

Oracle Coherence 3.6: Share and Manage Data In Clusters

Student Guide - Volume I

D66791GC11

Edition 1.1

October 2012

D79398

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

I

Course Introduction

Oracle Coherence 3.6

ORACLE

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

Audience

The target audience includes:

- Java developers
- Java architects
- System Administrators
- Other technical personnel who are responsible for scalable applications using Oracle Coherence



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

Class Introductions

Please introduce yourself:

- Name
- Title/position
- Company
- Experience using Java, application development, and Eclipse
- Reasons for attending



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

Goal

Goals of this course include learning to install, configure, and develop scalable applications using Java and Oracle Coherence.



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

Course Objectives

After completing this course, you should be able to:

- Install Oracle Coherence
- Configure Coherence Caches and caching schemes
- Develop Java entity objects optimized for Coherence
- Optimize Java entity objects, stored in Coherence caches, using various techniques, including:
 - Java `Serializable`, `ExternalizableLite`
 - Coherence Portable Object Format
- Implement the various Coherence event and processing models, including pre- and postprocessing of cache events as well as “in-place” data processing
- Describe, configure, and deploy local, replicated, distributed, and near cache topologies



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

Course Objectives

After completing this course, you should be able to:

- Extend Java applications to perform cache queries and aggregations using Filters and the Coherence Query Language
- Extend Java Coherence applications to support security and transactions
- Implement a cache that is backed by a persistent store such as a database
- 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



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

Course Objectives

After completing this course, you should be able to:

- Identify and describe the basic tasks for performance tuning Coherence cache clusters
- Identify and describe the various uses and components of Coherence*Web and Coherence*Extend



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

Prerequisites

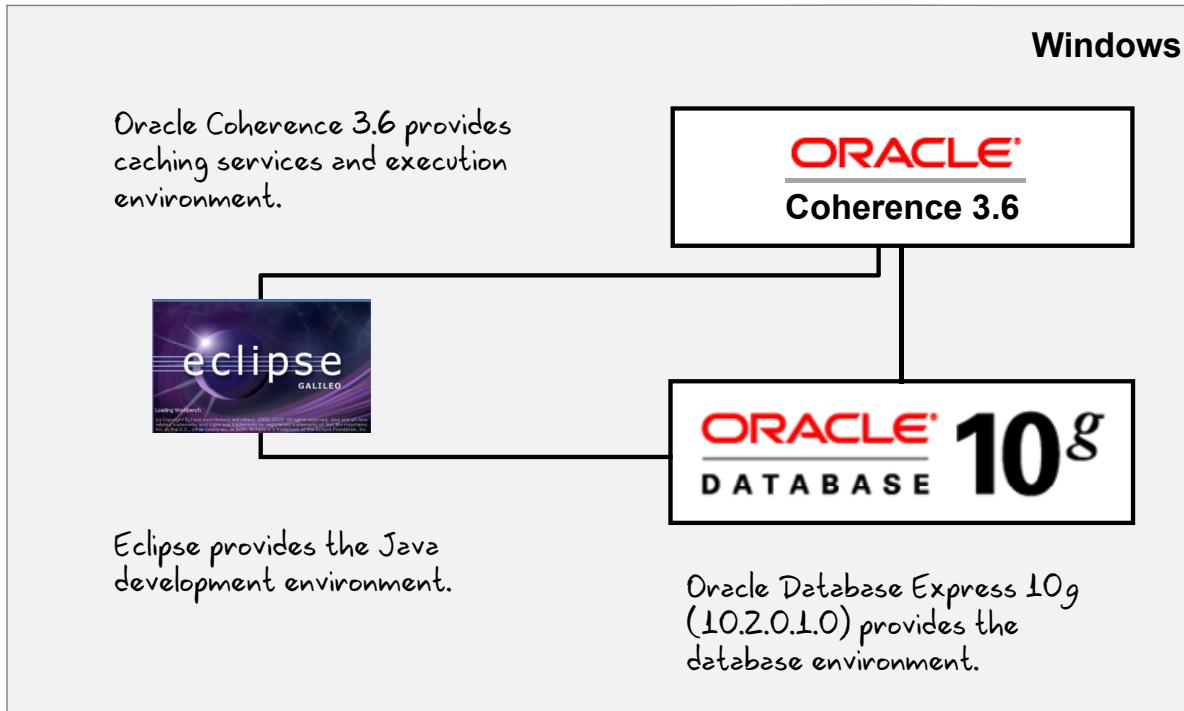
To successfully complete this course, you must meet the following requirements. You must have:

- Working experience in the Java programming language, including the Java collections classes and template concepts
- Knowledge of XML and XML DTD is helpful but not required
- Knowledge of UML and Eclipse is helpful but not required



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

Course Environment



ORACLE

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

Course Environment

In this course, the following products are used for the lesson practices:

- Microsoft Windows
- Oracle Coherence 3.6
- Oracle Database XE 10.2.0.1.0
- Eclipse
- Java JDK 1.6.0_20

Course Conventions

The following conventions are used throughout this course:

- Courier New is used for the names of commands, files, directories, programming code, and on-screen computer output.
For example: Use `dir /w` to list files in wide format
- Courier New is also used to indicate programming constructions, such as class names, methods, keywords, or code fragments.
For example:
`for (int i =0; i< 10; i++) { . . . }`
- Courier New Bold is used for characters and information you enter.
For example: `c:\> dir`



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

Course Conventions

The following conventions are used throughout this course:

- **Courier New Bold** is also used for lines of programming code. For example:

```
import java.io.*;
import com.tangosol.*
import com.oracle.education.*;
```

- **Courier New Italic Bold** is used variables or values to be entered by the student. For example:
Type `chmod a+rwx filename` to grant read, write, and execute rights for `filename` to world, group, and users.



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

Course Schedule

Session	Module
Day 1	Lesson 0: Course Introduction Lesson 1: Introduction to Coherence Lesson 2: Getting Started Lesson 3: Working with Objects
Day 2	Lesson 4: Configuring Caches Lesson 5: Cache Events
Day 3	Lesson 5: Cache Events (continued) Lesson 6: Querying and Aggregating Data in the Cache Lesson 7: In-place Processing and Entry Processors



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

Course Schedule

Session	Module
Day 4	Lesson 8: Coherence and Transactions Lesson 9: Data Integration Lesson 10: Typical Caching Architecture
Day 5	Lesson 11: Coherence*Extend Lesson 12: Coherence Administration Lesson 13: Understanding Coherence Security



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

How Can I Learn More?

Topic	Website
Education and Training	http://education.oracle.com
Product Documentation	http://www.oracle.com/technology/documentation
Product Downloads	http://www.oracle.com/technology/software
Product Articles	http://www.oracle.com/technology/pub/articles
Product Support	http://www.oracle.com/support
Product Forums	http://forums.oracle.com
Product Tutorials	http://www.oracle.com/technology/obe
Sample Code	http://www.oracle.com/technology/sample_code
Coherence Knowledge Base	http://coherence.oracle.com
Coherence Incubator Project	http://coherence.oracle.com/display/INCUBATOR/Home



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

How Can I Learn More?

Be mindful of the underscore in the URL for /sample_code, and the case-sensitive /INCUBATOR/Home.

1

Introduction To 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 concepts of system performance and scalability
- Describe how Coherence addresses scalability and performance
- Describe the architecture and concepts of Coherence



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

Agenda

- **Performance Concepts, Caching, and Scalability**
- Introduction to Coherence



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

System Performance Degradation

- Most enterprise-class systems suffer some form of performance degradation in their lifetime
 - They are typically unexpected, unacceptable, and inconvenient
 - The solutions are “business critical”
- The reason you are here today:
 - You have observed, suspect, or have been told, “The system is slow,” and you need to find a solution
 - You are embarking on a new project and would like to avoid such issues
 - You are interested in such technologies
 - Your manager told you to attend



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

System Performance Degradation

- You have already...
 - Upgraded to the latest versions of components
 - Adopted the latest and most efficient language features
 - Implemented the most efficient data-structures and algorithms
 - Deployed on the best hardware available
 - Tuned the operating systems
- When everything else fails...
... perhaps it's time to introduce some caching?



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

System Performance Degradation

As experience demonstrates, performance problems often come as a complete surprise to a developer and usually at an inconvenient time. For example, you've embraced the latest design/implementation principles to avoid known performance issues, you've performed routine and systematic performance testing, you've used highly optimized algorithms and deployed the system on high performance infrastructure, and yet all of a sudden the system starts to struggle under load.

So what do you do? Typically one of the common answers is to, "add some caching," because conventional wisdom says that caching makes things run faster. Is this really true? Is adding caching the "right" thing to do? Is caching enough?

When Performance Problems Aren't Performance Problems

- When told, “The system is slow,” it doesn’t mean the system is performing poorly.
 - It’s an observation about response time
 - Often unrelated to system performance
 - Many factors contribute to perceived performance degradation
- Observed “slowness” is based on the relative response time increases between uses of a system.
 - “I used the system this morning and it took n seconds, but now it’s much slower”
- This does not mean a system is actually performing worse.



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

When Performance Projects Aren't Performance Problems

The first thing to realize about system performance problems, especially when you’re being told, “The system slow,” is that there is often little correlation between what is being observed and real system performance. This may sound strange, but as developers, you always need to appreciate that when a user, and indeed ourselves, observe “slowness,” what you are observing is perceived performance, due to an unexpected increase in response times for our requests.

Just because your requests are taking longer to be processed, it does not mean a system is performing slow.

It is important to understand that most systems typically don’t add:

- “`Thread.sleep()`”s and other deliberate delays
- New complex and unnecessary processing

while in operation.

But rather, most systems are completely predictable, which then begs the question, “*Where does the perceived ‘slowness’ originate?*”

When Performance Projects Aren't Performance Problems (continued)

Most users fail to:

- Recognize there is more than just them using a system
- Appreciate the “load” caused by other users
- Care (everyone wants everything immediately)

When you listen to users, you are told:

- “The system is slow”
- “We need to speed up the system”

Consequently you:

- Naturally believe there is a system performance issue
- Start looking for performance optimizations

When Performance Problems Are Capacity Problems

- Most perceived performance problems are a symptom of another more serious problem: ***lack of system capacity***.
 - For example:
 - A system performs acceptably for a single user
 - The system is observed to slow down with multiple users
 - Typically means that the system doesn't have the capacity to handle user requests as expected, and that:
 - User requests queue up
 - Users start to notice delays
 - Users comment, "This system is slow"

"Trust me, you have a capacity problem, not a performance problem." – Cameron Purdy, VP Coherence Engineering



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

When Performance Problems Are Capacity Problems

Here's how to think about it: If a system performs acceptably for a single user but when accessed by multiple users they consistently observe consistently higher response times, the system has a capacity problem, not a performance problem.

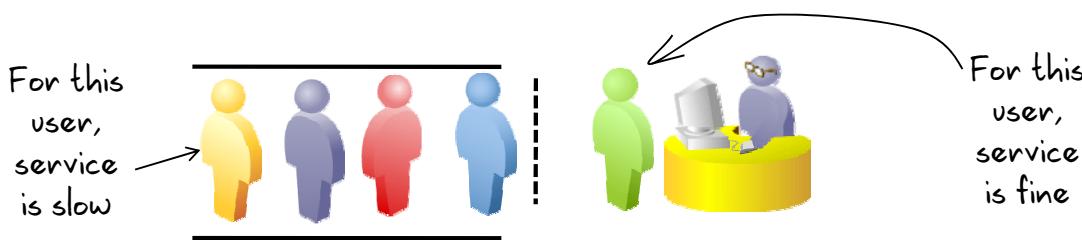
The system doesn't provide enough capacity to satisfy the number of users simultaneously attempting to access it. The consequence is higher response times as user requests queue up to be serviced.

If a system doesn't perform well for a single user, then there is a serious performance problem. It is unlikely to be a capacity problem (unless the user requests are significantly large).

Quantifying Capacity

Capacity can be quantified in terms of average *wait time*.

- Wait time = (average # of waiting users) * (average service time per request)
- For single users, with low processing time, service is *instantaneous*.
- For many, with the same processing time, they conclude the system is *slow* even though the service time is identical.



ORACLE

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

Quantifying Capacity

The concept of queues and understanding their behavior within a system is of great importance when attempting to understand the perceived performance degradation characteristics of a system. While beyond the scope of this course, it should be noted that there is an entire body of mathematics devoted to understanding the behavior of queues and the amount of time requests are waiting in such queues. Understanding just a small amount of this body of knowledge can be very useful when investigating system performance issues.

Little's Law, $L = \lambda W$, explains:

- The average time a request will wait in a queue
- The average size of a queue over time

where:

- L = Average number of requests waiting in the queue
- λ = Average rate of request arrival to the queue
- W = Average time to process a request

Quantifying Capacity (continued)

When there is only a single user on a system, the average wait time for the user is simply the time it takes a system to perform a request. If a single user makes 1 request per second (that is, $\lambda = 1$) and each request takes 1 second (that is, $W = 1$), the average number of users in the queue at any point in time will be 1 (that is, $L = \lambda W = 1 \times 1 = 1$).

Thus on average the user will wait 1 second (in the queue) for their request to be processed.

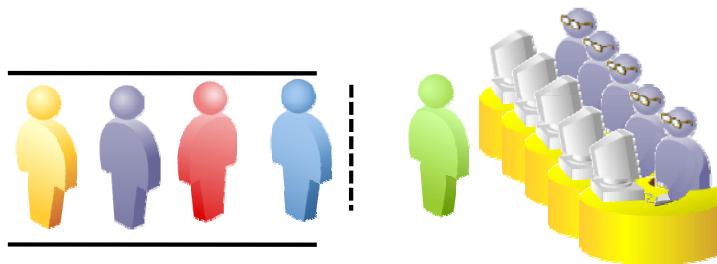
When there are multiple users of a system, things are very different.

For example if there are 10 requests per second being made (that is, $\lambda = 10$), and like last time, each request takes 1 second to be processed (that is, $W = 1$), the average number of requests in the queue at any point in time will be 10 (that is, $L = \lambda W = 10 \times 1 = 10$).

As the system processes requests sequentially, the requests/users in the queue will experience some delay. Consequently, the users will conclude that the system is slow, even though it is actually performing at the same rate as before. They just have to wait for their requests to be processed, whereas with a single user, they generally don't have to wait.

Quantifying Capacity

- How can you extend the single user performance to all users? Assume: Users arrive at a constant rate.
- To eliminate queue depth, performance must be increased by the queue depth. For example, assuming 1 second to process each entry, and a queue of 10, you must improve performance by a factor of 10x.



Moore's Law says it takes 18 months to double performance, or 3.5 years for 10x!

Scaling is the solution for system capacity problems!

ORACLE

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

Quantifying Capacity

Now let's say that you would like to ensure that only a single user is ever waiting in a queue, even though 10 requests per second are still arriving.

In this case, you know that $L = 1$ (maximum of one user waiting) and $\lambda = 10$ (number of requests being made) and thus $W = L / \lambda = 1 / 10 = 0.1$.

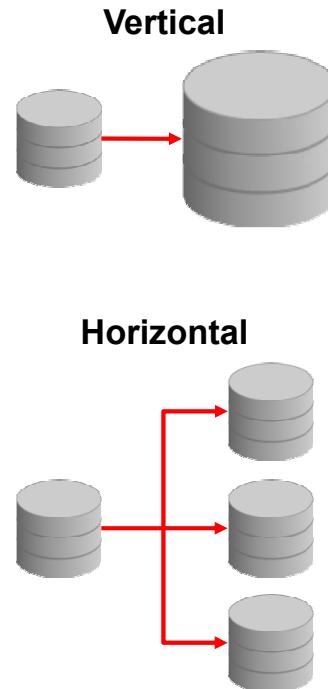
Consequently for the system to behave like it does for a single user (only one user waiting), it would have to process each request in 0.1 seconds instead of 1 second. The system would need to be 10x faster.

The harsh reality here is that this may be very difficult in a single server as the cost or even ability to acquire hardware that is 10x faster is non-trivial.

Moore's Law observes that hardware performance typically doubles every 18 months (round up to two years), thus you would need to wait approximately $(\log_2 10) = 3.5$ years for hardware to provide this level of performance without changing the system. Given that it is unlikely that an enterprise would want to wait that long to increase the processing capacity of the system, the problem must be solved in another way.

Scalability

- **Definition:** *Scaling*, adding or improving system resources so that a larger number of requests can be processed in the same period of time.
- **Scalability**
 - The *effort required to scale a system*
 - Desired property of a system so it can be incrementally improved to provide capacity as required



ORACLE

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

Scalability

While the term “scaling” is used to describe the solution for capacity problems, the term “scalability” is used to describe how easy/difficult, resource/time/investment intensive the act of scaling a system is. The generally accepted solution to the capacity problem as described above is called **scaling**. It is a relatively straightforward concept that simply involves adding more/improving resources in a system so that a larger number of user requests can be processed in a period of time. Scaling comes in two types: vertical and horizontal.

- **Vertical** scaling involves increasing the size or performance of a single resource. Vertical scaling involves replacing hardware components with faster models, for example, using faster processors, using faster memory, faster networks, or faster storage. When the size of a disk is increased to add capacity, it is considered vertical scaling. The strategy here is to run a system on faster components. In terms of our check-in counter example, this is the equivalent of replacing an agent with someone that can work faster. When in a hurry, everyone dislikes being placed in the queue where an agent is “in training” and you know is slow.

Scalability (continued)

- **Horizontal** scaling involves adding new hardware components (of similar capacity). Adding a second or subsequent disk would be considered horizontal scaling. The strategy here is to run a system on many components (in parallel). In terms of our flight check-in counter example, this is the equivalent of adding new agents to the counter.

Horizontal and vertical scalability have their own advantages and disadvantages, including cost. Choosing the “right” approach requires a detailed understanding of a system

It is important to note that both strategies are useful in their own right. Both have their advantages and disadvantages and the “right one” to use to scale a system requires careful consideration. It is also important to understand that both have their own costs.

For example: skilled agents that can check-in customers quickly are often paid more than unskilled. Having many agents is more expensive than just one or two. Often some combination of both is required.

Scaling Example

- Imagine an airline check-In counter with:
 - Large number of customers who want to check in
 - Only a single airline agent at the counter
- Customers:
 - Will naturally queue, they cannot check in all at once
 - Will experience delays (waiting) in the queue for the Agent
 - Will grumble, “Why is this taking so long?”, “This queue is too slow.”

Customers suggest:

- Open more counters with more agents
- Ask the agents to work faster

Customers are saying *scale* the system

Agent comments:

- “Too many customers arriving at once”
- “There aren’t enough agents to fulfill demand”
- “Working as fast as we can”

Agents are saying *not enough capacity*

ORACLE

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

Scaling Example

Consider an airline check-in counter where a large number of customers want to check-in. If there is only a single airline agent, a queue of customers will naturally form waiting to be checked-in (serviced), simply because an airline agent cannot process all of the customers at once (as described by Little’s Law). If you ask the customers about their experience (in the queue), they will usually say things like, “It’s too slow,” “What’s taking so long?”, “I wish they’d hurry up,” thus indicating a performance problem, even though the airline agent may be working as fast as physically possible.

Origins of Performance Problems

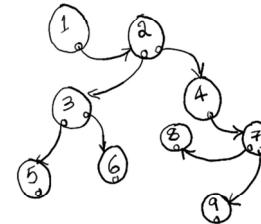
Performance problems typically result from:

- Poor choice of data structures
- Poor or underperforming algorithm(s)
- External interaction latency and locking

These are tightly related.
Changing one, often effects the other.

Solved via Caching!

Improving data structure and/or algorithms often yield better performance versus hardware improvements alone.
And remember, hardware is expensive and slow to improve!



ORACLE

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

Origins of Performance Problems

There are three principle causes of performance problems:

1. Poor/inefficient choice of data-structure(s) for a solution
2. Poor/inefficient algorithm implementation(s) for a solution
3. Latency caused by interaction with an external system(s) that are resource constrained

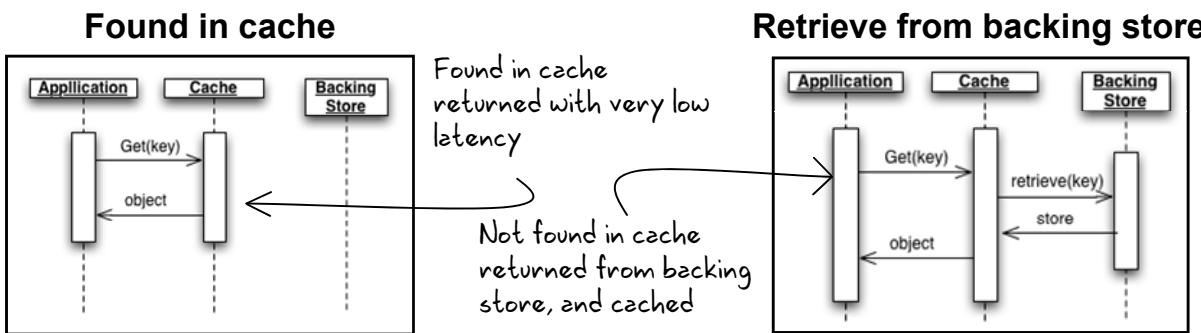
While #1 and #2 are often closely related, mainly because changing one inextricably effects the other, optimizing both will almost always lead to the most significant performance improvements for a system over say, increasing hardware performance. For example, it is possible to improve an algorithm performance by several orders of magnitude, but doing so with hardware may not be possible, or financially practical. Unfortunately, when #1 and #2 are already optimal, you need to consider the effects of #3.

To solve #3, the typical approach is to adopt some form of caching in order to minimize round-trip communication/waiting for with the high-latency external system. The goal of caching is to keep as much of the commonly used information from the external system "as close as possible to" the system so that the cost of retrieving that information from the external system is minimized, or better still, eliminated.

Note: Although caching is commonly used as the solution to #3, it is equally applicable to #1 and #2, as information from high-latency data-structures and algorithms are equally cacheable.

Definition: Cache

- Cache: A *local memory space*, holding *frequently requested* data, that might be otherwise *expensive to return or compute*
 - A cache hit occurs when the requested data is available.
 - A cache miss occurs when the requested data is *not* available.



ORACLE

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

Definition: Cache

A cache is simply a data structure that holds often requested data in memory. In the simplest case, a cache uses a `put` type operation, to store data and then later returns that same data via a `get` operation. If the object is in cache, then the `get` operation returns the object with almost zero latency. The slide shows a `get` on a cache which implements a read-thru operation. If the object is in the cache, its returned immediately, if not, a `read` operation, which is typically orders of magnitude slower, is performed to load the object into cache and return it. A third operation, where the object is not found in either the cache or the backing store is not shown. In that scenario, the “not found” result is returned.

Cache Benefits:

- **Performance:** If data is in the cache, then latency of a cache read is almost zero.
- **Specialized Eviction policies:** Data can be marked with an eviction policy and ejected when the policy is satisfied, managing cache size.

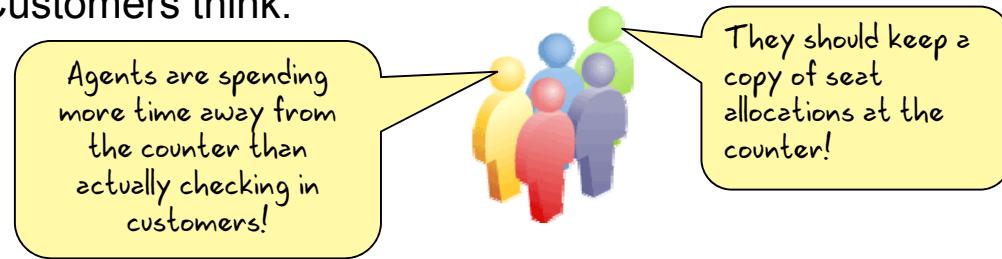
Definition: Cache (continued)

Drawbacks:

- **Consistency:** If the backing store is accessible in multiple ways, then the data can become inconsistent with the backing store.
- **Concurrency:** If multiple users or threads access the cache, concurrent access must be managed.
- **Memory:** Depending on eviction policy its possible to use large amounts of memory if the cache becomes very full.

Scaling Example Revisited

- Assume the agent must walk to/from back-office to confirm seat allocations.
 - Back-office is a long distance from counter.
 - Travel time is additional latency in the process.
- Customers think:



Keeping a copy locally is called "caching."
By "caching," we could eliminate travel latency.

ORACLE

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

Scaling Example Revisited

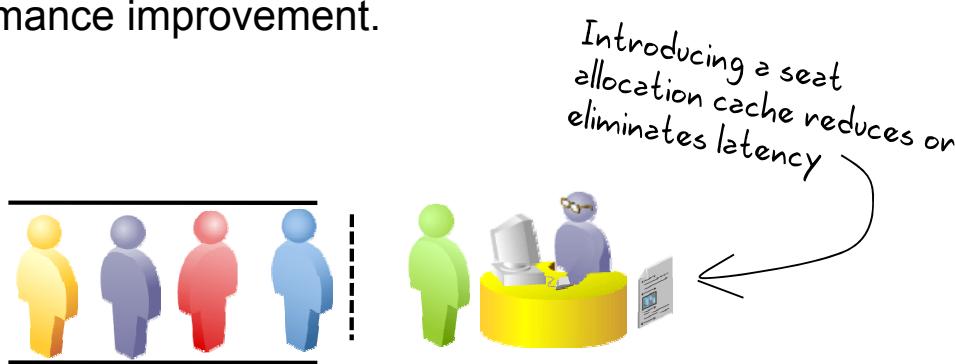
Let's return to the airline check-in example to see the effectiveness of caching on perceived performance.

Although this does not usually happen, let's say that when a customer wants to check-in for a flight, the airline agent needs to physically walk to the back-office to confirm individual seat allocations. If the back-office was a long distance away from the check-in counter, customers would naturally come to realize that the agent is probably investing more time in travelling to and from the back-office than actually processing the reservations at the counter.

Further, it's possible that they (and the agent) would naturally come to realize that the time (that is, latency) to walk to and from the back-office could be optimized by simply keeping a copy of, or the actual seat allocations, at the check-in counter instead of in the back-office, thus saving a lot of travel time.

Caching to Scale

- Assuming introducing a seat allocation cache eliminates travel latency and reduces check-in time by 50%, throughput is then increased by 1.5 or 2.
- In this scenario, adopting caching provides a 2x performance improvement.
- If caching reduced latency by 90%, that would be a 10x performance improvement.



ORACLE

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

Scaling Example Revisited (continued)

System throughput is related to Little's Law, and is defined as: $T = 1 / W$
where:

W = Average time to process a request

T = Average throughput/request processes per unit of time

If it takes 2 seconds to process a request, then the throughput $T = 1 / 2 = 0.5$ requests per second.

Notes on Caching

- Can help reduce/eliminate latency
- Can increase throughput
- Does not eliminate queuing, just reduces it
- Does not solve capacity problem itself, just reduces it
- No such thing as eliminating a capacity problem
 - Just making it more manageable
 - Just reducing the delays to be acceptable



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

Notes on Caching

While caching reduces the processing time for each customer and thus increases the throughput (number of check-ins processed in a period of time) for the agent, it does not eliminate the waiting time for customers. It simply reduces the wait time. Queuing will still occur. The capacity problem has not been *solved*, just *reduced*.

Amdahl's Law describes the expected improvement to a system when a component is improved, and is expressed as:

$$I = 1 / ((1-P) + P / S), \text{ where}$$

P = the proportion of a system being improved

S = the factor of improvement for P

I = over all system improvement

In computer science terms, the purpose of caching is to provide increases in system throughput by eliminating the latency associated with consistently slow operations. More specifically, caching is a strategy designed to change the largest possible component P of a system by a factor of S (according to Amdahl's Law) to yield the largest increase in system throughput.

Notes on Caching (continued)

Applying Amdahl's Law to scaling shows that applying scaling to the wrong component can result in little savings.

Assume 10% of the check-in time was in talking to customers ($P = 0.1$).

Assume you could avoid talking to customers, improving performance by 100x ($S = 100$).

The overall improvement

$$\begin{aligned} I &= 1 / ((1 - 0.1) + 0.1/100) \\ &= 1 / (0.9 + 0.001) \\ &= 1.1 \end{aligned}$$

Avoiding communication by 100x only yields 1.1 improvement... **be careful what you optimize.**

Assume 90% of the check-in time was in confirming seat allocations ($P = 0.9$).

By introducing caching, you could improve performance by 100x ($S = 100$).

The overall improvement

$$\begin{aligned} I &= 1 / ((1 - 0.9) + 0.9/100) \\ &= 1 / (0.1 + 0.009) \\ &= 9.2 \end{aligned}$$

Thus caching may improve capacity by 9.2x, which is *much better than waiting 3.5 years for hardware.*

Scalability Strategy Comparison

Strategy	Advantages	Disadvantages
Vertical (scale up)	<ul style="list-style-type: none">• Fast to implement. It is often simply the cost of upgrading hardware.• No need to re-architect the system. Just run on the new hardware.• Most systems will respond reasonably to some form of Vertical Scaling.	<ul style="list-style-type: none">• Focus is improving hardware performance and not solving capacity problem.• May hit physical limit of achievable performance• Expensive, as doubling hardware performance is usually more than 2x cost.• The relationship between performance and cost is not linear. It may be exponential.



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

Scalability Strategy Comparison

Strategy	Advantages	Disadvantages
Horizontal (scale out)	<ul style="list-style-type: none">• Predictable capacity gains as you add more resources. The relationship between hardware cost and capacity improvements is often linear.• Predictable hardware cost.• Physical limits for expanding capacity are significantly greater than Vertical Scaling model.	<ul style="list-style-type: none">• Typically requires system architecture strategy for scale-out. If not designed for scale-out, a “re-architecture” may be required. This may be costly and time consuming.



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

Scaling Example Revisited: Caching and Scaling Horizontally

- Agent has seat allocation document at their counter
- Then open more counters (add more agents)
- Agent now has to share seat allocation amongst agents!
 - Increases check-in latency due to passing seat allocation between agents to keep it consistent
- Alternative
 - Each Agent has a copy of the seat allocations and updates master when required
- Main challenge: How to keep “copies” consistent and coherent?



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

Scaling Example Revisited: Caching and Scaling Horizontally

For a moment let's return to the flight check-in example to discuss the effects of caching when horizontal scaling.

If you change the system as discussed above to allow the agent to keep the seat allocation document at their counter (instead of in the back-office), what would the effect be on performance if you added more agents to the counter?

The challenge here is no longer the latency to travel between the check-in counter and the back-office. It's now the latency for agents to share the seat allocations between each other (as only one agent can really be modifying the seat allocation document so that it remains consistent). You can only imagine what a flight check-in counter would look like if this were how they actually operated: each agent running backwards and forwards passing the document to each other for each check in!

An alternative approach to the problem would be for each check-in agent to have their own copy of the seat allocations document so that they can make independent check-ins. The main challenge with this approach is how to keep all of the “copies” of the allocations the same, or “coherent.”

Agenda

- Performance Concepts, Caching, and Scalability
- **Introduction to Coherence**
 - Coherence*Extend
 - Coherence*Web
 - Editions



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

Welcome to Coherence

Coherence:

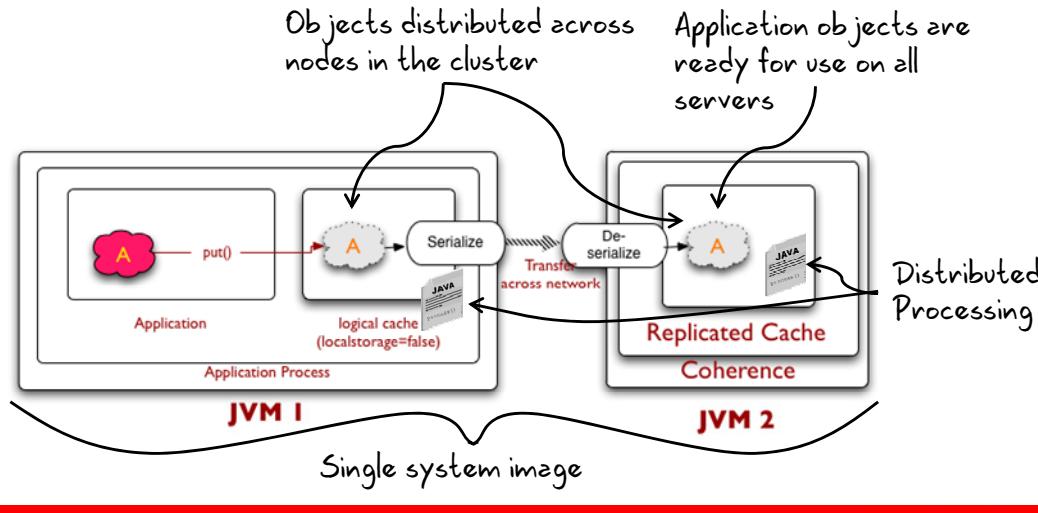
- Provides highly available and predictably horizontally scalable infrastructure for managing application data
- Ensures information managed by a collection of servers in a system is kept coherent such that the servers may operate (including fail/recover) independently of each other, providing the necessary processing capacity as required
- Ensures that caches remain coherent across the collection of servers (if required)



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

What is Coherence?

Coherence is an *in-memory* data grid which allows for the elimination of *single points of failure* and *single points of bottlenecks* in an application by distributing *application objects* and *logic* across multiple physical servers



ORACLE

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

What is Coherence?

