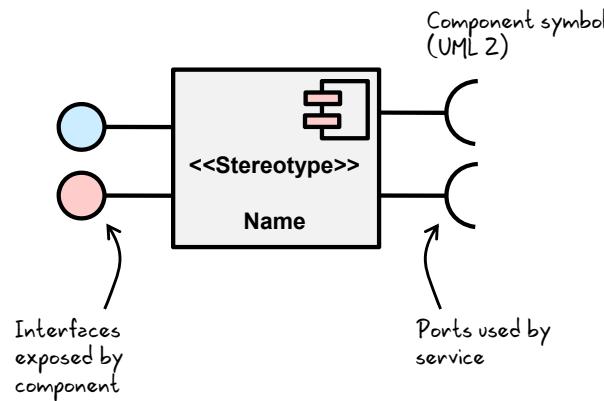
Aggregation and Composition

Aggregation is used when the relationship is meant to represent containment rather than a simple association. A car is an excellent example of containment in that its component parts represent the car as a whole. With Aggregation, the relationship is shown using a diamond. An empty diamond represents a relationship where the parts can exist without the whole. A filled diamond represents a relationship where the parts *cannot* exist without the whole.

Component Diagrams

Component diagrams:

- Represent the components or physical elements of a system
- Contain a stereotype, interface, port, and relationship information



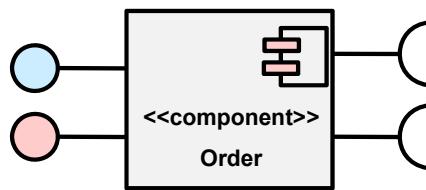
ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Component Diagram Anatomy

Component diagrams include:

- Name: A name for the component
- The component symbol (UML 2) 
- Stereotype: A term inside <> representing the thing the component extends
- Interfaces: Represented by a circle, denoting the services exposed by the component
- Ports: Represented by an arc, denoting the services consumed by the component



ORACLE

Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Component Diagram Anatomy

Component diagrams are described by five main elements:

- **Stereotype:** The type from which this component derives
- **Name:** The name of the element. If this component is itself a stereotype, the name may appear as another component's stereotype.
- **Interfaces:** Denoted by an open or empty circle, representing the interfaces or “ports” defined or exposed by this component. The interfaces are typically named by the interface they implement or expose.
- **Ports:** Denoted by a an arc, representing the interfaces or ports required by this component
- **The component symbol**

Summary

In this lesson, you should have learned how to read:

- Sequence diagrams
- Structure diagrams
- Component diagrams



Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

JB Coherence 3.7 New Features Overview

ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the themes of Coherence 3.7
- Enumerate the new features of Coherence 3.7



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Themes

Coherence 3.7 themes include:

Ease of Use

**Reliability
Availability
Scalability
Performance**

Integration

Innovation



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

New Features

Theme	Features
Ease Of Use	<ul style="list-style-type: none">Automatic proxy discovery for clients (WKA for *Extend)Client connection information now available via JMXAutomatic attempt to reconnect by clientsDynamic load balancing of client connections (*Extend)XML Schemas (XSD) for configurationPOF attributes
Integration	<ul style="list-style-type: none">Load balancer integration (F5)Native Coherence*Web Glassfish integrationREST support
RASP	<ul style="list-style-type: none">Query monitoring (via JMX)Partition-level transactions
Innovation	<ul style="list-style-type: none">Elastic Data (storage with Flash support)



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The new features of Coherence fall into four main areas: ease of use; integration; reliability, availability, scalability, and performance; and innovation. Each of these major improvements is supported by many more minor improvements.

Ease of Use New Features

Ease of Use



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Automatic Proxy Discovery

Feature:

- Coherence*Extend clients now only need to be configured with a small list of proxy node addresses.
- A client connects to a proxy, and is then redirected to the least utilized proxy server.

Benefits:

- Reduces client configuration
- Enables proxy servers to be added to the environment without changing client configuration

Requirement:

- Minimized or minimal configuration

Ease of Use

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Clients currently discover proxy servers via a static list of proxy listen addresses and ports. When additional proxy servers are added to the cluster, client configuration must be updated for the new capacity to be utilized. This could be improved only by using a WKA-like approach. The client configuration would need the address of only one proxy server. This proxy server could then redirect connect requests to another proxy, perhaps based on connection count or server utilization. This will result in better distribution of clients across proxy servers.

Dynamic Load Balancing for Coherence*Extend

Feature:

Using connection count, daemon pool size, and message backlog

- Default (proxy) implementation balances client connections based on a variety of metrics.
- Custom pluggable *Extend connection load balancing is supported.

Benefits:

- New connections are directed at the least utilized proxy.
- Develop and provide your own connection load balancer.

Requirement:

Using application specific metrics

- Minimal configuration
- Custom development for pluggable load balancing

Ease of Use

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Load balancing has been provided in Coherence 3.7 in two ways. First, the default proxy-based mechanism has been improved for better load balancing.

In general, the proxy-based load balancing algorithm distributes client connections equally across proxy service members. The algorithm redirects clients to the proxy services that are being utilized the least. The following factors are used to determine a proxy's utilization:

- **Connection Utilization** – This utilization is calculated by adding the current and pending connection counts. If a proxy has a configured connection limit and the current plus pending connection counts equal the connection limit, the utilization is considered to be infinite.
- **Daemon Pool Utilization** – This utilization equals the current number of active daemon threads. If all daemon threads are currently active, the utilization is considered to be infinite.
- **Message Backlog Utilization** – This utilization is calculated by adding the current incoming and current outgoing message backlogs.

Coherence 3.7 also saw the addition of the com.tangosol.coherence.net.proxy package, which includes the APIs that support development of custom plug-ins to balance client load across proxy service members based on specific client algorithms.

XML Schemas for Validating Configuration

Feature:

- Coherence now provides XSD files to support configuration validation. DTDs are now deprecated.
- XSD validation is optional, but strongly recommended.
- In the future, all configurations will be validated.

Benefits:

- All the typical development environment benefits
- Runtime validation of configuration (when declared) and fail-fast
- All XSDs available online at
<http://xmlns.oracle.com/>

Auto-complete,
documentation hovers,...

Ease of Use

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Prior versions of Coherence provide Document Type Definition (DTD) files that define the structure of the different configuration files used by the product. DTDs are limited in definition capabilities—a feature that limits the level of validation (for example, type checking) that can be performed on the configuration files. Providing XSD files will improve the end-user experience for Coherence because end users will be able to validate their configuration files more accurately by using XML editors.

XML Schemas for Validating Configuration

Include schema in XML-aware editors.

```
<cache-config
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
    xsi:schemaLocation=
        "http://xmlns.oracle.com/coherence/coherence-cache-config
         coherence-cache-config.xsd">
.
.
</cache-config>
```

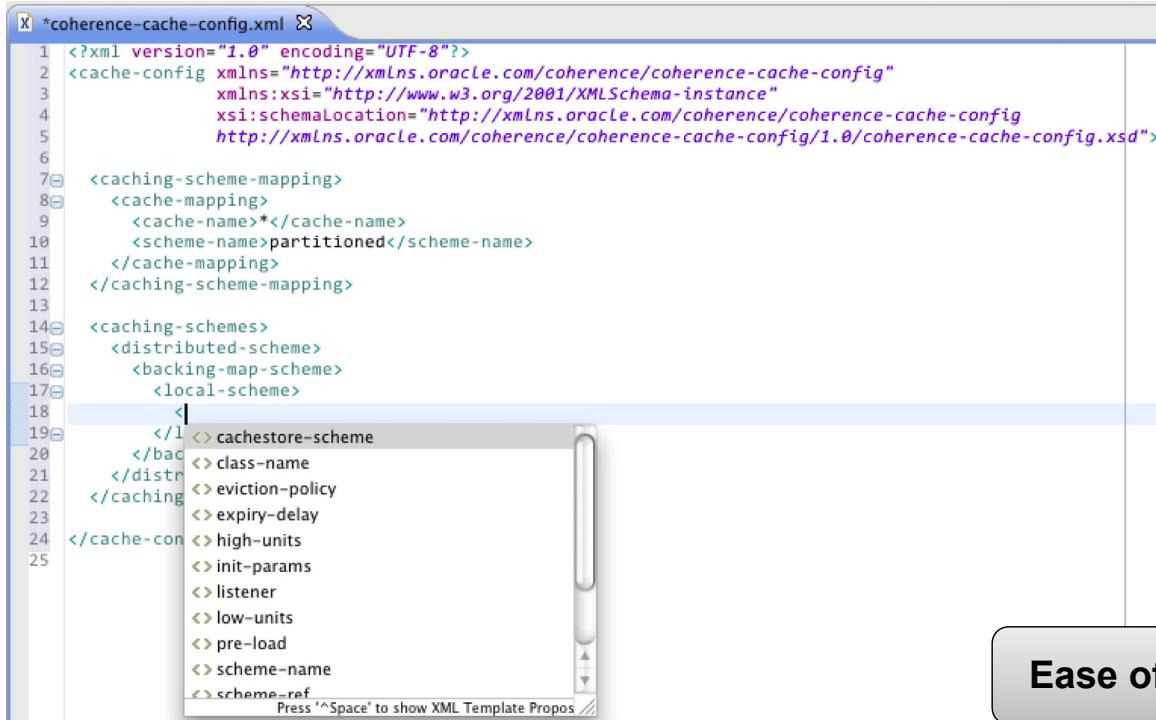
Editor can now validate for you.

Ease of Use

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

XML Schemas for Validating Configuration



The screenshot shows an XML editor window with the file name "coherence-cache-config.xml". The code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<cache-config xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
http://xmlns.oracle.com/coherence-cache-config/1.0/coherence-cache-config.xsd">
    <caching-scheme-mapping>
        <cache-mapping>
            <cache-name>*</cache-name>
            <scheme-name>partitioned</scheme-name>
        </cache-mapping>
    </caching-scheme-mapping>
    <caching-schemes>
        <distributed-scheme>
            <backing-map-scheme>
                <local-scheme>
                    <!-->
                    <!--> cachestore-scheme
                    <!--> class-name
                    <!--> eviction-policy
                    <!--> expiry-delay
                    <!--> high-units
                    <!--> init-params
                    <!--> listener
                    <!--> low-units
                    <!--> pre-load
                    <!--> scheme-name
                    <!--> scheme-ref
                </local-scheme>
            </backing-map-scheme>
        </distributed-scheme>
    </caching-schemes>
</cache-config>
```

A tooltip box labeled "Ease of Use" is overlaid on the right side of the editor.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Portable Object Format (POF) Improvements

Feature:

- Annotation support to automate POF serialization

Benefit:

- Simplified single-source POF definition

Requirement:

- Class must be registered in pof-config.xml.

```
@Portable
public class Person {
    @PortableProperty(0)
    public String getFirstName() { return m(firstName); }
    private String m(firstName);

    @PortableProperty(1)
    private String m(lastName);

    @PortableProperty(2)
    private int m(age);
}
```

Ease of Use

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Two new annotations have been added to support automated generation of POF serializers, which greatly simplifies POF development.

Two attributes are available to indicate that a class and its properties are POF serializable:

- @Portable – Marks the class as POF serializable. The annotation is permitted only at the class level and has no members.
- @PortableProperty – Marks a member variable or method accessor as a POF-serialized attribute. Annotated methods must conform to accessor notations (get, set, and is).

Integration New Features

Integration



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence*Web Support for GlassFish

Feature:

- Coherence*Web integration with GlassFish

Benefits:

- RASP for HTTP sessions
- No code change installation and/or configuration
- Flexible deployment models
- Many supported session models
- Various session-locking modes
- Session and session attribute sharing

Integration

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Starting with Coherence 3.7 and Oracle GlassFish Server 3.1, there is a new feature of Coherence*Web called ActiveCache for GlassFish. ActiveCache for GlassFish provides Coherence*Web functionality in web applications that are deployed on Oracle GlassFish Servers.

F5 Load Balancing Integration

Feature:

- Out-of-the-box support for Load Traffic Manager (LTM) integration

Use hardware to load balance application
*Extend client connections across a cluster.

Benefits:

- Reduction of Coherence overhead because of hardware-based load balancing
- SSL Acceleration

Offload CPU-intensive encryption and decryption of Coherence*Extend SSL.

Integration

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence now supports load balancing via F5.

The F5 BIG-IP LTM is a hardware device that sits between one or more computers running Coherence*Extend clients and one or more computers running Coherence*Extend proxy servers. The LTM distributes client connections across multiple clustered proxy servers by using a broad range of techniques to secure, optimize, and load balance application traffic.

Representation State Transfer (REST)

Feature:

- Annotate object model classes to support REST.

Benefits:

- Provides cache access via the HTTP protocol
- Exposes cache data to other languages, such as Python, Ruby, and so on
- Provides support for XML and JSON as input and output
- Provides support for fine-grain control over formats via the JAXB and Jackson attributes

Requirement:

- Cache and REST configuration

Integration

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST provides easy access to Coherence caches and cache entries over the HTTP protocol. It is somewhat similar to Coherence*Extend, because it enables remote clients to access data that is stored in Coherence without being members of the cluster themselves. However, unlike Coherence*Extend, which is a proprietary protocol that requires POF serialization to be used both on the client and within the cluster, Coherence REST uses HTTP as the underlying protocol, and can marshal data in multiple representation formats, such as JSON and XML. The benefit of Coherence REST is that it enables applications written in other languages, such as Ruby and Python (that are not natively supported by Coherence), to interact with cached data.

Representation State Transfer (REST)

Annotations on classes can be in JAXB or Jackson.