Coherence is an *in-memory* data grid which allows for the elimination of *single points of failure* and *single points of bottlenecks* in an application by distributing *application objects* and *logic* across *multiple physical servers*. In order to understand what Coherence actually is, and how applications benefit from its features, you need to examine the above statement carefully. Breaking down the ideas:

- **In-memory:** Access to memory is fast, orders of magnitude faster than database or disk. This achieves the requirement of low latency.
- **Application Objects:** The core data elements of an application are its application objects, which are distributed across multiple servers.
- **Logic:** By distributing logic as well as data, you can move the processing to the data, rather than the data to the processing. Distributing logic allows you to remove single points of bottleneck.
- **Single points of failure:** By having many servers, all with the same data, single points of failure are removed.
- **Single system view:** Because Coherence maintains a single view on all nodes in the cluster, all applications have the same view of the data.

What is Coherence? (continued)

In general, Coherence:

- Provides highly available and predictably horizontally scalable infrastructure for managing application data
- Ensures information managed by a collection of servers in a system is kept coherent such that the said servers may operate (including fail/recover) independently of each other, providing the necessary processing capacity as required
- Ensures that caches remain coherent across the collection of servers (if required)

Coherence Principles

Requirement	Description
Reliable	<ul style="list-style-type: none">Designed for continuous operationData is fault tolerantSelf repairing. For example: replication and backup between grid elements is automatic
Scalable	<ul style="list-style-type: none">Dynamic capacity: can seamlessly grow capacityData is always preservedNo service interruption when adding capacityCost effective
Unified Application View	<ul style="list-style-type: none">All applications, regardless of language, see same dataOperations are reflected on all instancesSimple, consistent data model from all applications
Data Support	<ul style="list-style-type: none">Supports XA style transactionsRobust event modelRobust security Model
Integration	<ul style="list-style-type: none">Broad integration support including: multiple languages, multiple platforms, multiple databases (as well as JPA) and application servers



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

Coherence Features

Coherence Support a variety of features including:

- A variety of caching strategies
- A rich set of event models
- A rich set of parallel processing models
- Security support using JAAS across a Coherence Cluster
- Reporting and administrator support via JMX and associated tools
- Rich language support via Coherence*Extend
- Tight integration with application servers using Coherence*Web
- Tight backing database support, including support for the Java Persistence Architecture, Toplink, and others



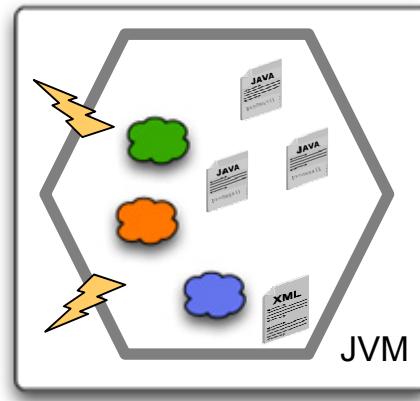
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is a Coherence Cluster Node?

A Coherence *Cluster Node*:

- Is a Java process which either:
 - Joins a cluster, or
 - Is an instance of Coherence server
- Can contain data, run processing and serve events
- Is often defined by a cache configuration
- Uses well-known ports or addresses to communicate with other nodes

Sometimes referred to as a member or cluster member



ORACLE

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

What is a Coherence Cluster Node?

A coherence node is either an instance of the Coherence class `DefaultServer`, running in a JVM, or a node may also be a Java application, which has joined a cluster. Java applications that join a cluster, but are storage disabled, are not typically referred to as a cluster node, but rather as a cluster client.

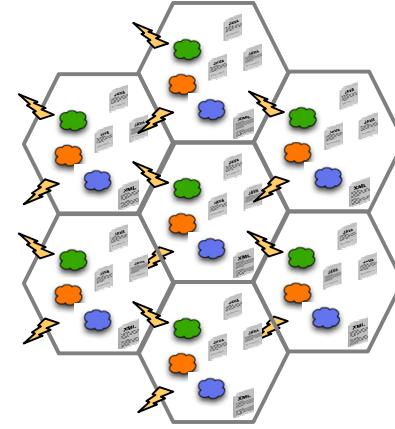
In general, a Coherence node is:

- An instance of Default Server
- Running within a JVM
- Managing data: application objects
- Managing processing: code co-located with its data
- Servicing events: sending notifications of well-known events such as `put`, `get`, or `remove`
- Defined by a set of configuration XML files, which may be overridden by defaults in the `coherence.jar` file
- Communicating using well-known addresses, ports, or both, using specially designed protocols

What is a Coherence Cluster?

A Coherence *cluster*:

- Is a group of:
 - Storage enabled cluster nodes
 - Specified by a common cluster name
- Can be expanded seamlessly by adding new cluster nodes
- Work together to perform the services of a data grid



ORACLE

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

What is a Coherence Cluster?

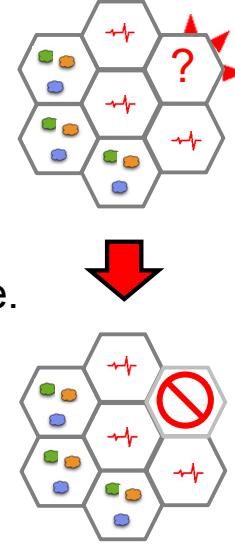
A Coherence cluster is a group of nodes all using the same cluster name.

In a cluster:

- A number of storage enabled nodes combine to form a named data grid.
- Each node specifies the same cluster name using configuration such as a command line override (-Dtangosol.coherence.cluster=name) or via a Coherence configuration file.
- Each node communicates with, and monitors the health of, other nodes.
- Additional nodes can be added or removed seamlessly without affecting the cluster itself.
- Automatically rebalances when a new node enters or leaves, regardless of the reason.

Coherence Data Grids and Fault Tolerance

- All members have the same responsibility and work together to maintain consensus.
- Cluster members maintain consensus at all times.
- Servers monitor the health of each other.
- If a server fails or is unhealthy, other servers cooperatively diagnose the state.
- The healthy servers immediately assume the responsibilities of the failed server.
- When a server fails, there is continuous operation without interruption of service or loss of data.



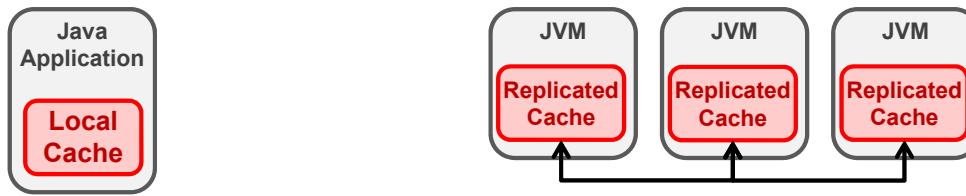
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Data Grids and Fault Tolerance

All members within a Coherence cluster work together to provide fault tolerance via the notion of *consensus*. In general, servers monitor each other's health and automatically pass health information back and forth using a protocol called Tangosol Cluster Management Protocol (TCMP). Within a cluster, each cluster node is aware of other nodes and the data they contain. If one of the servers or JVM fails, automatic recovery begins, specifically automatic re-backup and repartitioning of data. If a server fails or is unhealthy, other servers cooperatively diagnose the state. Cluster nodes work together to determine the health of any individual node. If a node appears unhealthy, other nodes diagnose and remove it from the grid. If a node fails on its own, other nodes discover that it is no longer part of the grid and act accordingly. The healthy servers immediately assume the responsibilities of the failed server, resulting in continuous operation with no interruption of service or loss of data. All this happens with no user interaction or intervention to recover servers. Even if the server that has the data fails during a request, Coherence knows where the backup is and provides the response to the application. So from the application's perspective, it does not matter if the server fails during a data request.

Coherence Cache Topology Examples

Requirement	Description
Local	<ul style="list-style-type: none"> A cache maintained on the local JVM only Not clustered, other nodes in the cluster do <i>not</i> see the data Support events and in-place processing Supports a variety of data eviction strategies
Replicated	<ul style="list-style-type: none"> A cache which replicates all data on all members of the cluster Is designed for fast read performance Provides transparent (to the application) replication to all members in the cluster Provides zero-latency access because all data is local to members



ORACLE

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

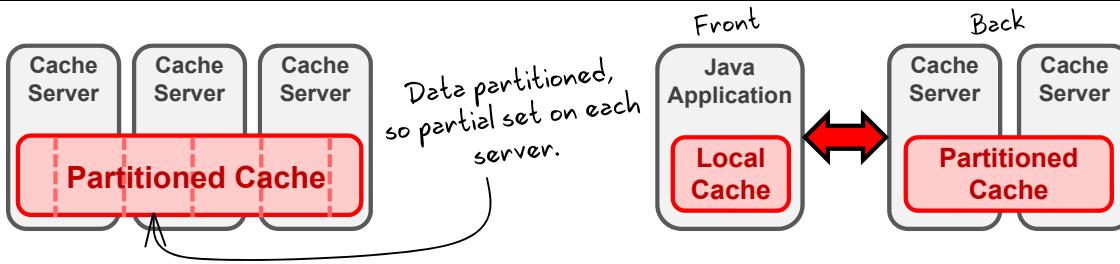
Coherence Cache Topology Examples

Coherence supports a number of cache topologies including:

- **Local Cache:** An in-memory cache that remains local to the Coherence node. Each Coherence node in the cluster contains their own copy of the local cache, but the data contained in each node is different from the data in the other nodes. The Coherence nodes are not aware of what data is contained in the other cluster members. A local cache can be backed by a data store. When a cache is backed by a data store, requests for data that does not exist in the cache are then loaded from the data store into the cache, and then returned to the calling application. The local cache implementation is thread-safe, and allows for highly concurrent use among multiple threads.
- **Replicated Cache:** A replicated cache replicates its data to all Coherence nodes. The replicated topology supports reads at in-memory speeds (100,000 reads per second or more).

Coherence Cache Topology Examples

Requirement	Description
Partitioned	<ul style="list-style-type: none"> A cache that partitions data into primary and backup copies and spreads those copies across the cluster Places backup copies on a separate physical machine for failover Provides predictable performance and scalability
Near	<ul style="list-style-type: none"> A cache representing a combination of a partitioned cache, referred to as the front tier, for very fast read access and a back tier cache containing all data. The front tier is generally a fast, size-limited cache, while the back tier is slower, but with much higher capacity.



ORACLE

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

Coherence Cache Topology Examples

Coherence supports a number of cache topologies including:

- **Partitioned Cache:** A cache that partitions the overall data set into smaller subsets, called partitions, that are spread across the entire cluster. Each node in the cluster holds a share of partitions. As a result, the partitioned cache can handle much larger data sets than a replicated cache, or what can be held in a single JVM. The partitioned cache also creates a backup copy of the partitions and stores them on another physical machine whenever possible for data resiliency. When an object is updated in a primary partition, the backup partition is updated synchronously.
- **Near Cache:** The near scheme defines a local cache that sits in front of a partitioned cache for very fast read access. The near scheme defines a two-tier cache consisting of a front tier that caches a subset of a back tier cache. The front tier is generally a fast, size-limited cache, while the back tier is slower, but with much higher capacity. A typical deployment may use a local scheme for the front tier and a distributed scheme for the back tier. The result is that a portion of a large partitioned cache is cached locally in memory allowing for very fast read access.

Event and Parallel Processing

Coherence supports two background processing models:
Events and *Parallel Processing*.

Parallel Processing includes:

- Entry Processors: Equivalent to "agents" executing services in parallel on the data in the cluster
- Invocation Services: Agents which execute on cluster members but are typically not data related

Events include:

- Map Listeners which are called **asynchronously after** data changes
- Map triggers which are called **synchronously before** data changes



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

Events and Parallel Processing

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.

Events and Parallel Processing

Coherence supports three specific event styles in addition to its parallel processing support.

The types of event support includes:

- **Map Listeners:** Asynchronous event processors which are invoked *after* an mutation occurs on a cache such as a `put`, `get` or `remove`
- **Map Trippers:** Synchronous events which occur *before* a mutation has occurred and which can change the nature and scope of the event
- **Backing Map Listeners:** Similar to map listeners, but occurring against backing maps

Queries and Filters

Coherence supports two query mechanisms:

- A filter mechanism, useful for events and event filters

MapEventFilter	Evaluates the content of a MapEvent object according to the specified criteria, includes support for event types such as insert, update and delete.
AndFilter	Provides logical AND between two filters

- A SQL-like mechanism known as the *Coherence Query Language*, useful for queries against caches

```
SELECT result-set | *
  FROM "cache-name"
    [ WHERE conditional-expression ]
```



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

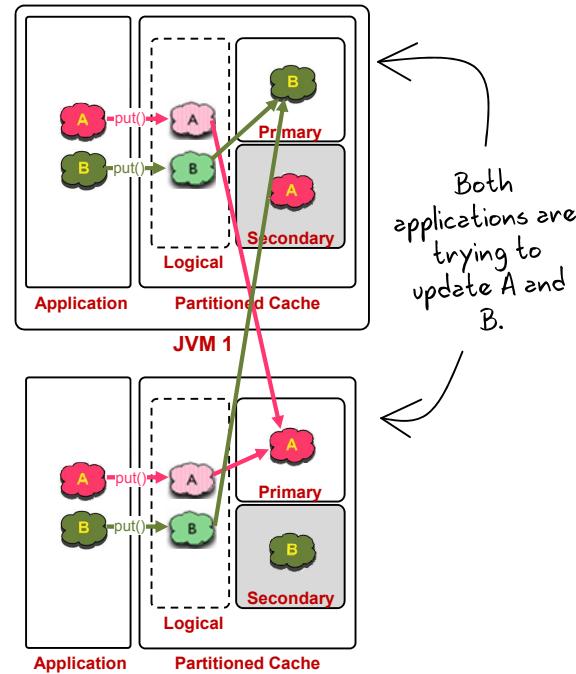
Queries and Filters

Filters extend the capability of events to select which events should be processed. Custom filters can be easily developed by implementing the Filter. The filter Evaluate method is passed an event object, which can be used to determine how to process the filter.

Caches can be queried using a mechanism similar to SQL statements, supporting both hard coded and run-time variables. While the syntax for SELECT was SHOW, SELECT, UPDATE, and DELETE statements are supported with various WHERE clauses, sorting options and other traditional SQL features.

Coherence Transactions

- Coherence provides several transaction options, each with specific transactional guarantees.
- Transactions options include:
 - Explicit locking via ConcurrentMap
 - Lock-free programming via EntryProcessors
 - Coherence Transaction Framework
 - Resource Adapter
 - Data Source integrations



ORACLE

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

Coherence Transaction Support

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 may be multiple clients, each accessing copies of data, it's important that the value of an object being read is changed atomically.

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
- **Entry Processors:** 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.
- **Transaction Framework API:** Coherence Transaction Framework API is a connection-based API that provides atomic transaction guarantees across partitions and caches even in the event of a client failure. The framework supports the use of NamedCache operations, queries, aggregation, and entry processors within the context of a TX.

Coherence Transaction Support (continued)

- **Resource Adapter:** The Coherence resource adapter leverages the Coherence Transaction Framework API and allows Coherence to participate as a resource in XA transactions that are managed by a JavaEE container's transaction manager.
- **Data Source Integration:** Write-behind and write-through strategies are used in conjunction with a database. Transactional guarantees are provided by the database, but the strategies must be used properly to ensure data consistency.

Coherence Security

The Coherence security framework:

- Manages access to cluster resources, such as:
 - Creating a new clustered cache or service
 - Joining a existing cache or service
 - Destroying an existing cache or service
- Uses standard Java security features to:
 - Authenticate
 - Grant or deny access to a protected cluster resource
 - Encrypt and decrypt communication based on caller credentials

Coherence security encompasses transport, access, proof of identity, and proof of trustworthiness.



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

Coherence Security

Security Framework in Coherence is based on a concept of Clustered Access Controller, which can be turned on (activated) by a configurable parameter or command line attribute.

The Access Controller manages access to the "clustered resources," such as clustered services and caches and controls operations that include (but not limited to) the following:

- Creating a new clustered cache or service
- Joining an existing clustered cache or service
- Destroying an existing clustered cache

The Access Controller serves three purposes:

- Grant or deny access to a protected clustered resource based on the caller's permissions
- Encrypt outgoing communications based on the caller's private credentials
- Decrypt incoming communications based on the caller's public credentials

Coherence uses a local `LoginModule` to authenticate the caller and an Access Controller on one or more cluster nodes to verify the caller's access rights.

Coherence Management

- Coherence includes facilities for managing and monitoring Coherence resources via JMX and provides a tool called Reporter
- Coherence Reporter:
 - Provides OOTB JMX reporting
 - Contains default reports to help administrators and developers manage capacity and troubleshoot
 - Supports custom reports
- Coherence editions support differing levels of reporting:
 - Standard Edition: Statistics for the local node only
 - Enterprise and Grid Editions: statistics for the entire cluster from any member

JMX administrators can monitor Coherence Cluster with any JMX-compliant tool such as JConsole, MBeans and others.



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

Coherence Management

Coherence offers management and monitoring facilities by using Java Management Extensions (JMX). A detailed set of statistics and commands is maintained in the API documentation for `com.tangosol.net.management.Registry`.

Coherence provides a JMX reporting capability tool called *Reporter*. *Reporter* provides out-of-the-box reports that help administrators and developers manage capacity and troubleshoot problems. Custom reports can also be created.

Coherence*Extend

- 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
- Coherence*Extend clients can be written in a variety languages, including:
 - Java
 - C++
 - .NET



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

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. 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*Web

- Coherence*Web:
 - Is an HTTP session state-management framework for any J2EE application server
 - Is certified with WebLogic, WebSphere, JBoss, Tomcat, SunOne, and so on
 - Replaces the existing HTTP session state replication
- The benefits of Coherence*Web for application servers:
 - Policy based sophisticated state management
 - 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.

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. Then Coherence manages those session objects with all its features and capabilities.

Coherence*Web is certified with WebLogic, Oracle OC4J, JBoss, IBM WebSphere, Apache Tomcat, and SunOne.

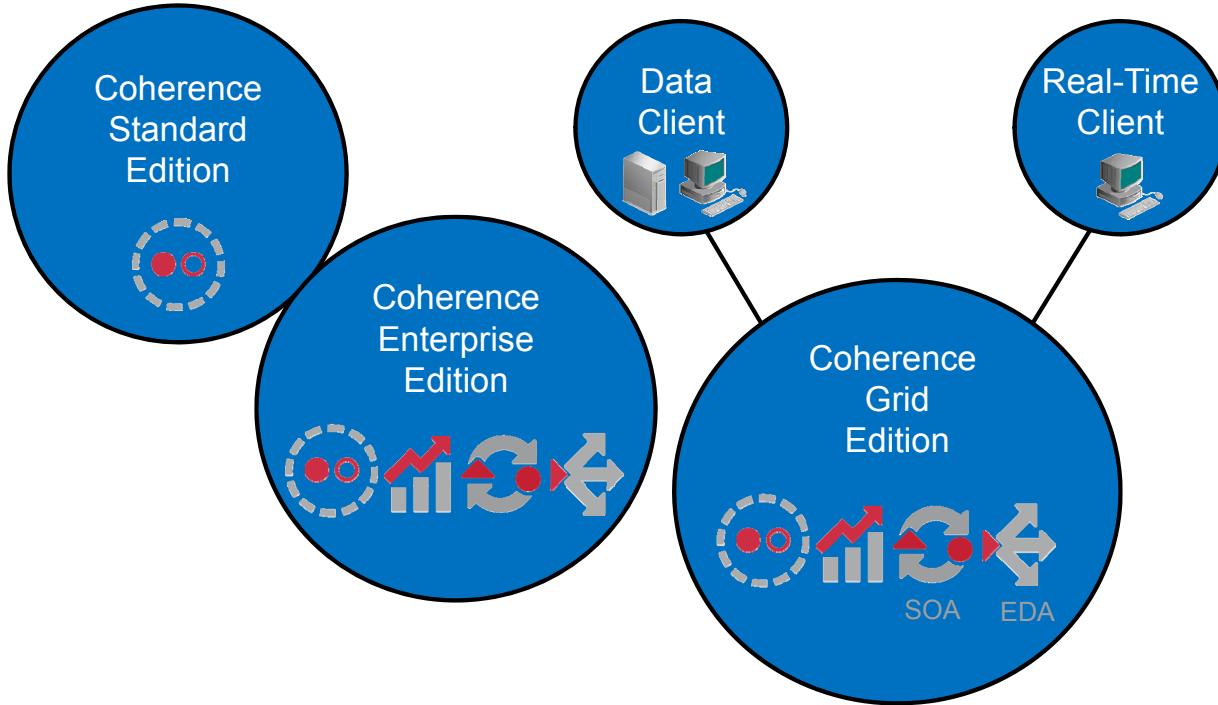
It plugs directly into Oracle WebLogic Server. Note that this capability is only to manage HTTP session objects. However, with your own application code, you can use the full capability of Coherence. If you want to have stateful EJB replication, you can use the existing application server infrastructure. The value that this 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 (continued)

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 Data Grid Solution Set



ORACLE

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

Coherence Data Grid Solution Set

There are three server solutions and two client options that define a Coherence product set. These three distinct product editions cater to different market needs. The three server solutions include the following:

- **Standard Edition**: Coherence Standard Edition provides a distributed data caching solution. You can perform simple data management with a partition cache, perform simple insert and extract operations, and provide support for simple clients.
- **Enterprise Edition**: Coherence Enterprise Edition provides an application server caching solution. In the Enterprise Edition, you can query the cache, perform parallel queries and parallel aggregations.
- **Grid Edition**: Coherence Grid Edition serves as the shared data services infrastructure platform within the data center. It includes the functionality of Standard Edition, Enterprise Edition, real-time clients, and other features.

Coherence Data Grid Solution Set (continued)

The two client options are:

- **Data client:** Data grid clients can be used anywhere and provide access to data and services on the data grid.
- **Real-time client:** Real-time clients enable Java, Java EE, and .NET applications to connect to a data grid running the Coherence Grid Edition. There is real-time access to data feeds, including near caching of data on the client as well as continuous query caching.

Real-time client provides full-client access to the information and services of the data grid, and acts as a bridge for platform and language interoperability. Real-time client is designed to also provide continuous updates of data from the back-end data grid to the client environment.

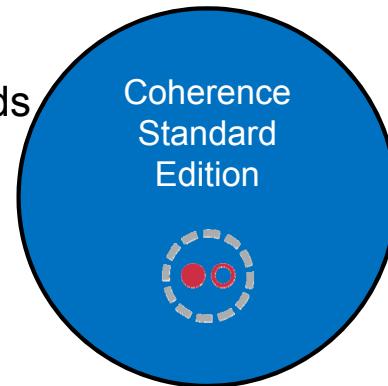
Standard Edition

Coherence Standard Edition:

- Provides entry-level application caching solution
- Brings the fault tolerance to data caching

Standard Edition benefits include:

- Manageability for embedded use
- Scalability and reliability
- Ability to access Coherence data grids



ORACLE

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

Standard Edition

Coherence Standard Edition is an entry-level product that is used by small-scale applications for distributed data caching and sharing clustered data in applications. Coherence Standard Edition:

- Enables faster access by the application to frequently accessed data
- Reduces load on shared data sources in the back-end data tier
- Establishes a manageable and scalable host for the application data cache
- Guarantees consistent data and data integrity in the middle tier for the applications' use
- Enables broad industry support as a plug-in cache for many popular application server Object Relational Mappings (ORMs) and frameworks such as Hibernate, TopLink, and SPRING
- Is used by companies that reach maximum processing or I/O capabilities in their back-end data tier due to increasing middleware processing

Examples of solution implementations include offloading reference data from the back-end data tier to memory in the middle tier, reducing back-end load, and accelerating application business logic with frequently used data in memory.

Coherence Enterprise Edition

Coherence Enterprise Edition:

- Provides mid-level application data management features such as:
 - Fault-tolerant data caching
 - Data management, transactions, analytics, and events
- Supports application server caching

Enterprise Edition benefits include:

- All the benefits of Standard Ed.
- Manageability for embedded use
- Support for queries, transactions, different types of caching, and compute grid features



ORACLE

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

Coherence Enterprise Edition

Coherence Enterprise Edition is a mid-level product that is aimed at medium to large-scale deployments. The focus of this product is on distributed data management for line-of-business applications. Organizations involved in medium to large-scale application server deployments typically use the Coherence Enterprise Edition for its deeper integration, such as persistence and transactions, with the application server infrastructure. Coherence Enterprise Edition serves as the application server caching solution. Coherence Enterprise Edition:

- Offloads and accelerates application servers
- Enables massive scaling of stateful Web applications
- Co-locates business logic and data for efficiency
- Processes transactional data in real time
- Is useful for companies that reach maximum processing or I/O capabilities in their back-end data tier
- Is useful for companies looking to significantly increase performance and/or scalability of data-intensive transaction processing or analytical applications

Example of solutions include:

- **Online travel booking:** Session state in-memory data grid
- **Online insurance broker:** In-memory transactional system

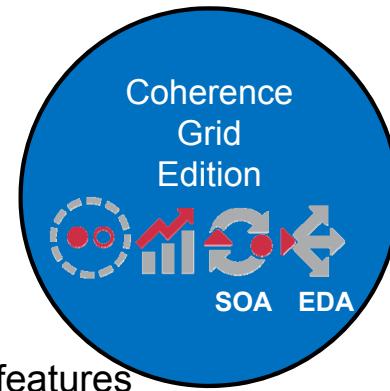
Coherence Grid Edition

Coherence Grid Edition:

- Is a shared data services infrastructure platform within the data center
- Enterprise-wide data management and integration platform

Coherence Grid Edition benefits include:

- All the benefits of Enterprise Edition
- Support for multisite data grid infrastructure
- Cross-platform real-time client support:
 - Data client
 - Real-time client
- Support for queries, transactions, write-behind caching, and compute grid features



ORACLE®

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

Coherence Grid Edition

Coherence Grid Edition, the premier edition of the product line, is a shared enterprise-wide data services platform. Organizations typically use Coherence Grid Edition as a separate tier or core piece of infrastructure in large-scale deployments that focus on real-time analytics, transaction management, event infrastructure, and sophisticated application data-caching implementations. Unlike the Standard and Enterprise Editions, the goal of Grid Edition is to have a clustered shared service that multiple clients can share for computation and transaction processing. Coherence Grid Edition serves as the shared data services infrastructure platform within the data center. Coherence Grid Edition:

- Enables a highly available, reliable, transactional in-memory system
- Enables analytics through massive parallel processing across the grid
- Enables transaction rates beyond what a database can handle
- Is useful for companies that reach maximum processing or I/O capabilities in their back-end data tier
- Is useful for companies looking to increase performance and/or scalability of data-intensive transaction processing or analytical applications

Coherence Grid Edition (continued)

- Is useful for companies looking for a high-performance infrastructure to allow application data sharing with low latency
- Is useful for companies interested in event-driven architecture (EDAs)
- Is useful for Service-Oriented Architecture (SOA) and Web 2.0 technologies

Examples of solutions include:

- **Financial Services:** Risk management (including real-time risk), algorithmic trading, order books, event feeds, matching, and reconciliation
- **Travel and Hospitality:** Profit-optimization engines (price discrimination, dynamic up-sell, utilization-driven pricing)
- **Online Gaming:** Real-time event matching

Oracle Fusion Middleware



ORACLE

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

Summary

In this lesson, you should have learned how to:

- Describe the concepts of system performance and scalability
- Describe how Coherence addresses scalability and performance
- Describe the architecture and concepts of Coherence



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

Getting Started with Oracle Coherence



ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Install Coherence
- Start, stop, and use Coherence cache servers and the cache console
- Develop Java clients that use a Coherence cache
 - Common interfaces
 - Order of events
 - Cache and application lifecycle
- Enumerate development environment considerations



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

Agenda

Coherence Installation

- Typical Installation
- Directory Structure and scripts
- Introduction to Coherence Operational Configuration

Working with Caches

Developing Coherence Applications



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

Installing Oracle Coherence Server

Oracle Coherence Server:

- Is downloadable from www.oracle.com/technology/products/coherence/index.html
- Requires JDK version 1.5 or later
- Is packaged zipped, and installed by unzipping

```
C:\> mkdir oracle  
C:\> cd oracle  
C:\> unzip coherence-java-<version>.zip
```

Typically /opt/oracle or
/opt/coherence under Linux



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

Installing Oracle Coherence Server

The installation of Oracle Coherence is a two-step process. The first step involves downloading the Coherence software itself. Coherence is provided in server and client versions. Navigate to the URL given, select the download page, and then download the **Oracle Coherence for Java** version.

Coherence is typically installed beneath an Oracle root directory such as `c:\oracle` under Windows or `/opt/oracle` under Linux. The root directory, often referred to as the Oracle Middleware directory, is commonly used to support the installation of multiple Oracle products such as Coherence, WebLogic Server, JDeveloper and others. While not required, it is considered a best practice to use these root directories.

Install Coherence by unzipping the downloaded file.

Note: There are number of other downloads for Coherence, in support of client side development for C++, .NET, and so on. None of these other downloads are covered in this specific course, but are covered in various Oracle By Example (OBE) tutorials.

Coherence Directory Structure

Directories and Files	Description
<pre> graph TD coherence[coherence] --> bin[bin] coherence --> lib[lib] coherence --> doc[doc] coherence --> examples[examples] coherence --> api[api] bin --> startCacheServer["startCacheServer.sh"] bin --> stopCacheServer["stopCacheServer.sh"] doc --> api </pre>	<p>Coherence Root Directory</p> <p>Binary directory, contains all command scripts to start cache servers, as well as other command line tool scripts</p> <p>Contains all required jar files including coherence.jar</p> <p>Product documentation in PDF format,</p> <p>Javadoc interface documentation</p> <p>Java, Configuration, and JSP samples</p>



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

Coherence Directory Structure

Once unpacked, Coherence contains four main directories containing all the elements of an installation. The directory structure is as follows, all under the coherence root directory.

- **bin:** The bin directory contains all the command scripts required to start Coherence servers, run command line tools, such as the query tool and otherwise work with Coherence from the command line.
- **lib:** The lib directory contains Coherence jar files such as the coherence.jar used by application clients. Additionally supporting jar files for other aspects of Coherence are stored here, such as support for persistence layers such as JPA and Hibernate.
- **doc:** The doc directory contains the full suite of Coherence documentation in Portable Document Format (PDF). Additionally the doc directory contains an api subdirectory, which houses all the Coherence API documentation as Javadoc.
- **examples:** The examples directory contains a variety of Java, JSP, and configuration samples and examples.

Coherence Script Basics

The Coherence `bin` directory contains the baseline script files required to start and interact with Coherence Server instances

Command	Description
<code>cache-server. [cmd sh]</code>	Starts instances of Coherence server
<code>coherence. [cmd sh]</code>	Script for interacting with, or performing operations against a cache
<code>query. [cmd sh]</code>	Starts the command line query tool
<code>*-test. [cmd sh]</code>	Test scripts for validating support for datagram and multicast.

Ensure `java_home` points to the base of a JDK install such as
D:\oracle\jdk1.6.0_20!



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

Coherence Script Basics

Coherence ships with a core set of script files for use with both Windows and Linux. The scripts can be used as provided, directly from the install, to start and interact with instances of Coherence. The scripts are:

- `cache-server`: This script is used to start instances of Coherence from the command line. You can start multiple instances of this script to create multiple cache servers for testing, and so on.
- `coherence`: This script is used to interact with a running Coherence cluster. Depending on settings, the script may also start a Coherence instance
- `query`: This script starts an interactive query session again the running Coherence cluster. Queries and the use of the query tool will be discussed in detail in later lessons.
- `datagram-test`: This script starts a test, which validates if the current installation provides the required datagram support. 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.

Coherence Script Basics (continued)

- **Multicast-test:** This script starts a test which validates if the current installation provides the required multicast support. 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.

Note: All scripts require that the `java_home` variable point to an appropriate Java JDK installation.

Starting Coherence

To start an instance of Oracle Coherence:

1. Open a command prompt to %COHERENCE_HOME%/bin
2. Ensure that JAVA_HOME is set appropriately
3. Run cache-server. [cmd | sh]

```
D:\oracle\coherence> cache-server.cmd
java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Server VM (build 16.3-b01, mixed mode)
2010-09-29 17:03:35.707/0.578 Oracle Coherence 3.6.0.0 <Info>
(thread=main, member=n/a): Loaded operational configuration from
"jar:file:/D:/Oracle/coherence/lib/coherence.jar!/tangosol-
coherence.xml"...
Oracle Coherence Version 3.6.0.0 Build 17229
Grid Edition: Development mode
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights
reserved....
```



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

Starting Coherence

Coherence instances are started via the `cache-server` script, which is provided for both Windows and Linux. The script is provided in the `bin` directory beneath where Coherence was installed. Note that the `JAVA_HOME` environment variable must be set prior to running the command.

Understanding Startup

```
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Server VM (build 16.3-b01, mixed mode)

2010-05-20 13:59:44.093/0.922 Oracle Coherence 3.6.0.0 <Info>
.
.
.
Oracle Coherence Version 3.6.0.0 Build 17229
Grid Edition Development mode
.
.
.
M
Group{Address=224.3.6.0, Port=36000, TTL=4} Matches Coherence version
to avoid clashing with older versions.

MasterMemberSet
(
    ThisMember=Member{Id=1, Timestamp=2010-05-20 14:00:01.484,
                      Address=192.168.
    OldestMember=Member{Id=1, Timestamp=2010-05-20 14:00:01.484,
                      Address=192.16
    ActualMemberSet=MemberSet{Size=1, BitSetCount=2
        Member{Id=1, Timestamp=2010-05-20 14:00:01.484,
               Address=192.168.91.134:80
    }
    RecycleMillis=1200000
    RecycleSet=MemberSet{Size=0, BitSetCount=0
    }
)
```



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

Understanding Startup

Upon starting, Coherence produces a large amount of information, most of which is covered in later lessons, but some of which is important.