```
@XmlRootElement(name="Address")
@XmlAccessorType(XmlAccessType.PROPERTY)
public class Address { . . . } JAXB
```

```
@JsonTypeInfo(use=JsonTypeInfo.Id.CLASS,
               include= JsonTypeInfo.As.PROPERTY,
               property="@type")
public class Address { . . . } JSON
```

REST configuration defines how or what caches are exposed.

```
<?xml version="1.0"?>
<rest . . . >
  <resources>
    <resource>
      <cache-name>rest-example</cache-name>
      <key-class>java.lang.String</key-class>
      <value-class>example.Address</value-class>
    </resource>
  </resources>
</rest>
```

Maps an object to a key for a given cache

Integration

Note: Proxy definition required in cache configuration, and not shown

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST supports both XML and JSON formats as input and output. To use these formats, correct bindings are required when a user type is created. Objects that are represented in XML must have the appropriate JAXB bindings defined in order to be stored in a cache. Objects that are represented in JSON must have the appropriate Jackson bindings or JAXB bindings defined in order to be stored in a cache. The default Coherence REST JSON marshaller gives priority to Jackson bindings, and if it is not found, fails safe to JAXB bindings. Using Jackson annotations gives users more power over controlling the output JSON format. In cases where both XML and JSON formats are needed, JAXB annotations can be used which supports both formats.

RASP New Features

**Reliability
Availability
Scalability
Performance**



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Query Optimization: Explain Plans

Features:

- Is used to evaluate query cost and effectiveness
- Produces a query report 
Contains information about each filter as it was applied
- Uses the existing API

Benefits:

- Examine query performance before release.
- Determine exactly what indexes are required.
- Isolate existing query performance and test index benefits.

RASP

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence 3.7 introduces a new feature to Coherence queries which might be familiar to most database developers, the **explain plan** and **trace** command.

The EXPLAIN PLAN FOR and TRACE commands are used to create and output query records that are used to determine the cost and effectiveness of a query. A query explain record provides the estimated cost of evaluating a filter as part of a query operation. A query trace record provides the actual cost of evaluating a filter as part of a query operation. Both query records take into account whether or not an index can be used by a filter.

Query Optimization: Explain Plans

- Comes in two forms:
 - EXPLAIN PLAN – Provides an estimated cost of evaluating the query
 - TRACE – Runs the query and provides the actual cost of execution

Form

```
EXPLAIN PLAN FOR select statement | update statement | delete statement  
TRACE select statement | update statement | delete statement
```

For Example:

```
TRACE SELECT * from "MyCache" WHERE age=19
```

RASP**ORACLE**

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Query Optimization: Monitoring

Overview

- Supports new JMX metrics to monitor queries and indexes
- Helps identify poorly performing and un-indexed queries

New Metrics

- | | |
|---|--|
| <ul style="list-style-type: none">• OptimizedQueryCount• NonOptimizedQueryCount• OptimizedQueryMillis• NonOptimizedQueryMillis | <ul style="list-style-type: none">• OptimizedQueryAverageMillis• NonOptimizedQueryAverageMillis• MaxQueryDescription• MaxQueryThresholdMillis |
|---|--|

RASP

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence 3.7 introduces a number of new JMX monitoring variables that are focused on query optimization, such as the following:

- **OptimizedQueryCount** – The total number of queries that were fully resolved using indexes since the statistics were last reset
- **NonOptimizedQueryCount** – The total number of parallel queries that could not be resolved (or that were partially resolved) using indexes since the statistics were last reset
- **OptimizedQueryAverageMillis** – The average duration, in milliseconds, per optimized query execution since the statistics were last reset
- **NonOptimizedQueryAverageMillis** – The average duration, in milliseconds, for non-optimized query execution since the cache statistics were last reset
- **MaxQueryDescription** – A description of the query with the longest duration that exceeds the MaxQueryThresholdMillis attribute since the statistics were last reset

Query Optimization: Monitoring

Attribute values	
Name	Value
EventsDispatched	0
EvictionCount	0
IndexInfo	SimpleMapIndex: Extractor=.toString(), Ordered=true, Footprint=781KB,
InsertCount	10000
ListenerFilterCount	0
ListenerKeyCount	0
ListenerRegistrations	0
LocksGranted	0
LocksPending	0
MaxQueryDescription	Partially optimized query (5004 keys, scan of 4994, 444 matches) for AndFilter(GreaterEqualsF
MaxQueryDurationMillis	7
MaxQueryThresholdMillis	0
NonOptimizedQueryAverageMillis	7
NonOptimizedQueryCount	1
NonOptimizedQueryTotalMillis	7
OptimizedQueryAverageMillis	1
OptimizedQueryCount	8
OptimizedQueryTotalMillis	14
RefreshTime	Fri Dec 10 11:54:47 EST 2010
RemoveCount	0
TriggerInfo	java.lang.String[0]

RASP

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Partition-Level Transactions

Benefits:

- Development of entry processors that process multiple entries atomically across different caches (in the same partition)
- Efficient entry back-ups
 - All changes communicated in a single message “for free”
- Safe and efficient re-entrance support
 - Access and mutate other entries and caches from an entry processor, without:
 - Using heavy transactions
 - Worrying about thread safety
 - Identify and prevent Dead Locking

RASP

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Partition-Level Transactions

Features:

- Entry processors that use BinaryEntry can access:
 - Entries in other caches
 - Other Entries in the same partition

```
public void process(Entry entry) {  
    //update an entry in the same partition  
    ((BinaryEntry)entry).getBackingMap()  
        .put("otherkey", value);  
  
    //get an entry in the same partition  
    InvocableMap.Entry entry =  
        ((BinaryEntry)entry).getBackingMapContext()  
            .getBackingMapEntry("somekey");  
}
```

RASP

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Innovation New Features

Innovation



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

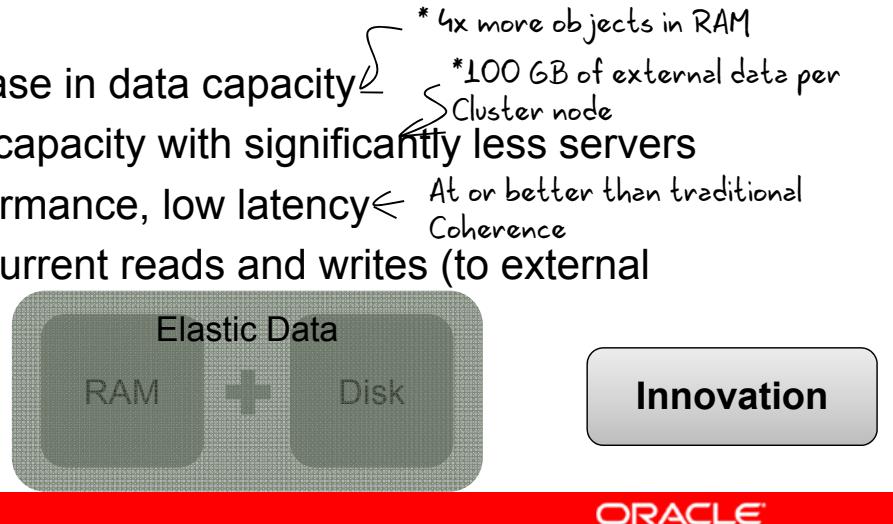
Elastic Data

Features:

- New storage implementation for server-side caches
- Seamless and efficient bridging between memory and disk
- Transparent storage (with massive capacity)

Benefits:

- Dramatic increase in data capacity
 - * 4x more objects in RAM
 - * 100 GB of external data per Cluster node
- Greater cache capacity with significantly less servers
- Very high performance, low latency
 - At or better than traditional Coherence
- Massively concurrent reads and writes (to external storage)



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The benefits of elastic data versus a simple distributed scheme are many:

- Because of the use and nature of journaling, up to 4x objects may be stored in memory as compared to prior versions of Coherence.
- With journaling and compression, datasets of 100 GB can be managed on single-cluster members.
- Self-tuning – Elastic data can self-tune and track object use, rebalancing the internal tree structures to maintain the most commonly used data in memory.
- Java garbage collection is minimized as a result of the tree-based journals used to manage data.
- It is very easy to use based on several sample schemes provided.

Elastic Data

Benefits:

- Dramatically simplified configuration
 - Specify on-heap memory size.
 - Specify storage location.
- Elimination of “out-of-memory” errors

Limitations:

- Not a persistence solution
- Indexes not stored using elastic data
- Cache eviction not supported
- Care needed when dealing with large data set aggregations and entry processors

Innovation

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Elastic data, introduced in Coherence 3.7, allows for the use of solid state devices (most typically flash drives) to provide spill over capacity for a cache. Using elastic data, a cache is specified to use a specific type of backing map based on a RAM or DISK journal. After it is defined, the cache can support massive amounts of data per node, often as much as 100 GB. Elastic data has been extensively tested and is proven to have almost no performance difference with totally RAM-based solutions.

Additionally, elastic data is simple to configure. In the base case, a caching scheme need only specify that the backing map be RAM journal based. However, a variety of configuration options exist for specifying how much memory to use for the backing map, what device to use, and others.

Elastic Data

Configuration:

```
<cache-config  
    xmlns:xsi=". . ."  
    xmlns=". . ."  
    xsi:schemaLocation=". . .">  
. . .  
  
<distributed-scheme>  
    <scheme-name>elastic-data</scheme-name>  
    <service-name>DistributedCacheElasticData</service-name>  
    <backing-map-scheme>  
        <ramjournal-scheme/>  
    </backing-map-scheme>  
</distributed-scheme>  
. . .  
</cache-config>
```



In its simplest form, just define a ram-journal scheme as the backing map scheme.

Innovation

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The simplest form of an elastic cache is based on the RAM journal, which can be added to the **backing-map-scheme** element to define an elastic data-based cache. When memory is exceeded, the RAM journal automatically delegates to a flash journal. Specifying a flash journal results in its immediate use rather than the usage being based on running out of memory as is the case with a RAM journal.

Both flash journals and RAM journals can be controlled using command-line arguments and override files. The most common override is the size of the journal itself, typically some percentage of the heap. However, a variety of other values can be controlled, such as how large a value to keep in memory, what directory to use for storage, and similar variables.

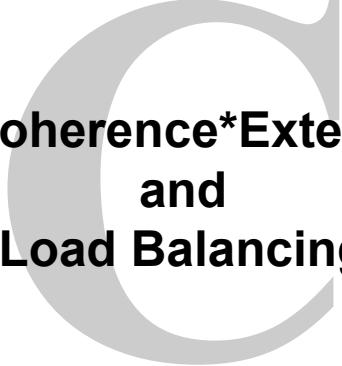
Summary

In this lesson, you should have learned how to:

- Describe the themes of Coherence 3.7
- Enumerate the new features of Coherence 3.7



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.



**Coherence*Extend
and
Load Balancing**

ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Enumerate load balancing concepts
- Configure server and client-side load balancing
- Develop custom load balancing



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Dynamic Load Balancing for Coherence*Extend

- Features:
 - Default (proxy) implementation balances client connections based on a variety of metrics.
 - It has support for custom pluggable *Extend Connection load balancing.
- Benefits:
 - New connections are directed at the least used proxy.
 - It supports the ability to develop and provide your own connection load balancer.
- Requirements:
 - Minimal configuration
 - Custom development for pluggable load balancing

Using connection count, daemon pool size, and message backlog

Client and proxy load balancing are supported.

Using application specific metrics

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.



Load balancing has been provided in Coherence 3.7 in two ways. First, the default proxy-based mechanism has been improved to enable better load balance.

In general, the proxy-based load balancing algorithm distributes client connections equally across proxy service members. The algorithm redirects clients to proxy services that are being utilized the least. The following factors are used to determine a proxy's utilization:

- **Connection Utilization** – This utilization is calculated by adding the current and pending connection counts. If a proxy has a configured connection limit and the current plus pending connection counts equal the connection limit, the utilization is considered to be infinite.
- **Daemon Pool Utilization** – This utilization equals the current number of active daemon threads. If all daemon threads are currently active, the utilization is considered to be infinite.
- **Message Backlog Utilization** – This utilization is calculated by adding the current incoming and current outgoing message backlogs.

Coherence 3.7 also saw the addition of the com.tangosol.coherence.net.proxy package, which includes the APIs that enable the development of custom plug-ins to balance client load across proxy service members based on specific client algorithms.

Connection Load Balancing

Coherence*Extend supports:

- Client-based load balancing
 - Uses a client address provider implementation
 - Accesses each proxy randomly until a connection is made
- Proxy-based load balancing
 - Is the default strategy
 - Is used to balance client connections
 - Is weighted by using the following factors in each proxy:
 - Existing connection count
 - Daemon pool utilization
 - Message backlog
- Custom load balancing



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Proxy-Based Load Balancing

Proxy-based load balancing:

- Is the default if no method is specified
- Is based on the load-balancer element that is set to proxy

```
<proxy-scheme>
    <service-name>ExtendTcpProxyService</service-name>
    <acceptor-config>
        <tcp-acceptor>
            <local-address>
                <address>192.168.1.5</address>
                <port>9099</port>
            </local-address>
        </tcp-acceptor>
    </acceptor-config>
    <load-balancer>proxy</load-balancer>
    <autostart>true</autostart>
</proxy-scheme>
. . .
```

coherence-cache-config.xml



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Proxy-based load balancing is the default strategy that is used to balance client connections between two or more proxy services. The strategy is weighted by a proxy's existing connection count, its daemon pool utilization, and its message backlog.

The proxy-based load balancing strategy is configured within a `<proxy-scheme>` definition using a `<load-balancer>` element that is set to proxy.

Client-Based Load Balancing: Per Proxy

Client-based load balancing can be specified:

- Per proxy
- As the system default

```
...
<proxy-scheme>
    ...
        <load-balancer>client</load-balancer>
    ...
</proxy-scheme>
...

```

coherence-cache-config.xml



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The client-based load balancing strategy relies on a client address provider implementation to dictate the distribution of clients across proxy service members. If no client address provider implementation is provided, the extend client tries each configured proxy service in a random order until a connection is successful. The client-based load balancing strategy is configured within a `<proxy-scheme>` definition using a `<load-balancer>` element that is set to `client`.

Client-Based Load Balancing: Systemwide

Client-based load balancing can be specified as the system default.