The screen shot shown here highlights a number of important characteristics of this single member cluster.

First, it is running Coherence version 3.6, grid edition, in development mode.

Second, the screen shot points out which member of the cluster this instance represents, since there is only a single instance running, its member id 1.

Lastly, the instance is running with multicast address 224.3.6.0, on port 36000, with TTL 4. More about each of these settings will be discussed in later lessons.

Coherence Configuration

Coherence requires two forms of configuration:

- ***Operational* Configuration** Configured in the coherence Jar, DLL or .Lib depending on platform
 - Server instances only
 - Configuring network, security, services, and so on
 - Sensible defaults are provided out-of-the-box
- ***Cache* Configuration** Configured in XML and used by both client applications and servers
 - Definition of caches, invocation, other custom configuration

Production Systems:

- May not require *Operational* configuration
- Will require custom *Cache* configuration



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

Coherence Configuration

Coherence configuration falls into two main categories: Operational and Cache.

- Operational configuration is configuration related to the server instances. By definition such configuration has to do with the run-time environment associated with a cache. For example, what port numbers are used, how many hops multicast will use and other operational information and configuration. For a given installation, you might *not* need to perform much, if any, operational configuration.
- Cache Configuration is configuration related to definition of caches, caching schemes, backing stores, custom processing, and so on. For a given installation you *will* need to perform cache configuration.

Coherence attempts to provide appropriate defaults for configuration out-of-the-box, such configuration is bound into one of the supporting .jar, .dll, or .lib files provided with Coherence, depending on the platform being used.

Coherence Configuration

Configuration is set via a combination of:

- Command line properties
- XML files, providing defaults and overrides

File	Description
tangosol-coherence.xml	Default configuration descriptor, not normally updated. Loaded first. Packaged in coherence.jar. Defined by coherence.dtd.
tangosol-coherence-override-{mode}.xml	Mode specific override file, loaded from the classpath, contains override settings specific to either development (dev) or production (prod) mode as defined by tangosol.coherence.mode. Loaded after the defaults.
tangosol-coherence-override.xml	Optional operational override file. Loaded from the classpath and contains overrides to the standard default settings. Loaded after the mode specific overrides , and before any mode specific overrides.
coherence-cache-config.xml	The cache configuration deployment descriptor is used to specify the various types of caches which can be used within a cluster. The default is packaged in coherence.jar. Defined by cache-config.dtd.



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

Coherence Configuration (continued)

Coherence can load up to three operational configuration files, depending on mode. These configuration files define a large number of well-defined Coherence-specific operation settings such as those previously set via the command line. However, the XML versions of these values are a far more flexible mechanism.

First and foremost, the tangosol-coherence.xml descriptor is loaded. This descriptor is packaged into the coherence.jar file and contains reasonable default settings for Coherence server instances. After the defaults, anything found in the tangosol-coherence-override.xml is loaded. This file normally contains overrides which are common to both development and production instances. Once both the defaults, and common overrides are loaded, either development or production overrides are loaded. Note that in either case these files are loaded from the classpath and are specific to the current mode.

Once all operational configuration is loaded, the coherence-cache-config.xml is loaded, defining the caches themselves.

Default Configuration

- Default configuration files are in coherence.jar and /coherence/lib. They include:
 - coherence-cache-config.xml (cache-config)
 - tangosol-coherence.xml (cluster-config)
- Configuration files should be overridden rather than edited

```
...
java -Dtangosol.coherence.override=
/config/tangosol-coherence-mycluster.xml
-Dtangosol.coherence.cacheconfig=
/config/coherence-cache-config.xml
-Dtangosol.coherence.distributed.localstorage=true
-verbose:gc
-Xms1024m -Xmx1024m -server com.myproject.myApplication
...
.
```

application.cmd



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

Default Configuration

Coherence requires two XML configuration files. By default, they are inside the coherence.jar file, and inside coherence/lib directory.

The two files are:

- coherence-cache-config.xml: the cache configuration file
- tangosol-coherence.xml: the cluster configuration file.

It is recommended that you do *not* extract the two XML files from coherence.jar. For operational configuration, you should produce an override file, which sets only the configuration items you are concerned with, allowing the defaults to be retrieved from the original operational configuration embedded within the JAR.

For cache configuration, it is recommended that you create a fully custom version which describes your cache topologies, the version included in the JAR is not intended for production use.

You have an example in the slide for the Java command where the two command line options are:

- -Dtangosol.coherence.override with the location of the tangosol-coherence.xml file
- -Dtangosol.coherence.cacheconfig where you put the cache-config.xml file

It is recommended that you have the same configuration file for all members of the cluster

Command Line Properties

Coherence supports a number of overrides on the command line.

Property	Description	Values
mode	Development or production mode	dev (default), or prod
localport	Starting port number	8088 (default), 1-65535
localhost	Unicast IP Address	Localhost
ttl	Multicast Time to live	4 (default), 0-255
log.level	Logging Level	5 (default), -1 (none), 0-9 0=severe, 1=error, 2=warning, 3=info, and so on
cluster	The name of the cluster	Generated, must match for all cluster members

A few examples, all use the `tangosol.coherence.` prefix.



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

Command Line Properties

Coherence is configured normally through a set of configuration files, written in XML, and supplied on the classpath. However, many common properties can be overridden at the command line. Using the command line overrides allows developers to quickly specify some common values and avoid creating or modifying XML. The slide shows a small set of properties which are often overridden. For a complete set see:

<http://wiki.tangosol.com/display/COH33UG/Command+Line+Setting+Override+Feature>.

Note that each of these properties must be specified using the `-D` syntax and passed on the `java` command line. Additionally all of these properties are prefixed with `tangosol.coherence.`. For example, to override the `mode` parameter and run Coherence in production mode, pass

`-Dtangosole.coherence.mode=prod` to the `java` command.

Coherence Port Use

Coherence:

- Defaults to:
 - Port 8088 for TCP and UDP A common problem is port conflicts
 - Port 36000 for multicast
- Increments port by one for each additional instance on a given machine
- Can be overridden using the `tangosol.coherence.localport` property

Continued for clarity, should be on one line

```
...  
set java_opts="-Xms%memory% -Xmx%memory%  
-Dtangosol.coherence.localport=9100"  
"%java_exec%" -server -showversion "%java_opts%" -cp  
"%coherence_home%\lib\coherence.jar" com.tangosol.net.DefaultCacheServer  
%1  
...  
cache-server.cmd
```

ORACLE

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

Coherence Services

Coherence Services: ← These are what is started when we run coherence-cache!

- Are uniquely named, asynchronous threads
- Provide specific services in support of caches and other Coherence functions, such as events and invocation
- Are defined and configurable in Operational Configuration files
- May be started automatically or on demand

```
.
.
<services>
    <service id="1">
        <service-type>ReplicatedCache</service-type>
        <service-component>ReplicatedCache</service-component>
        .
        .
    </service>
.
.
</services>
```



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

Coherence Services

Coherence services are much like the operating system counterparts. The purpose of a Coherence service is to provide run-time infrastructure and services for caching, events, invocation, and any other functional area of Coherence. When running, Coherence services are basically named Java threads, each specifically tasked with providing an infrastructure for a defined functional area, on a given instance within a specific cluster.

A word of caution: developers should never change any information or configuration associated with services without first contacting a Coherence support engineer. Typically service definitions are not user-configurable components.

Service Types

Coherence service types include:

- ReplicatedCache : Maintains copies of entries on all cluster nodes
- DistributedCache : Evenly partitions entries across the cluster
- InvocationService : A non-cache service for performing operations on remote cluster nodes

This is only a partial list of service types

*Service instances are NOT caches,
but provide services FOR caches!*

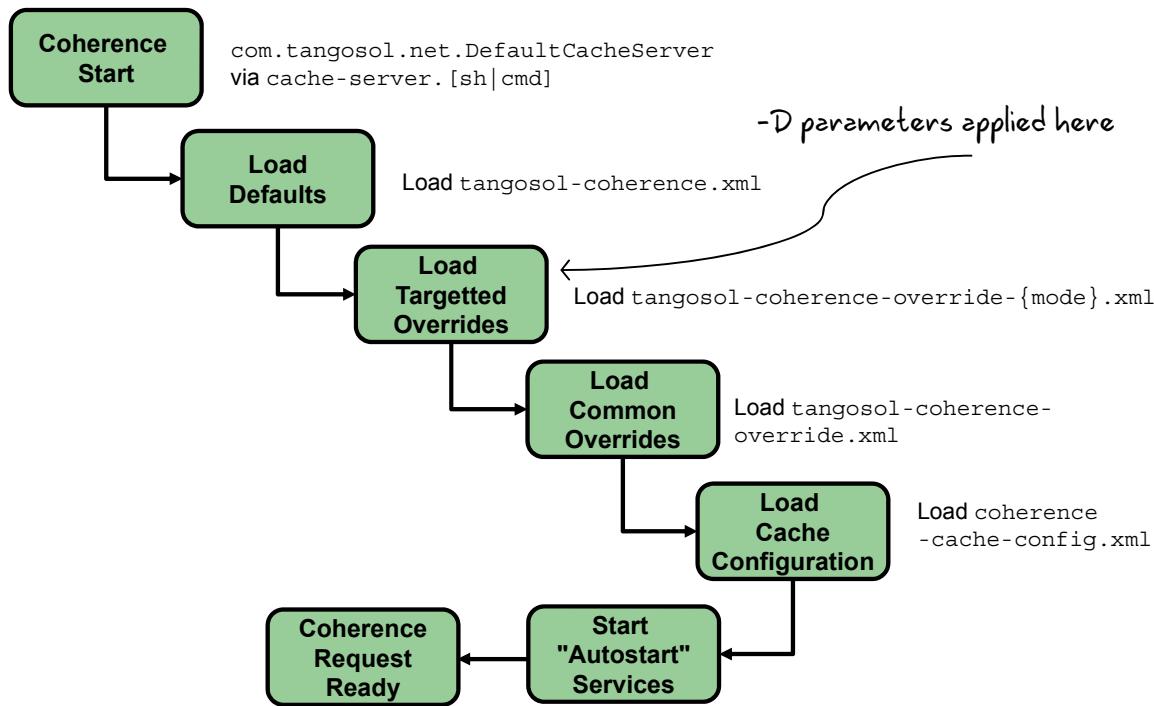
ORACLE

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

Service Types

Coherence starts services as defined in the various configuration files. These services fall into two specific categories. Cache services provide the basic caching that is expected of Coherence. Invocation services provide a mechanism for executing commands on a given cluster member. Each of these will be detailed in later lessons. However, it is important to understand that these services, based on their specific configuration, are started when the coherence-cache script is executed.

Coherence Start Sequence



ORACLE

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

Coherence Start Sequence

Note that autostart is only relevant for the DefaultCacheServer and is otherwise ignored, defaulting to lazy start.

Example tangosol-coherence-override.xml

Example override:

- Defines a local cluster via time-to-live of 0
- Specifies a 3 second wait before starting a cluster of its own
- Disable logging messages

```
<?xml version='1.0'?>
<coherence>
  <cluster-config>
    <multicast-listener>
      <time-to-live>0</time-to-live>
      <join-timeout-milliseconds>3000</join-timeout-milliseconds>
    </multicast-listener>
  </cluster-config>

  <logging-config>
    <severity-level>-1</severity-level>
  </logging-config>
</coherence>
```



tangosol-coherence-override.xml

ORACLE

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

Example tangosol-coherence-override.xml

The example on this slide defines an override file, used with both development and production, which specifies a local cluster with a 3 second wait for other members, and disables logging messages.

Quiz

Which of the following are true of Coherence starting? (Select all that apply)

- a. -D or command line parameters are loaded first
- b. Generic overrides (`tangosol-coherence-override.xml`) are loaded last
- c. Cache configuration and startup is performed last



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

Answer: a, c

When Coherence starts, it does so in the following order:

1. Load command line parameters
2. Load `tangosol-coherence.xml`
3. Load mode specific overrides. Load `tangosol-coherence-override-{mode}.xml`
4. Load generic overrides `tangosol-coherence-override.xml`
5. Load cache configuration and start services.

Agenda

Coherence Installation

Working with Caches

- What are Named Caches?
- Coherence Console
- Introduction to Coherence Cache Configuration

Developing Coherence Applications



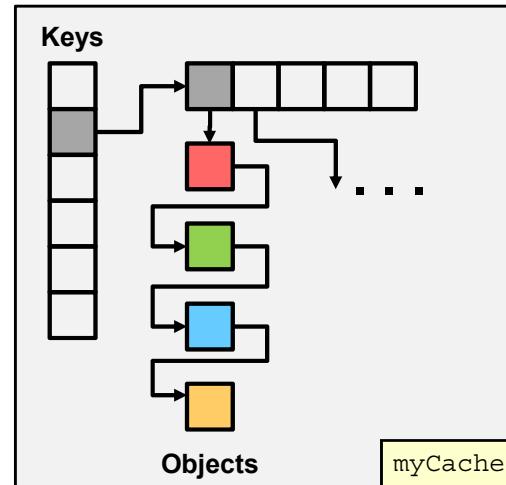
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Coherence Programming Model

The Coherence Programming Model:

- Stores key/value-based data structures
- Uses get and put style operations
- Is completely random access, but mapped to the Coherence cluster
- Works identically to a `java.util.Map` or .NET `IDictionary` data structure

Only one key can map to one object in Coherence.



ORACLE

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

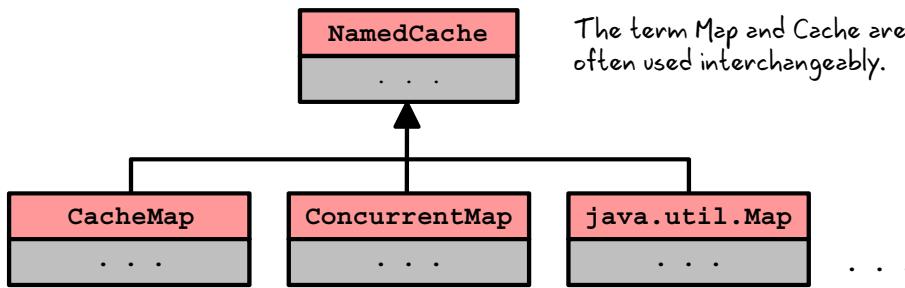
Coherence Programming Model

The Coherence programming model is a simple one from an application standpoint. Applications access a map of data, performing get and put style operations against the map. Both get and get_all return single items or multiple items based on a key. Both put and put_all perform similar operations, but store data into the map. From a single application statement, a Coherence cache is simply a map, however when multiple applications are accessing the same map, via its name, then the same data can be shared, introducing an entirely new access paradigm.

Named caches

Named Caches:

- Are Maps, that is, a set of identity/value pairs
- Hold resources shared by cluster members and managed in memory
- Are typically defined in coherence-cache-config.xml/cache-config elements



ORACLE

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

Named Caches

Named caches are one of the most common elements used by Coherence applications. Named maps implement the concepts behind the Coherence programming model and provide the shared data access Coherence application require. When applications first connect to Coherence, they request a specific cache by name. Coherence then returns an object which can be used to access the cache.

Configuring Caches

Caches Definitions:

- Are defined in `coherence-cache-config.xml`
 - Are based on `cache-config.dtd`
 - Are packaged, by default, in `coherence.jar`
 - Will be loaded from the classpath (if found)
 - Must be the same for all members of the cluster
 - Are defined in two steps:
 1. Deciding the “names” of caches
 2. Defining the **schemes** of the caches
- Wildcards may be used for matching names*
- The specification of the cache at runtime*



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

Cache Definitions:

Cache definitions define, a priori, the name type and general characteristics of a cache. The lesson titled “Configuring Coherence Caches” provides the details of cache definition, but in brief, a cache is defined to be one of the several types, each represented by one of the services previously described. Schemes are a way of grouping common characteristics of a set of caches together, and then reusing those characteristics as cache baselines. For example, a scheme might be defined for all the partitioned caches in a cluster, and then some number of actual caches defined from it. A scheme could be considered like a java class, and the cache an instance of that scheme.

In general, a scheme specifies:

- Service to “provide” the infrastructure of the cache
- Data Topology, for example, distributed, replicated, and others
- Data Management, such as on-heap, off-heap, cache store based, and others
- Eviction Policy, for example, how long should data be managed before being considered stale?
- Many other characteristics specific to a given topology or service

Minimal Cache Example

```

<cache-config>
  <cache-scheme-mapping>
    <cache-mapping>
      <cache-name>example-cache</cache-name>
      <scheme-name>example-scheme</scheme-name>
    </cache-mapping>
  </cache-scheme-mapping>
  ...
  <cache-schemes>
    <distributed-scheme>
      <scheme-name>example-scheme</scheme-name>
      <service-name>DistributedCache</service-name>
      <backing-map-scheme>
        <local-scheme>
        </local-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
  </cache-schemes>
</cache-config>

```

coherence-cache-config.xml

ORACLE

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

Minimal Cache Example

The example above is a simple cache definition. Each cache definition starts with a scheme on which it is based. The example is a distributed change scheme defined by the name `example-scheme`. This scheme is based on the distributed cache service, previously mentioned, and set to `autostart`, so that it is available when a Coherence server instance starts. A cache is then defined, based on the scheme, and using the name `example-cache`. Applications can then use the provided name to access the cache.

The default configuration file provides a number of schemes and mappings, which can be used as examples to creating caches specific to a given installation.

The Coherence Console

The Coherence console:

- Provides command line access to a Coherence cluster and its associated cache objects
- Is started using the bin/coherence.[sh|cmd] script
- Joins or starts a Coherence cluster on startup

```
C:\Oracle\coherence\bin>coherence.cmd
** Starting storage disabled console **
java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Server VM (build 16.3-b01, mixed mode)

2010-05-25 13:11:07.830/0.547 Oracle Coherence 3.6.0.0 <Info>
(thread=main,
.
.
2010-05-25 13:11:12.533/5.250 Oracle Coherence GE 3.6.0.0 <D5>
(thread=Invocation:Management, member=1): Service Management joined the
cluster with senior
service member 1

Map (?) :
```



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

The Coherence Console

The Coherence console is a development-only tool for creating and accessing caches. In fact, if run against a production instance, cache operations will throw Java exceptions. However, the console is an excellent tool for testing, checking the size of, adding and removing elements of a cache, and so on.

Remember, when adding elements to a cache instance via the console, both the key and data value are always Java strings.

Console Commands

Command	Description	Example
bye	Exit the console	bye
help	List all commands	help
cache	Create or map current the named cache	cache myTestCache, cache replicated:repCache
put	Add or replace named pair in the current cache	put "1" "value for 1"
get	Return the value associated with the provided key from the current cache	get "1"
list	List all the entries in the current cache	list
clear	Remove all entries from the current cache	clear
destroy	Remove the current cache from the cluster.	destroy



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

Quiz

True or False? During initial cache loading, the `put` operation on a NamedCache is more efficient than `putAll` because it minimizes network traffic.

- a. True
- b. False



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

Answer: b

`putAll` is more efficient than `put` for cache loading because many entries are added to the cache at once, thus avoiding multiple serialization operations and network traffic, specifically the returning of data across the network.

Practice.02.01 Overview: Installing, Configuring and Starting a Coherence Cluster

This practice covers the following topics:

- Unpackaging and installing Oracle Coherence
- Specifying command line arguments to the Coherence server start script
- Creating and using a Coherence development configuration, which creates a local cluster
- Starting a Coherence server instance in development mode using a custom configuration script



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

Practice.02.02 Overview: Using the Coherence Console

This practice covers the following topics:

- Starting and stopping the Coherence console
- Creating a cache
- Adding items to a cache
- Sizing and examining the contents of a cache



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

Agenda

Coherence Installation

Working with Caches

Developing Coherence Applications

- Coherence Application Basics
- Common Interfaces
- Bulk loading and data loaders



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

Coherence Application Basics

Coherence Applications:

- Can be written in Java, C++, or .NET
- Use a language-specific Coherence library
- Follow the same structure:
 1. Obtain a cache
 2. Perform put/get operations on the cache

```
import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;

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

SomeObject value = (SomeObject) cache.get(key);

. .
cache.put(key, value);
```

application.java



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

Coherence Application Basics

Coherence applications all have the same basic structure. First and foremost, one of the application programming-specific libraries, such as `coherence.jar`, must be associated with the application. This association in Java is via `classpath`; other languages may use other ways. An application then obtains an instance of a cache and performs traditional map style operations on the elements of the cache.

Coherence Application Overview

```

import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;

public class TestCache {
    public static void main(String[] args) {
        String key = "key";
        CacheFactory.ensureCluster();

        NamedCache cache=
            CacheFactory.getCache("myCache");
        cache.put(key,value);
        Object o = cache.get(key);

        CacheFactory.shutdown();
    }
}

```

1 Required imports, resolved via coherence.jar

2 Optional method to force cluster initialization.

3 Obtain a cache instance of name 'myCache'

4 Interact with cache

5 Shutdown the cache and release resources

TestCache.java



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

Coherence Application Overview

The anatomy of a simple Coherence application is quite straightforward:

1. Import the packages for the CacheFactory and NamedCache classes. These are resolved via the coherence.jar.
2. The optional CacheFactory.ensureCluster() method forces the initialization of the cluster which is done, by default, lazily.
Note that the ensureCluster() method is a best practice in Web Applications to ensure that all caches are initialized before a first request is made.
3. Using the cache factory, obtain a named cache, the single parameter to the getCache static method is the name of the cache to return.
4. Using the java.util.Map interface methods get and put, interact with the map as needed. In the example shown, a String is inserted using a String key object.
The lesson titled “Working with Objects” shows other mechanisms and describes the requirements for object managed in Coherence caches.
5. The optional CacheFactory.shutdown() method releases all currently allocated resources. Note that accessing the cache object after this call would raise an exception.

CacheFactory Useful Methods

```
package com.tangosol.net;  
public class CacheFactory {  
    public static NamedCache getCache(String);  
        Returns an instance of a cache based on the provided name from the loaded Cache Configuration. If the cache was previously loaded a reference is returned.  
    public static Cluster ensureCluster();  
        Optional method returning an instance of the current cluster and forcing cluster initialization, making cache requests predictably faster  
    public static void releaseCache( NamedCache);  
        Releases resources associated with the NamedCache making the reference invalid.  
    public static void log(String , int);  
        Load a message with the provided severity to the log associated with caches logging configuration.  
}
```

CacheFactory.java



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

Cache Factory Useful Methods

CacheFactory.getCache (name) performs the following steps in order:

1. Loads the Cache Configuration, if required
2. Attempts to find a previously-constructed NamedCache for the provided name and returns it if found.
3. Looks up the definition of the specified cache in the loaded Cache Configuration file
4. Uses the associated scheme to construct a suitable NamedCache instance
5. Stores a reference to the newly-constructed NamedCache
6. Returns the constructed NamedCache instance

NamedCache Interface Useful Methods

```
package com.tangosol.net;  
  
public interface NamedCache {  
  
    public void destroy();  
  
    Release and destroy this instance of NamedCache.  
  
    public Object put(Object key, Object value[, long cMillis]);  
    Insert the provided value into the cache associating it with the provided key. Optionally  
    provide the expiration of the data item.  
  
    public static void releaseCache( NamedCache );  
    Releases local resources associated with the NamedCache .  
  
    See also the super interfaces: CacheMap, ConcurrentMap, InvocableMap,  
    java.util.Map, ObservableMap, QueryMap  
  
}
```

NamedCache.java



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

NamedCache extends various interfaces

- The NamedCache class extends a variety of interfaces including:

Interface	Description
CacheMap	Provides mechanisms for adding data to a cache with explicit expirations
ConcurrentMap	Adds support for concurrency and concurrent access including methods such as <code>Lock()</code> .
InvocableMap	Adds support for executing processing and processing agents against a map.
ObservableMap	Adds the ability to registering event listeners on a map. Covered in a later lesson
QueryMap	Add the ability to perform queries, including CohQL queries against a map.
<code>java.util.Map</code>	An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.



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

Data Loading Efficiencies

A common paradigm is to prepopulate caches using code similar to:

```
 . . .
public void loadData(NamedCache cache) {
    // data from some source such as database
    for (SomeObject o: setofObjects) {
        cache.put(o.getKey(), o);      Better approach is to use putAll().
    }
} . . .
```

Which results in:

- Round trip, perhaps to multiple nodes, per `put`
- Return of data, ignored in the example, per `put` to meet the `java.util.Map` interface requirements



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

Data Loading Efficiencies

The example provided here is a perfectly reasonable one. An element will be added to the cache for each `put` and the result will be a cache populated as expected. However, this code has a number of side effects.

Side effects include:

- Round tripping and serialization overhead: For each data object inserted into the cache, the data must be deserialized, pushed across the network to other members of the cluster, including backups and potentially other caches (perhaps all in the case of a replicated cache).
- Discarded returns: The map interface

Data Loading and `putAll()`

- `putAll` can be used, rather than `put`, to avoid single element overhead using code similar to:

```
 . . .
public void loadData(NamedCache cache) {
    // data from some source such as database

    Map buffer = new HashMap(); ← Buffer all data locally in a HashMap
    int count = 0;
    for (SomeObject o: setofObjects) {
        buffer.put(o.getKey(), o);
        count++;
        if (count > 0 && count % 100) {
            cache.putAll(buffer); ← Write to cache in sets of 100
            buffer.clear();
        }
    }
    if (!buffer.isEmpty()) ← Write final stragglers
        cache.putAll(buffer);
}
. . .
```



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

Data Loading and `putAll()`

A better, more efficient, approach to data loading is to use a local non-Coherence buffer to cache data and then to send the data in reasonable size sets to Coherence to be cached. The actual size of the data set to send is dependent on the object size. Smaller objects can be sent in larger sets, and larger objects in smaller sets.

In general, the process is:

1. Create a buffer to hold the data.
2. Add data to the buffer, tracking the number of elements.
3. Write to the cache at reasonable intervals.
4. Send any straggler data to the cache to complete the operation.

Reading entries in a cache

- NamedCache methods can be used to return data or keys, both of which can be filtered
 - entrySet returns all the entries
 - keySet returns keys only

```
. . .

NamedCache cache = . . .

Set<Type> keys = cache.keySet();
. . .

Set<Map.Entry> entries = cache.entrySet();
for (Map.Entry entry: entries) {
    System.out.println("Returned '" + entry.getKey()
        + "' for '" + entry.getValue() + "'");
}
. . .
```



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

Reading entries in a cache

Entries or keys for entries can be returned all together using the `entrySet` and `keySet` methods.

The `keySet` method: Returns a set view of the keys contained in this map for entries that satisfy the criteria expressed by the filter. Unlike the `Map.keySet()` method, the set returned by this method may not be backed by the map, so changes to the set may not reflect in the map, and vice versa.

The `entrySet` methods (of which there are three): Return a set view of the entries contained in this map that satisfy the criteria expressed by the filter. Each element in the returned set is a `Map.Entry`. The various `entrySet` methods support filter, sorting, or both.

Local or Private Clusters

Coherence applications should be developed in clusters with greater than one node.

To create a private cluster:

- Create a `tangosol-coherence-override.xml`
- Define a `time-to-live` element with value 0
- Alternately, if using well-known addresses, specify
 `-Dtangosol.coherence.wka=localhost`

Can be done with `-D` parameters, but gets messy when multiple parameters are required

```
<?xml version='1.0'?>
<coherence>
  <cluster-config>
    <multicast-listener>
      <time-to-live>0</time-to-live>
    </multicast-listener>
  </cluster-config>
</coherence>
```



ORACLE

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

Local or Private Clusters

When working with Coherence, it is important to develop and test in environments which contain two or more running instances. Since the footprint of Coherence is small, it is relatively easy to achieve this. However, in typical networked environments, if two developers, working on the same subnet, both start Coherence server instances, then both will see each other's caches. While this may be the goal, more often it is not, and each should be developing in his or her own sandbox. To create a local or private cluster, all that needs to be done is to set the packet `time-to-live` to 0. This effectively limits transmission of packets to only localhost.

Local or Private Clusters (continued)

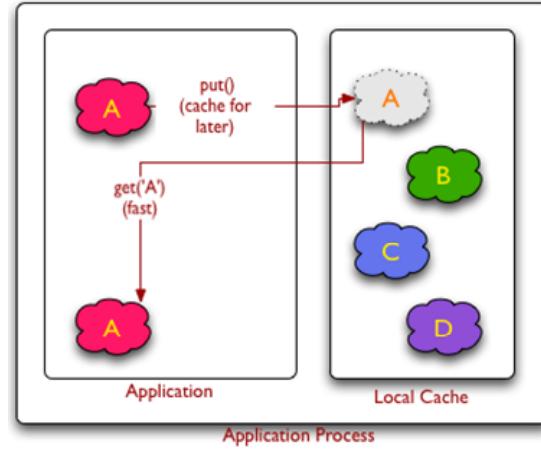
Alternately, in environments where well-known addresses are used rather than multicast, you can set the address element of the well-known-addresses element of cluster-config to localhost. For example:

```
<?xml version='1.0'?>
<coherence>
  <cluster-config>
    <unicast-listener>
      <well-known-addresses>
        <socket-address id="1">
          <address>localhost</address>
          <port>8088</port>
        </socket-address>
      </well-known-addresses>
    <unicast-listener>
  </cluster-config>
</coherence>
```

Local Storage

Local storage:

- Is Coherence cache data that is collocated in the same JVM as an application
- Can cause performance issues when servers join and leave the cluster due to repartitioning or redistribution of data
- May cause data thrashing for transient clients
- Can be disabled using
`-Dtangosol.coherence.distributed.localstorage=false`



Can also be set via an override XML file

ORACLE

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

Local Storage

Each instance of Coherence, whether it be via an application client, or via a server, can have a local cache of data. This local cache contains elements representing the named cache loaded in an application or the caches defined in the cache configuration. However often a local cache can cause performance problems or make debugging and testing more difficult.

For example, a transient client, one which starts and stops regularly, can cause a cluster to thrash as data is constantly transferred to the new member, or repartitioned to account for an additional node. Coherence also supports the concept of a remote code execution.

Processing can be run on any Coherence node. Debugging such a problem requires that the code execute on a remote instance. Disabling local storage ensures that these problems are avoided.

Note that local store only pertains to partitioned caches and is very useful for testing serialization.

Coherence and TCMP

The Tangosol Cluster Management Protocol, or TCMP, is:

- Used by cluster instances for Server Discovery, Cluster Management, Data Transmission and Service provisioning
- Completely asynchronous
- Uses a combination of multicast, unicast and TCP

Protocol	Usage
Multicast	Cluster discovery: Is there a cluster already running that a new member can join? Cluster heartbeat: The most senior member in the cluster issues a periodic heartbeat via multicast; the rate is configurable and defaults to once per second. Message delivery: Messages that need to be delivered to multiple cluster members will often be sent via multicast, instead of unicasting the message one time to each member.
unicast	Direct point-to-point communication, including messages, asynchronous acknowledgments asynchronous negative acknowledgments and peer-to-peer heartbeats. Under some circumstances, a message may be sent via unicast even if the message is directed to multiple members.



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

Coherence and TCMP

TCMP uses a combination of UDP/IP multicast, UDP/IP unicast and TCP/IP as follows:

Multicast

- Cluster discovery: Is there a cluster already running that a new member can join?
- Cluster heartbeat: The most senior member in the cluster issues a periodic heartbeat via multicast. The rate is configurable and defaults to once per second.
- Message delivery: Messages that need to be delivered to multiple cluster members will often be sent via multicast, instead of unicasting the message one time to each member.

Unicast

- Direct member-to-member ("point-to-point") communication, including messages, asynchronous acknowledgments (ACKs), asynchronous negative acknowledgments (NACKs) and peer-to-peer heartbeats. Under some circumstances, a message may be sent via unicast even if the message is directed to multiple members. This is done to shape traffic flow and to reduce CPU load in very large clusters.

Coherence and TCMP (continued)

TCP

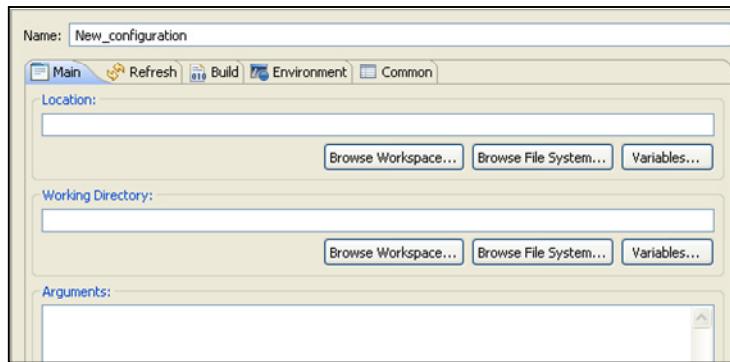
- An optional TCP/IP ring is used as an additional "death detection" mechanism, to differentiate between actual node failure and an unresponsive node, such as when a JVM conducts a full garbage collection (GC).
- TCP/IP is not used as a data transfer mechanism due to the intrinsic overhead of the protocol and its synchronous nature.