```
<?xml version='1.0'?>
<coherence xmlns:xsi=". . .">
    <cluster-config>
        <services>
            <service id="7">
                <init-params>
                    <init-param id="12">
                        <param-name>load-balancer</param-name>
                        <param-value>client</param-value>
                    </init-param>
                </init-params>
            </service>
        </cluster-config>
    </coherence>
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

To set the client strategy as the default strategy for all proxy services if no strategy is specified, override the `load-balancer` parameter for the proxy service type in the operational override file. Note that 7 is the ID of the proxy service and 12 is the ID of the `load-balancer` parameter.

Proxy-Based Load Balancing Algorithm

The default proxy load balancing algorithm attempts to distribute client connections evenly based on:

- Connection Utilization – Is calculated by adding the current and pending connection counts
- Daemon Pool Utilization – Equals the current number of active daemon threads
- Message Backlog Utilization – Is calculated by adding the current incoming and current outgoing message backlogs

Each proxy service maintains a list of all proxy services ordered by their utilization. The ordering is weighted first by connection utilization, then by daemon pool utilization, and then by message backlog.



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Each proxy service maintains a list of all proxy services ordered by their utilization. The ordering is weighted first by connection utilization, then by daemon pool utilization, and then by message backlog. The list is re-sorted whenever a proxy service's utilization changes. The proxy services send each other their current utilization whenever their connection count changes or every 10 seconds (whichever comes first).

When a new connection attempt is made on a proxy, the proxy iterates the list as follows:

- If the current proxy has the lowest connection utilization, the connection is accepted; otherwise, the proxy redirects the new connection by replying to the connection attempt with an ordered list of proxy servers that have a lower connection utilization. The client then attempts to connect to a proxy service in the order of the returned list.
- If the connection utilizations of the proxies are equal, the daemon pool utilization of the proxies takes precedence. If the current proxy has the lowest daemon pool utilization, the connection is accepted; otherwise, the proxy redirects the new connection by replying to the connection attempt with an ordered list of proxy servers that have a lower daemon pool utilization. The client then attempts to connect to a proxy service in the order of the returned list.

- If the daemon pool utilizations of the proxies are equal, the message backlog of the proxies takes precedence. If the current proxy has the lowest message backlog utilization, the connection is accepted; otherwise, the proxy redirects the new connection by replying to the connection attempt with an ordered list of proxy servers that have a lower message backlog utilization. The client then attempts to connect to a proxy service in the order of the returned list.
- If all proxies have the same utilization, the client remains connected to the current proxy.

Custom Load Balancing

Custom load balancing:

- Is specified via the `instance` element
- Requires a class that implements
`com.tangosol.coherence.net.proxy.ProxyServiceLoaderBalancer`

```
<proxy-scheme>
    . .
    <load-balancer>
        <instance>
            <class-name>
                com. . . MyCustomLoadBalancer
            </class-name>
        </instance>
    </load-balancer>
    . .
</proxy-scheme>
. . .
```

coherence-cache-config.xml

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The client-based load balancing strategy relies on a client address provider implementation to dictate the distribution of clients across proxy service members. If no client address provider implementation is provided, the extend client tries each configured proxy service in a random order until a connection is successful. The client-based load balancing strategy is configured within a `<proxy-scheme>` definition using a `<load-balancer>` element that is set to `client`.

ProxyServiceLoadBalancer Interface

```
package com.tangosol.net.proxy;

import com.tangosol.net.Member;
import com.tangosol.net.ProxyService;
import java.util.List;
public interface ProxyServiceLoadBalancer {

    public abstract void init(ProxyService proxyService);
    Is called once on initialization

    public abstract void update(Member member,
                               ProxyServiceLoad proxyServiceLoad);
    Updates the load balancing strategy in response to a change in a ProxyService utilization

    public abstract List getMemberList(Member member);
    Returns an ordered list of Members to which the new client should be redirected. It is called
    when a new client connects to the proxy service.

}
```

ProxyServiceLoadBalancer.java

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

AbstractProxyServerLoadBalancer Class

```
package com.tangosol.net.proxy;

import com.tangosol.net.*;
import com.tangosol.util.Base;

public abstract class AbstractProxyServiceLoadBalancer
    extends Base
    implements ProxyServiceLoadBalancer {

    protected Member getLocalMember() { . . . }

    Returns the Member object that represents the local cluster member

    protected boolean isLocalMember(Member member) { . . . }

    Returns true if the specified Member object represents the local member of the
    cluster

    public ProxyService getService() { . . . }

    Returns the enclosing Proxy Service

    protected ProxyService m_service;
}
```

AbstractProxyServerLoadBalancer.java

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

ProxyServiceLoad Interface

ProxyServiceLoad:

- Encapsulates information about the current usage of a ProxyService
- Is used to implement load balancing algorithms

```
package com.tangosol.net.proxy;

public interface ProxyServiceLoad extends Comparable {

    public abstract int getConnectionString();
    public abstract int getConnectionStringPendingCount();
    public abstract int getConnectionStringLimit();
    public abstract int getDaemonCount();
    public abstract int getDaemonActiveCount();
    public abstract int getMessageBacklogIncoming();
    public abstract int getMessageBacklogOutgoing();
}
```

ProxyServiceLoad.java



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

DefaultProxyServiceLoadBalancer

```
package com.tangosol.net.proxy;p  
import com.tangosol.net.Member;  
import java.util.*;  
  
public class DefaultProxyServiceLoadBalancer  
    extends AbstractProxyServiceLoadBalancer {  
  
    protected ProxyServiceLoad m_loadLocal;  
    protected final Map m_mapLoad;  
    protected final SortedMap m_mapMember;  
  
    public DefaultProxyServiceLoadBalancer() {  
        this(null);  
    }  
  
    public DefaultProxyServiceLoadBalancer(Comparator comparator) {  
        m_mapLoad = new HashMap();  
        m_mapMember = new TreeMap(comparator);  
    }  
    . . .
```

DefaultProxyServiceLoadBalancer.java

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

DefaultProxyServiceLoadBalancer

```
    . . .
    public void update(Member member, ProxyServiceLoad load) {
        if(isLocalMember(member)) {
            if(load == null) {
                m_mapLoad.clear();
                m_mapMember.clear();
            }
            m_loadLocal = load;
        } else {
            Map mapLoad = m_mapLoad;
            Map mapMember = m_mapMember;
            ProxyServiceLoad loadOld =
                (ProxyServiceLoad)mapLoad.remove(member);
            if(loadOld != null)
                mapMember.remove(loadOld);
            if(load != null) {
                mapLoad.put(member, load);
                mapMember.put(load, member);
            }
        }
    }
    . . .
}
```

DefaultProxyServiceLoadBalancer.java

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

DefaultProxyServiceLoadBalancer

```
    public List getMemberList(Member client) {
        ProxyServiceLoad loadLocal = m_loadLocal;
        if(loadLocal == null)
            return null;
        Map mapMember = m_mapMember;
        if(mapMember.isEmpty())
            return null;
        List list = new ArrayList(mapMember.size());
        Iterator it = mapMember.entrySet().iterator();
        do {
            if(!it.hasNext())
                break;
            Entry entry = (Entry)it.next();
            ProxyServiceLoad loadThat =
                (ProxyServiceLoad)entry.getKey();
            if(loadThat.compareTo(loadLocal) >= 0)
                break;
            list.add(entry.getValue());
        } while(true);
        if(!list.isEmpty())
            list.add(getLocalMember());
        return list;
    }
}
```

DefaultProxyServiceLoadBalancer.java

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Enumerate load balancing concepts
- Configure server and client-side load balancing
- Develop custom load balancing



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and ORACLE CORPORATION use only



Representational State Transfer

REST

ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe Representational State Transfer
- Modify Java objects to support REST
- Configure Coherence to return cache objects via REST
- Start Coherence REST Servers stand-alone or within a Servlet container such as that provided by WebLogic Server
- Query REST data using GET, PUT, and DELETE operations



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

What Is REST?

Representational state transfer (REST):

- Is an architecture style of large-scale networked software
- Is based on clients sending requests and servers sending responses
- Is very loosely coupled and scales well
- Takes advantage of WWW technologies and protocols, including HTTP 1.0/1.1, JAXB, and JSON

Using cURL, do an HTTP PUT.

Data is being sent in JSON.

```
$ curl -i -X PUT \n      -H "Content-type:application/json" \n      -d '{"name":"chris","age":"30"}' \n      http://localhost:8080/somecache/1
```

Using a well-defined address and port

Add to somecache by using key
↓

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Representational state transfer or REST is a style of software architecture that is based on existing standards such as XML, HTTP, JSON, and others to support transfer of data and/or object state in well-defined ways. In general, REST is defined by a set of basic concepts, which, if adhered to by applications, make the system “RESTful.” Common REST concepts include:

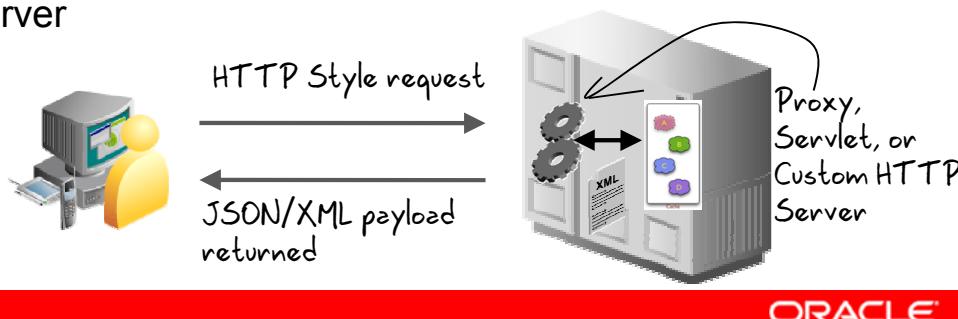
- **Data elements** – Resources, most likely data objects, and resource IDs (addresses and partial URLs) are accessible via standard interfaces such as HTTP.
- **Components** – Servers, clients, gateways, and proxies can communicate by transferring representations of resources using standard operations such as PUT and GET. In fact, REST is often considered more supportive of HTTP than most browser HTTP implementations because it uses operations such as DELETE, HEAD, POST, and others.
- **Stateless** – All requests are stateless, meaning that every request must be stand-alone and not dependent on any other request. With REST, a message must include all the information required to understand its context.

In the example shown, the cURL utility, which is available natively on *nix-based operating systems, can be used to make URL requests—in this case, to do a PUT operation on a cache with a key of 1, using the representation JSON.

Coherence REST

And others!

- Uses HTTP operations such as PUT and GET to send requests to and receive responses from caches
- Uses the `<rest/>` configuration to define what cache objects are available
- Uses one of the following to serve requests:
 - Proxy servers – Defined in cache configuration
 - Custom servers – Based on various HTTP server definitions
 - Servlet Container – A specific servlet hosted with WebLogic Server



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST is no different conceptually from a generic REST implementation as far as clients are concerned, but requires a number of elements on the server. These include:

- **Cache** – The cache itself, with objects modified to define how they will return REST resources
- **Server** – A server for service cache objects returned as REST. This can be any of a number of possible combinations: Coherence supports proxy server definitions, similar to Coherence*Extend proxie; Servlets running in the WebLogic Servlet container; Completely custom HTTP implementations or custom implementations based on the provided Coherence base classes.
- **Configuration** – Regardless of the proxy or server, Coherence needs to know what objects are being served as REST resources. These objects are defined in a `<rest/>` configuration file, which is specified at cache instance start time.