Running External Applications in Eclipse

Eclipse can run external applications using **External Tools Configurations**.

External Tools Configurations:

- **Specify a location:** The application to run
- **Working directory:** Where to run
- **Arguments:** Passed to the application



ORACLE

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

Running External Applications in Eclipse

Eclipse is capable of running a variety of applications directly within the IDE itself via a facility known as External Tools Configurations.

External tools configurations specify all the components of an application as you would expect from the command line.

In general, each external configuration has:

- Name: A symbolic name for the configuration
- Location: The location term is somewhat misleading, and actually specifies the fully qualified name of the application to run.
- Working directory: The working directory represents the directory where the application is run.
- Arguments: A set of command line arguments passed to the application when it is started.

Configurations can take advantage of Eclipse variables. Variables can be inserted directly into the set of arguments used for the application by using a `${variable_name}` syntax.

Launching Cache Servers from Eclipse

The default cache server can be launched from within Eclipse, taking advantage of the Eclipse environment

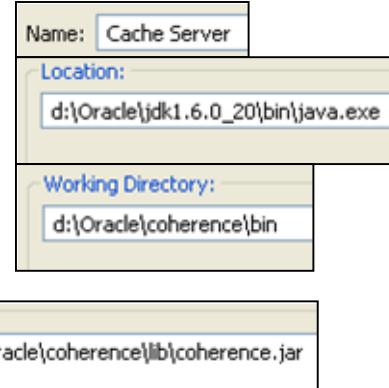
To run DefaultCacheServer, either:



Create a new External Configuration using

Run > External Tools > External Tools Configurations, or

1. Provide an appropriate name
2. Specify java.exe as the location
3. Specify \coherence\bin as the working directory
4. Provide appropriate arguments



ORACLE

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

Launching Cache Servers from Eclipse

To launch a cache server from Eclipse, an external tool must be specified which:

- Uses the java application
- Has a classpath pointing to coherence.jar (at a minimum)
- Specifies the com.tangosol.net.DefaultCacheServer class.

Arguments for running DefaultCacheServer

When running a cache server via Eclipse, the argument set controls how the class will execute.

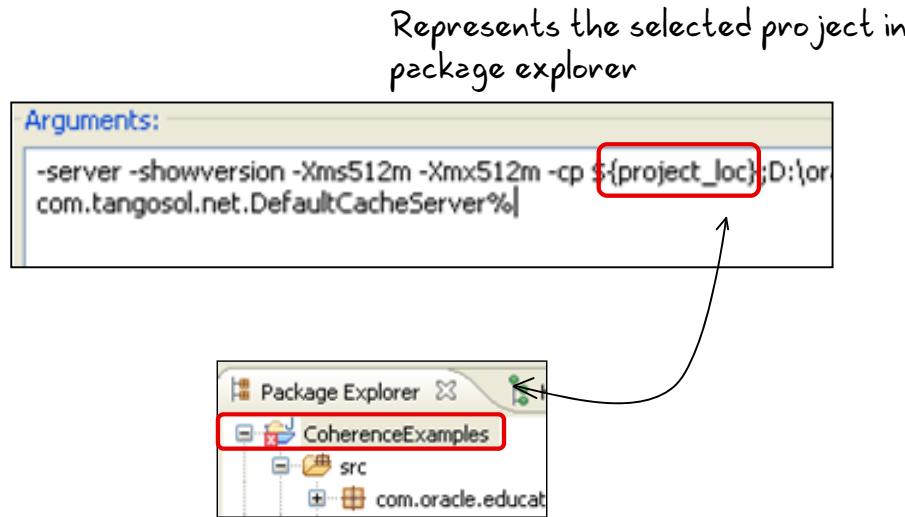
Argument	Meaning
-server	Run the Java executable using the the Server VM
-showversion	Show the current version of the Java VM
-Xms	Specify the initial java heap size
-Xmx	Specify the maximum heap size
-cp	Specify the classpath. This must at a minimum include coherence.jar
com.tangosol.net.DefaultCacheServer	The class to run, in this case DefaultCacheServer



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

Eclipse Environment Variables

Eclipse supports a number of variables, which can be passed to external applications.



ORACLE

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

Eclipse Environment Variables

Eclipse supports a large number of environment variables, which can be passed to scripts and external applications.

When running a Coherence cache server, it is often convenient to run using a specific set of configuration files, which are specified by classpath.

These configuration files, such as a tangosol default file, can be placed into the root of a application and then specified on the command line via the \${project_loc} variable.

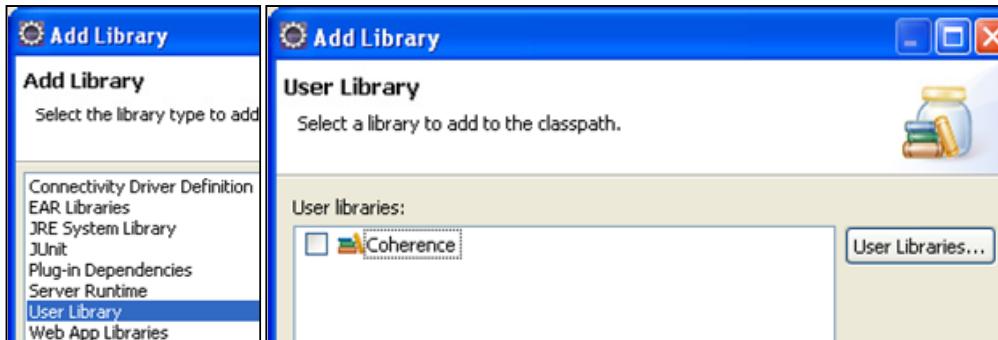
When a specific project is selected and an external tool is run, the execution is in the context of the project and will pick up its environment. In this way, many environments can be configured and run selectively.

Eclipse Libraries

Eclipse supports the concept of User Libraries.

User Libraries:

- Are collections of .jar files and associated Javadoc
- Are workspace specific
- Simplify development by allowing users to add .jar files to projects in sets



ORACLE

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

Eclipse Libraries

Eclipse supports the concept of a user library.

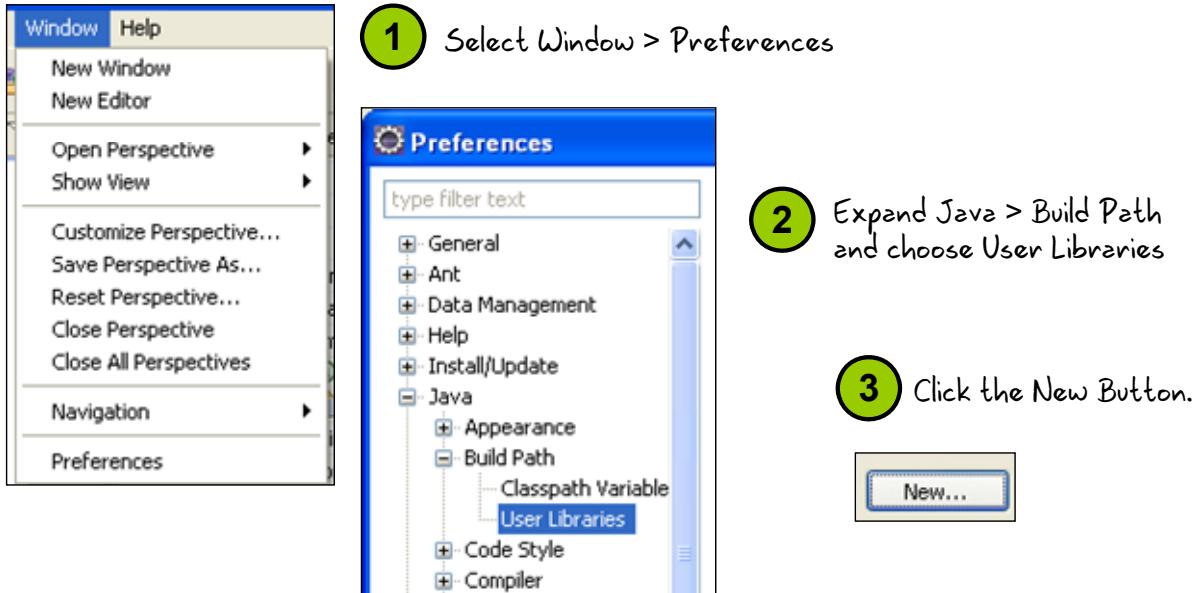
User Libraries are collections of .jar files, and other artifacts which can be added to a project as a whole. Libraries simplify development by grouping sets of files together which represent some common functionality. For example, if all projects require a core set of .jar files such as coherence.jar, apache commons, log4j, and so on, these .jar files can be combined into a single library and then added to all projects which require them.

Once defined, a library can be added to a project by:

1. Choosing the project
2. Select its properties
3. Selecting Java Build Path
4. Choosing Add Library, and then navigating to the specific library to add

Defining Libraries

To define a library:



ORACLE

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

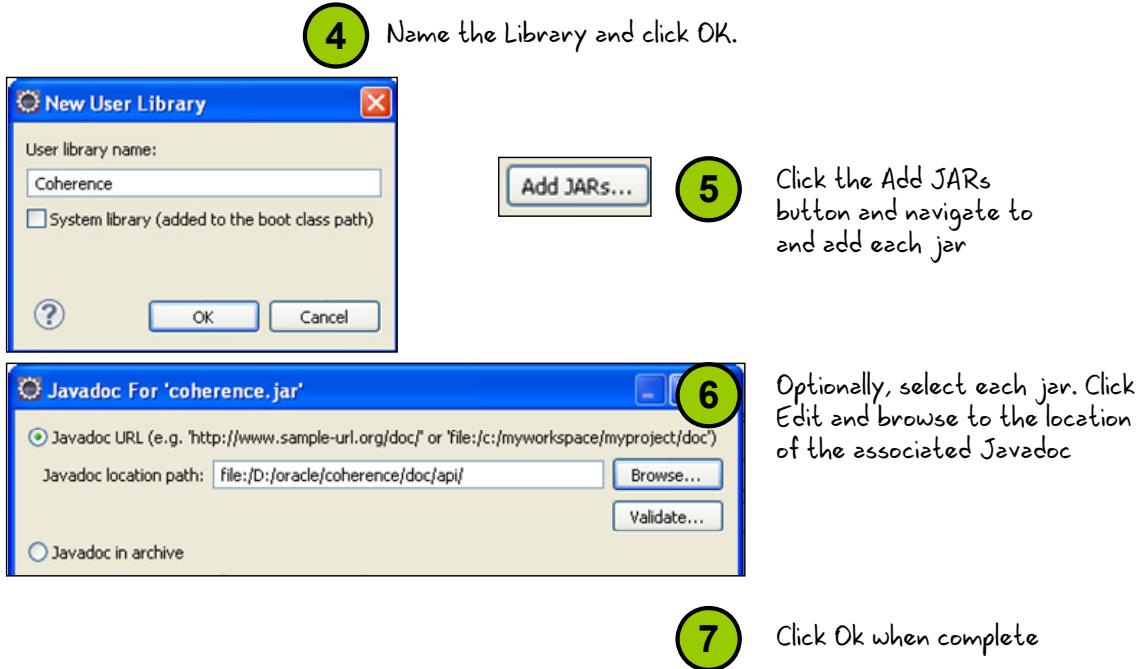
Defining Libraries

Eclipse Libraries are sets of .jar files and classpath entries grouped together for ease of management.

To create a library:

1. Select Window > Preferences
2. Expand Java > Build Path and select User Libraries
3. Click New and add the appropriate .jars or directories.

Defining Libraries



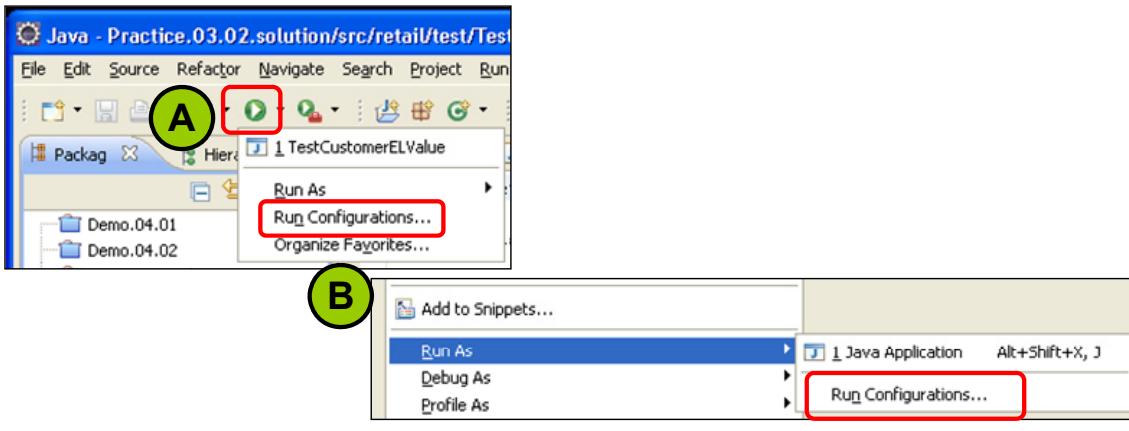
ORACLE

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

Eclipse Run Configurations

Eclipse Run Configurations:

- Are project-specific definitions
- Specify environment for running applications
- May specify application and JVM arguments
- Support custom classpath settings



ORACLE

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

Eclipse Run Configurations

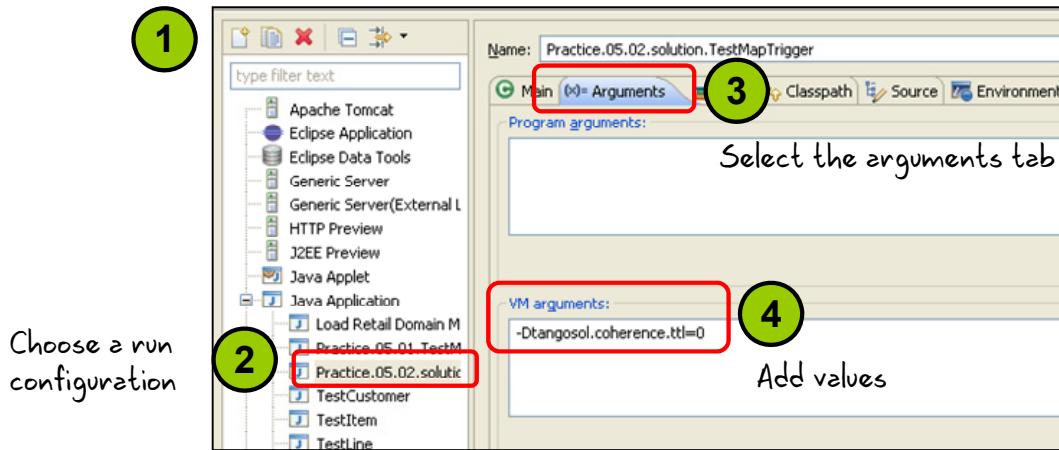
Eclipse supports a feature known as a run configuration. Run configurations are custom environments for specifying the details for running an application. Support is provided for details such as parameters, both to the application and JVM, classpath, environment particular to the running instance, and other information.

Run configurations are stored by project, but all the run configurations for all projects are displayed at once. There are two ways to access run configurations:

1. Right-click in a Java code and choose Run As > Run Configurations. The list of known configurations is shown.
2. Select the white arrow in the green circle from the tool bar menu. Then choose Run Configurations.

Run Configurations and Coherence

- Run configurations can be used to specify application arguments
- Application arguments are important with Coherence to minimize development environment interaction



ORACLE

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

Run Configurations and Coherence

Run configurations are very important to Coherence in development because of how Coherence clusters are formed. To eliminate machine-to-machine interaction, it is suggested that most applications specify `-Dtangosol.coherence.ttl=0` to the VM.

To specify a parameter:

1. Open Run Configurations (not shown).
2. Choose a specific run configuration (remember these are stored by project).
3. Select the Arguments tab.
4. Specify appropriate VM arguments.

Summary

In this lesson, you should have learned how to:

- Install Oracle Coherence
- Start, stop and use Coherence cache servers and the cache console
- Develop Java clients that use a Coherence cache
- Enumerate development environment considerations



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

Practice.02.03 Overview: Coherence “Hello World”

This practice covers the following topics:

- Creating an Eclipse project
- Adding the Coherence Jar and associated API documentation to a project
- Adding a Java class to a project
- Interacting with a Coherence cache using:
 - com.tangosol.net.CacheFactory
 - com.tangosol.net.NamedCache



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

3

Working with Objects

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Implement Java objects that are stored in Coherence caches using `Serializable`, `ExternalizableLite`, and Portable Object Format techniques
- Describe the purpose of deserialized caches
- List which cache types are not serialized



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

Agenda

Developing Objects

- Minimum Requirements
- Objects and Identity
- Aggregate and value/embedded objects
- Data Affinity concepts

Improving Performance Using Serialization



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

Java Objects and Coherence

A Coherence Java Object:

- Is a Plain Old Java Object (POJO)
- Must support some form of serializable
- May support Comparable
- Should provide support for identity

```
public class AirPort implements Serializable, Comparable {  
  
    private int id;  
    private String code;  
    private String city;  
    private String name;  
    private String country; }  
  
    public Airport() { }  
    . . .  
}
```

All must be serializable,
child classes must also
be serializable!

AirPort.java

ORACLE

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

Java Objects and Coherence

Coherence objects, that is Java objects which can be inserted into a cache, have a single hard requirement: they must be serializable in some way. Because cached objects are sent across the network, they must be serializable in one form or another. The standard `java.io.Serializable` interface is technically all that is required, however, in reality several other requirements exist which make Coherence objects usable, such as:

- `java.lang.Comparable`: An interface which imposes a total order on the objects of the class it implements. In essence, `Comparable` allows you to sort objects against each other, and is a requirement for certain SQL-like queries, which include `ORDER BY` clauses.
- Identity: Identity is not a strict requirement, however, much like any database or persistent store, caches are storage. Storing an object that cannot be uniquely returned is of little value. Typically, objects can be identified by simple classes such as `String`, `Integer`, and so on, but may be identified by more complex objects.
- `toString`: A method which prints an object's content in a human-friendly manner. `toString()` is purely a convenience.

AirPort Object Example

```

public class AirPort implements Serializable, Comparable {
    private static final long serialVersionUID = . . . ;
    private int id;
    .
    .
    private String country;
    public AirPort(int id, String code,
                  String city, String name, String country) {
        this.id = id;
    }
    .
    .
    public AirPort() { }

    public String getCode() { . . . }
    public void setCode(String code) { . . . }
    public String getCity() { . . . }
    public void setCity(String city) { . . . }
    public String getName() { . . . }
    public void setName(String name) { . . . }
    public String getCountry() { . . . }
    public void setCountry(String country) { . . . }
    .
    .
}

```

Must implement some form of serialization.

One or more constructors.
Note no-arg constructor

'Standard' JavaBean getter/setters

AirPort.java

ORACLE

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

AirPort Object Example

The AirPort class shown here, defines all the requirements of a baseline JavaBean which can be used with Coherence. It has the following characteristics:

1. **Serializable:** While inefficient compared to other serialization mechanisms, the Java serializable interface meets the requirements for serialization.
2. **Constructors:** Has as a minimum a no-argument constructor, other constructors are ignored. Note that it is a requirement for loading instance of this class by Spring.
3. **Setters and Getters:** Has standard JavaBean getter/setter methods. These can be generated by both Eclipse and JDeveloper.

AirPort Object Example

```
 . . .
public int compareTo(Object o) {
    AirPort ap = (AirPort)o;
    return name.compareTo(ap.getName());
}

public String toString() {
    String sb = new String();
    sb.append("Airport: \n");
    sb.append("\t").append("ID:").append(id).append("\n");
    . .
    return sb.toString();
}
```

Optional Comparable support.

AirPort.java



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

AirPort Object Example (continued)

The remainder of this example is optional, but recommended.

The class implements the Comparable interface, which requires a compareTo method and a toString method.

Objects and Identity

Objects become *entities* when they are assigned an *identity*.

Identity:

- Is used to uniquely identify one object from another
- Can be an attribute of an object, known as a *natural* identity
- Can be assigned by the system, known as a *surrogate* identity
- Must not change over the lifetime of the object

```
public class AirPort implements . . . {  
    private int id;  
    . . .  
    public AirPort(int id, . . . ) { this.id = id; }  
    . . .  
    public int getId() { . . . }  
}
```

AirPort.java

ORACLE

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

Objects and Identity

Objects in Coherence are persistent entities. What this means is that there must be a way to identify an object so it can be returned when needed. Typically an identity is some sort of generated value, which is uniquely assigned to each entity at creation time. In databases, you can typically `INSERT` a new row into a table, conceptually similar to adding a object to a Coherence cache, and on insert, some sort of ID generator is used to assign the new row a unique ID. However, in Coherence the identity must be assigned up front, before the object is inserted into the cache. It is the responsibility of the developer to assign a unique identity to each element, rather than the database.

There are two types of identity: natural and surrogate. A natural identity is one which is part of the object, a combination of one or more attributes which are would be part of the object no matter how it was stored. A surrogate ID is one which exists for no other reason than to make the entity unique.

The final requirement of identity is that it must not change over time. Since Coherence Caches use map semantics, the values must be 1:1 mapped between key and object, any new object, inserted with the same key, replaces the old object.

Identity Types

Identity:

- May be based on a single primitive type, or their wrapper classes
- May be based on a class composed of the above
- Must be serializable
- Must implement `equals()` and `hashCode()`

```
public class LoggingKey implements Serializable {  
    private Long severity;  
    private Timestamp time;  
    public LoggingKey (long severity) {  
        this.severity = severity;  
        this.time = new Timestamp(System.currentTimeMillis());  
    }  
    ... // hashCode and equal omitted  
}
```

LoggingKey.java



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

Identity Types

Most developers that are familiar with databases, are familiar with the concept of a unique identifier using primitive types such as `integer` or `long`. However Coherence happily works with composite keys built from some number of serializable data elements.

When implementing `equals()` and `hashCode()`, always ensure these methods use all fields for testing, not a subset. Keys are used extensively in Coherence for partitioning of data and poorly coded `hashCode` and `equals` methods can cause unexpected behaviors.

Implementing Entities

Classes can be forced to provide identity using an interface

```
public interface CoherenceEntity<T> {
    T getId();
}
```

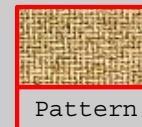
Typed interface with required get method.

```
public class AirPort implements CoherenceEntity<Long> . . . {
    . . .
    private Long id;
    public Long getId() { return id; }
    . . .
}
```

Pattern suggests:

1. define an interface with a typed getId() method
2. Classes implement interface

Pattern forces all classes to provide identity!



Pattern

ORACLE

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

Implementing Entities

While not strictly a requirement, it's helpful to force entities object to provide an identity via a simple interface such as the one shown. In this example a template interface requires the implementation of the `getId()` method which returns the templated type `<T>`.

Identity generation

Key generation:

- Is supported in most databases, but not explicitly with Coherence
- Can be cumbersome for multipart keys
- Can be simulated using sequence generators

Sequence generators must be:

- Accessible: Easy to use from multiple clients across multiple cluster nodes
- Concurrent: Must support concurrent access and never return duplicate identifiers
- Performant: Support returning the next sequence number quickly
- Reliable: Never returning the same sequence ID twice, even on system failure.



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

Identity generation

One of the more difficult problems for new Coherence developers is the lack of support for sequence number generation. Developers from a JPA or database background take sequence generation for granted, expecting to be able to insert rows or objects into the store without assigning them an explicit identifier.

For those objects which require a key, and which have no natural identifier, a sequence generator is perfect. However, usable sequence generators need to have four specific characteristics:

- **Accessible:** The sequence generator must be easy to use and accessible from all nodes in the cluster. This means an easy to use API and a .jar file provided to all instances of the cluster.
- **Provide for concurrency:** Since any number of threads must access the sequence generator, it must be thread-safe.
- **Performant:** Sequence generators have the possibility of being used constantly, so they must perform well.
- **Reliable:** In the event of a system crash or other problem, the sequence generator should continue to function, and provide appropriate sequence values.

Using Sequence Generators

```
import com.seovic.coherence.identity.IdentityGenerator;
import com.seovic.coherence.identity.sequence.SequenceGenerator;

public class AirPort implements Serializable, Comparable {
    .
    .
    private static IdentityGenerator<Long> idGen =
        new SequenceGenerator("airport.id", 20);
    private long id;
    .
    public AirPort(String code, . . . , String country) {
        this.id = idGen.generateIdentity();
        this.code = code;
        this.city = city;
        this.name = name;
        this.country = country;
    }
}
```

AirPort.java

Include appropriate imports.

Specify a identity generator.

*Use the identity generator to create an id,
note lack of id in constructor.*

This example is based on <http://code.google.com/p/coherence-tools>



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

Using Sequence Generators

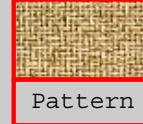
The example given uses the freely available sequence generator written by Alek Seovic, however, the Coherence incubator project also contains support for sequence generators. Students are directed to Coherence Common JAR which contains the `com.oracle.coherence.common.sequencegenerators` package. This package contains a set of interfaces and classes to simplify the creation of sequence numbers, either locally in a Java Virtual Machine, or across a cluster.

Sequence Generators and Entities

```

public class AirPort implements CoherenceEntity<Long> . . . {
    . . .
    private Long id;
    . . .
    public Long getId() { return id; }
    . . .
    private static IdentityGenerator<Long> idGen = . . . ;
    public static AirPort create( String Code . . . ) {
        return new AirPort(Code . . . );
    }
    private AirPort( String code . . . ) {
        this.id = idGen.generateIdentity();
        this.code = code;
        this.city = city;
        this.name = name;
        this.country = country;
    }
    . . .
}

```



Pattern

Pattern forces use of
create and identity
generator!

Pattern suggests:
 1. define a sequence generator
 2. Write a 'create' method,
 which uses the c'tor
 3. Make the c'tor private
 Forcing the use of create

AirPort.java

ORACLE

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

Sequence Generators and Entities

Combining a sequence generator and the prior example is simple enough, but it needs to ensure that objects cannot be constructed without a generated sequence number. In the example shown, the constructor is now `private`, and a `static create` method was added, which uses the constructor to create an instance, ensuring that the sequence generator is used to create the instance.

Quiz

True or False? Objects inserted into a NamedCache are required to provide an explicit identity method such as `getId()`:

- a. True
- b. False



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

Answer: b

False, when inserted the object requires a key which represents its identity, however the object itself does not need to provide it explicitly, even though it should.

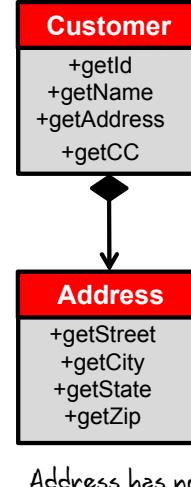
Aggregate Objects

Aggregate entities are:

- Composite entities
- Composed of a root entity and one or more dependent or weak entities
- Typically stored in the same cache

Dependent entities typically have no meaning without their owner. Examples include:

- Order lines, which have no meaning without their order
- Accounts, which have no meaning without their owner
- Phone call details, which have no meaning without their associated phone line



Address has no meaning without the associated customer!

ORACLE

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

Aggregate Objects

Aggregate objects are those objects which contain one root or parent object and one or more associated or dependent objects. A common example might include banking accounts which have a set of associated transactions. The transactions have no meaning without their associated account record. Similarly, an order and its associated lines might be considered dependent objects.

In Coherence, the first reaction is to separate these two objects into separate caches, using the same logic used in databases where they would be in separate tables, with a foreign key association. However such an approach may cause consistency concerns with updating both objects in the scope of a transaction. A second concern, and a reason to separate the two entities into different caches, might be sizing or differing caching retention policies. It may make sense to cache the last 90 days of banking transactions only, but not all bank accounts.

Value Objects

Value objects:

- Have no identity outside their containing object
- Represent complex or hierarchical values within an entity
- Are often considered attributes of an entity
- Are typically immutable
- Must implement `equals()` and `hashCode()` when being used as in an indexed cache

More on indexes in a later lesson



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

Data Affinity

Data affinity describes the concept where one object is in some way related to another.

Coherence data affinity:

- Describes the concept of ensuring that a group of related entities are contained within a single cache partition
- Improves query performance ← *Queries don't need to span multiple caches*
- Improves concurrency ← *Transactions are managed locally, avoiding synchronization issues*
- Is supported by several interfaces, including:
 - KeyAssociation *Details in a later lesson*
 - KeyAssociator



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

Data Affinity

Data affinity is an important concept in Coherence, and rounds out any discussion about object associations. Data affinity says that this object is related to some other object in some way. For example: order lines are related to the order. Coherence performance can be improved or take advantage of data affinity via several interfaces. Entities that are related to other entities can use the KeyAssociation and KeyAssociator interfaces to tell Coherence about the specific entities they are related to.

Properties of Relationships

Relationships include a number of core properties.

- **Cardinality:** What is the relationship between two objects?
Typically expressed as 1:1, 1:Many, Many:1, Many:Many
- **Dependency:** Does one object depend on or own another? Typically thought of as:
 - *Association:* Entities in a relationship exist independently of the relationship.
 - *Composition:* Entities on one side of the relationship exist only if the relationship exists.
- **Direction:** Which elements reference which in a relationship? Typically thought of as:
 - Unidirectional or one way
 - Bidirectional or two way



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

Properties of Relationships

In order to understand how two entities relate to each other, it is important to understand what the core properties of relationships are.

First is cardinality. Cardinality defines the “how many” are involved between two entities. For example, a phone number applies to one phone, but a person may have many phones. So the relationship of phone to phone number is 1:1, but the relationship of person to phone is 1:M.

Next is dependency. Can one object exist without the other? Certainly phones may exist without an owner, in which case the relationship is an association. However, in many cases, the relationship cannot exist without one or the other element. For example, you cannot have children without parents.

Finally, there is direction, which defines how objects are referenced. A parent-child relationship is bidirectional. Parents have children; children have parents. Similarly, people have addresses, but addresses do not have people. Such a relationship is unidirectional or one way.

Modeling Relationships in Java

Relationships in RDBMS:

- Are implicitly bidirectional
- Can use keys to find object owners
- Are modeled using foreign keys

PERSONS
PID
ADDRID
...

ADDRESSES
ADDRID
INTEGER
...

Relationships in Java:

- Are implicitly unidirectional
- Can be made bidirectional using two unidirectional relationships

Objects do not know who owns them!

```
public class Address {
    int addrid;
    ...
}

public class Person
    Address address;
    ...
}
```

ORACLE

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

Modeling Relationships in Java

Bidirectional relationships are modeled in Java by two unidirectional relationships.

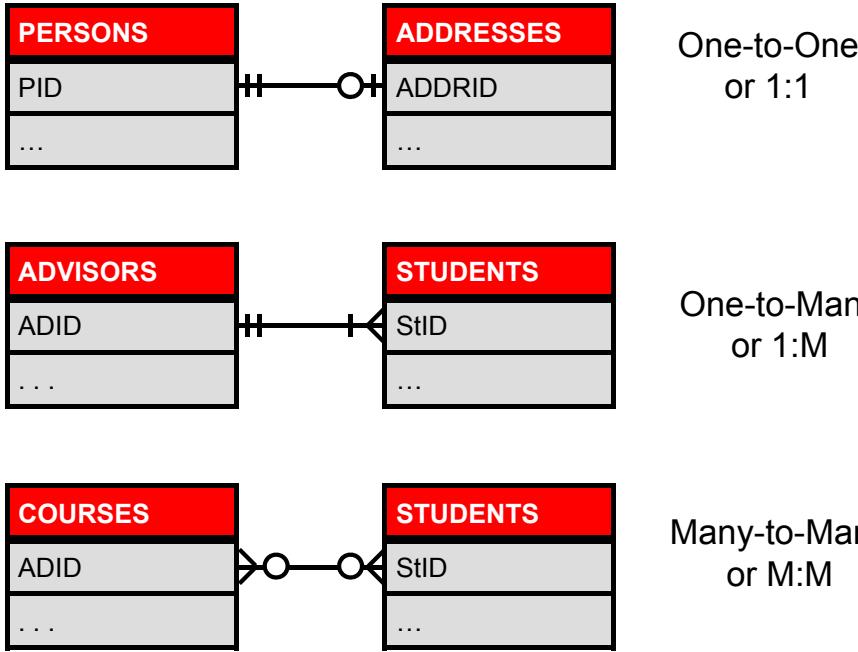
Two references in memory model the single value in the relational database.

- On read, JPA updates both values in memory based on one value in the database.
- On write, JPA uses the “owning side” of the relationship in memory to update the database.

The application should manage the bidirectional relationship in Java.

The application’s burden to manage bidirectional relationships entails design choices. A bidirectional one-to-many relationship is typically modeled as a collection with a back reference. For example, the invoice has a collection of line items, and each line item has a reference to its invoice. It would be a dubious design choice to allow code outside the model to add directly to the collection of line items, because this code may not observe the constraint to update the line item’s back reference to the invoice. Failure to maintain the back reference means that the relationship is lost during persistence, because in the case of a bidirectional one-to-many relationship, the back reference is always the owning side of the relationship. Note, however, that Kodo does provide management of relationships as a vendor feature.

Cardinality Examples

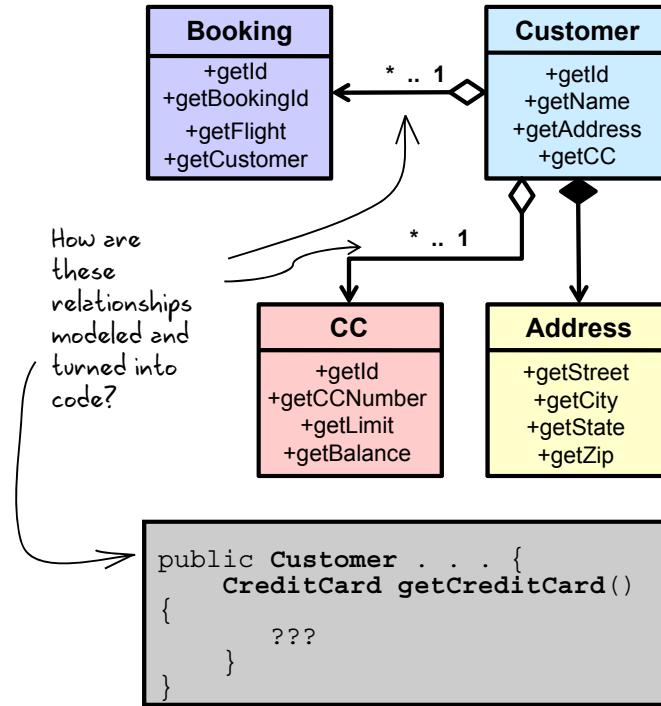


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ORACLE

Modeling Relationships in Coherence

- Relationship modeling requires a mechanism to return related objects
- Coherence does not implicitly provide support for modeling relationships



ORACLE

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

Modeling Relationships in Coherence

Turning a well-defined data model into a Coherence object model requires some thought. Coherence does not require nor enforce any specific mechanism. However, several patterns exist which allow for modeling such relationships.

Solving the `getXXX` problem

To return an associated object, a class needs:

- A key or object identifier for the object
- A lookup or retrieve mechanism for that object

```
public Customer . . . {  
    Type ccID;  
    Type creditCards;   
    . . .  
    CreditCard getCreditCard() {  
        return creditCards.get(ccID);  
    }  
}
```

How is the repository of
credit card defined?



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

Solving the `getXXX` problem

To implement an object model, objects which have relationships with objects need some mechanism to return them. Typically such mechanisms use a `getXXX` style method signature to get an instance.

How do you implement this method? The obvious solution is to embed an instance of a cache inside the containing class. However, this approach has many negative side effects:

- The class now needs to implement all the required methods to get, set, update, and so on, the associated object being retrieved.
- The object is now tied directly to Coherence.
- The methods for returning the associated object are not reusable.

A better approach is to use the 'repository' pattern to implement a typed repository.

Customer with Repository Example

- Customers use a 'repository' to return Credit Cards
- The 'credit card repository' insulates Customer objects from Coherence

```

public Customer . . . {
    . . .
    private Collection<Long> ccIds = new HashSet<Long>();
    private transient CreditCardRepository creditCards;
        ↗ Transient, so not persisted when serialized
    public Collection<CreditCard> getCreditCards() {
        return getCreditCardRepository().getAll(ccIds);
    }

    private CreditCardRepository getCreditCardRepository() {
        if (creditCards == null) {
            creditCards = new CoherenceCreditCardRepository();
        }
        return creditCards;
    }
}

```

Order may have many credit cards

Convenience method to obtain credit card repository

Customer.java

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

Customer with Repository Example

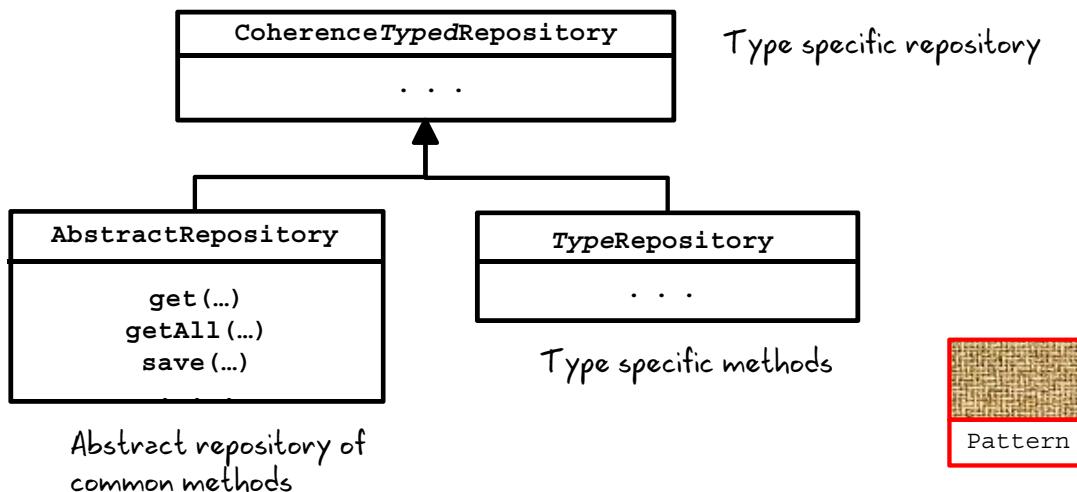
Implementing a class with repository is simple from the data model perspective.

The class requires:

- A key or set of keys for returning related objects: In the example shown a customer might have multiple credit cards, so a collection of keys is used.
- A repository: In the example shown the repository is marked `transient`, since this class contains references which cannot be serialized. Marking it transient has the effect of it being ignored.
- Some mechanism to boot strap the repository: In this example a convenience method is provided which returns the repository if already loaded, or loads it.

Getting Objects via Repository

A repository pattern can be used to define a repository which insulates clients from the specifics of Coherence and maintains the integrity of the data model



ORACLE®

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

Getting Objects via Repository

The prior slide has a glaring omission: the definition of the repository itself. Repositories are actually type specific instances based on a simple pattern. Each repository is based on a type specific interface which defines methods particular to the type, which are implemented by the concrete implementation. Additionally, the repository has some set of methods which are common to all repositories such as `get`, `getAll`, `saveAll`, `remove`, and so on. These are defined in a generic abstract class and then inherited.

Example AbstractRepository

```
>Returns Key/Value pairs of type entities
public abstract class
    AbstractCoherenceRepository<K, V extends Entity<K>> {
    public abstract NamedCache getCache(); Gets them from a specific cache
    public V get(K key) {
        return (V) getCache().get(key);
    }
    public Collection<V> getAll(Collection<K> keys) {
        return getCache().getAll(keys).values();
    }
    . . .
}
```



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

Example AbstractRepository

The object model from a type specific repository was still rather vague. In the example above, a abstract generic class provides much of the functionality required for all repositories.

In the example shown, the `AbstractCoherenceRepository` is a generic class based on the earlier defined `Entity` class, but having a key and value type associated with.

The partial implementation provides `get()`, `getAll()`, `save()` and `saveAll()` methods based on the Java generic class mechanism, and then is used by concrete implementations.

Example AbstractRepository

```
    . . .
    public void save(V value) {
        getCache().putAll(Collections.singletonMap(value.getId(),
                                                    value));
    }
    Can save instances
    public void saveAll(Collection<V> values) {
        Map batch = new HashMap(values.size());
        for (V value : values) {
            batch.put(value.getId(), value);
        }
        getCache().putAll(batch);
    }
    . . .
}
Might be able to delete and
update as well
```



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

Example AbstractRepository (continued)

`save()` and `saveAll()` methods implementations are straightforward and use the associated named cache to save the provided objects. Additional methods might be implemented to update, remove, and `removeAll` objects if required.

Example CreditCardRepository

CreditCardRepository:

- Defines the methods exposed by the concrete Coherence repository
- May add other type specific methods

```
public interface CreditCardRepository {  
  
    public CreditCard getById(Long key);  
    public Collection<CreditCard> getAll(Collection<Long> keys);  
    public void save(CreditCard cachedCreditCard);  
}
```



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

Example CreditCardRepository

An example credit card repository might have a `getById()`, `getAll()` and `save()` methods typed to a specific value as shown here. Other methods can be placed into the interface, specific to the repository.

Example CoherenceCreditCardRepository

```
public class CoherenceCreditCardRepository
    extends AbstractCoherenceRepository<Long, CreditCard> implements CreditCardRepository {  
  
    public static final String CACHENAME = "CreditCards";  
  
    private static final NamedCache m_cachedItems = CacheFactory.getCache(CACHENAME);  
    protected NamedCache getCache() { return m_cachedItems; }  
  
    public CreditCard getById(Long key) {  
        return super.get(key);  
    }  
  
    public Collection<CreditCard> getAll(Collection<Long> keys) {  
        return super.getAll(keys);  
    }  
}
```

Returns credit cards

with a long key

Uses internal
named cache



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

Example CoherenceCreditCardRepository

A Coherence-specific repository is developed by extending the abstract version, providing types for the key and value, and then implementing the methods required by the type specific interface. The combination of these classes solves the problems of providing associations between objects and their associations. The examples provided can be tailored to meet each different relationship type.

Quiz

Objects inserted into a Coherence cache must: (Select all that apply)

- a. Follow JavaBean rules
- b. Must implement Java Serializable
- c. Must support a form of serialization
- d. Must implement Comparable
- e. Must define hashCode() and equals() methods



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

Answer: a,c

Coherence entities must follow JavaBean rules, including a no-arg constructor and getter/setter methods for all fields, additionally they must provide a form of serialization, but it need not be Java serializable. Comparable, hash code and equals implementations are required for objects which are used as keys, but not for the object themselves.

Practice.03.01 Overview: Developing with Complex Objects

This practice covers the following topics:

- Creating identifiable, serializable objects and adding them to a cache
- Packaging the objects into a .jar which can be used with the console



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

Agenda

Developing Objects

Improving Performance Using Serialization

- Serialization and Performance
- Serialization Options
- Deserialized Caches
- Serialization Behaviors



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

Serialization Concepts

Serialization:

- Is the process of transforming an object into and out of a format appropriate for storage or transfer across a network
- Provides:
 - A convenient method for writing objects to storage
 - A method for detecting changes in objects versus implementations ← Can detect mismatches such as old data deserialized to a newer object definition
 - Support for overriding the process via implementation of the Externalizable interface.
- Operations are referred to as serialize and deserialize
- Is supported in java using `java.io.Serializable` interface

Marks the object 'ok' to serialize nothing more ↗

ORACLE

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

Serialization Concepts

Serialization is the process of converting a object to a storable format. Many languages provide mechanism for marking an interface serializable. For example, .NET uses the `Serializable` attribute, whereas Java uses the `java.io.Serializable` marker interface. Marking a class serializable doesn't actually do anything, but rather announces to the application and language that the class can be written to disk. Such a marking means that only serializable fields are used in the class. Java supports the ability to mark fields transient, such fields are not serialized.

The standard Java encoding method uses a simple translation of the fields into a byte stream. Primitives as well as non-transient, non-static referenced objects are encoded into the stream. Each object that is referenced by the serialized object and not marked as transient must also be serialized; and if any object in the complete graph of non-transient object references is not serializable, then serialization will fail. Developers can influence this behavior by marking objects as transient, or by redefining the serialization for an object via `Externalizable`.

The process of serialization is sometimes referred to as *marshalling* and *un-marshalling*.

Serialization and Performance

Serialization mechanisms impact:

Area	Impact
CPU	Some serialization forms use significantly more CPU than others to serialize the same object
Network & Memory	The more compact a serialized object is, the less network bandwidth that is required to transmit it and the less memory required to store it, speeding garbage collection.
Cache Capacity	The more compact the serialization format is, the more objects that can be managed by the grid for a given amount of memory.



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

Serialization and Performance

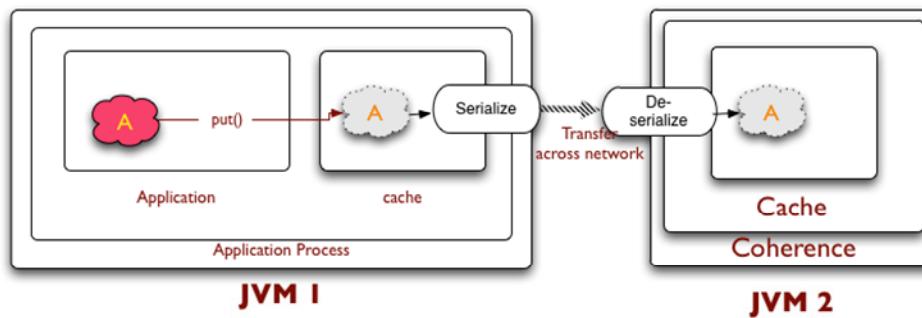
Every object is transmitted and stored in memory in its serialized form. The overall performance of your cache is impacted by the way your objects are serialized. The following properties of the serialization mechanism that you choose affect your cache throughput and latency:

- **CPU utilization:** Some serialization forms take significantly more CPU than others to serialize the same object.
- **Network utilization:** The more compact a serialized object is, the less network bandwidth that is required to transmit it.
- **Garbage collection:** A more memory-efficient serialization mechanism can allow for less garbage collection.
- **Cache capacity:** The more compact the serialization format is, the more objects that can be managed by the grid for a given amount of memory.

Serialization Options with Coherence

Coherence supports a number serialization options, including:

- Java serialization: `java.io.Serializable`
 - `ExternalizableLite`: a variation on Java `Externalizable`
 - Portable Object Format (POF): Fast, compact, platform independent serialization
- Others as well, such as using XML to define the serialized format (not serialized into XML)!



ORACLE

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

Serialization Options with Coherence

Coherence provides the following options for serialization:

- Standard Java Serialization: The Java Serialization API provides a standard mechanism to handle object serialization. An object is marked serializable by implementing the `java.io.Serializable` interface. When you use the `Serializable` interface, your class is serialized automatically by default.
- Externalizable interface: Object serialization also uses the `Externalizable` interface. Java gives you control over the serialization process through the `java.io.Externalizable` interface.
- ExternalizableLite: Coherence provides `ExternalizableLite`, which is a highly optimized `Serializable` interface, along with `ExternalizableHelper` to serialize and deserialize attributes. `com.tangosol.io.ExternalizableLite` is very similar to `java.io.Externalizable`, but offers better performance and less memory usage by using a more efficient I/O stream implementation. The `com.tangosol.run.xml.XmlBean` class provides a default implementation of this interface.
- Portable Object Format (POF): POF is an extremely efficient binary format. It has many advantages over the standard Java serialization.

For backward compatibility reasons, `ExternalizableLite` remains available, but where possible, POF should be used as it has many benefits over `ExternalizableLite`.

Serialization Comparison

Type	Pros	Cons
Serializable	Built-in	Slow; large files; not cross platform
ExternalizableLite	Performs better than Java serialization, by a factor of 6:1, sizes are on the order of 3:1 smaller.	Must code <code>readExternal()</code> and <code>writeExternal()</code> methods on entities; not cross platform.
XmlBean	Similar size to, but performs slightly better than ExternalizableLite	Requires minor configuration; classes and subclasses must implement XmlBean; not cross platform.
Portable Object Format (POF)	Cross platform; efficient and compact; highest performing of all serialization mechanisms, often 10:1 over Java serializable.	Requires configuration and assignment of identifiers to each object, also supports run time registration; must code <code>readExternal()</code> and <code>writeExternal()</code> methods on entities.



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

Implementing Java Serialization

To implement Java Serialization:

- Member variables must be serializable
- Class members must be serializable
- Must implement `java.io.Serializable`

```
import java.io.Serializable;
public class AirCraft implements . . . Serializable {
    .
    .
    private Long id;
    private Long capacity;
    private String type;
    private String description;
}

```

Add implements Serializable to class.

Members must be serializable. Class members must implement serializable

AirCraft.java

No specialized code required.



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

Implementing Java Serialization

Classes which implement Java serializable are perhaps the simplest of all serializable classes. Classes which implement Java serialization must:

- Define that they are serializable with the `java.io.Serializable` marker interface.
- Member variables, including class variables, must be serializable.
- Variables which are not serializable, such as HttpSession, Sockets, and so on, must be marked transient.

Lastly, no special coding is required. Coherence can determine type information via reflection and write out the class contents appropriately.

Implementing ExternalizableLite

Classes implementing ExternalizableLite:

- Must provide a no-argument constructor and setters for all fields
- Must implement `com.tangosol.io.ExternalizableLite`
- Must implement `readExternal()` and `writeExternal()` methods
- Can only serialize primitive, primitive wrappers, and classes which themselves implement `ExternalizableLite`
- Can use `ExternalizableHelper` to assist with serialization of certain types, arrays, child classes and others

Can expect a 6 to 8x improvement in performance and a 3:1 reduction in size over Java Serialization!



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

Implementing ExternalizableLite

Classes which implement ExternalizableLite have a number of requirements beyond standard Java serialization and are responsible for defining how they serialize themselves.

Classes which implement Coherence ExternalizableLite must:

- Define that they are serializable with the `com.tangosol.io.ExternalizableLite` interface
- Must implement `readExternal()` and `writeExternal()` methods which are called to externalize and recreate class instances
- May use the `ExternalizableHelper` class to assist in serializing content

ExternalizableLite Example

```

import java.io.DataInput; <----- Required by read/writeExternal
import java.io.DataOutput;

import com.tangosol.io.ExternalizableLite; <----- Implements...
import com.tangosol.util.ExternalizableHelper;

public class AirCraft implements . . . ExternalizableLite {
    . . .
    public void readExternal(DataInput in) throws IOException {
        id = new Long(in.readLong());
        capacity = new Long(in.readLong());
        type = ExternalizableHelper.readSafeUTF(in);
        description = ExternalizableHelper.readSafeUTF(in);
    } <----- Repopulates object
    public void writeExternal(DataOutput out) throws IOException {
        out.writeLong(id.longValue());
        out.writeLong(capacity.longValue());
        ExternalizableHelper.writeSafeUTF(out, type);
        ExternalizableHelper.writeSafeUTF(out, description);
    } <----- Stores object
    . . .
} <----- Data is read and written in the same order

```

AirCraft.java



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

ExternalizableLite Example

Classes which implement the ExternalizableLite can expect to serialization improvements along the lines of 6 or 8:1 in terms of speed and approximately a 3:1 reduction in size over standard Java serialization.

The two read and write routines required by the interface take DataInput and DataOutput objects for reading and writing content respectively. These are the same two classes used by the Java Externalizable object and behave for the most part the same way. However, Coherence provides a second helper class for handling certain object types. In the example given, strings are read and written using the readSafeUTF() and writeSafeUTF() methods. Other methods also exist for reading and writing child objects which themselves implement ExternalizableLite.

Portable Object Concepts

Portable Object Format (POF):

- Is language-independent binary format created for Coherence
- Has APIs for Java, .NET, and C++
- Serializes data faster and produces smaller results than other methods
- Supports data versioning for forward and backward compatibility
- Requires external serialization



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

Portable Object Concepts

The Portable Object Format (POF) is a language-independent binary format with accompanying APIs. It supports Java, C++, and .NET. POF has many advantages over the standard Java serialization.

POF is far more efficient than the standard Java serialization. On the average, POF is 10 to 12 times faster than the standard serialization and produces a much more compact result. A standard single Java Integer requires 76 bytes in serialized form, whereas with POF it takes about 4 bytes. Note that the number of bytes required varies depending on the type of object that is serialized.

POF data types can be versioned and are backward compatible. If you add a data element to an object, you can increment the version number of `EvolvablePortableObject` and it still works with the older versions. By using POF, the serialization logic can be externalized, giving you the added flexibility of not having to directly modify classes that support POF. The POF binary format is self-descriptive and indexed, which enables the extraction of values without having to fully deserialize the object. Though this functionality is not available in the product currently, POF is the foundation for future improvements. Similar to `Externalizable`, POF also requires that you implement the serialization methods manually.

PortableObject Requirements

Classes implementing PortableObject:

- Must Implement `com.tangosol.io.pof.PortableObject`
- Must Implement `readExternal()` and `writeExternal()`
- Can serialize primitive, primitive wrappers, arrays, and child objects
- Must be registered declaratively or programmatically

Can expect a 10 to 12 improvement in performance and a 6:1 reduction in size over Java Serialization!



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

PortableObject Requirements

Classes which implement PortableObject have a number of requirements beyond standard Java serialization and are responsible for defining how they serialize themselves.

Classes which implement Coherence PortableObject must:

- Define that they are serializable with the `com.tangosol.io.ExternalizableLite` interface.
- Must implement `readExternal` and `writeExternal` methods which are called to externalize and recreate class instances.
- May use the `ExternalizableHelper` class to assist in serializing content

PortableObject Example

```

import com.tangosol.io.pof.PofReader;           Required imports
import com.tangosol.io.pof.PofWriter;
import com.tangosol.io.pof.PortableObject;

public class AirCraft implements . . . PortableObject {
    . . .

    public void readExternal(PofReader in) throws IOException {
        id = in.readLong(0);
        capacity = in.readLong(1);
        type = in.readString(2);
        description = in.readString(3);
    }

    public void writeExternal(PofWriter<out> out) throws IOException {
        out.writeLong(0,id);
        out.writeLong(1,capacity);
        out.writeString(2,type);
        out.writeString(3,description);
    }
}

Each data element is assigned a numeric identifier, and read and
written by its identifier.

```

AirCraft.java

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

PortableObject Example

With portable object, developers are required to develop serialization code themselves. Each class must implement the portable object interface and its two required methods `readExternal` and `writeExternal`. These methods repopulate and externalize any object, and when coupled with registration, allow Coherence to efficiently read and write.

Portable Object provides two classes, `PofReader`, and `PofWriter` to support reading and writing object contents. These two classes provide a set of support methods for reading and writing primitive types, arrays of types, collections of types, and so on.

POF Indices Requirements

POF attributes indices should be assigned using the following guidelines:

- Order reads and writes from lowest index to highest.
- Read and write in the same order.
- Indices can be noncontiguous, but must be written in order.
- Indices cannot change over the life of the object class.
- Indices between a class and its encapsulated classes are independent and may overlap. For example a `Customer` class may use values 1 to 4, and an included class such as `Address` may use the same values



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

POF Indices Requirements

In addition to the points given, inheritance should be avoided in POF-based classes because it requires the derived objects to be aware of the indices used by the parent objects, breaking the basic inheritance model.

Registering Portable Objects

Classes implementing `PortableObject` must be registered.

Registration can be:

- Programmatic, using `PofContext()` instance
- Declarative, using `pof-config.xml`

```
SimplePofContext ctx = new SimplePofContext();
ctx.registerUserType(1000, AirCraft.class,
                    new PortableObjectSerializer(1000));
```

app.java

```
<pof-config>
  <user-type-list>
    <include>includes</include>
    ...
    <user-type>
      <type-id>1000</type-id>
      <class-name>com.oracle...AirCraft</class-name>
    </user-type>
  ...
</pof-config>
```



pof-config.xml

ORACLE

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

Registering Portable Objects

Objects that implement `PortableObject` must be registered in order for Coherence to take advantage of the custom serialization code. Registration can happen in one of two ways: programmatically or declaratively.

To register a class Programmatically:

1. Create a new instance of `SimplePofContext()`;
2. Call the `registerUserType()` method with an identifier, the class which implements Portable Object and a serializer.

To register a class Declaratively:

Use user-type entries in the `pof-config.xml` configuration file.

In both cases the serializer must be assigned an identifier. Identifiers are any integer 1000 or above. In the example provided, the same identifier is used in both cases: 1000.

Choice of identifier is somewhat arbitrary, but need only be 1000 or above.

Selected PofWriter Methods

```
package com.tangosol.io.pof;  
public interface PofWriter {  
    public PofContext getPofContext();  
  
    public void writeBoolean(int iProp, boolean value);  
  
    public void writeChar(int iProp, char value);  
  
    public void writeShort(int iProp, short value);  
  
    public void writeLong(int iProp, long value);  
  
    . . .  
}
```

An informative subset of
PofWriter methods.
See docs for complete list!

The PofContext interface represents a set of user types that can be serialized to
and deserialized from a POF stream.

Writes a boolean from the input stream

Writes a character from the input stream.

Writes a short integer from the input stream.

Writes a integer from the input stream.

All throw IOException on error.

ORACLE

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

Selected PofWriter Methods

The methods of the PofWriter interface provide the basis for reading data from an input stream to repopulate an objects. In addition to a variety of `public void write<Type>(int iProperty, <Type> value)` methods, the interface also provides `getPofContext()`, which provides information about the underlying class being processed.

Selected PofReader Methods

```
package com.tangosol.io.pof;
```

```
public interface PofReader {
```

```
    public PofContext getPofContext();
```

The PofContext interface represents a set of user types that can be serialized to and deserialized from a POF stream.

```
    public boolean readBoolean(int iProp);
```

Reads a boolean from the input stream

```
    public char readChar(int iProp);
```

Reads a character from the input stream.

```
    public short readShort(int iProp);
```

Reads a short integer from the input stream.

```
    public long readLong(int iProp);
```

Reads a integer from the input stream.

```
    . . .
```

An informative subset of
PofReader methods.
See docs for complete list!

ORACLE®

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

Selected PofReader Methods

PofReader methods are similar to PofWriter, but perform the inverse operation, reading data *into* variables rather than storing data *from* variables.

Quiz

True or False? When implementing POF serialization, parent objects, containing child objects must not re-use indices used by the child.

- a. True
- b. False



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

Answer: b

False, the parent object may reuse the same index values used by the parent because the indices themselves are differentiated by object type.

External Serialization

Serialization code can be externalized via `PofSerializer`.

Serialization code might be externalized because:

- The same code is repeated over and over
- Serialization code is complex and obscures entity functionally
- Other classes need to be serialized, which are outside of the control of the current development
- Other reasons

```
public class AirPortSerializer implements PofSerializer {  
    public Object deserialize(PofReader reader) throws IOException { . . . }  
    public void serialize(PofWriter writer, Object o) throws IOException{. . .}  
}
```



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

External Serialization

Coherence supports a variety of mechanisms for serialization, including the ability to separate, or externalize, the serialization of a class or set of classes.

There are a number of reasons why a developer might consider externalizing such as:

- Serialization code is complex or repeating and cannot be developed in a common abstract or parent class
- Serialization code needs to be applied to classes outside the control of the developer
- Serialization code obscures the underlying functionality of a class to such a degree that the class becomes overly complex

In such circumstances, serialization can be externalized via a class which implements `PofSerializer`.

PofSerializer Basics

```

import java.io.IOException;
    import com.tangosol.io.pof.PofContext; Required imports
    import com.tangosol.io.pof.PofReader;
    import com.tangosol.io.pof.PofSerializer;
    import com.tangosol.io.pof.PofWriter;

public class CustomSerializer implements PofSerializer { ← Implements...
    public Object deserialize(PofReader reader)← Creates object from stream
        throws IOException {
        . .
    }
    public void serialize(PofWriter writer, Object o)← Stores object to stream
        throws IOException {
        . .
    }
}

```

Typically checks input type to insure its correct via instanceof

CustomSerializer.java



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

Implementing the PofSerializer Interface

PofSerializer provides a way to externalize the serialization logic of a class. This is useful for classes that you do not have access to change.

To implement PofSerializer, you must provide the `deserialize()` and `serialize()` methods in the class definition.

The `deserialize()` method takes a `PofReader` as parameter. It reads the data from the underlying POF input stream to load an object's state.

The `serialize()` method takes a `PofWriter` stream and an object as parameters. It saves the content of the object by storing its state into the underlying POF stream through `PofWriter`.

Note that the object elements are indexed in POF, and it is important that each element is read from the same index that it was written to.

Registering POF Serializers

Class implementing `PofSerializer` must be registered.

Registration can be:

- Programmatic, using `PofContext()` instance
- Declarative, using `pof-config.xml`

```
SimplePofContext ctx = new SimplePofContext();
ctx.registerUserType(1000, AirCraft.class, new AirCraftSerializer(1000));
```

```
<pof-config>
  ...
  <user-type>
    <type-id>1000</type-id>
    <class-name>com.oracle...AirCraft</class-name>
    <serializer>
      <class-name>com.oracle...AirCraftSerializer</class-name>
    </serializer>
  </user-type>
  ...
</pof-config>
```



pof-config.xml

ORACLE

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

Registering POF Serializers

POF Serializers are registered in exactly the same way as `PortableObject` serializers are, either programmatically via a `PofContext` or declaratively via `pof-config.xml`.

The primary differences are:

- Programmatically: The registration of the class which implements the `PofSerializer` interface.
- Declaratively: A `serializer` element is added to the `user-type` element which specifies the name of the associated serializer class.

Evolvable Objects

Objects change over time. To handle such change, a cluster must:

- Be shut down and all data reloaded in the new format
- Objects which change must implement `Evolvable`

Evolvable objects must support three specific attributes:

- Implementation version: A constant within the class, incremented with each new structural change. For example: the addition of a new field.
- Data Version: Version number of the class data (serialized data)
- Future Data: Data within the POF stream which cannot be deserialized to older versions of the object

```
package com.tangosol.io;
public interface Evolvable { . . . }
```



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

Evolvable Objects

Evolvable Objects are objects which can be serialized and deserialized effectively by version. Newer instances of objects can be created, and using the version number, stored and serialized with older versions.

Evolvable requires three core pieces of data to function:

- Object version: Coherence must know the object's version in order to support serialization to/from that version. Typically the object version is a constant.
- Data Version: Coherence must know the version of the data itself. This version can be thought of as the version of the serialized form of the class. This attribute is set by the serializer during deserialization, based on the version found in the POF stream.
- Future data: Effectively this is a binary placeholder for data which cannot be serialized but exists in the POF stream.

Implementing Evolvable

```

import com.tangosol.io.Evolvable;

public class AirCraft implements . . . Evolvable {
    .
    .
    private static int VERSION = 1;
    private int dataVersion;
    private Binary futureData;
    .
    .
    public int getImplVersion() { return VERSION; }
    public int getDataVersion() { return dataVersion; }
    public Binary getFutureData() { return futureData; }
    public void setDataVersion(int dataVersion) {
        this.dataVersion = dataVersion;
    }
    public void setFutureData(Binary futureData) {
        this.futureData = futureData;
    }
}

```

Add attributes and required evolvable methods.

AirCraft.java

```

import com.tangosol.io.AbstractEvolvable;

public class AirCraft extends . . . AbstractEvolvable {
    .
    .
    private static int VERSION = 1;
    public int getImplVersion() { return VERSION; }
}

```

Since almost ALL Evolvable classes are the same, an abstract version exists!

AirCraft.java

ORACLE

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

Implementing Evolvable

Implementing the `Evolvable` interface is as simple as adding a version number and other attributes to a class and then implementing the methods as shown. Since almost every implementation uses the same values and definitions for the `dataVersion()` and `futureData()` areas, an abstract version exists which can be extended.

Evolvable and POF

Evolvable Objects require changes to serialization

Custom serializers must:

- When deserializing:
 - Read the data version from the POF stream and set the data version attribute
 - Read object attributes normally
 - Store into future data any remaining attributes, otherwise set the future data value to null
- When serializing:
 - Set the data version of the POF stream to the maximum of the implementation version or the data version
 - Write object attributes
 - Write future data if present



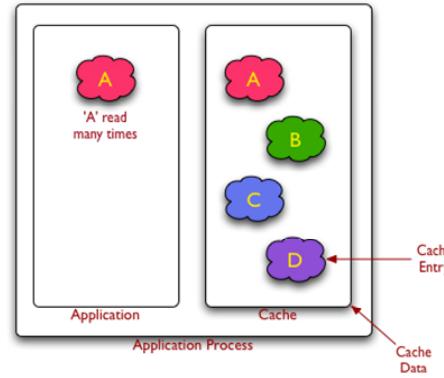
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Serialized versus Unserialized Caches

Not all Caches serialize data.

Caches which perform no serialization include:

- Near Cache: A form of cache which attempts to provide the best of replicated cache performance and partitioned cache scalability
- Local Cache: A cache which is completely contained within, or local to, a particular cluster node



ORACLE

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

Serialization Testing Support

Serialization can be exercised using:

- ExternalizableHelper:
Can be used with or without a SimplePofContext to convert to and from Binary representations of data
- Binary:
A serialized representation of an object, providing convenience routines such as length, `toString()`, and others
- Base:
A base class containing a variety of support functions, including `toHex()` returning a string representation of an object

All are in the `com.tangosol.util` package



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

Quiz

True or False? All Coherence cache types require and contain serialized objects.

- a. True
- b. False



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

Answer: b

False, several caches, including Local cache and the fronting cache of a Near cache may not contain serialized objects.

Serialization Testing Example

Serialization can be tested using ExternalizableHelper

```

import com.tangosol.util.ExternalizableHelper; ← Can use helper to
import com.tangosol.util.Binary; ← convert to/from
import com.tangosol.util.Base; ← Binary has convenience
... routines for examining data
AirCraft ac = new AirCraft (...);
Binary binary = ExternalizableHelper.toBinary(ac);

int length = binary.length();
Byte[] bytes = binary.getByteArray();

String hex = Base.toHexString(bytes,16); ← Base has convenience
                                                routines for conversion etc

AirCraft copy (AirCraft) =
    ExternalizableHelper.fromBinary(binary);

```



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

Serialization Testing Example

Serialization can be tested using both caching and retrieving objects, however ExternalizableHelper can be used to simplify the process by using its toBinary() and fromBinary() methods. In the example shown, an AirCraft object is constructed and then the method is used to force serialization, and return a binary version of the object. Likewise, the fromBinary can recreate an instance from its binary representation.