Coherence REST Requirements

Coherence REST requirements:

- Cached objects – Using annotations, describe how an object is represented
- Server – Responding to requests, either stand-alone or via WLS
- Configuration:
 - REST – Defining which objects types to expose
 - Operational – Defining a server to support the cache

```
@XmlRootElement(name="Address")
@XmlAccessorType(XmlAccessType.FIELD)
public class Address {
    ...
}
```

```
<?xml version="1.0"?>
<rest . . . >
    <resources>
        <resource>
            <cache-name>somecache</cache-name>
            <key-class>java.lang.String</key-class>
            <value-class>example.Person</value-class>
        </resource>
    </resources>
</rest>
```

```
<?xml version="1.0"?>
<cache-config . . . >
    <caching-scheme-mapping> . . .
    </caching-scheme-mapping>
    <caching-schemes>
        <distributed-scheme>
            <proxy-scheme> . . .
        </proxy-scheme>
    </caching-schemes>
</cache-config>
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence requires a number of elements in order to support REST.

- **Cached objects:** Any object that is intended to be returned via a RESTful interface must define its representation. Currently, two representations are supported: XML and JSON.
- **Servers:** Cached objects must be served somehow. Coherence provides two mechanisms for serving objects: using stand-alone servers or via WebLogic Server.
- **Cache configuration:** To support exposing a cache, a REST configuration file must be defined. The REST configuration defines how cache objects are exposed. Specifically, it defines a cache and its key/value pairs, which are available.
- **Operational configuration:** In a fashion similar to HTTP*Extend, Coherence must expose a proxy that acts on behalf of the cache to expose servers. This proxy is defined in an operational configuration file, and includes a definition of how to expose a given cache mapping scheme by using a proxy service that supports REST.

Supported Representations

Coherence supports two data representations when processing REST requests:

- eXtensible Markup Language, or XML

```
<person><age>30</age><name>chris</name></person>
```

- JavaScript Object Notation or JSON, a lightweight data interchange format based on name/value pairs

```
{ "name": "chris", "age": "30" }
```

The one that is appropriate depends
on the consuming application.



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence supports two data representations when processing REST requests: XML and JSON. For the most part, students are familiar with XML and its usage in Coherence. REST is not significantly different from any other usage. However, JSON is not as common. JavaScript Object Notation (JSON) is a simple lightweight data interchange format that is based on name/value pairs. JSON is fairly manually readable, relatively easy to generate and parse, and flexible in terms of what it can represent. JSON in its simplest form relates a name to a value. However, a value might be another object represented by nested braces, an array, a null value, a true/false value, a string, or a number. Each of these types is well defined. For more information about JavaScript Object notation, see <http://www.json.org>.

Annotations

Annotations:

- Are based on JSR 175, and were introduced in Java 5
- Relieve the developer from cumbersome configuration
- Avoid the need for boilerplate code by adding “annotations” or hints to the source code

```
public interface Animal {  
    String type;  
}  
Definition
```

```
@Animal(type="mammal")  
public class Bird { . . . }  
Usage
```

Coherence objects can be made REST-aware via annotations.

```
@XmlRootElement(name="Address")  
@XmlAccessorType(XmlAccessType.  
    PROPERTY)
```

JAXB

```
@JsonTypeInfo(use=JsonTypeInfo.Id.  
    CLASS,  
    include= JsonTypeInfo.As.PROPERTY,  
    property="@type")
```

JSON

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Annotations are one of the newer trends in Java development. Simply put, annotations are markers or hints to the compiler and the virtual machine that a class exhibits a certain behavior or characteristic. In essence, an annotation is a mechanism for associating metadata with program elements and allowing the compiler or the virtual machine to extend program behaviors from these annotated elements.

Coherence uses either Java Architecture for XML Binding (JAXB) or the Jackson open source Java JSON processor for annotating objects.

Class Requirements (JAXB)

To support Coherence REST JAXB, a class must:

- Have a public no-argument constructor
- Be serializable
- Be exposed as type `XmlRootElement(name="...")`
- Expose data using `XmlAccessorType` as one of:

Becomes the name of the element

<code>XmlAccessType</code>	Description
<code>PUBLIC_MEMBER</code>	Default access type. Expose all public fields, annotated fields, and properties.
<code>PROPERTY</code>	Expose annotated fields and properties.
<code>FIELD</code>	Expose fields and annotated properties.



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

To be used with REST via JAXB and returned as XML, a class must meet certain requirements, some of which overlap with Coherence.

1. The class must inherit or have a no-argument constructor.
2. The class must be annotated with `@XmlRootElement`, providing the `name` attribute, which defines the XML name of the class.
3. The class must expose fields in one of three ways, all using `@XmlAccessorType`.

The three possible accessor types are:

- `PUBLIC_MEMBER` – Expose any public field, specifically annotated fields and properties.
- `PROPERTY` – Expose public properties and annotated fields.
- `FIELD` – Expose public fields and annotated properties.

XmlAccessType.PUBLIC_MEMBER: Example

```
import java.io.Serializable;
import javax.xml.bind.annotation. . . ;
Annotations are in this package.

@XmlRootElement(name="address")
@XmlAccessorType(XmlAccessType.PUBLIC MEMBER)
public class Address implements Serializable {

    private String street;
    private String city;
    public String country;

    public String getCity() { return city; }
    public void setCity(String city) { this.city= city; }

    ...
}

<xml...>
<address >
    <country>country value</country>
    <city>city value</city>
</address>
```

All are exposed because they are marked public.



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

When accessor type is set as PUBLIC, any fields marked “public” will be exposed. Note that the specification of XmlAttribute on a field or property will result in that element being exposed as an attribute. XmlAttribute and XMLTransient will be discussed in greater detail later.

Use PROPERTY when you use a set of public property methods to expose content or when only public fields are being exposed.

XmlAccessType.PROPERTY: Example

```
import java.io.Serializable;
. . . ;
@XmlRootElement (name="Address")
@XmlAccessorType(XmlAccessType.PROPERTY)
public class Address implements Serializable {

    private String street;
    private String city;
    public String country; <-- Not exposed even though it is public

    public String getCity() { return city; }
    public void setCity(String city) { this.city= city; }
    . .
}
```

```
<xml...>
<address>
    <city>city value</city>
</address>
```

Only the public properties are exposed.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

When accessor type is set as PROPERTY, any public properties will be exposed.

Use PROPERTY when there are fields that are marked public, but should not be exposed.

XmlAccessType.FIELD: Example

```
import java.io.Serializable;
. . .
@XmlRootElement (name="Address")
@XmlAccessorType(XmlAccessType.FIELD)
public class Address implements Serializable {

    private String street;
    private String city,
    public String country;

    public String getCity() { return city; }
    public void setCity(String city) { this.city= city; }

    . . .

}

<xml...>
<address >
    <country>country value</country>
</address>
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

When accessor type is set as FIELD, any public fields will be exposed.

Use FIELD when there are public property methods that should not be exposed.

XmlAttribute and xmlTransient

- `XmlAttribute` and `XMLTransient` can be used to finely control how data is exposed.
- `XmlAttribute` can be associated with a field or property to generate attributes rather than child elements.
- `XMLTransient` can be used to force a field or property to be ignored.
- Both are applicable to fields or properties.

```
...  
@XmlRootElement (name="Address")  
@XmlAccessorType(XmlAccessType.FIELD)  
public class Address  
    implements Serializable {  
    ...  
    @XmlAttribute  
    private String street;  
    @XMLTransient  
    public String postalRoute ;  
    ...  
}
```

```
<xml...>  
<address street="value">  
    ...  
</address>
```

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

`XmlAttribute` and `XMLTransient` can be used when fine control is required over what fields are exposed.

`XMLTransient` has a clear use. It can be used when a field should be ignored even if it fits the accessor type—for example, generated fields.

`XmlAttribute` has a less clear use, but can be used to force fields to be generated as attributes again even if they violate the accessor type.

Class Requirements (JSON)

To support Coherence REST JSON, a class must:

- Have a public no-argument constructor
- Must be `Serializable`
- Be exposed using `@JsonTypeInfo` that must define the following attributes:
 - Serialization metadata `use=JsonTypeInfo.CLASS`
 - Type metadata `include=JsonTypeInfo.As.PROPERTY` with `property="@type"`

```
import java.io.Serializable;
import org.codehaus.jackson.annotate. . . ;

@JsonTypeInfo(use=JsonTypeInfo.Id.CLASS,
              include= JsonTypeInfo.As.PROPERTY,
              property="@type")
public class Address implements Serializable {
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

JSON requirements are much like JAXB in that the class must have a no-argument constructor. However, beyond that the annotations are completely different. `JsonTypeInfo` requires the `use` and `include` attributes, which define what metadata is used for marshalling and including other classes.

The `use` attribute defines how the JSON class exposes the original class, so that the JSON data can be marshaled back out of JSON. `use` can be one of the following:

- `Id.NONE` – Use no explicit type metadata; all typing is done using contextual data.
- `Id.CLASS` – Use a fully-qualified Java class as type identifier.
- `Id.MINIMAL_CLASS` – Use the Java class name with minimal path as the identifier.
- `Id.CUSTOM` – Use a custom scheme.
- `Id.NAME` – Use the logical type name as the type information. Name will need to be separately resolved to a concrete type.

The `include` attribute defines how type metadata, if any, is included and can be one of the following:

- `As.PROPERTY` – Uses a single configurable property, along with the data, as a separate property
- `As.WRAPPER_OBJECT` – Wraps the typed JSON as a single entry with an identifier and a value as JSON
- `As.WRAPPER_ARRAY` – Wraps the typed JSON as an array of typed ID and JSON value

The `property` property is used with `As.PROPERTY` and specifies the name of the property to use when marshalling and unmarshalling from JSON.

JSON Mapping: Example

```
package example;

import org.codehaus.jackson.annotate.*;

@JsonTypeInfo(use=JsonTypeInfo.Id.CLASS,
              include= JsonTypeInfo.As.PROPERTY,
              property="@type")

public class Address implements serializable {
    private String street;
    private String city;
    private String country;
}

{ "@type": "example.Address"
  "street": "10 van deGraf",
  "city": "burlington",
  "country": "USA" }
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In the example shown, the Java class, `example.Address`, is mapped to JSON. In the example, the `use` attribute specifies that the fully-qualified class name should be used to identify the type. The `include` attribute specifies that the class name should be included as a property, like any other class property, and that the name of the property should be `@type`. The choice of `@type` is completely arbitrary and could have been anything.

JSonProperty and JSonIgnore

- `JSonProperty` and `JSonIgnore` can be used to finely-control how data is exposed.
- `JSonProperty` can be associated with a field or property to expose the property and/or change its name.
- `JSonIgnore` can be used to force a field or property to be ignored.



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The `JSonProperty` and `JSonIgnore` annotations can be used to expose data elements more carefully. Most often, the `JSonProperty` annotation is used to specify the name of the property for marshalling. In the example shown, the `JSonProperty` attribute was applied to the `private String street` member variable of the class to change the marshalling property name to **Street** from **street**. The attribute could have been placed on the field, as shown, or on a method, or at the class level. Likewise, the `JSonIgnore` attribute was applied to the `isValidAddress` property to ignore and not marshal the field.

Key Converters

Key converters:

- Are used to convert back and forth between external and internal representations of keys
- Are required when the key class is not one of the `java.lang` primitives such as `java.lang.[String|Long|Integer|...]` and others
- Are implemented using the `KeyConverter` interface

```
package com.tangosol.coherence.rest;

public interface KeyConverter{

    // Convert a string representation of a key
    // into its object form.
    public abstract Object fromString(String sKey);

    public abstract String toString(Object oKey);
}
```

KeyConverter.java



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

When an object is stored in the cache using a complex key, for example, one that is not one of the `java.lang` primitives, it must be converted into and out of JSON or XML. To convert a key, a key converter class is required. A key converter needs to implement both the `fromString` and `toString` methods. The actual implementation of these methods is up to the developer. A simple converter might take advantage of the object itself, for example, using the `toString` method to marshal and unmarshal key elements.

KeyConverter: Example

```
import com.tangosol.coherence.rest.KeyConverter;  
  
public class PersonConverter implements KeyConverter { Implements...  
  
    public Object fromString(String sKey) {  
        Person p= new person();  
        StringTokenizer st = new StringTokenizer(s, "_");  
  
        return new PersonId(  
            st.hasMoreTokens() ? st.nextToken() : null,  
            st.hasMoreTokens() ? st.nextToken() : null);  
    }  
  
    public String toString(Object oKey) {  
  
        if (oKey == null) return "";  
        PersonId pid = (PersonId)oKey;  
        name = pid.getName();  
        age = pid.getAge();  
        return name.toString() + "_" + age.toString();  
    }  
}
```

Return the object version
from the marshaled string

Return the string version
from the object version



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The example shown is rather simple but illustrates the concept of a key converter well.

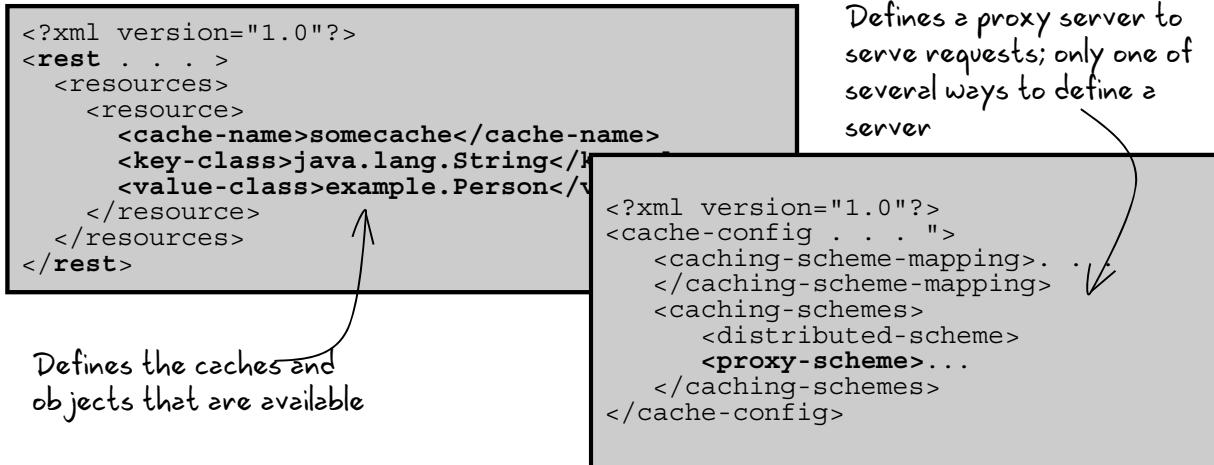
The class must implement the `KeyConverter` interface that requires the `fromString` and `toString` methods.

`fromString` is used to reconstitute a key from a string representation of a key. In the case of Persons, the `PersonId` field is a concatenation of the name and age fields. `FromString` is implemented by simply breaking the string into two parts using an underscore as a separator. `toString` is used to create a string representation from an object representation. In the example, the object is checked for null, and in theory should also be checked for the correct type before casting. It is then cast to the `PersonId` type, and the name and age are extracted. The fields are then returned as a string, with the fields separated by an underscore.

REST Configuration

Coherence REST requires the following:

- A rest configuration file that exposes the objects in a cache
- REST servers to act on HTTP requests



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST support requires two types of configurations: object level and server level.

- **Object Level:** Each object type that is accessible via REST must be defined in a REST configuration file. This file has a set of resource entries, each of which maps an object to a cache and a key type. Optionally, a key converter may be used when a cache object must be accessed using a complex key, which does not map to one of the `java.lang` primitive types (String, Long, Integer, and others).
- **Server Level:** REST requests must be handled in some fashion. Coherence REST supports a number of mechanisms to support serving REST requests, including:
 - Stand-alone servers, which are defined using cache configuration files and specifying a proxy scheme to handle requests
 - Custom classes, which are also defined using a cache configuration, which allow customization of the mechanisms used to handle requests
 - Registration of a provided HTTP Servlet within the WebLogic Server servlet container to serve requests

Registering Objects

- Object types are registered:
 - Using a rest configuration
 - In a resource element
- A rest configuration file has two sections:
 - Namespace definition
 - Resources definition

```

<?xml version="1.0"?>
<rest
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.oracle.com/coherence/coherence-rest-config"
  xsi:schemaLocation=
    "http://xmlns.oracle.com/coherence/coherence-rest-config
     coherence-rest-config.xsd">

  <resources>
    <resource> . . .
    <resource> . . .
    . . .
  </resources>
</rest>

```

rest-config.xml

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

ORACLE

REST configuration files are composed of two parts: the first being the namespace definition for the XSD that represents the configuration file and the second being a set of elements that identify the object types, which can be operated on via REST operations.

The REST namespace is defined by the `xmlns` and `xsi` attributes of the `rest` element itself. These attributes must always be included as:

```

<rest
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.oracle.com/coherence/coherence-rest-config"
  xsi:schemaLocation=
    "http://xmlns.oracle.com/coherence/coherence-rest-config
     coherence-rest-config.xsd">
```

The second section of the REST configuration is the nested set of resources/resource elements. Each resource element represents a triplet of the cache, the object, and its key, as well as an optional key converter.

REST resources

REST resources require:

- cache-name – The name of the cache containing the object instances
- key-class – The fully qualified path to a class representing the key
- value-class – The fully qualified path to a class representing the value

```
<?xml version="1.0"?>
< . . . >
<resources>
  <resource>
    <cache-name>some cache name</cache-name>
    <key-class>fully qualified key class</key-class>
    <value-class>fully qualified object class</value-class>
  </resource>
  . .
</rest>
```

rest-
config.xml

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

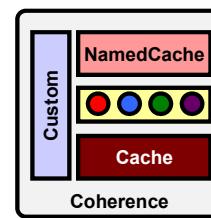
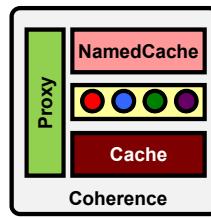
The REST configuration file resource elements define each of the actionable objects that can be returned by the REST service. Each REST configuration includes a single resources element, which itself includes one or more resource elements. The resource element then must contain the cache-name, a key-value element, and a value-class element. The values of these elements combine to represent an actionable item.

The key-class element bears some special consideration. This element represents the class object used for the key. For simple scalar key objects, classes from the java.lang package, such as java.lang.String or java.lang.Long, can be used. For objects that use a complex key, a key converter element must also be provided. Key converters, which are covered in later slides, take a complex key element and translate it into and out of String objects.

Serving Coherence REST Requests

Coherence REST requests can be served in several ways such as:

- Using an embedded HTTP Server that is defined within a proxy-scheme cache configuration file
- Using a custom HTTP Server by extending or implementing one of several provided classes, for example:
`com.tangosol.coherence.rest.server.DefaultHttpServer`
- Within the WebLogic Server Servlet container



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST requests must be served in some fashion. There are two main mechanisms for serving REST requests: either by using a proxy or by using a provided servlet within a servlet container such as the WebLogic Server servlet container. Additionally, rather than registering a proxy, an instance of a class implementing `com.tangosol.coherence.rest.server.HttpServer` can be used to serve REST requests. For the sake of convenience, the class `com.tangosol.coherence.rest.servcer.DefaultHttpServer` can be used as a starting point for a custom REST server, an example of which will be shown later in this lesson.

Configuring REST Proxies

REST proxies are the most common server mechanisms in stand-alone Coherence clusters.

To register a proxy in a Coherence cache configuration:

1. Add a proxy-scheme element
2. Within the proxy-scheme element, add an acceptor-config element that defines an http-acceptor to service requests.

```
 . . .
<proxy-scheme>
    <service-name>ExtendHttpProxyService</service-name>
    <acceptor-config>
        <http-acceptor>. . . </http-acceptor>
    </acceptor-config>
</proxy-scheme>
. . .
```

Must be →
Defines the
listen address

rest-cache-config.xml



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

REST proxies are defined within a cache configuration file by specifying a proxy scheme. The proxy scheme defines a single service, which is used to direct REST requests to an appropriate HTTP server. In order to configure a REST proxy, first define a proxy-scheme element specifying an ExtendHttpProxyService. Add an acceptor element that defines what class will accept the requests and an http-acceptor that defines the actual proxy. The next slide shows a complete example of the http-acceptor, but the most important element is the local-address element, which defines the actual address that the acceptor will listen on.

Configuration: Example

```
<?xml version="1.0"?>
<cache-config . . .>
    .
    .
    <caching-schemes>
        .
        .
        <proxy-scheme>
            <service-name>ExtendHttpProxyService</service-name>
            <acceptor-config>
                <http-acceptor>
                    <local-address>
                        <address>localhost</address>
                        <port>8080</port>
                    </local-address>
                </http-acceptor>
            </acceptor-config>
            <autostart>true</autostart>
        </proxy-scheme>
    </caching-schemes>
</cache-config>
```

Must be ExtendHttpProxyService

Enter a single local address representing the TCP address and port to bind to.

Mark the proxy auto-start.

rest-cache-config.xml

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The simplest of REST servers is the configuration of a `proxy-scheme` element. The `proxy-scheme` element defines a service, which supports serving REST requests. `proxy-scheme` includes an `acceptor-config` element, which wraps an `http_ancestor` element. The `http_ancestor` element itself includes a single `local-address` element that specifies the address and port that the acceptor should listen on.

There are a number of optional elements with the scheme and the acceptor itself. For example, a `thread-count` element, directly following the service name element, can be specified to describe how many threads should be created to service requests. In the absence of the `thread-count` element, all requests are served on the service's main thread. Note that a common practice is to specify a system property name for the address and port such that a script can override the values at proxy start time. Such elements might be structured as:

```
<address system-property="proxy.address">default address</address>
<port system-property="proxy.port">default port</port>
```

The properties could then be overridden at run time with `-D` statements. A similar mechanism might be used with the `auto start` element, to override whether a proxy is started by a server.

Starting a REST Proxy

REST proxy servers:

- Must define:
 - The cache configuration that includes a proxy-scheme
 - The REST configuration file specifying the REST-exposed objects, using the tangosol.coherence.rest.config property
- Often define:
 - Dtangosol.coherence.distributed.localstorage=false

```
CLASSPATH= . . . /coherence-rest.jar: . . .
java -D. . .
-Dtangosol.coherence.distributed.localstorage=false
-Dtangosol.coherence.cacheconfig=rest-cache-config.xml"
-Dtangosol.coherence.rest.config=rest-config.xml
com.tangosol.net.DefaultCacheServer
```

application.sh

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

REST proxies are instances of DefaultCacheServer and specify a cache configuration that includes properties specifying a cache configuration that itself includes a proxy-scheme and a rest_config element specifying the REST resources to expose. The rest properties are shown in the slide and include the following:

- tangosol.coherence.cacheconfig – A cache configuration file specifying the proxy-scheme element
- tangosol.coherence.rest.config – A configuration file specifying the REST resources to expose
- The coherence-rest.jar that specifies the other dependent jar files required by the Coherence REST system

REST Configuration Best Practices

- Only a single REST proxy should be started per address.
- Ensure single processes through use of system properties.

```
.
.
<proxy-scheme>
    <service-name>ExtendHttpProxyService</service-name> . . .
        <local-address>
            <address system-property="proxy.address">127.0.0.1</address>
            <port    system-property="proxy.port">8088</port>
        </local-address>
    .
    .
    <autostart system-property="proxy.enabled">false</autostart>
</proxy-scheme>
.
.
java -D. . .
    -Dproxy.address=localhost
    -Dproxy.port=8080
    -Dproxy.enabled=true
com.tangosol.net.DefaultCacheServer
```

Script to start proxy only

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

It is always best to have a single configuration file per Coherence cluster. However, if a single physical machine hosts multiple instances of DefaultCacheServer, it is necessary to take steps to ensure that only a single instance runs on each IP address. The simplest way to ensure a single instance is via system properties.

The first property involves auto-start that, if configured to false, will disable the proxy. On the instances that should start the proxy, the value can be overridden with a value of -Dproxy.enabled=true. As such, a single configuration file can be used on all instances, with minor differences in start scripts.

Similarly, the address and port can be overridden as required. In the example shown, the default address is the loop-back adapter, which is overridden with -Dproxy.address, which when combined with the UNIX hostname command, can be used to uniquely identify a host without changing any scripts. In a similar fashion, a default port can be assigned and overridden as required.

Custom REST Server

A custom REST server:

- Is typically used to implement specialized behaviors
- Often extends `AbstractHttpServer`
- Must implement the `start` and `stop` methods
- Defines an `HttpServer` instance to handle requests
- Is registered in a cache configuration using the `class-name` element within an `http-acceptor`

```
public class SimpleHttpServer extends AbstractHttpServer{  
    protected HttpServer server;  
    public void start() { . . . };  
    public void stop () { . . . };  
}
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST ships with two HTTP servers, along with the default proxy server.

- `com.tangosol.coherence.rest.server.DefaultHttpServer`, which is backed by the Oracle Lightweight HTTP server, is primarily for development and testing, and is not normally used in production environments
- `com.tangosol.cohernece.rest.Server.GrizzlyHttpServer`, which is backed by the Grizzly HTTP server

When developing a custom HTTP server, typically, the developer extends `AbstractHttpServer`, and implements the `start` and `stop` methods. The `start` method obtains an instance of a backing `HttpServer` to listen to and reply to requests. The `stop` method releases the server instance on shutdown.

Contract for AbstractHttpServer

Custom HTTP servers must:

1. Extend the `AbstractHttpServer` interface
2. Create and start an `HttpServer` instance

```
import com.sun.jersey.api.container.httpserver.HttpServerFactory;
import com.sun.net.httpserver.HttpServer;                                Required imports
import com.tangosol.util.WrapperException;
import com.tangosol.coherence.rest.server.AbstractHttpServer

import java.io.IOException;

public class ExampleHttpServer extends AbstractHttpServer {Implements...
    protected HttpServer m_server = null;

    public ExampleHttpServer() { . . . }
    public void start() { . . . }                                     Required start and stop
    public void stop () { . . . }                                     methods
}
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The simplest way to create a custom `HttpServer` is by extending `AbstractHttpServer` and obtaining an `HttpServer` instance from the Oracle HTTP server factory. To implement a REST server:

1. Include `HttpServerFactory` and `HttpServer`
2. Include `AbstractHttpServer` and extend the server instance from it
3. Define an `HttpServer` variable
4. Initialize the `HttpServer` variable instance in the `start` method, and then start the server
5. Stop the `HttpServer` in the `stop` method and release it

Note that the `com.sun.jersey.api.container.grizzly2.GrizzlyServerFactory` server factory can also be used to return a server factory. See the <http://grizzly.java.net> documentation for more details about the Grizzly HTTP server framework.