The Binary and Base classes provide a number of additional convenience methods for converting, examining, pretty-printing, and otherwise examining the serialized data.

Note that a second version of toBinary exists, which accepts a context to use with serialization, and can be used as shown below:

```

SimplePofContext ctx = new SimplePofContext();
ctx.registerUserType(1000, AirPort.class, new AirPortSerializer());

AirPort original = new AirPort(...);
Binary binary = ExternalizableHelper.toBinary(original, ctx); ←

```

Practice.03.02 Overview: Serialization using ExternalizableLite

This practice covers the following topics:

- Modifying a Serializable class to use ExternalizableLite
- Externalizing parent and child classes
- Testing an ExternalizableLite class



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

Practice.03.03 Overview: Serialization using Portable Object Format

This practice covers the following topics:

- Modifying a class to support POF
- Adding required method support to child and parent classes
- Testing using SimplePofContext and ExternalizableHelper



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

Summary

In this lesson, you should have learned how to:

- Implement Java objects that are stored in Coherence caches using `Serializable`, `ExternalizableLite`, and Portable Object Format techniques
- Describe the purpose of deserialized caches
- List which cache types are not serialized



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

Configuring Coherence Caches

4

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Describe, configure, deploy, and use local, replicated, partitioned, and near cache topologies
- Describe the overflow, optimistic, remote, transaction, and continuous query cache topologies
- Determine when and how to use each type of cache topology



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

Agenda

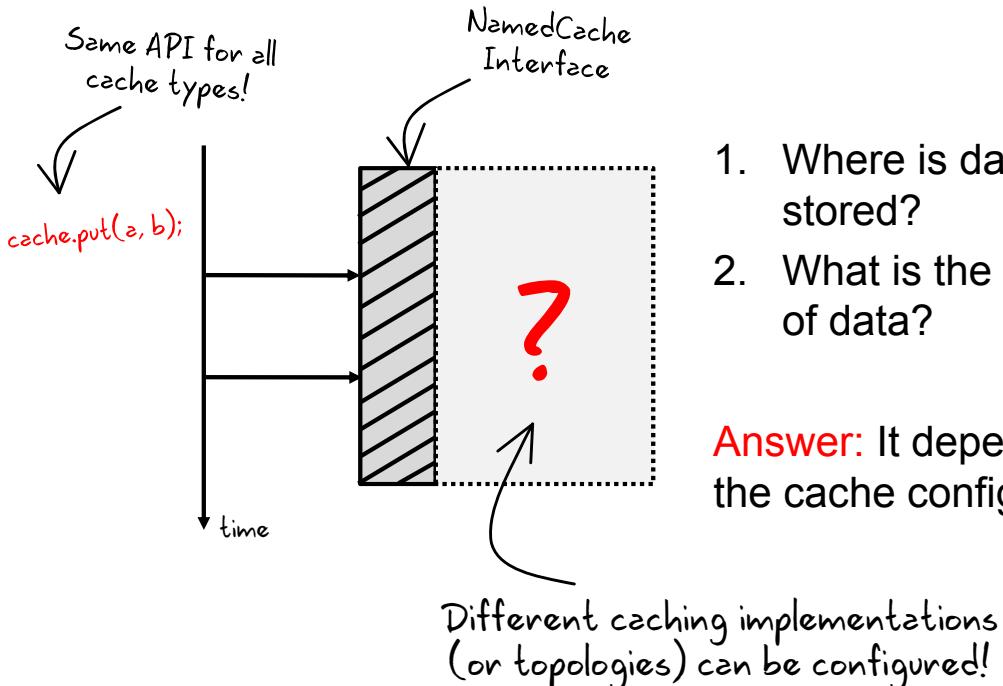
Coherence Cache Topologies

- **What Happens to Cached Data?**
- Cache Configuration Concepts
- Topologies
 - Local Cache
 - Replicated Cache
 - Partitioned Cache
 - Near Cache
 - Overflow Cache
 - Other Cache Types, Topologies Summary, and Advanced Configurations



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

What Happens to Cached Data?



1. Where is data stored?
2. What is the lifetime of data?

Answer: It depends on the cache configuration!

ORACLE

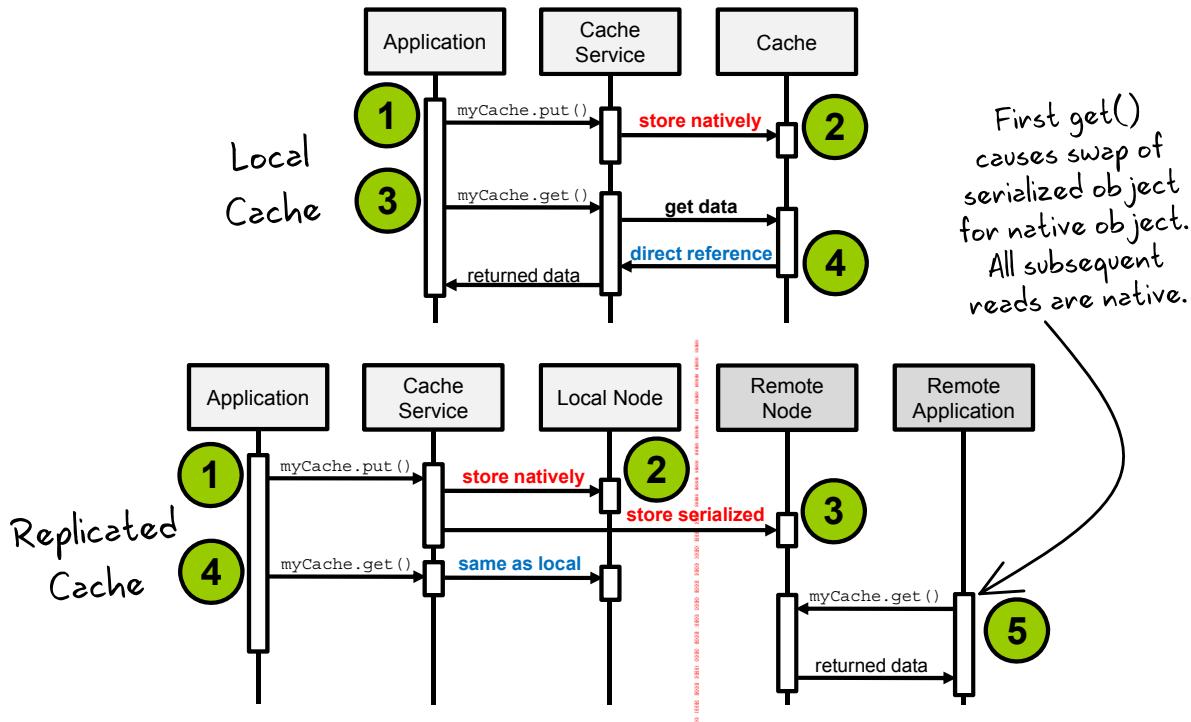
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What Happens to Cached Data?

When data is placed into a cache, where does it go? And after it gets there, what happens to it? The answers to these questions depend on the type of cache implementation that is configured. A cache implementation within Coherence is referred to as a topology. So you can think of a topology as an implementation of a Coherence cache.

Now that you know what a topology is, let's take a closer look at the different types that Coherence provides.

Local and Replicated Cache Semantics



ORACLE

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

Local and Replicated Cache Semantics

Different cache types store data in different forms within a Coherence cache. If the data has to travel on the network to another node, an application that gets data from a remote node always gets a copy of the original object, and not a reference to the original object itself.

Local Cache

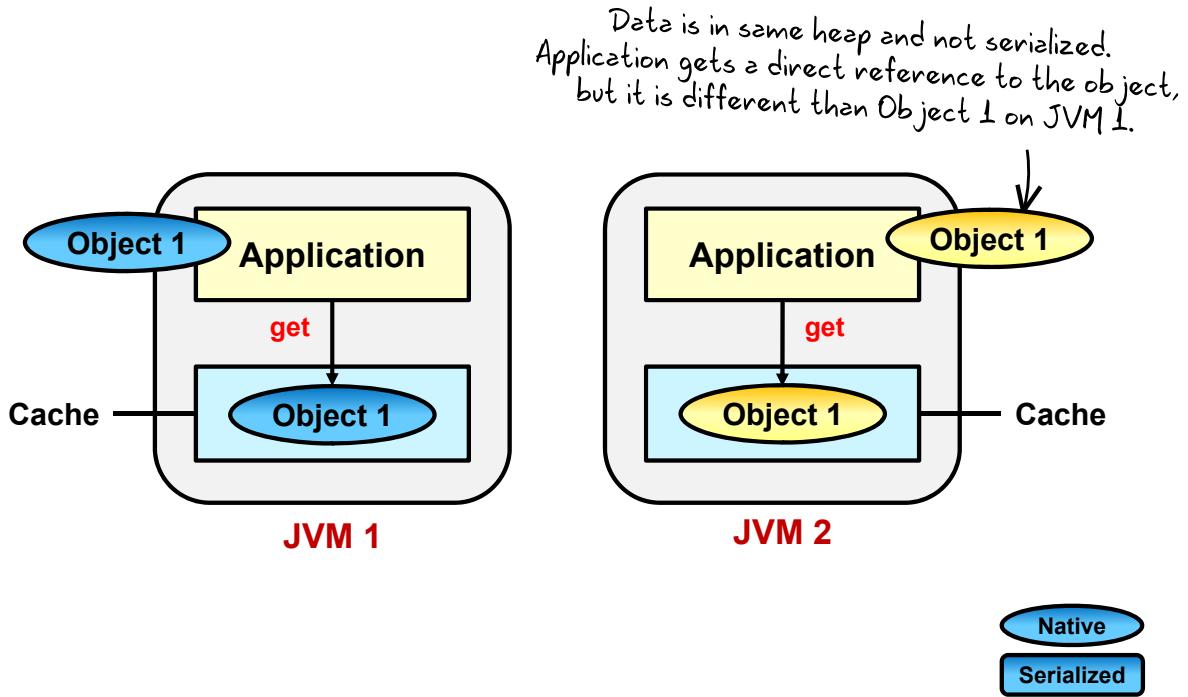
1. The application calls `put()` on a cache entry to store an object in Coherence.
2. The local cache service stores the data in native object format.
3. The application calls `get()` on the cache entry to retrieve the object from Coherence.
4. Coherence returns the object as a direct reference to the original object in memory.

Local and Replicated Cache Semantics (continued)

Replicated Cache

1. The application calls `put()` on a cache entry to store an object in Coherence.
2. The replicated cache service stores the data in native object format on the same local node where the application calls `put()`.
3. The replicated cache service replicates the data in serialized form to any remote cluster nodes. The data is stored in serialized form.
4. The same application calls `get()` on that cache entry, and Coherence returns the object as a direct reference to the original object, just as it does with a local cache.
5. The remote application calls `get()` on the cache entry to retrieve the object from Coherence. The cache service retrieves the object from the same node where the application calls `get()`. In the process, the serialized form of the object in the cache is deserialized and the native object form replaces the serialized form in the cache. All subsequent `get()` calls avoid deserialization costs and receive a direct reference to the copied object on that cluster node.

Local and Replicated Cache Semantics



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



Local and Replicated Cache Semantics (continued)

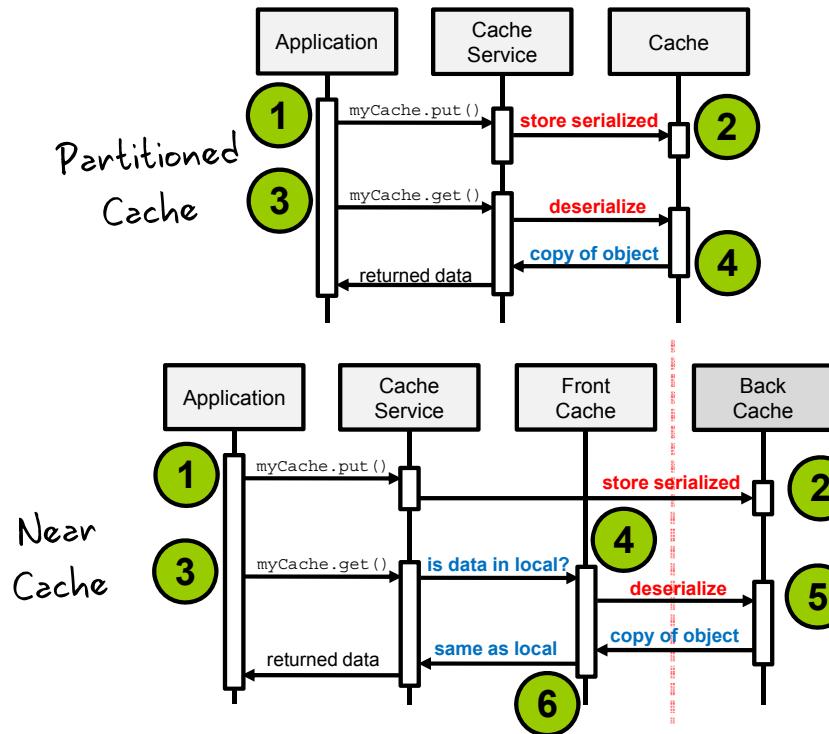
Local Cache:

- Focus only on JVM 1
- Object is stored directly in memory and not serialized
- The Application in JVM 1 performs a `get` and receives a direct reference to the object.

Replicated Cache:

- Focus on JVM 1 and JVM 2
- A replicated cache contains the same object on each Coherence node (represented by each JVM in this diagram).
- When data is put into a replicated cache, it is placed directly in memory on the local node, then it is serialized and sent to the other cluster nodes, and stored in serialized form.
- When the Application in JVM 2 performs a `get`, the object is deserialized and placed in the cache. The serialized version is discarded, and the object is returned to the Application.
- The Application receives a copy of the original object, but always receives the same reference for each `get`.

Partitioned and Near Cache Semantics



ORACLE

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

Partitioned and Near Cache Semantics

Partitioned Cache

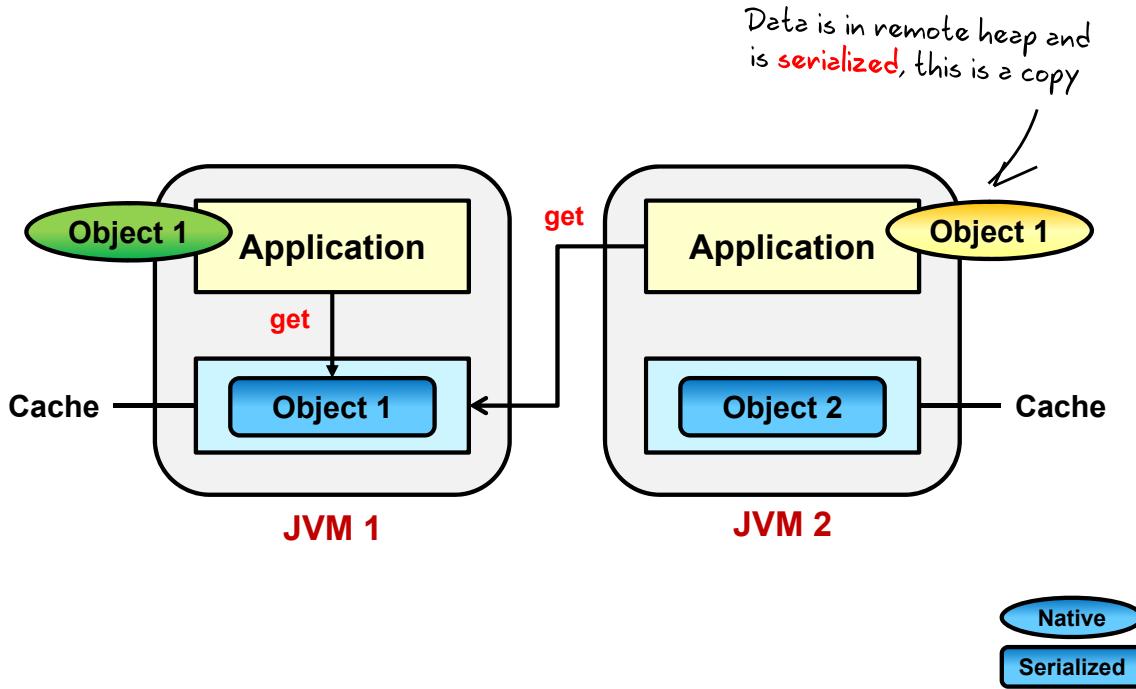
1. The application calls `put()` on a cache entry to store an object in Coherence.
2. The partitioned cache service stores the data in serialized form.
3. The application calls `get()` on the cache entry to retrieve the object from Coherence.
4. The object is deserialized and Coherence returns a copy of the object.

Partitioned and Near Cache Semantics (continued)

Near Cache

1. The application calls `put()` on a cache entry to store an object in Coherence.
2. The near cache service stores the data in serialized form directly on the back tier.
3. The application calls `get()` on the cache entry to retrieve the object from Coherence.
4. Coherence checks the front tier of the cache to see if the data is available locally. If it is, then go to #6.
5. If the data is not in the front tier, then Coherence checks the back tier to get the data. The object is deserialized, and the copy of the object is stored in the front tier in native form.
6. Coherence returns the object as a direct reference to the in-memory copy of the original object.

Partitioned Cache Semantics



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

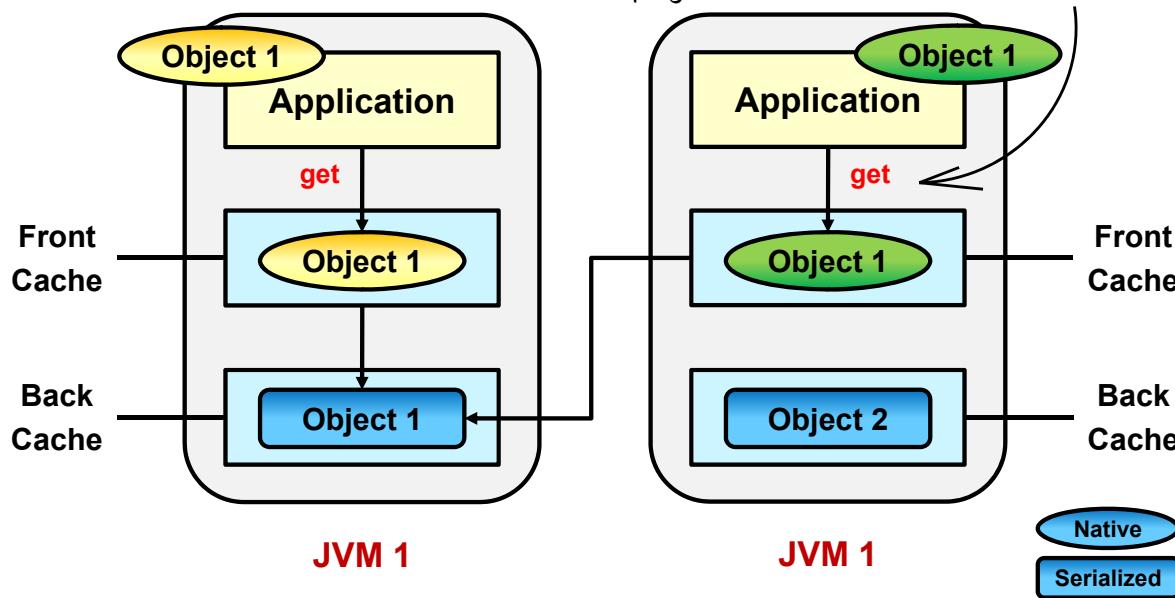
ORACLE

Partitioned Cache Semantics

- Data set is divided into partitions which are spread evenly across a cluster in primary and backup copies.
- Data is stored in serialized form.
- All get requests performed on a partitioned cache return a new copy to the Application.
- Object 1 on JVM 1 is different than Object 1 in the cache and on JVM 2.

Near Cache Semantics

Data is in remote heap and is **serialized**, this is a copy. However, the near cache is local and application always gets direct access to local reference.



ORACLE

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

Near Cache Semantics

- A near cache combines two cache types to work together. The most common is to define a front scheme as a local cache, and a back scheme as a partitioned cache.
- Data is stored in object form on the local cache.
- Data is stored in serialized form on the partitioned cache.
- All get requests performed on a partitioned cache return a new copy to the local "near" cache, which is returned to the Application.
- All subsequent calls on the local cache return that same copy.
- The local cache on each cluster node will have its own copy of Object 1.

Java Object Access Semantics

Java typically supports two calling mechanisms:

Pass By Value and *Pass By Reference*. With:

- Pass by Value: a copy of the object is provided to the method or requestor
- Pass by reference: the requestor has access to the same object

Coherence Objects can use either semantic depending on cache type



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

Java Object Access Semantics

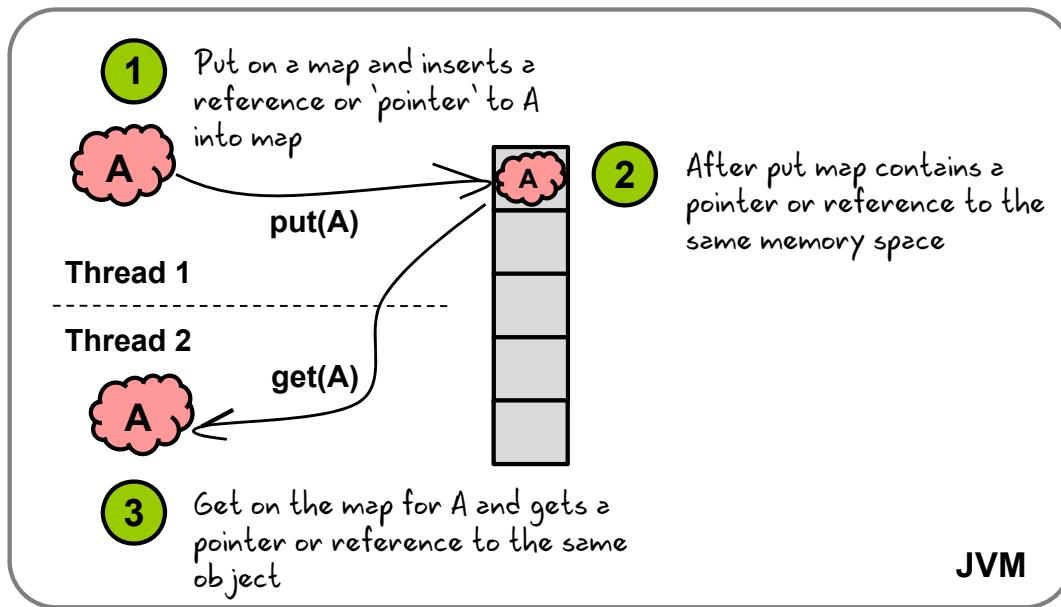
Computer languages support two argument passing semantic types: by value and by reference.

With pass by value, on call to a method, a copy of an object is created, pressed onto the argument stack and provided to the method. The method can then change the copy without concern for any impacts up the call stack.

With pass by reference, a reference or pointer to the object is passed, changes in the underlying method directly impact upper elements in the call stack.

Typically Java supports only pass by reference. However, both JEE and Coherence muddy this distinction. With Coherence, an object might be pushed into the cache locally, or remotely. Likewise the object might be serialized or not. All these characteristics depend on how the cache topology was defined.

Pass by Reference and Maps



Thread 1 and Thread 2 are now accessing the same object!
Changes in one thread affect the other!

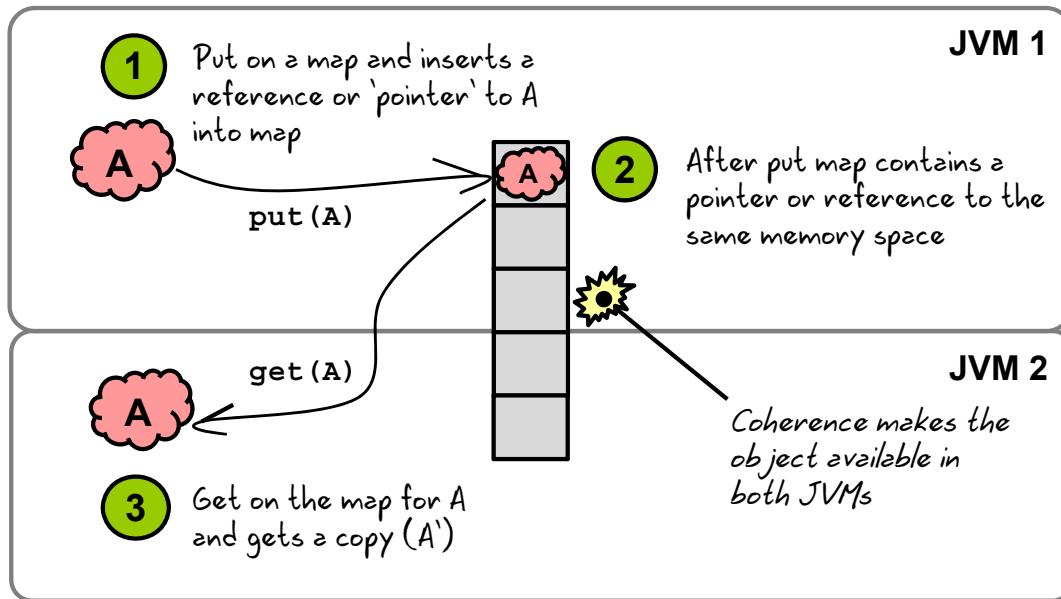
ORACLE

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

Pass by Reference and Maps

In this example, an object is put into a map. Because Java is the language, the map contains an actual reference to the object and not a copy. Later, in step three in the diagram, when the object is extracted on a different thread, but in the same JVM, the second thread now has a reference to the original object. Changes to the original object, in thread one, impact thread two's copy and vice versa.

Pass by Copy and Maps



Thread 1 and Thread 2 are now accessing the different but identical objects! Changes in one JVM do NOT effect the other.

ORACLE

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

Pass by Copy and Maps

While Java does not support an implicit pass by copy, you can accomplish the same thing using Coherence. In the example given, a thread in JVM 1 places an object into a cache. The element is then replicated, or copied, to other nodes in the cluster. Later, when a reference is made in JVM 2, a copy of the object with the same value is provided. Changes in JVM 2 have no impact on JVM 1 until the object is put back to the cache. In fact, even after a `put`, the changes have no impact, unless JVM rereads the object, replacing the original with the updated copy.

Agenda

Coherence Cache Topologies

- What Happens to Cached Data?
- **Cache Configuration Concepts**
- Topologies
 - Local Cache
 - Replicated Cache
 - Partitioned Cache
 - Near Cache
 - Overflow Cache
 - Other Cache Types, Topologies Summary, and Advanced Configurations

ORACLE®

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

Revisiting coherence-cache-config.xml

- Default coherence-cache-config.xml is stored in coherence.jar.

WARNING: If you fail to provide a coherence-cache-config.xml (on the front of your classpath), Coherence will use the default!

ADVICE: To avoid Coherence using the default, remove the coherence-cache-config.xml from coherence.jar for production!

- Override the default coherence-cache-config.xml:

```
java -Dtangosol.coherence.cacheconfig=my-cache-config.xml ...
```



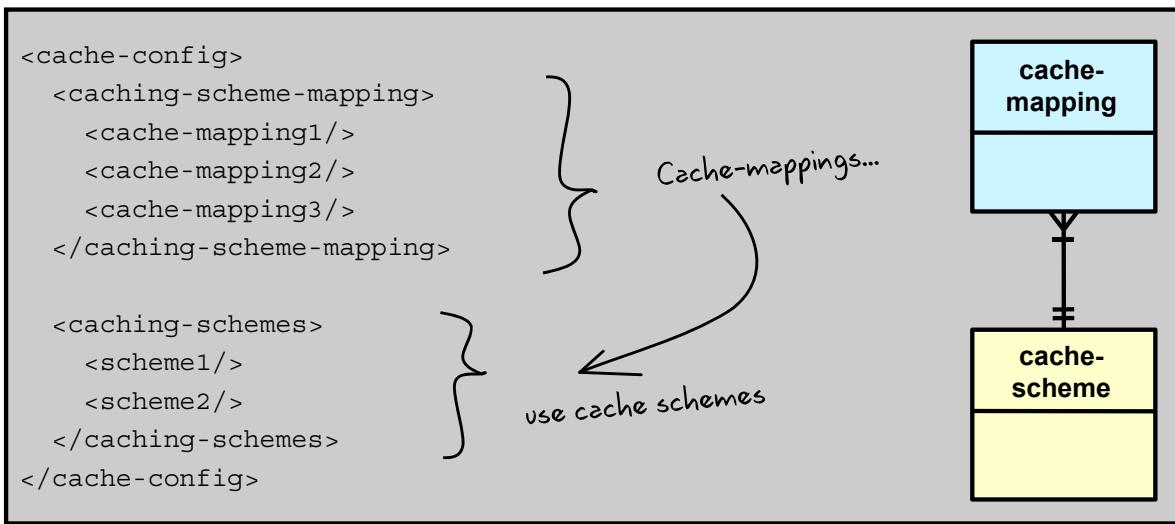
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Revisiting coherence-cache-config.xml

The lesson titled “Getting Started with Oracle Coherence” briefly described how the coherence-cache-config.xml file works. This section reviews some of the major points again, because this lesson is all about the details of using this file to configure the different caches provided by Coherence. It is also possible to specify a URI that points to a centralized configuration file using HTTP.

Anatomy of a Cache Configuration

- Two major sections to a cache configuration file:
 - Cache scheme mappings
 - Cache scheme definitions



ORACLE

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

Anatomy of a Cache Configuration

A Coherence cache configuration file consists of two major sections:

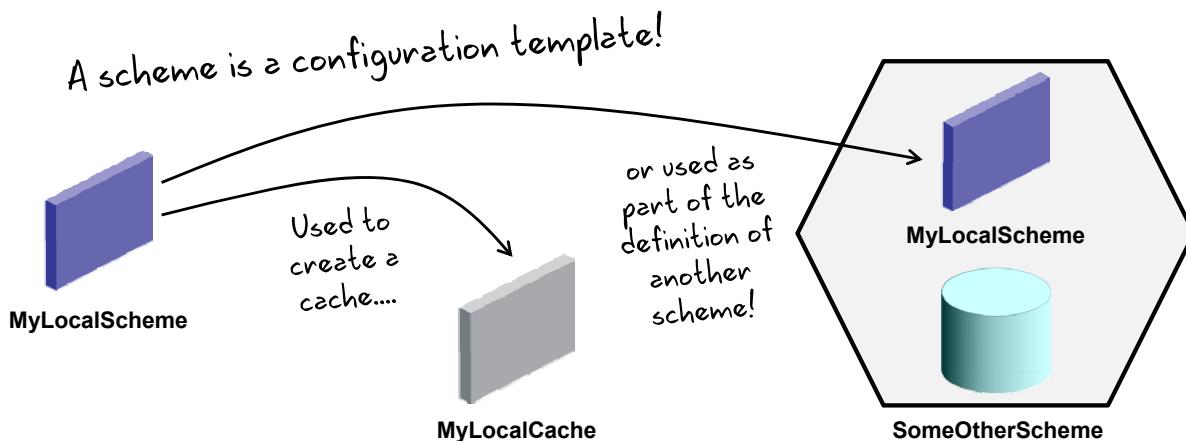
- Cache scheme mappings that map cache names to a particular cache scheme definition.
- Cache scheme definitions that define individual caching schemes.

Many cache scheme definitions can be defined, but a cache mapping definition can only reference one.

Many cache mapping definitions can reference the same cache scheme definition.

Note: <cache-mapping#> and <scheme#> are pseudo code and not actual Coherence configuration XML elements.

What is a Scheme?



Coherence defines a couple of different scheme types:

- Cache schemes
- Invocation schemes
- Proxy scheme



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

What is a Scheme?

A scheme captures configuration information about a specific type of resource. Resources defined by schemes only become realized when they are requested by an application. A scheme is like a template (cookie cutter). It does not become realized until a cache that is mapped to it is requested. A scheme can be inherited or “reused” to define another specialized scheme, such as a LocalCache scheme being used as a backing map scheme for a DistributedCache.

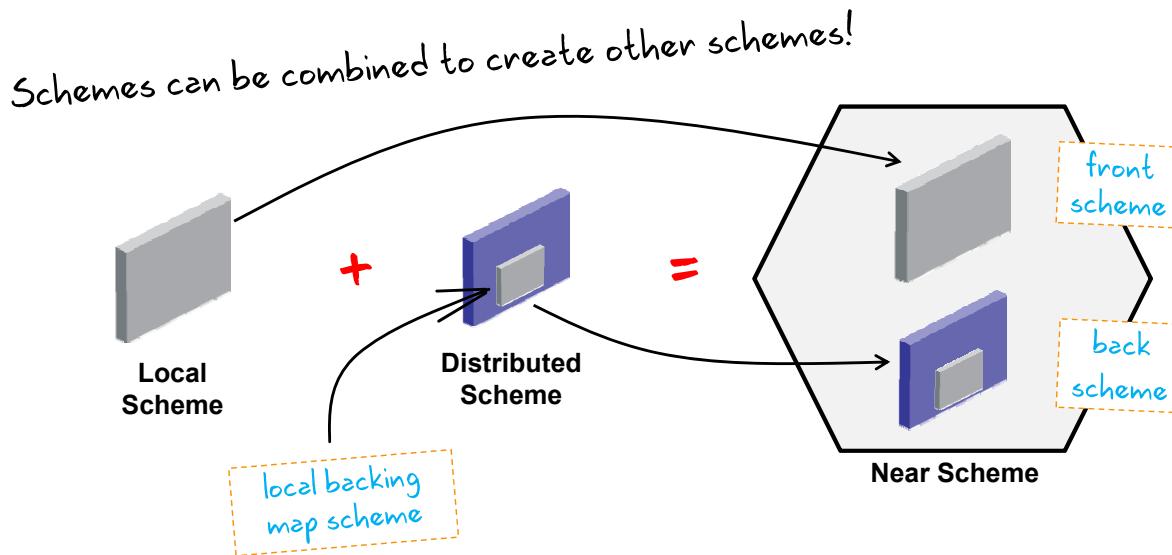
Coherence defines a few different types of schemes for different functions:

- Cache schemes to define the various different types of caches that Coherence supports.
- Invocation schemes to define an invocation service that is used to perform custom operations in parallel on any number of Coherence nodes.
- Proxy scheme which defines the connection acceptance point and behavior for Coherence*Extend clients to join the TCMP cluster.