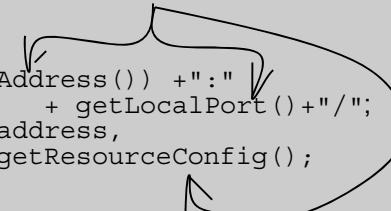
Start Method

The start method of AbstractHttpServer:

1. Obtains an instance of an `HttpServer` specifying the listen address and a resource configuration
2. Starts the instance

```
...  
public class ExampleHttpServer extends AbstractHttpServer {  
    protected HttpServer m_server = null;  
...  
    public void start() {  
        if (m_server) != null return;  
        String address = "http://"+ getLocalAddress() + ":"  
                        + getLocalPort() + "/";  
        m_server = HttpServerFactory.create(address,  
                                            getResourceConfig());  
        m_server.start();  
    }  
...}
```

Provided by `AbstractHttpServer`



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The start method of the example in the slide is used to create an instance of an `HttpServer`, and then start the instance. The `AbstractHttpServer` object provides a number of convenience methods, which simplify the creation of the instance.

`getLocalAddress()` returns the local address variable as bound into the proxy configuration. Likewise, `getLocalPort()` returns the port. The `create` method on `HttpServerFactory` can be used with only an address or with a `ResourceConfiguration` as shown. The resource configuration can be used by the `create` method to provide other resource information.

Stop Method

The `stop` method of `AbstractHttpServer`:

1. Stops an instance of an HTTP server
2. Releases the instance

```
...
public class ExampleHttpServer extends AbstractHttpServer {
    protected HttpServer m_server = null;
    ...
    public void stop() {
        if (m_server) != null return;
        m_server.stop();
        m_server = null;
    }
    ...
}
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The `stop` method is considerably simpler than the `start` method in that it stops the underlying HTTP server if there is one, and then releases the instance.

Registering a Custom REST Server

The `class-name` element of the `http-acceptor` element is used to specify the name of a custom class.

```
 . . .
<http-acceptor>
    <class-name>example.ExampleHttpServer</class-name>
    <local-address>
        <address>localhost</address>
        <port>8080</port>
    </local-address>
</http-acceptor>
. . .
```

rest-cache-config.xml



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

A custom server is registered by adding a `class-name` element to the HTTP acceptor and specifying the fully qualified name of the class. Note that both the Grizzly HTTP Server and the Oracle Lightweight HTTP server can be specified using their fully qualified class names, `com.tansosol.coherence.rest.server.DefaultHttpServer` and `GrizzlyHttpServer`, respectively.

WebLogic Server Deployment

To deploy Coherence REST to WebLogic Server:

1. Define a `web.xml` deployment specifying the Oracle Jersey Lightweight servlet container
2. Define a `weblogic.xml` deployment descriptor, which specifies to prefer the `web-inf` classes
3. Package the Coherence classes into a WAR file with the deployment descriptors
4. Deploy the WAR file to WebLogic



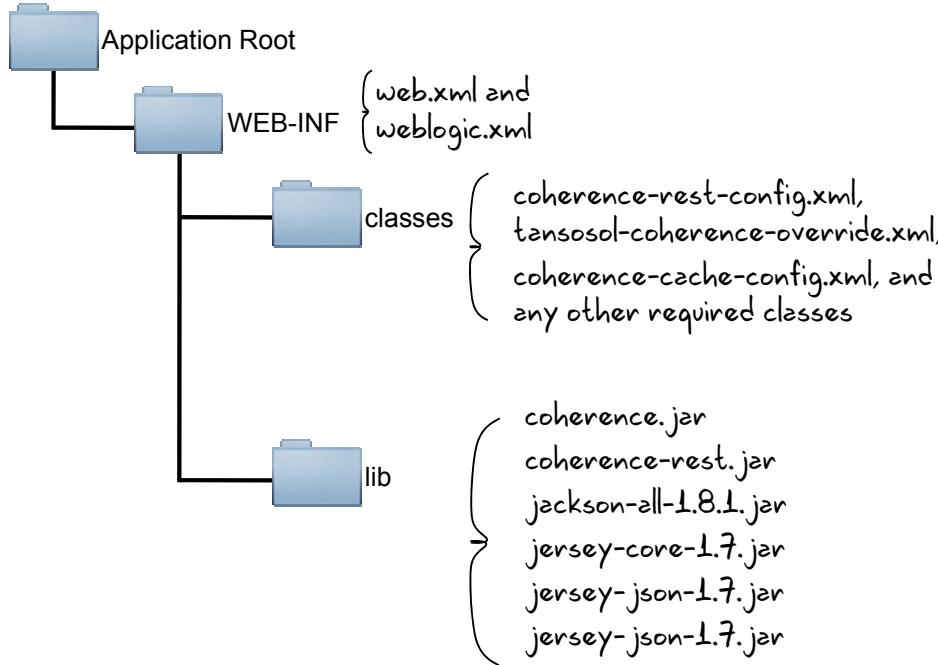
Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST can be deployed to WebLogic using a simple servlet packaged with a WAR file. The components required, detailed in the next slides, include:

- `web.xml` – A Web application deployment descriptor specifying the name of the web application, the servlet which backs it, and an appropriate URL mapping
- `weblogic.xml` – A WebLogic application deployment descriptor specifying appropriate WebLogic settings—in the case of Coherence, to prefer the version of Coherence packaged within the WAR over classpath and so on
- Coherence – The Coherence REST jars and appropriate configurations

After it is packaged, the WAR file can be deployed to WebLogic Server using the console or WebLogic Scripting Tool (WLST).

REST Web Application Structure



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The REST web application structure follows the standard web application packaging format but must include various files in the correct locations.

- WEB-INF – This directory must include `web.xml` and `weblogic.xml` as shown.
- WEB-INF/classes – This directory must include the Coherence configuration files, and is searched by the classloader first. The default file names must be used, including:
 - `coherence-rest-config.xml` – The REST configuration, defining what caches and objects to expose
 - `tansosol-coherence-override.xml` – Overrides as appropriate
 - `coherence-cache.xml` – Cache definitions. Note that no proxy is required when using a servlet.
- WEB-INF/lib – This directory must include the Coherence jar files, including: `coherence.jar`, `coherence-rest.jar`, `ackson-all-1.8.1.jar`, `jersey-core-1.7.jar`, `jersey-json-1.7.jar`, and `jersey-json-1.7.jar`

web.xml Deployment Descriptor

```
<?xml version = '1.0' >
<web-app . . . >
  <servlet>
    <servlet-name>Coherence REST</servlet-name>
    <servlet-class>
      com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>
        com.sun.jersey.config.property.resourceConfigClass
      </param-name>
      <param-value>
        com.tangosol.coherence.rest.server.DefaultResourceConfig
      </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>jersey</servlet-name>
    <url-pattern>/jersey/*</url-pattern>
  </servlet-mapping>
</web-app>
```

web.xml

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The traditional web application deployment descriptor that is used to define the servlet, which serves the REST requests, is shown in the slide. The main points of the slide include:

- **Servlet class** – The servlet class defines the class, which serves the requests, and must be `com.sun.jersey.spi.container.servlet.ServletContainer`.
- **Initialization** – The `ServletContainer` class requires two initialization parameters that must be:
 - `com.sun.jersey.config.property.resourceConfigClass`
 - `com.tangosol.coherence.rest.server.DefaultResourceConfig`
- **Auto start** – Lastly, the servlet must be marked to load at startup as shown.

weblogic.xml Deployment Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<wls:weblogic-web-app
    xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-web-app"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd
    http://xmlns.oracle.com/weblogic/weblogic-web-app
    http://xmlns.oracle.com/weblogic/weblogic-web-app/1.0/weblogic-web-
app.xsd">
    <wls:container-descriptor>
        <wls:prefer-web-inf-classes>true</wls:prefer-web-inf-classes>
    </wls:container-descriptor>
</wls:weblogic-web-app>
```

weblogic.xml



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The `weblogic.xml` deployment descriptor instructs the container to prefer the WEB-INF/classes directory over the classpath when resolving classes. This forces the loading of the configuration files specified in this directory.

Deploying the Web Application

Deploy the web application using the WLS console or WLST.

wls:/...> deploy(appName='restApp',path='~/coherence/')

wlst.deploy example

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

After it is constructed and packaged, the web application can be deployed by using either the WebLogic Server console or the WLST deploy command. See the WebLogic Server documentation for complete information about deploying web applications.

Single Object Operations

Single object operations allow users to retrieve, insert, update, or delete one single cache object

Operation	Description & Example	Result
GET	Retrieves a single object GET <code>http://localhost:8080/acache/1</code>	Returns the object or 404 if not found
PUT	Inserts or replaces a single object PUT <code>http://localhost:8080/acache/1</code> <code>'{@type': 'example.Person', 'name': 'chris', 'age': '30'}'</code>	Returns 200(Ok) on update, 201(created) on insert
DELETE	Deletes a single object DELETE <code>http://localhost:8080/acache/1</code>	Returns 200 on success, 404 if not found



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST supports three single object operations:

- **GET** – Given a cache and a key, returns the object. The typical format of a get is: GET `http://address:port/cache/key`. It returns 200 (OK) on success or 404 if not found.
- **PUT** – Inserts or replaces an object into a cache. The typical format is PUT `http://localhost:8080/acache/1` with Json and `'{@type': 'example.Person', 'name': 'chris', 'age': '30'}'` with Json payload.
- **DELETE** – Deletes the object in the cache. The typical format is: DELETE `http://localhost:8080/acache/1`.

Multiple Object Operations

Multiple object operations allow users to retrieve or delete multiple objects with a single request.

Operation	Description & Example	Result
GET	Retrieves multiple objects GET <code>http://localhost:8080/acache/{1,2}</code>	Returns the objects in an undefined order with code 200; empty set if nothing is found
DELETE	Deletes multiple objects DELETE <code>http://localhost:8080/acache/{1,2}</code>	Returns 200



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Partial Results

Partial results can be returned by specifying a `p` parameter after a key specification supplying the field names.

In general:

```
GET http://localhost:8080/somecache/1;p=field[,field...]
```

Assuming:

```
Public class person {  
    public String name;  
    public String age;  
    public String sex;  
}
```

GET:

```
GET http://localhost:8080/somecache/1;p=name,age
```

Returns:

```
{ "age":30, "name": "chris" }
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

REST GET requests can request data subsets by specifying `p=field1, [field2]` as a suffix for a GET operation. In the example shown in the slide, assuming the person class has three fields—name, age and sex—the GET operation returns only the specific fields requested.

Queries

Queries can be:

- Performed by supplying a query as a `q` parameter
- Ordered by using the `sort` parameter
- Paged by using `start` and `count`

In general:

```
GET http://localhost:8080/somecache/  
;sort={field[:asc|:desc]}  
;start={. . . }   
;count={. . . }  Integer values  
;q={query}
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Coherence REST supports queries by using the `q={query}` parameter. In general, queries can be:

- **Order** – Specifying `sort=[field, [field[:asc|:desc]]]` indicates the sort fields and order (ascending or descending). Ascending is the default.
- **Starting point** – Specifying `start=integer` determines where in the result set to start returning results. For example, 1-10 returns first the 10 instances and 11-20 returns the next 10 instances.
- **Returned objects** – Specifying `count = integer` determines how many objects are returned.
- **Query** – This is specified by using `q=(query)`.

Summary

In this lesson, you should have learned how to:

- Describe Representational State Transfer
- Modify Java objects to support REST
- Configure Coherence to return cache objects via REST
- Start Coherence REST Servers in stand-alone or within a Servlet container such as that provided by WebLogic Server
- Query REST data by using the GET, PUT, and DELETE operations



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Partition-Level Transactions

ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Enumerate atomic partition-level transaction concepts
- Describe Cache and EntryProcessor features
- Implement Coherence 3.7 Atomic Operations support



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Partition-Level Transactions

Benefits:

- Development of entry processors that process multiple entries atomically across different caches (in the same partition)
- Efficient entry back-ups
 - All changes communicated in a single message “for free”
- Safe and efficient re-entrance support
 - Access and mutate other entries and caches from an entry processor, without:
 - Using heavy transactions
 - Worrying about thread safety
 - Identify and prevent Dead Locking



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Atomic Partition-Level Transactions

Partition-level transactions are:

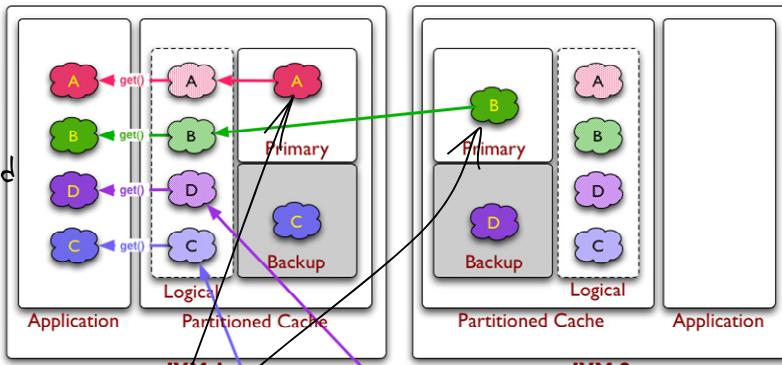
- Written in the context of an entry processor
- Either all complete or none complete
- Used to mutate or change multiple entries in a single operation
- Used to access multiple caches within a single operation



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

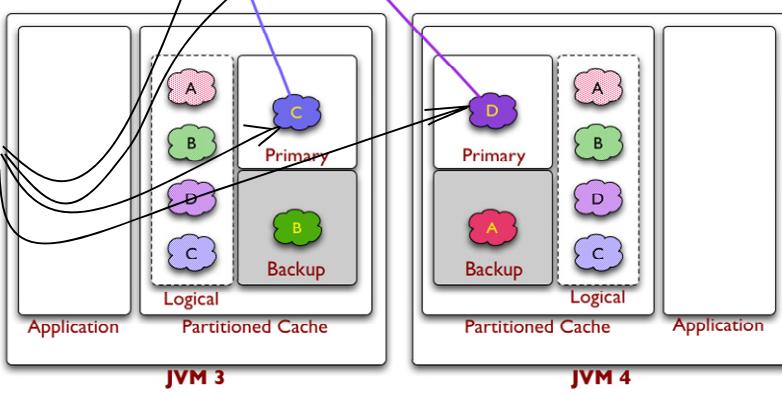
Partitioned Cache: Review

1 Data is distributed across storage-enabled members by partitions.



3 Each member is responsible for a unique set of primary and backup partitions.

2 A specific key is bound to a specific partition for the life of the cache.



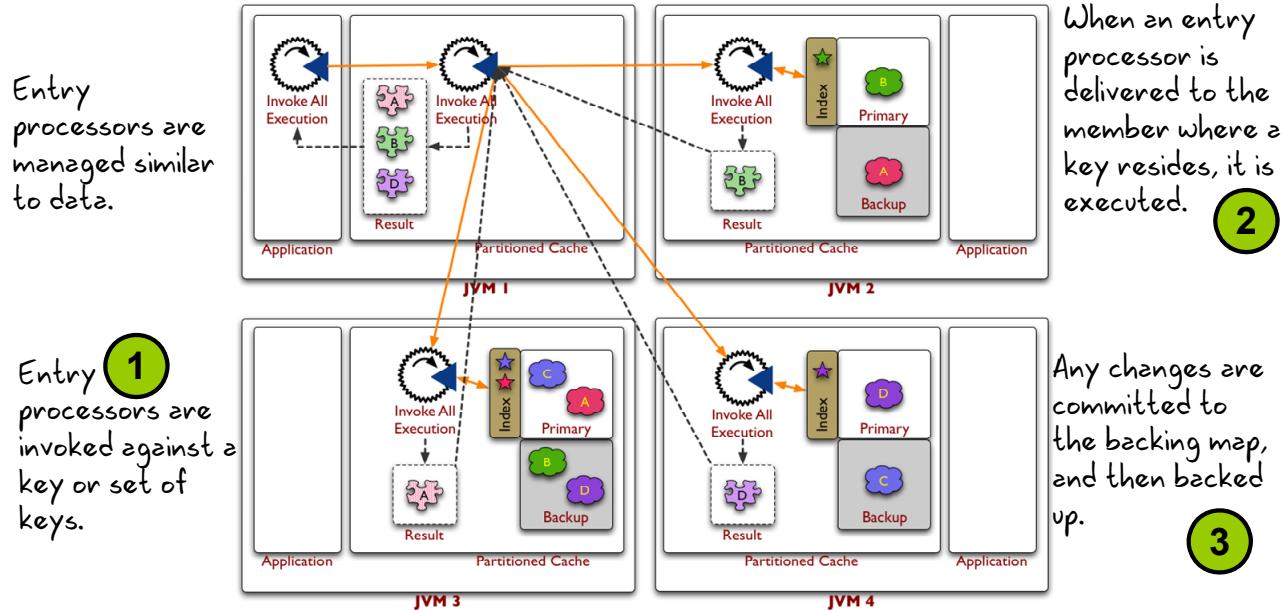
ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In a partitioned cache, data is distributed approximately equally across all storage-enabled members. Each member is responsible for a unique set of primary and backup partitions. A specific key is bound to a specific partition for the entire life of the cache. Likewise, a specific key is bound to a specific backup.

Entry Processor: Review

Entry processors are implemented via the InvokableMap API.



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Entry processors are managed in a fashion that is similar to data.

1. First, an entry processor is invoked against a key or set of keys.
2. The entry processor is then delivered to the member where the key resides and is executed there.
3. Any changes made to the data are then committed and backed up.

Concurrent Processing and Entry Processors

- Entry processor methods are executed concurrently.
- The server isolates the operation from all other operations on the same key.

```
public Object process(Entry entry) {  
    // get the value of the entry  
    Integer iValue = (Integer)entry.getValue();  
  
    // Update the value  
    iValue++;  
  
    // Update the value, holding the lock on this entry  
    entry.setValue(iValue);  
  
    return iValue;  
}
```

Entry processors implicitly hold a lock on the key for which they are executing.

As a result, any code executing within the entry processors process method is atomic.



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Because an entry processor's process method is held against a specific key, the method can be considered to hold a lock against that key. As a result, the developer does not need to be concerned about concurrency issues. All processing within the process method can be considered atomic.

Partition-Level Operations Pre 3.7: Example

```
public Object process(Entry entry) {  
  
    // modify the checking value  
    BinaryEntry checking = (BinaryEntry)entry;  
    Integer newAmount = (Integer) checking.getValue - m_iAmount;  
    checkingEntry.setValue(newAmount);  
  
    // Possible race condition  
    // All data in interval format  
    Map savingsMap = checking.getContext().getBackingMap("savings");  
    BinaryEntry savings =  
        (BinaryEntry)savingsMap.get(m_iSavingsAccountId);  
    newAmount = (Integer)savings.getValue() + m_iAmount;  
    savingsMap.put(m_iSavingsAccountId, savings.getContext().  
        getValueToInternalConverter().convert(newAmount));  
  
    return true  
}
```

All these operations are suspect!



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, a simple checking to savings transfer is implemented and involves the following steps:

1. The checking account key is processed by the entry processors; as a result, all operations on this object are atomic.
2. The checking entry is used to obtain the backing map for the savings account.
3. The savings account data is then accessed; however, operations are *not* atomic and data is in internal format.
4. The PUT operation on the savings account could result in a race condition if another access occurred at the same time.
5. In addition, data must be converted back to the internal format because backing map objects are stored in binary form.

Atomic-Level Operation Support

- Coherence added support for accessing multiple keys atomically in 3.7.
- `BackingMapContext.getBackingMapEntry(Object key)` can be used to access non-Entry data.
- Considerations and requirements are as follows:
 - Keys must be located in the same partition. Continue to use `KeyAssociation`.
 - All operations are atomic in the context of a single `EntryProcessor`.
 - All entries that access via this method are locked.
 - Transaction-read consistency is maintained.
 - Deadlock detection is now supported.

Note: Always use `BackingMapContext` to access other data.



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Partition-Level Operations 3.7 and Later: Example

```
public Object process(Entry entry) {  
  
    // modify the checking value  
    BinaryEntry checking = (BinaryEntry)entry;  
    Integer newAmount = (Integer) checking.getValue - m_iAmount;  
    checkingEntry.setValue(newAmount);  
  
    // access is safe, and no longer requires format translation  
    BinaryEntry savings =  
        ((BinaryEntry)checking).getContext()  
            .getBackingMapContext("savings")  
            .getBackingMapEntry(m_iSavingsAccountId);  
    newAmount = ((Integer)savings.getValue()) + m_iAmount;  
    savings.setValue(newAmount);  
  
    return true;  
}
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, a simple checking to savings transfer is implemented using `getBackingMapEntry ()`.

1. The checking account key is processed by the entry processors; as a result, all operations on this object are atomic.
2. The checking entry is used to obtain the backing map for the savings account.
3. The savings account data is then accessed. Operations are now sandboxed and part of the same transaction.
4. The `setValue` operation now uses the external value.

Accessing Other Keys in the Same Cache

Partition-level transactions can also be applied to other keys in the same cache.

```
...
public void process(Entry entry) {

    ((BinaryEntry)entry).getBackingMap()
        .put("otherkey", value);

    //get an entry in the same partition
    Entry otherentry =
        ((BinaryEntry)entry).getBackingMapContext()
            .getBackingMapEntry("somekey");
}

...
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned to:

- Describe atomic partition-level transactions
- Review a partitioned cache and an entry processor
- Explain Coherence 3.7 atomic operations support



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.



ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Enumerate the features of Elastic Data
- Configure Elastic Data



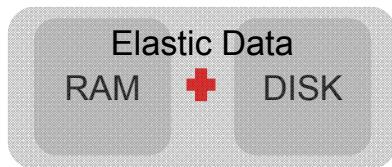
Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

What Is Elastic Data?

Elastic Data:

- Is a seamless, efficient, and transparent bridge between memory and disk
- Is tuned for Solid State Disks
- Supports massive storage capacity
- Shows virtually no performance difference between flash and memory
- Eliminates “out-of-memory” errors

A specific amount of memory is specified for a cache.
The rest spills over into flash.



Simplifies configuration:

- Specify on-heap memory.
- Specify storage location.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Elastic data, which was introduced in Coherence 3.7, allows for the use of solid state devices (most typically flash drives) to provide spillover capacity for a cache. Solid state disks or SSDs use integrated circuit assemblies as memory to store data persistently rather than the spinning platter that is used in typical hard disk devices (HDD). SSDs are considerably faster, more resistant to shock, and silent in comparison to their HDD cousins. However, they are also considerably more expensive.

Using Elastic data, a cache is specified to use a specific type of backing map based on a RAM or DISK journal. After it is defined, the cache can support massive amounts of data per node, often as much as 100 GB. Elastic data has been extensively tested and is proven to have almost no performance difference with totally RAM-based solutions.

Additionally, Elastic data is simple to configure. In the base case, a caching scheme need only specify that the backing map be RAM journal based. However, a variety of configuration options exist for specifying how much memory to use for the backing map, what device to use, and others.

Benefits

Elastic Data:

- Stores up to four times as many objects in RAM compared to past default cache definitions
- Stores up to 100 GB of data per disk per cluster member while maintaining low latency and high throughput
- Self tunes
- Minimizes GC impact and performs on par or better than default schemes for pure in-memory datasets
- Is very easy to configure

Elastic data is ideal for state-of-the-art SSDs such as those provided with Oracle Exalogic.



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The benefits of elastic data versus a simple distributed scheme are many.

- Because of the use and nature of journaling, up to 4x objects may be stored in memory as compared to previous versions of Coherence.
- With journaling and compression, datasets of 100 GB can be managed on single-cluster members.
- Elastic data self tunes and tracks object use, rebalancing the internal tree structures to maintain the most commonly used data in memory.
- Java garbage collection is minimized as a result of the tree-based journals used to manage data.
- It is very easy to use, based on several sample schemes provided.

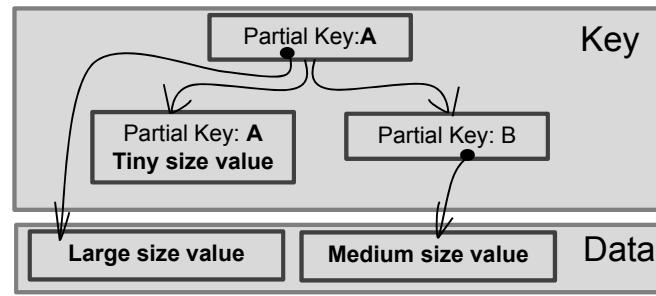
Elastic Data and Journaling

Elastic data is based on journals. Journals:

- Are a technique for storing object state changes
- Record values for a specific key
- Use in-memory trees to store pointers to values for keys
- Are purged at regular intervals to remove stale data
- Batched file writes based on device block size
- Are provided in two forms: RAM or flash journals

Tiny, less than seven bytes, values are stored directly in the tree.

```
put(A, large size value);
put(AA, tiny size value)
put(AB, medium size value)
```



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

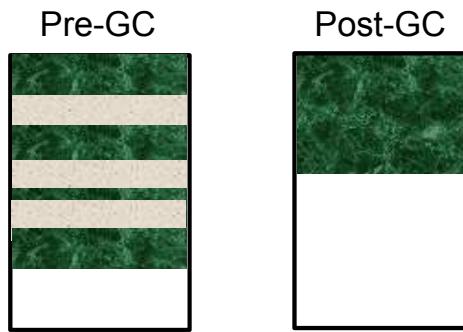
To enable elastic data, all you need to do is specify a backing map scheme that uses a flash or RAM journal. Coherence provides two journal schemes, the RAM journal scheme and the flash journal scheme. Each of these schemes works in a similar way. In general, journals record the actual data object or data for a specific key. An in-memory map is then used to specify which journal contains the most recent entry for a given key. Over time, as mutations occur, journal elements become outdated, and are purged by Coherence.