Review the documentation to see the other types of schemes that Coherence provides.

Scheme Composition

The act of combining two or more caching schemes:



ORACLE

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

Scheme Composition

Most schemes are composed of other schemes.

For example:

The `<near-scheme>` requires a `<front-scheme>` (for "local" caching) and a `<back-scheme>` (for the "back" tier caching).

The `<distributed-scheme>` requires a `<backing-map-scheme>` to specify how primary data will be stored (locally).

Almost every part of Coherence may be replaced (with another scheme or a class-scheme).

The possibilities for Coherence Cache Configuration are essentially limitless.

Declaring Cache Mappings

Declare your caches and what schemes they use:

ADVICE: Lock your caches down by **NOT** using wildcards!

```

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>MyLocalCache</cache-name>
      <scheme-name>MyLocalCachingScheme</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <local-scheme>
      <scheme-name>MyLocalCachingScheme</scheme-name>
    </local-scheme>
  </caching-schemes>
</cache-config>
```

Multiple cache-mappings can refer to the same scheme! They just use different cache names.

coherence-cache-config.xml



ORACLE

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

Declaring Cache Mappings

As mentioned previously, a scheme does not become realized until a cache that is mapped to it is requested. A single scheme can be used by multiple cache mappings, which facilitates reuse of a scheme for different caches.

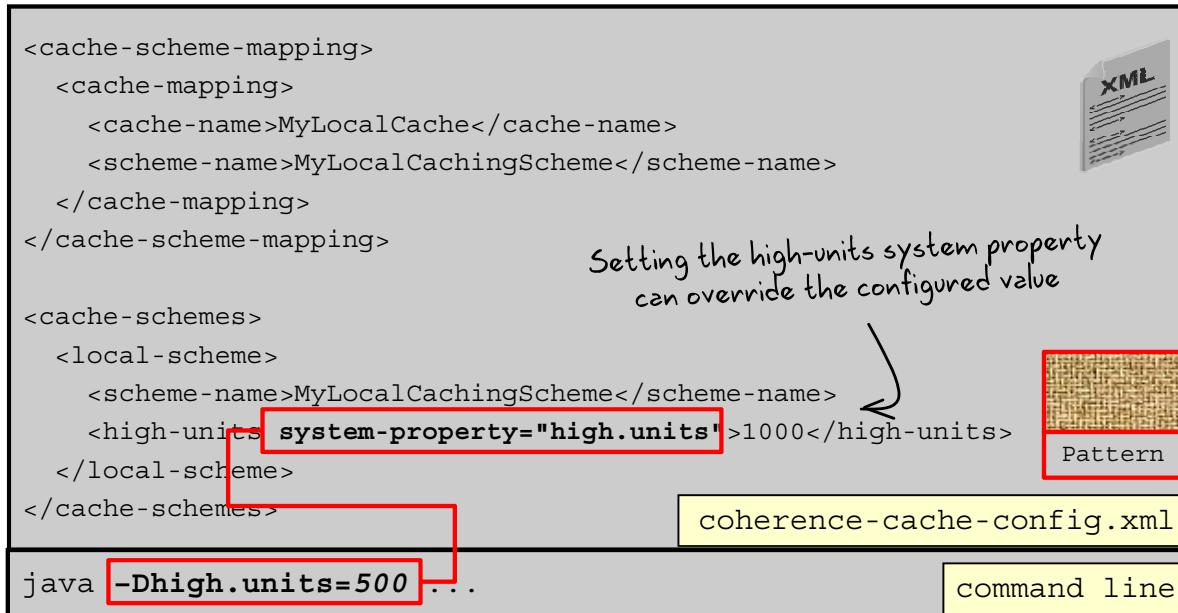
For example, two caches (passenger and baggage) which are both constructed using the same scheme:

1. The MyLocalCache cache mapping in this slide uses the MyLocalCachingScheme local-scheme.
2. Create another cache mapping for passenger and baggage caches by copying the MyLocalCache cache mapping stanza twice, and provide each new cache mapping configuration with the cache-names passenger and baggage respectively.

Now that you know the basic structure of the coherence-cache-config.xml file let's take a closer look at the configuration of a local cache.

Using System Properties to Override XML Elements

System property overriding example:



ORACLE

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

Using System Properties to Override XML Elements

Coherence provides a way to override configuration values using system properties. This slide shows the same local cache example with a `<high-units>` parameter configured. A closer look at the `<high-units>` element shows that an XML attribute is configured that reads `system-property="high.units"`. This allows overriding of hard-coded configuration values with the Java command line using `-Dhigh.units=value`. The "high.units" property name in this example is user-defined. There is no requirement to call the parameter "high.units". The property name can be anything, and Coherence just substitutes the values.

Using Macro Parameters to Override XML Elements

```

<cache-scheme-mapping>
  <cache-mapping>
    <cache-name>MyLocalCache</cache-name>
    <scheme-name>MyLocalCachingScheme</scheme-name>
    <init-params>
      <init-param>
        <param-name>back-size-limit</param-name>
        <param-value>1000</param-name>
      </init-param>
    </init-params>
  </cache-mapping>
</cache-scheme-mapping>

<cache-schemes>
  <local-scheme>
    <scheme-name>MyLocalCachingScheme</scheme-name>
    <eviction-policy>HYBRID</eviction-policy>
    <high-units>{back-size-limit 0}</high-units>
    <expiry-delay>{back-expiry 1h}</expiry-delay>
    <flush-delay>1m</flush-delay>
    <cachestore-scheme></cachestore-scheme>
  </local-scheme>
</cache-schemes>

```

coherence-cache-config.xml



Uses hybrid size limitation and expiry eviction policies for removing data from the cache



Pattern

ORACLE

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

Using Macro Parameters to Override XML Elements

This is an example of a local cache scheme definition and cache mapping with a few more parameters. The {back-size-limit 0} and {back-expiry 1h} values are macro parameters. Each may be defined somewhere else in the configuration file, usually within the cache-mapping element. If either is not defined, then the value on the right of the macro serves as its default value.

In this example, the MyLocalCache configuration defines the back-size-limit initialization parameter, and assigns it a value of 1000. However, it does not define the back-expiry parameter. So when the cache is initialized, it will have a back-size-limit of 1000, which limits the cache to 1000 entries, and the back-expiry parameter will default to 1h, which gives all unaccessed cache entries one hour to exist in the cache before they expire.

Backing Maps

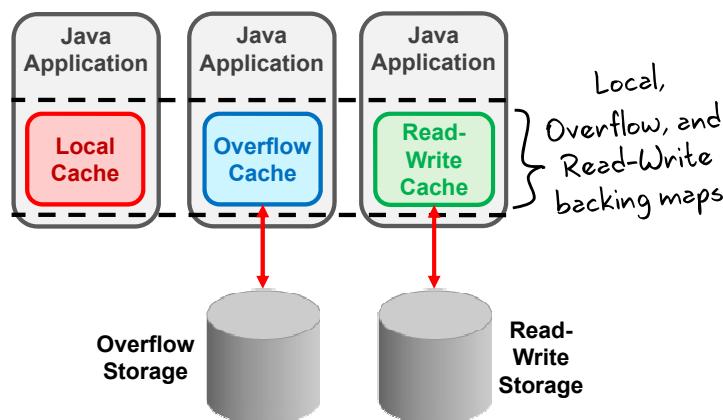
Q: What is a backing map?

A: It is the set of data structures that control where data is stored.

- All caches are backed by a map.
- Configure using the `backing-map-scheme` element.

Backing map types:

- LocalCache
- Read-Write
- Overflow
- External
- Paged External



ORACLE

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

Backing Maps

- LocalCache is the recommended backing map in most situations.
- Backing a cache with `read-write-backing-map` allows for storing data in memory and persisting it via a `CacheStore`.
- An `overflow` backing map is a two-tier backing map with a fast in-memory front scheme backed by a slower back scheme on disk. You can think of this backing map like banking overdraft protection for a checking account. When the front tier's memory is full, the back tier is used to hold the data that spills over the front tier's capacity. This is meant for temporary situations as an insurance policy against data loss, not for storing larger data sets on the back tier for regular usage. A `read-write` backing map is much better for that purpose. The difference between a `read-write` and `overflow` backing map is that the `overflow` back tier is managed by Coherence in binary form and the `read-write` back tier is managed by a `CacheStore` implementation that can work with any data store.

Backing Maps (continued)

No Longer Used: These backing map types were intended for avoid problems with Java garbage collection and heap size limitations. As of Java 6, this is no longer an issue as garbage collection and larger heap support are improved.

- An External backing map provides the ability to store cache data off the JVM heap. This greatly increases the storage capacity by separating the data storage from the heap limitations of a JVM. Several out of the box external backing map storage strategies exist: NIO Memory Manager, NIO File Manager, and Berkely Database Store Manager. You can also write your own custom implementation.
- A Paged External backing map is like the External backing map, except it uses paging to optimize the Least Recently Used (LRU) eviction process. This process involves breaking up the backing store into separate pages. The most recent page is the current page and is written to until a new page is created. The configured number of pages and time to create pages establishes how long data stays in the cache. Once the page count is reached, the items in the oldest page are evicted from the cache one page at a time.

Agenda

Coherence Cache Topologies

- What Happens to Cached Data?
- Cache Configuration Concepts
- **Topologies**
 - Local Cache
 - Replicated Cache
 - Partitioned Cache
 - Near Cache
 - Overflow Cache
 - Other Cache Types, Topologies Summary, and Advanced Configurations

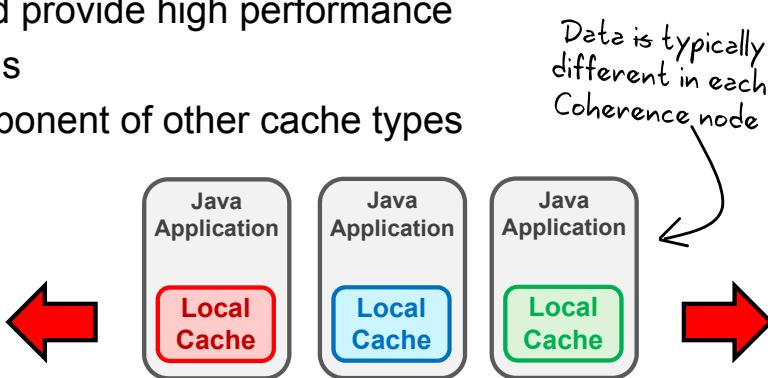


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is a Local Cache?

Local caches:

- Are completely contained within a Coherence node
- Are not clustered (Data is isolated and thus may be different in each Coherence node.)
- Entries are maintained on-heap
- Entries may be loaded from or stored to a cache store
- Are thread safe and provide high performance
- Support notifications
- Are used as a component of other cache types
- Support eviction:
 - LFU
 - LRU
 - Hybrid
 - Custom



ORACLE

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

What is a Local Cache?

A local cache is an in-memory cache that remains local to the Coherence node. Each Coherence node in the cluster contains their own copy of the local cache, but the data contained in each node is different from the data in the other nodes. The Coherence nodes are not aware of what data is contained in the other cluster members. A local cache can be backed by a data store. When a cache is backed by a data store, requests for data that does not exist in the cache is then loaded from the data store into the cache, and then returned to the calling application. The local cache implementation is thread safe and allows for highly concurrent use among multiple threads.

Local caches support the Coherence event notification mechanism, which allows for applications to react to data inserts, updates, and deletes. Coherence events are covered in the lesson titled “Observing Data Grid Events.”

Local caches are used by other cache types to store actual data. Local caches support evicting data from the cache: Least Frequently Used (LFU), Least Recently Used (LRU), a hybrid of the two, or a custom eviction implementation.

Objects stored in a local cache are stored by reference.

When To Use a Local Cache

Use a local cache when:

- Fault tolerance is not required.
- Instant read and write performance is required.
- All entries may be managed on-heap.
- Application can tolerate stale data.
- Used with other cache types:
 - Examples: Replicated, Partitioned, and Near Caches

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
No	Very fast	Very fast	All entries may be managed on-heap	No



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

When To Use a Local Cache

A local cache is useful for typical application caching to avoid the performance hit of going to a data store for frequently repeated reads and writes. A local cache is a good fit for an application where the data can simply be read in again if the data is lost, and it is not important if other application members are aware of the same data.

When a local cache is used, the capacity of the cluster is only as great as ***the number of nodes running in the cluster * the amount of RAM on each node***. If the data size requirements of the application get too large, then the number of nodes required to hold the data grows rapidly. This quickly breaks down the ability to predictably scale the application once the number of machines, and the related costs, grow to support the data set. As a result, a local cache is best used for small to somewhat large data sets, but used with caution on larger data sets.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

When To Use a Local Cache (continued)

A Coherence node using a local cache is not aware of the data in the local cache of another Coherence node. As a result, when data is updated in one cache, the same data object stored in another cache contains stale data. The Coherence event listener feature can get around this, but by default this is the behavior. Events are discussed in the lesson titled “Observing Data Grid Events.”

Additionally, the local cache is typically used by other cache types, such as with replicated, partitioned, and near caches. These caches provide services above and beyond what the local cache offers, but still store data within a local cache on each Coherence node where the data resides.

When To Not Use a Local Cache

Do not use a local cache when:

- Fault tolerance is required.
- Information needs to be consistent across all servers.
- The cache data size exceeds the available heap size.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
No	Very fast	Very fast	All entries may be managed on-heap	No



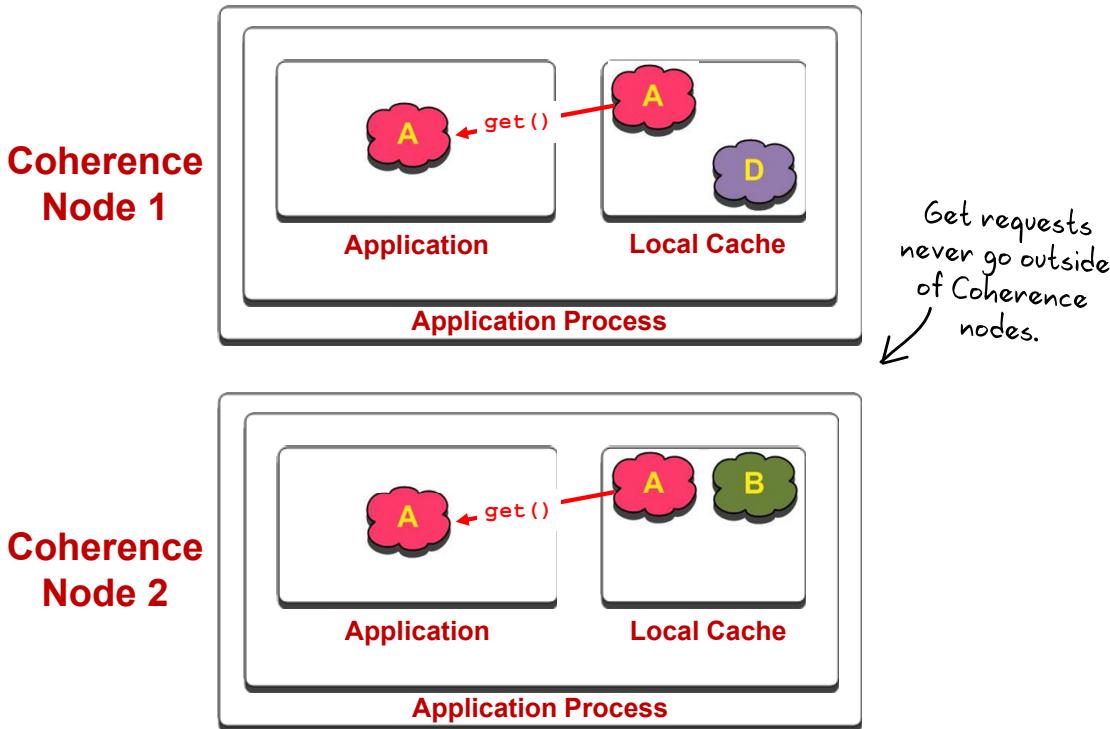
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

When To Not Use a Local Cache

A local cache should not be used alone if the data must survive the loss of a cluster instance, or if the data in the cache needs to be available and consistent across all Coherence nodes. This is also important for predictable scalability. If there are five Coherence nodes, and each one contains a copy of the cached data, then the application can only scale as high as the memory available across all five Coherence nodes. If the cached data set is going to be very large, then a different cache type will fit better.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

Local Cache get Diagram



ORACLE

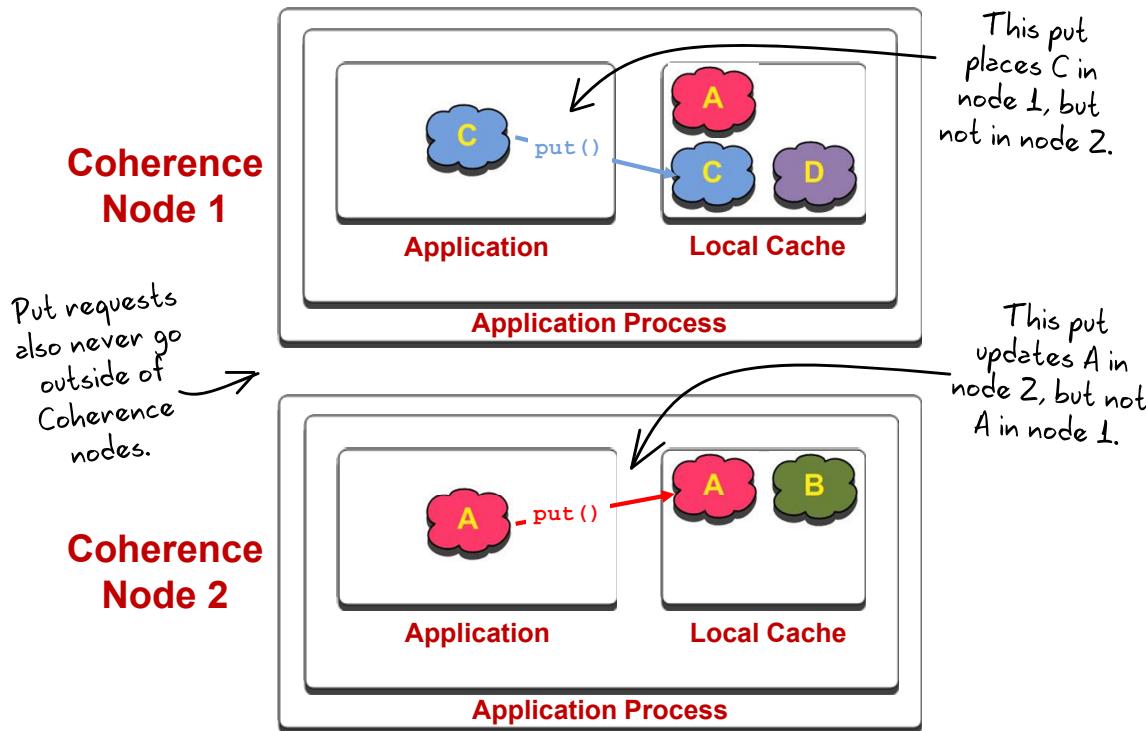
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Local Cache get Diagram

This diagram shows two Coherence nodes that each use the same local cache configuration:

- Node 1's local cache has objects A and D in its cache.
- Node 2's local cache has objects A and B in its cache.
- When the application in each node performs a `get ('A')` operation using Coherence, a local copy of A is returned.
- Although the diagram shows that node 1 and node 2 both have a copy of A in their local caches, neither node is aware of the data in the other node's cache, and the two nodes never communicate as part of the `get ('A')` operation.

Local Cache put Diagram



ORACLE

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

Local Cache put Diagram

This diagram shows the same two JVM processes, but this time the applications are performing a `put()` operation:

- Initially, node 1's local cache has objects A and D in its cache.
- Initially, node 2's local cache has objects A and B in its cache.
- When node 1 performs a `put('C')` operation, the C object does not already exist in its local cache, so it is inserted as a new object in the cache. Nothing happens on node 2.
- When node 2 performs a `put('A')` operation, the A object already exists in its local cache, so the object in the cache is updated with the new object that is being 'put' into the cache. Nothing happens on node 1, and object A stored in its cache is now **stale**.
- The two nodes never communicate as part of the `put()` operation.

Configuring a Local Cache

Example:

```

<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>MyLocalCache</cache-name>
      <scheme-name>MyLocalCachingScheme</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <local-scheme>
      <scheme-name>MyLocalCachingScheme</scheme-name>
    </local-scheme>
  </caching-schemes>
</cache-config>

```

coherence-cache-config.xml



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

Configuring a Local Cache

This is an example of a simple local cache configuration.

When the application makes a call to `getCache ("MyLocalCache")`:

1. A cache named `MyLocalCache` must be configured using the `<cache-name>` element within the `<cache-mapping>` element.
2. The `MyLocalCache` mapping configuration refers to a cache scheme definition named `MyLocalCachingScheme` using the `<scheme-name>` element.
3. The `MyLocalCachingScheme` definition is configured within the `<caching-schemes>` element, and is:
 - a. Defined as a local caching scheme using the `<local-scheme>` element
 - b. Identified by the name `MyLocalCachingScheme`, using the `<scheme-name>` element

Defining Local Cache Parameters

Parameter	Description
local-scheme	Specifies that this scheme is a local scheme type
scheme-name	Specifies the name of the scheme. Must be unique
scheme-ref	Specifies the name of another scheme to inherit for this scheme
service-name	Specifies the service that manages caches created with this scheme
eviction-policy	Specifies the type of eviction to perform: LRU, LFU, HYBRID, custom
high-units	Limits the size of the cache to a number of entries
low-units	The size a cache is trimmed to when expiring data from the cache
expiry-delay	Causes cache entries to expire if not accessed within a time interval
flush-delay	Specifies a time interval after which all expired cache entries are removed
cachestore-scheme	Indicates a cache is backed by an external data source



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

Defining Local Cache Parameters

This slide shows some of the most common parameters that can be used with a local cache scheme. For information about all possible parameters see the *Coherence Developer Guide*.

How `expiry-delay` and `flush-delay` work:

Expiry refers to the period when an entry in a cache is *visible* to an application. For example, when an entry has an expiry time of 5 seconds, that means it will be available to applications for a period of five seconds since it was last accessed. The catch here is that `expiry-time` only refers to *visibility* of an entry and not whether it is actually removed from the cache. Expired entries are only removed from a cache when required. They are not removed automatically. *Flushing* is what removes expired entries from a cache.

When an application attempts to read an entry that has expired, from the perspective of the application, that entry does not exist and thus nothing (null) will be returned. If a `CacheLoader/CacheStore` is configured, then an attempt to read the entry from the `CacheLoader/CacheStore` will be made.

Defining Local Cache Parameters (continued)

Why aren't expired entries removed immediately when they expire? It is because often removal is an expensive operation. Coherence would have to "watch and/or scan" every entry in a cache constantly to ensure that expired entries are removed. Instead, Coherence is lazy, and only actually removes entries from a cache when they are *flushed*. Thus the `flush-delay` is the period of time between attempts to clean up *expired* entries.

That said, if a Cache becomes full (assuming that there is a high-units specified), Coherence will automatically begin flushing.

Quiz

True or False? Coherence reads its configuration from a central registry that must be installed on a separate machine.

- a. True
- b. False



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

Answer: b

Coherence gets its configuration from XML files. Coherence has no central registry or central server. That being said, a URI can also be used to specify the location of Coherence XML configuration files. This allows the files to be hosted anywhere that can be accessed via a URI.

Quiz

What is a Coherence scheme?

- a. A mechanism to map caches to templates using the `<cache-mapping>` element
- b. A template that defines a caching topology using an element within the `<cache-schemes>` element
- c. A reusable named cache that uses the partitioned topology
- d. A template that maps caches to a named cache using the `<map-to-cache>` element



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

Answer: b

A Coherence scheme is like a cookie cutter; a template that defines a caching topology which can have caches mapped to it using the `<cache-mapping>` element.

Quiz

Which of the following is a characteristic of a local cache?

- a. In-memory cache that is shared by all Coherence nodes
- b. In-memory cache that is backed up on another Coherence node
- c. In-memory cache that contains the same data on each Coherence node
- d. In-memory cache whereby entries may be different on each Coherence node



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

Answer: d

A local cache is an in-memory cache that exists on each Coherence node, and the data in the cache may be different for each node.

Practice.04.01 Overview: Configuring a Local Cache Scheme

This practice covers the following topics:

- Configuring a local cache scheme
- Running some clients that form a Coherence cluster that uses your local cache scheme
- Viewing the results of how and where data is stored in the cluster when using a local cache



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

Practice.04.01 Overview: Configuring a Local Cache Scheme

This practice demonstrates configuring a local cache, running two client applications that serve as Coherence cluster members that use the configuration, and analyzing the output of the clients to see how data is stored and retrieved from local caches in a cluster.

Agenda

Coherence Cache Topologies

- What Happens to Cached Data?
- Cache Configuration Concepts
- **Topologies**
 - Local Cache
 - **Replicated Cache**
 - Partitioned Cache
 - Near Cache
 - Overflow Cache
 - Other Cache Types, Topologies Summary, and Advanced Configurations

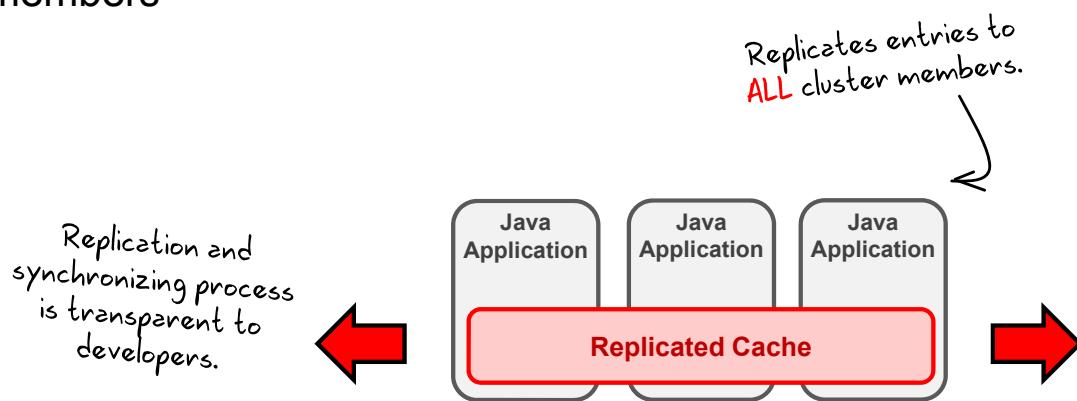


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is a Replicated Cache?

A replicated cache:

- Is the simplest form of clustered caching
- Is designed for fast read performance
- Provides zero latency access as all data is local to members



ORACLE

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

What is a Replicated Cache?

The replicated scheme is the first topology introduced by Coherence. It was an instant success due to its ability to handle data replication, concurrency control, and failover in a cluster, all while delivering in-memory data access speeds. A replicated cache replicates its data to all Coherence nodes. The replicated topology supports reads at in-memory speeds (100,000+ reads per second or more).

Whenever you write an object into a local JVM using `put()`, it gets replicated to all the other JVMs on the cluster. The data is stored locally on every JVM and every JVM has a copy of all the data. Applications retrieve the object in memory with near zero latency. A local copy is stored and there is no serialization or over-the-network access involved *for reads*. When the application requests an object, it gets the object from the local JVM.

The lesson titled “Getting Started with Oracle Coherence” introduced the concept of a service. At runtime, Coherence may have many instances of these services running, each of which is uniquely named. Some caches require a service, such as the replicated cache. To create a replicated cache, you can configure a service name, or let it use the default.

When To Use a Replicated Cache

Use a replicated cache when:

- Fault tolerance is needed.
- Application is read-only or read-mostly.
- All entries may be managed on-heap.
- Application cannot tolerate stale data.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
Yes	Very fast	Expensive (not scalable)	All entries may be managed on-heap	Yes



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

When To Use a Replicated Cache

The replicated cache uses multiple JVMs to achieve fault tolerance and data consistency. Whenever an application writes an object into a local JVM using `put()`, it gets replicated to all the other JVMs on the cluster. The data is stored locally on every JVM and every JVM has a copy of all the data.

The replicated topology supports reads at in-memory speeds (100,000 reads per second or more). Applications retrieve the object in memory with near zero latency. When data is replicated to a node in the cluster, it is stored in serialized form at first. When the first read occurs, the data is deserialized and the native object is swapped into the cache as a local copy. Subsequent reads require no serialization or over-the-network access. When the application requests an object, it gets the object from the local JVM.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

When To Use a Replicated Cache (continued)

Conversely, the replicated scheme is useful only for small caches that fit inside one JVM. Therefore, if one JVM is configured with a 1 GB heap, it is limited to a few hundred MB of data with a replicated cache. It is very good for read-intensive systems. So if the application has relatively small caches with very intense reads (hundreds of thousands of reads per second), the replicated topology will suit its needs. However, it scales poorly for writes, just as the replication topology of other products. Every time the application performs a write, Coherence has to write data to every JVM that is in the cluster. Therefore, this scheme type is limited as more Coherence nodes are added to the cluster. Each new cluster node adds an extra write operation for a replicated cache. As the number of nodes increases, the network traffic and CPU bandwidth of the cluster becomes busier. This limits the scalability of a replicated cache.

When To Not Use a Replicated Cache

Do not use a replicated cache when:

- Support of heavy writing is needed.
- The cache data size exceeds the available heap size.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
Yes	Very fast	Expensive (not scalable)	All entries may be managed on-heap	Yes



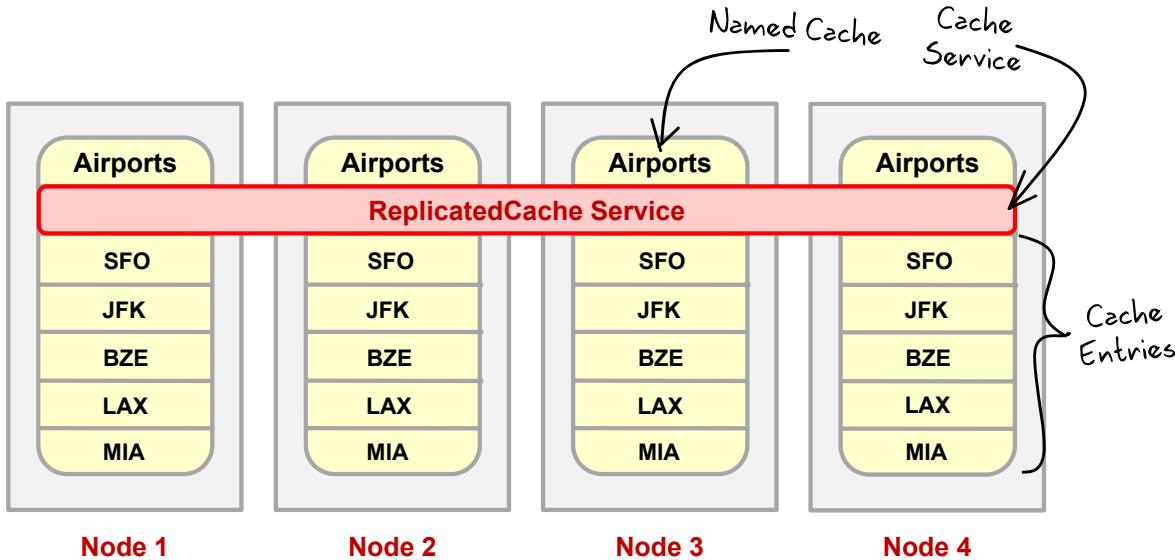
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

When To Not Use a Replicated Cache

Because the replicated cache performs all writes to all cluster members, it is not the best fit for applications that perform a lot of updates. The replicated cache is also not a good fit for large cache sizes.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

Visualizing Data in a Replicated Cache



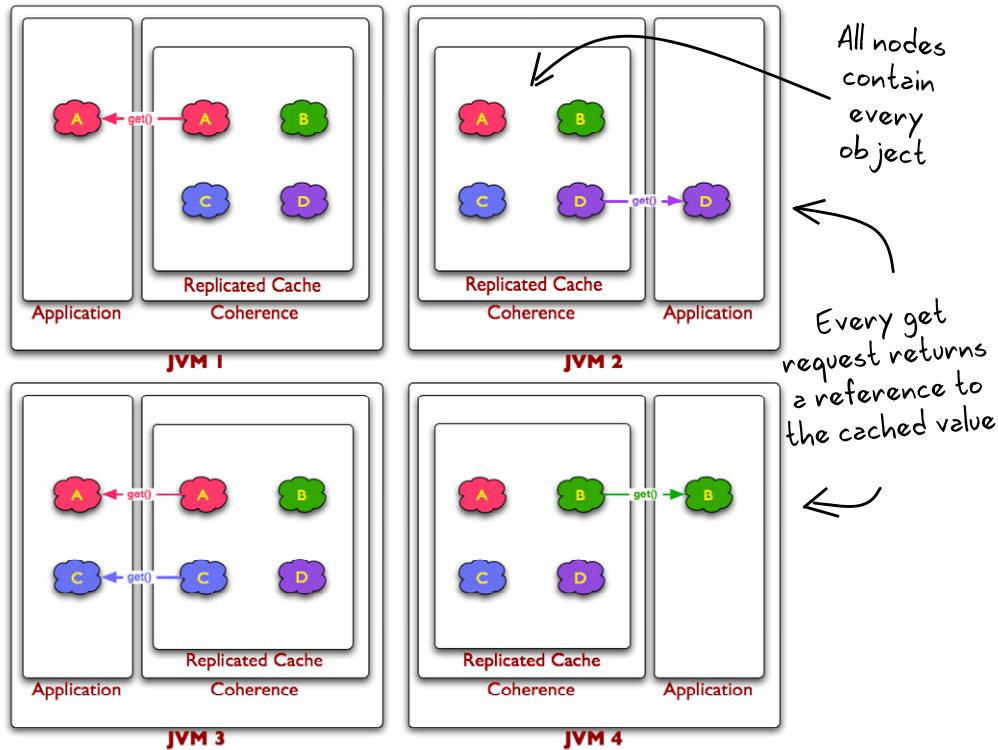
ORACLE

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

Visualizing Data in a Replicated Cache

This diagram shows how the replicated cache stores the same data in the Airports NamedCache consistently across all Coherence nodes. The ReplicatedCache Service manages the data across all Coherence cluster members. The ReplicatedCache service is responsible for ensuring that cache entries are consistent across all Coherence nodes.