In the example shown, three puts with small, medium, and large values, are done to a cache. As shown, the small value is stored directly in the tree, whereas the other two values are stored in a journal file.

Note that when journals are used, additional capacity is required for performing queries and aggregations. See the administrators' guide for details.

Journaling and Garbage Collection

- As values change, old values are invalidated and become garbage.
- When the garbage ratio threshold is reached, garbage collection occurs concurrently.
- Files are recycled during the garbage collection process.



ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

As values change, garbage is created within a journal file. Coherence uses an internal garbage collection algorithm to track whether a garbage threshold is reached. When the threshold is reached, a garbage collection thread executes concurrently to remove old values. During this process, files are recycled and reused.

Configuring a Journaling Backing Map

Example:

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>Journaled</cache-name>
      <scheme-name>JournalScheme</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>JournalScheme</scheme-name>
      <service-name>DistributedCache</service-name>
      <backing-map-scheme>
        <[flash/ram] journal-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme></cache-config>
```

coherence-cache-**config**.xml



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The simplest form of an elastic cache is based on the RAM journal, which can be added to the backing-map-scheme element to define an elastic data-based cache. When memory is exceeded, the RAM journal automatically delegates to a flash journal. Specifying a flash journal results in the immediate use of a flash journal rather than being based on running out of memory.

Both flash and RAM journals can be controlled by using command-line arguments and override files. The most common override is the size of the journal itself, typically some percentage of the heap. However, a variety of other values can be controlled, such as how large a value to keep in memory, what directory to use for storage, and similar variables.

Controlling Journaling Behavior

Journal behavior is:

- Controlled by a journal manager
- Defined in an operational override or via properties

```
-Dtangosol.coherence.ramjournal.size={value}
```

Where value is %of heap or size. For example:

```
-Dtangosol.coherence.ramjournal.size=25%
-Dtangosol.coherence.ramjournal.size=10gb
```

cache-server. [cm|sh]



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The most common variable to control with a journal is its size. Coherence supports defining the size of a RAM journal, which is inherited by a flash journal as well, as either a percentage of RAM or as a fixed size. The `tangosol.coherence.ramjournal.size` property can be used on the command line to define a size. However, a variety of other elements can also be defined when using an override file, such as the maximum size of a journal value entry.

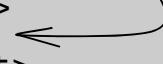
Controlling Journaling Behavior

Journaling can be finely controlled by using the `journaling-config` element within a cluster configuration.

```
<?xml version='1.0'?>
<coherence>
  <cluster-config>
    . .
    <ramjournal-manager>
      <maximum-value-size>64K</maximum-value-size>
      <maximum-size>2G</maximum-size>
    </ramjournal-manager>
    <flashjournal-manager>
      <block-size>256KB</block-size>
      <directory></directory>
      <async-limit>16MB</async-limit>
    </flashjournal-manager>
  </cluster-config>
. .
</coherence>
```

tangosol-coherence-override.xml

Partial list



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

RAM and flash journals can be finely controlled by using an operational override file.

The RAM journal manager subelements control the overall size as well as the maximum value size stored in memory. The two elements of a `ramjournal-manager` element are:

- **maximum-value-size** – The maximum size of an element that can be stored in memory. The default is 65 K and the maximum is 4 MB.
- **maximum-size** – The maximum amount of RAM to be used by the journal. The value can be specified as an actual value or a percentage. Values must be in the 16 MB to 64 GB range. Percentages are a fraction of max heap, typically specified using `-Xmx`.

The flash journal manager subelements include:

- **maximum-value-size** – The maximum size, in bytes, of binary values that are to be stored in the flash journal. The value cannot exceed 64 MB. The default value is 64 MB.
- **block-size** – It specifies the size of the write buffers in which writes to an underlying disk file occur. The size should match or be a multiple of the physical device's optimal block size and must be a power of two. The value must be between 4 KB and 1 MB. The default value is 256 KB.

- **maximum-file-size** – It specifies the maximum file size of the underlying journal files. The value must be a power of two and a multiple of the block size. The value must be between 1 MB and 4 GB. The default value is 2 GB.
- **maximum-pool-size** – It specifies the size, in bytes, for the buffer pool. The size does not limit the number of buffers that can be allocated or that can exist at any point in time. The size determines only the amount of buffers that are recycled. The pool size cannot exceed 1 GB. The default value is 16 MB.
- **Directory** – It specifies the directory where the journal files should be placed. The directory must exist and is not created at run time. If the directory does not exist or is not specified, the JVM or the operating system's default temporary directory is used. The suggested location is a local flash (SSD) drive. Specifying a directory that is located on a drive, which is shared by other applications or system operations increases the potential for unplanned space usage. Use a directory location on a nonshared disk partition to ensure a more predictable environment.
- **async-limit** – It specifies the maximum size, in bytes, of the backlog. The backlog is the amount of data that is yet to be persisted. Client threads are blocked if the configured limit is exceeded and remain blocked until the backlog recedes below the limit. This helps prevent out-of-memory conditions.

Note: The maximum amount of memory used by the backlog is at least twice the configured amount, because the data is in binary form and rendered to the write-behind buffers. The value must be between 4 KB and 1 GB. The default value is 16 MB.

Considerations for Elastic Data

Elastic data:

- Typically should be sized 1:10 memory to disk
- Uses files, but is not a persistence solution
- Indexes are stored in memory increasing memory use
- Does not support Cache Eviction
- Should not be used with aggregations and entry processors



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Summary

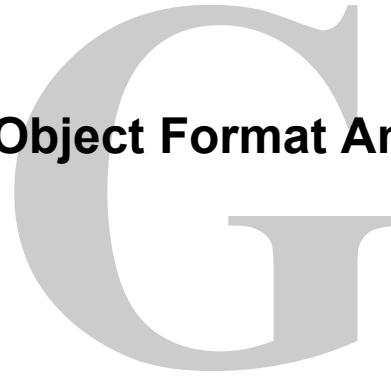
In this lesson, you should have learned how to:

- Enumerate the features of Elastic Data
- Configure Elastic Data



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Portable Object Format Annotations



ORACLE®

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Enumerate supported annotations
- Annotate classes to support serialization
- Register annotated classes
- Test POF annotations and indexing



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Annotations

Annotations:

- Are based on JSR 175 and were introduced in Java 5
- Relieve the developer from cumbersome configuration
- Avoid the need for boilerplate code by adding “annotations” or hints to the source code

```
public interface Animal {  
    String type;  
}  
Definition
```

```
@Animal(type="mammal")  
public class Bird { . . . }  
Usage
```

Entity objects can be made serializable annotations.

```
...  
@Portable  
public class Person {  
    ...  
    @PortableProperty(2) private int m_age;  
}
```

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Annotations are one of the newer trends in Java development. Annotations, simply put, are markers or hints to the compiler of a virtual machine that a class exhibits a certain behavior or characteristic. In essence, an annotation is a mechanism for associating metatags with program elements and allowing the compiler or the virtual machine to extract program behaviors from these annotated elements.

Coherence can use annotations to specify that a class is serializable and to specify the properties and methods that should be serialized.

Portable Object Format: Annotations

POF annotations:

- Provide an automated mechanism to serialize and de-serialize data
- Are an alternative to coding against the `PortableObject` and the `PofSerializer` interfaces

POF Annotation Based classes are serialized and de-serialized using `PofAnnotationSerializer`, which implements `PofSerializer`



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

POF Serialization supports annotations to define which classes, and within each class, which methods and properties should support serialization. The underlying class, `com.tangosol.io.pof.PofAnnotationSerializer`, can read and act on the annotations within a class to serialize and deserialize its elements. This class, along with its specific annotations, provides developers with an alternative to developing against the `PortableObject` and `PofSerializer` interfaces themselves. Using these annotations, a class can be quickly and easily marked for serialization. While complex serializations will still require implementation of `PortableObject` and so on, simple entity classes can be marked for serialization quickly using POF annotations.

Specifying Portable Object Annotations

POF annotations include:

- Class-level annotation – Portable
- Property-level annotation – PortableProperty

```
package com.tangosol.io.pof.annotation;
@Target(value=TYPE)
@Retention(value=RUNTIME)
public interface Portable extends Annotation {}
```

Can be applied only to classes

```
package com.tangosol.io.pof.annotation;
. . .
@Target(value={FIELD,METHOD})
@Retention(value=RUNTIME)
public interface PortableProperty
extends Annotation {
    public abstract int value();
    public abstract Class codec();
}
```

Maybe be applied on fields or methods

Index value for this field or method



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Two new annotations have been added to support automated generation of POF.

These annotations are available to indicate that a class and its properties are POF serializable.

- @Portable: Marks the class as POF serializable. The annotation is permitted only at the class level and has no members.
- @PortableProperty: Marks a member variable or method accessor as a POF-serialized attribute. Annotated methods must conform to accessor notation (get, set, and is).

This annotation takes two possible parameters, `value` and `codec`. `value` specifies the index of the field, whereas `codec` is used to specify a concrete implementation of a class when an object returns an interface. An example of when to use a `codec` might be when a method return type is a `Map`. In such a situation, a `codec` would be used to instruct the system to return a specific implementation of the `Map` interface such as `LinkedHashMap`.

POF Annotations: Example

POF annotations must be added to a class and registered in pof-config.xml.

```
import com.tangosol.io.pof.annotation.*;
@Portable
public class Person {
    @PortableProperty(0)
    public String getFirstName() { return m.firstName; }
    .
    .
    @PortableProperty(2)
    private int m_age;
}
```

Person.java


```
<?xml version='1.0'?>
<pof-config xmlns:xsi=". . .">
    <user-type-list>
        <include>coherence-pof-config.xml</include>

        <!-- User types must be above 1000 -->
        <user-type>
            <type-id>1001</type-id>
            <class-name>. . .Person</class-name>
        </user-type>
    </user-type-list>
</pof-config>
```

pof-config.xml



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Specifying annotations requires two steps. First, a class must be annotated. In the example shown, the class is annotated, along with both the methods and properties. Note that only some of the fields are shown in the example; however, indexes would be specified as consecutive integers.

After it is annotated, the class needs to be noted in a pof-config.xml file. In the example shown, pof-config shows the class being assigned a type ID of 1001.

POF Automatic Indexing

POF Supports the ability to automatically apply indexes to classes. Automatic indexing:

- Removes the need to assign and manage indexes
- Still requires @PortableProperty, but without a value
- Properties with a value use the explicit value
- Must be explicitly enabled in pof-config.xml

```
<?xml version='1.0'?>
<pof-config xmlns:xsi=". . . " >
  <user-type-list>
    <user-type>
      <type-id>1001</type-id>
      <class-name>. . . Person</class-name>
      <serializer>
        . . .
        </serializer>   <-- Detailed on next slide
      </user-type>
    </user-type-list>
  </pof-config>
```



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

To further simplify the process of using POF annotations, developers can auto assign index values to class fields and methods rather than hard coding values into a class. When using the auto indexing feature, a developer ignores the value attribute of the PortableProperty field. The serializer then automatically assigns indexes to properties. Note that you can still selectively use indexes in your class, with the POF serialization filling in any missing indexes. The POF Serializer class, by default, does *not* implement auto indexing. To use the auto indexing feature, enable auto indexing by using a serializer element as described in the next slide.

POF Serializer Configuration

POFAnnotationSerializer is configured in the pof-config.xml, once per user class.

```
<user-type>
  <type-id>1001</type-id>
  <class-name>...Person</class-name>
  <serializer>
    <class-name>com.tangosol.io.pof.PofAnnotationSerializer</class-
name>
    <init-params>
      <init-param>
        <param-type>int</param-type>
        <param-value>{type-id}</param-value>
      </init-param>
      <init-param>
        <param-type>class</param-type>
        <param-value>{class}</param-value>
      </init-param>
      <init-param>
        <param-type>boolean</param-type>
        <param-value>true</param-value>
      </init-param>
    </init-params>
  </serializer>
</user-type>
```

Parameters conform to the order specified in the class constructor. See the Javadoc for PofAnnotationSerializer for a complete list.

AutoIndex is the third parameter of PofAnnotationSerializer.

pof-config.xml



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

To use the auto index feature of Coherence without indexes, you must configure the serializer class to enable auto indexing. To enable this feature, you must first add a `serializer` element to the configuration for the class to be serialized. This element supports the `class` subelement. The `class` element supports defining the class associated with a given feature or function—in this case, the `com.tangosol.io.pof.PofAnnotationSerializer` class. This class has two constructors, the second of which supports enabling auto indexing by setting the third parameter to true as shown in the slide.

Summary

In this lesson, you should have learned how to:

- Explain POF annotations
- Annotate classes to support serialization
- Register annotated classes
- Explain POF annotations and indexing



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and ORACLE CORPORATION use only