Replicated Cache get Diagram



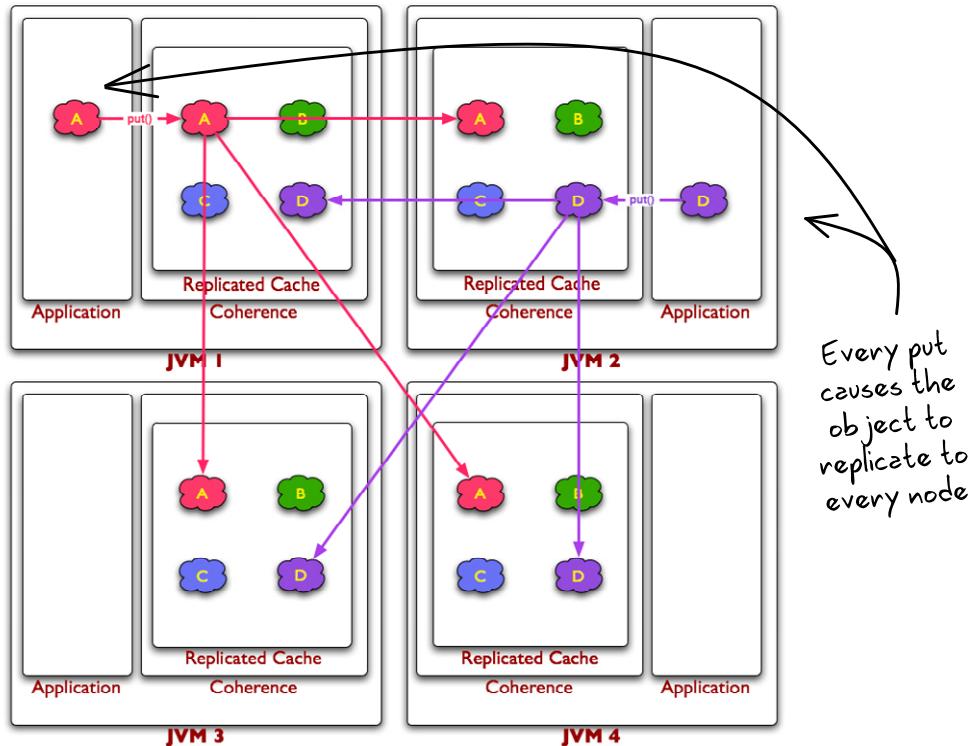
ORACLE

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

Replicated Cache get Diagram

This diagram shows four JVMs running as Coherence replicated cache nodes. Again, it shows that all nodes have all objects stored locally: A, B, C, and D. When a `get()` is performed by the application within any of the JVMs, a reference to the cached entry is returned to the application immediately. Keep in mind that the data in every node is kept in sync by the `ReplicatedCache` service.

Replicated Cache put Diagram



ORACLE

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

Replicated Cache put Diagram

This diagram shows the same Coherence cluster that was shown on the previous slide, but instead of showing the interaction of the `get()` API, it shows how the `put()` operation works with a replicated cache. When a `put()` is called by the application within any of the JVMs, the data that is managed by the `ReplicatedCache` service must be replicated on every Coherence node that has joined the service. This means that memory utilization (the Java heap size) is increased for each Coherence node, which can negatively impact performance. Additionally, replicated caches with a high incidence of updates will not scale linearly as the cluster grows. Conversely, the cluster suffers diminishing returns as Coherence node are added. When a `put()` is performed, the object is stored natively on the cluster node where the `put()` occurred. Then the node serializes the data and synchronously sends it to the cluster leader. At this point, the `put()` command returns control to the calling application. Behind the scenes, the cluster leader asynchronously sends the data to be stored on all other cluster nodes in serialized form.

Configuring a Replicated Cache

Example:

The diagram shows the XML configuration file `coherence-cache-config.xml`. It defines a replicated cache scheme named `MyReplicatedScheme` with service name `ReplicatedCache`. This scheme uses a `backing-map-scheme` which refers to a local scheme named `MyLocalCachingScheme`. A handwritten note with an arrow points to this reference with the text "Inherit properties from this local scheme". To the right of the XML code, there is a graphic representation: a grey rectangle labeled "XML" contains a smaller red-bordered box labeled "Pattern". Below the XML code, a yellow box contains the file name "coherence-cache-config.xml".

```

<caching-schemes>
  <replicated-scheme>
    <scheme-name>MyReplicatedScheme</scheme-name>
    <service-name>ReplicatedCache</service-name>

    <backing-map-scheme>
      <local-scheme>
        <scheme-ref>MyLocalCachingScheme</scheme-ref>
      </local-scheme>
    </backing-map-scheme>

    <autostart>true</autostart>
  </replicated-scheme>

  <local-scheme>
    <scheme-name>MyLocalCachingScheme</scheme-name>
  </local-scheme>
</caching-schemes>

```

ORACLE

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

Configuring a Replicated Cache

Here is a simple example of a replicated cache configuration. To conserve space, only the `<caching-scheme>` section is shown.

To declare a replicated cache:

- Use the `<replicated-scheme>` to indicate the scheme is for a replicated cache.
- Just as with the local cache scheme, a scheme name is specified that is used by a `<cache-mapping>` element to create an instance of this replicated cache.
- The `<service-name>` element is an optional parameter used to name the service that manages this cache. By default, the replicated cache service is named `ReplicatedCache`.
- The `<backing-map-scheme>` element specifies what backing map is used by this scheme to store data on each node. A `LocalCache` scheme is defined using the `<local-scheme>` element. The `<scheme-ref>` element specifies that this replicated cache will inherit and reuse the local cache scheme named `MyLocalCachingScheme`.
- The `<autostart>` element instructs Coherence to start up the service by default when the node starts. The autostart is only for the `DefaultCacheServer`, otherwise it is ignored and services start when requested.

Defining Replicated Cache Parameters

Parameter	Description
replicated-scheme	Specifies that this is a replicated scheme type
service-name	Specifies the name of the service which will manage caches created from this scheme
backing-map-scheme	Specifies what type of Map will be used within the cache server to store the entries
scheme-ref	Specifies the name of another scheme to inherit
autostart	Specifies whether or not the cache services associated with this cache scheme should be started automatically



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

Defining Replicated Cache Parameters

This slide shows some of the most common parameters that can be used with a replicated cache scheme. For information about all possible parameters see the *Coherence Developer Guide*.

Quiz

A replicated cache stores data on:

- a. Every node in the cluster
- b. Half of the nodes in the cluster
- c. The master node with read-only copies on other nodes
- d. One node in the cluster and one copy on disk



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

Answer: a

A replicated cache is designed to store cache entries on every node in the cluster.

Quiz

What are the primary reasons for using a replicated cache?
(Select all that apply)

- a. High performance writes
- b. High performance reads
- c. Data consistency across the cluster
- d. High performance replication



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

Answer: b, c

A replicated cache is used for extremely fast reads and since the data is automatically replicated across the entire cluster, it is good for data consistency.

Quiz

True or False? A replicated cache is perfect for an overall data set of 500GB.

- a. True
- b. False



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

Answer: b

A replicated cache limits the overall data size to the largest size of a single JVM heap. Usually this is a useful maximum of 4GB on some high-bit architecture systems.

Quiz

What stores the actual data in a Coherence cache?

- a. Backing map
- b. Partitioned cache
- c. Database
- d. Replicated cache



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

Answer: a

A backing map is the set of data structures that manages the data behind a caching topology.

Practice.04.02 Overview: Configuring a Replicated Cache Scheme

This practice covers the following topics:

- Configuring a replicated cache scheme
- Running some clients that form a Coherence cluster that uses your replicated cache scheme
- Viewing the results of how and where data is stored in the cluster when using a replicated cache



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

Practice.04.02 Overview: Configuring a Replicated Cache Scheme

This practice demonstrates configuring a replicated cache, running two client applications that serve as Coherence cluster members that use the configuration, and analyzing the output of the clients to see how data is stored and retrieved from replicated caches in a cluster.

Agenda

Coherence Cache Topologies

- What Happens to Cached Data?
- Cache Configuration Concepts
- **Topologies**
 - Local Cache
 - Replicated Cache
 - **Partitioned Cache**
 - Near Cache
 - Overflow Cache
 - Other Cache Types, Topologies Summary, and Advanced Configurations

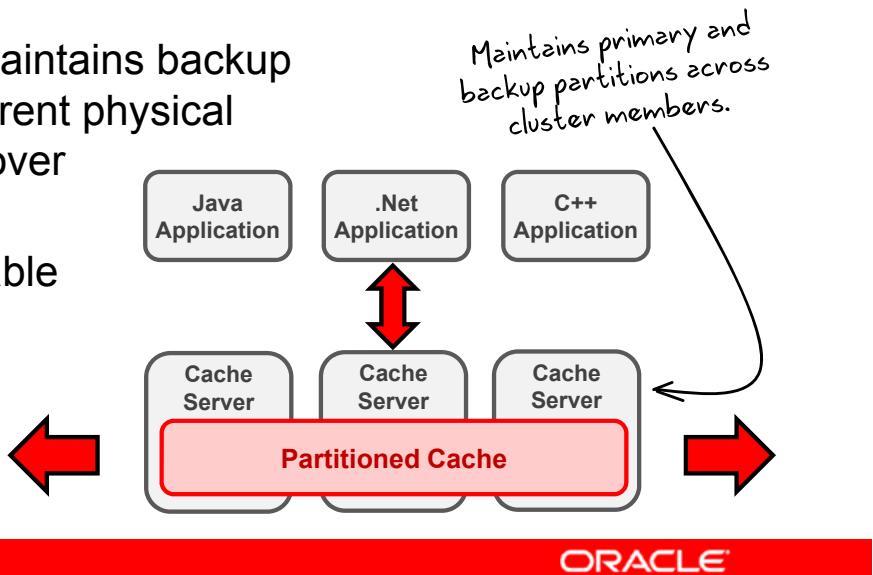


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

What is a Partitioned Cache?

A partitioned cache:

- Organizes cache entries into partitions
- Evenly balances the partitions across members of the cluster
- Synchronously maintains backup partitions on different physical machines for failover and recovery
- Provides predictable performance and scalability



ORACLE

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

What is a Partitioned Cache?

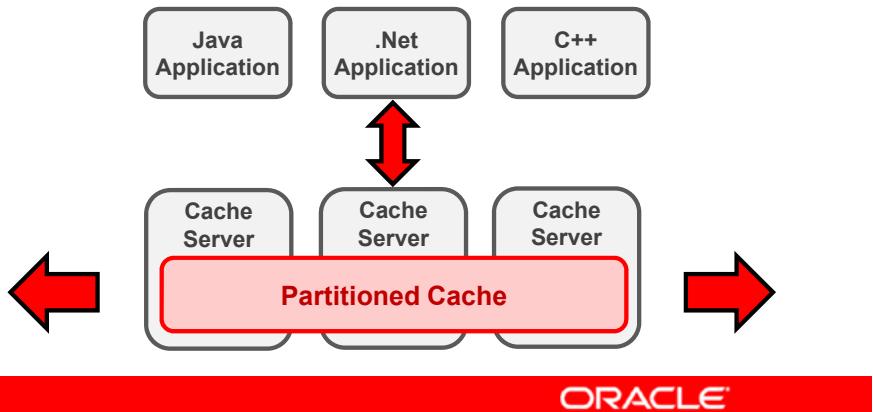
A partitioned cache partitions the overall data set into smaller subsets, called partitions, that are spread across the entire cluster. Each node in the cluster holds its fair share of partitions. As a result, the partitioned cache can handle much larger data sets than a replicated cache, or what can be held in a single JVM. The partitioned cache also creates a backup copy of the partitions and stores them on another physical machine whenever possible for data resiliency. When an object is updated in a primary partition, the backup partition is updated synchronously.

Because data is stored across the cluster, most of the time the primary copy will be managed by a JVM other than the calling application's JVM. This means that most reads will incur a network hop, and most writes will incur two network hops to update the primary and backup copies. Although this is slower than the replicated cache for reads, it is still relatively fast, and provides much faster writing than the replicated cache. Also, because the partitioned cache creates a smaller data footprint per object, the amount of data that it can manage is much greater than a replicated cache, because at all times only two copies of the data exist in the entire cluster.

What is a Partitioned Cache?

Partitioned caches:

- All members manage their fair share of both primary and backup partitions.
- Cache entries are stored and managed in binary form.



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

What is a Partitioned Cache? (continued)

This distributed storage enables the partitioned cache to hold more data, but also makes it predictably scalable and provides predictable performance even as the number of Coherence nodes increases.

One thing to keep in mind with a partitioned cache is that as cluster members join and leave the cluster, the partitioned cache data is automatically rebalanced across the cluster. This ensures that remaining members of the cluster have their fair share of partitions. This process may degrade cluster performance, so it is a good idea to keep cluster members that drop in and out frequently as storage-disabled members.

The API for accessing data in a partitioned cache is the same for all other caches. The implementation of how and where data is stored is completely transparent to the developer and the users. Coherence bases the location of the partitioned data on an entry key, so all cluster members always know exactly where the data resides, which allows Coherence to access the data in a single network hop at most.

Due to its distributed nature, data in a partitioned cache is always stored in serialized form, which introduces more latency for getting and retrieving data. The lesson titled “Working with Objects” covered this in great detail.

When To Use a Partitioned Cache?

Use a partitioned cache when:

- Fault tolerance is needed.
- Predictable read and write performance is needed.
- Data scalability is needed.
- Large data set support is needed.
- Application cannot tolerate stale data.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
Yes	Fast	Inexpensive (scalable)	Larger than a single heap	Yes



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

When To Use a Partitioned Cache?

The partitioned cache uses primary and backup copies of data stored on different physical machines to achieve fault tolerance and data consistency.

Whenever a calling application in a JVM calls `get()`, the primary copy of the data is located on another member of the cluster in most cases, requiring at most one network hop, and the data is deserialized for use in the application. Although this is slower than replicated caches, it is still fast. Coherence provides other caching mechanisms that can be used with a partitioned cache that reduce reading latency, which is covered in the next section. When `put()` is called the primary copy is updated, requiring at most one network hop, and synchronously updates the backup copy, requiring at most one more network hop. As a result, both reads and writes have predictable latency and the partitioned cache topology scales regardless of how many Coherence nodes are added to the cluster.

Because the partitioned cache stores at most two copies of the data within the entire cluster, it is capable of handling extremely large data sets with predictable performance and scalability. As of Coherence 3.5, data sets in the terabyte range are possible.

All applications work with the same primary and backup copies of the data, so the data is always consistent across the cluster.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

When To Not Use a Partitioned Cache?

Do not use a partitioned cache when:

- Fault tolerance is not needed.
- The application is read intensive, but not write intensive, and the data set size is small to medium sized.
- Application can tolerate stale data.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
Yes	Fast	Inexpensive (scalable)	Larger than a single heap	Yes



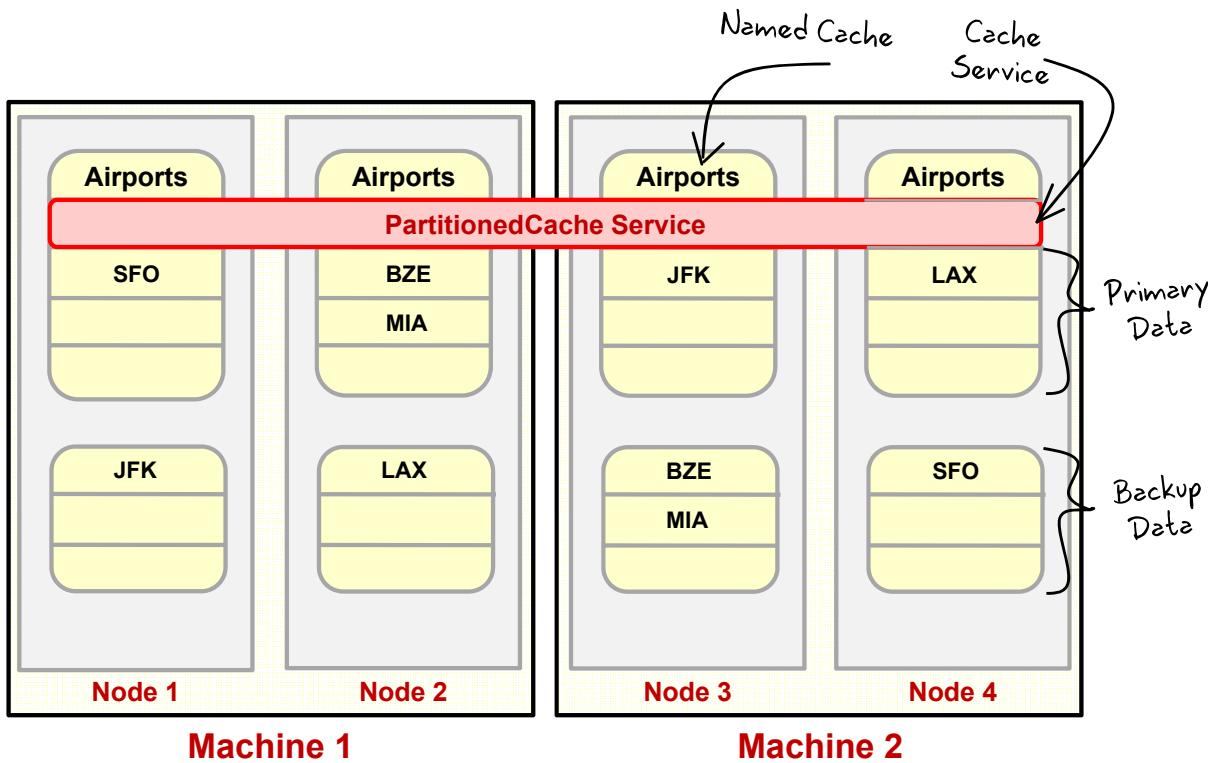
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

When To Not Use a Partitioned Cache?

Because the partitioned cache is designed for fault tolerance, large data sets, and predictable performance and scalability, it is not the best fit for applications that are smaller in nature and do not have these requirements.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

Visualizing Data in a Partitioned Cache



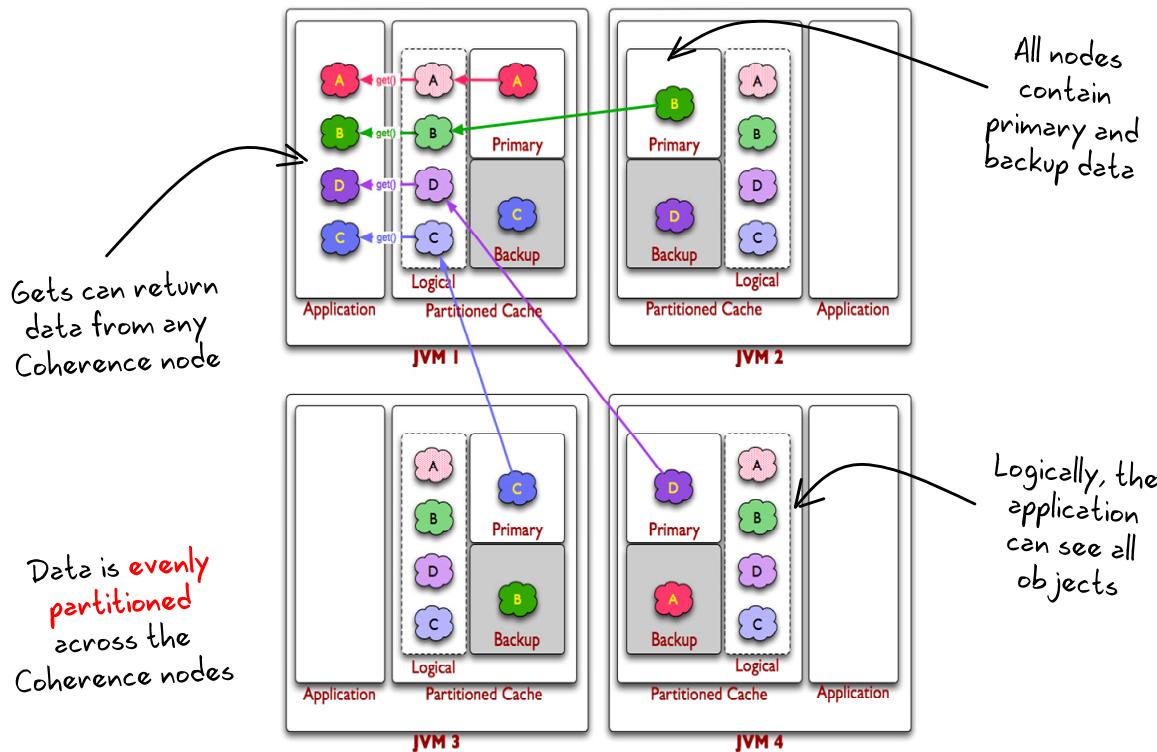
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ORACLE

Visualizing Data in a Partitioned Cache

This diagram shows how the partitioned cache stores the data in the Airports NamedCache in primary and backup copies across the Coherence nodes. Exactly two copies of the data exist in the cluster at one time. Coherence places backup data on a separate machine whenever possible, so that if one machine crashes, the other machine retains a copy of the data. Notice how the data requires less space than a replicated cache to store the same data. The PartitionedCache Service manages the data across all Coherence cluster members. It is important to note that whatever data exists in a primary partition is the same data that exists in the backup partition. Note how Node 2's primary partition contains the BZE and MIA airport codes. Its corresponding backup on Node 3 also contains BZE and MIA.

Partitioned Cache get Diagram



ORACLE

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

Partitioned Cache get Diagram

The following slides on partitioned topology explain how Coherence works and a lot of the functionality provided builds on this concept.

To address the potential scalability limits of the replicated cache service, both in terms of memory and communication bottlenecks, Coherence has provided a partitioned cache service since release 1.2. Coherence defines a partitioned cache as a collection of data that is distributed (or partitioned) across any number of Coherence nodes such that exactly one node in the cluster is responsible for each piece of data in the cache, and the responsibility is distributed (or load-balanced) among the Coherence nodes. The key points to consider about a partitioned cache:

- **Partitioned:** The data in a partitioned cache is spread out over all the servers in such a way that no two servers are responsible for the same piece of cached data. This means that the size of the cache and the processing power associated with the management of the cache can grow linearly with the size of the cluster. Also, it means that operations against the data in the cache can be accomplished with a "single hop," in other words, involving at most one other server.
- **Load-balanced:** Because the data is spread out evenly over the servers, the responsibility for managing the data is automatically load-balanced across the cluster.

Partitioned Cache get Diagram (continued)

- **Location transparency:** Although the data is spread out across Coherence nodes, the same API is used to access the data, and the same behavior is provided by each of the API methods.
This is called location transparency, which means that the developer does not have to code based on the topology of the cache, because the API and its behavior is the same as that of a local JCache, a replicated cache, or a distributed cache.
- **Failover:** All Coherence services provide failover and fallback without any data loss, and this includes the distributed cache service. The distributed cache service allows the number of backups to be configured. As long as the number of backups is one or higher, any Coherence node can fail without loss of data.

For example, objects A, B, C, and D are stored in storage JVMs. Every object has a primary storage JVM. In this example, it is JVM 1 for object A, and a backup storage, which in this example is JVM 4 for object A. When an application makes a request for an object, the code calls `get()` from the Java Collections API. There is no need to know where object A is or to partition the application. When `get()` is called, there is a logical pointer to that data.

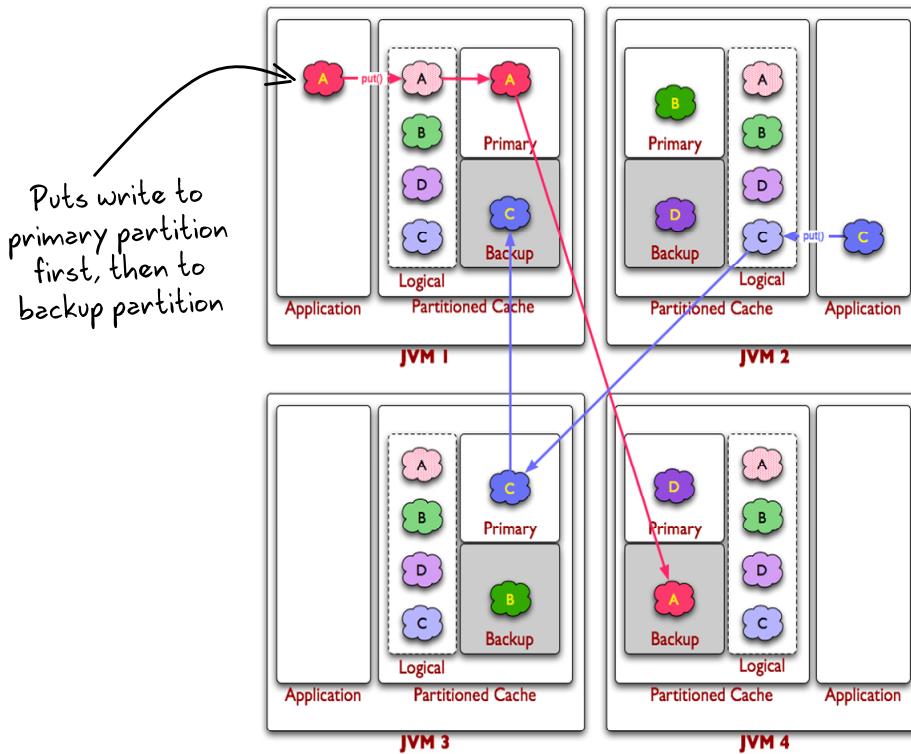
Coherence automatically gets that data out of wherever the JVM is located. When `get()` is called for object B, Coherence automatically gets the object from JVM 2, sends it across the network, deserializes it, and the `get` call returns the object to the calling application.

When `get()` is called for object D, Coherence knows that the object is located on JVM 4, and brings that object into the application. Your application does not need to know where the object is, nor does it have to know anything about how the data is partitioned. Coherence handles that automatically.

This is a peer-to-peer architecture. There is nothing in the middle, there is no central registry, no central server, and no database attached. No lookup is performed with any store or central registry. Every JVM knows its responsibilities and every JVM knows where the objects are located because of the Coherence consensus algorithm. When object D is requested, there is no need to go to a central server. Coherence automatically goes to JVM 4 and gets that data.

This activity requires two network hops. One network hop to get the data and the other network hop to bring it back. Therefore, there are two network hops per read operation. If the number of JVMs is doubled from four JVMs to eight, there are still only two network hops per read operation. This is a peer-to-peer communication across a network to get the object, bring it back, and put the object in your local JVM. This is why Coherence is scalable: the number of network hops and the number of CPU instructions remain constant as JVMs are added. Adding JVMs does not introduce scalability limitations, but increases capacity instead.

Partitioned Cache put Diagram



ORACLE

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

Partitioned Cache put Diagram

Similarly, a cache update operation can utilize the same single-hop point-to-point approach, which addresses one of the two known limitations of a replicated cache: the need to push cache updates to all Coherence nodes. Here the data is being sent to a primary Coherence node and a backup Coherence node. This is for failover purposes, and corresponds to a backup count of one (the default).

If the cache data were not critical, which is to say that it can be reloaded from disk, the backup count can be set to zero, which allows some portion of the distributed cache data to be lost in the event of a Coherence node failure. If the cache were extremely critical, a higher backup count, such as two, can be used. Oracle does not recommend using a backup count greater than one, as it protects from the unlikely scenario of two machines failing at exactly the same instant, and is not used in practice. The backup count affects the performance of only cache modifications, such as those made by adding, changing, or removing cache entries.

Partitioned Cache put Diagram (continued)

Modifications to the cache are not considered complete until all backups have acknowledged receipt of the modification. This means that there is a slight performance penalty for cache modifications when using distributed cache backups; however, it guarantees that if a Coherence node were to unexpectedly fail, data consistency is maintained and no data is lost. This cost however is always consistent, unlike the Replicated Scheme, where the cost of an update is proportional to the size of the cluster.

In this example, when object C is put into the cache:

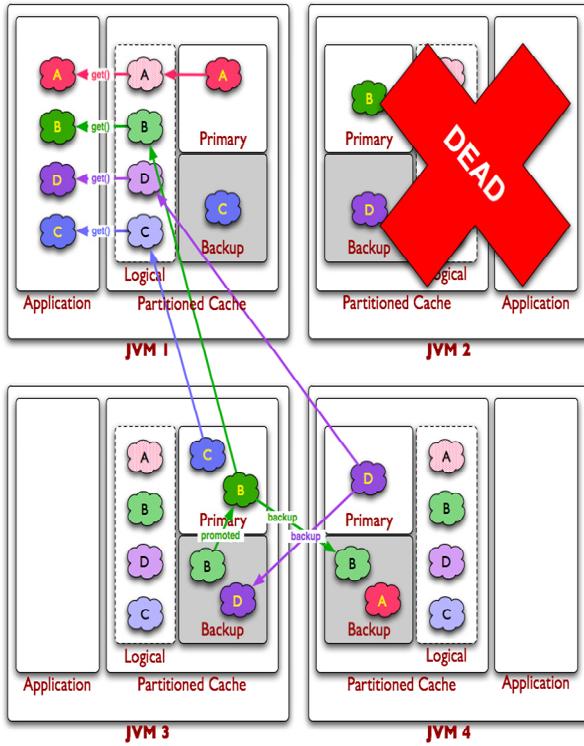
- The data is sent over to the storage JVM 3 process.
- In a synchronous operation, a backup copy is placed on another JVM. By default, it is one backup on another server.

When object A is put into the cache:

- The object is placed in JVM 1 as the primary copy.
- The backup copy is placed in JVM 4.

All Coherence nodes know where the data is through the consensus algorithm. Therefore, there is no need to lock anything. There is no need to lock object A to have a consistent transaction. If another user updates object A, the update queues up. Coherence automatically queues that transaction and executes it in serial order. The application can lock the object using the transactional API provided, but that is not required. There is a consistent view of the data without using locks.

Partitioned Cache Fault Tolerance



- Membership changes (as new members join or leave).
- Recovery and rebalancing of partitions across members occurs in parallel.
- No inflight operations are lost.
- There can be some latencies (due to higher priority of recovery).
- Priorities include reliability, availability, scalability, and performance.
- Performance of some requests get degraded.

ORACLE

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

Partitioned Cache Fault Tolerance

The failover of a partitioned cache involves promoting backup partitions to become primary partitions, and establishing new backup partitions. When a Coherence node fails, the remaining Coherence nodes determine the partition that the failed Coherence node had primary responsibility for, and identify the corresponding partition that is held in backup. This partition becomes the primary responsibility of the Coherence node that was the backup for the partition.

If there are multiple levels of backup, the first backup becomes responsible for the partition, the second backup becomes the new first backup, and so on. Just as with the replicated cache service, lock information is also retained in the case of server failure, with the sole exception being that the locks for the failed Coherence node are automatically released.

Partitioned Cache Fault Tolerance (continued)

What happens when a JVM crashes or a system goes down? The partitioned service automatically rebalances the partitions across the cluster. For example, there are two objects (object B and object D) that are in JVM2, which has just crashed. The Coherence cluster automatically detects, with its death detection mechanism, that a JVM is unavailable. The entire server could also be down. Automatically, Coherence fails over to the backup partition containing the copy of object B, thus converting the backup partition to a primary partition, and then automatically makes another backup partition in another JVM on another server.

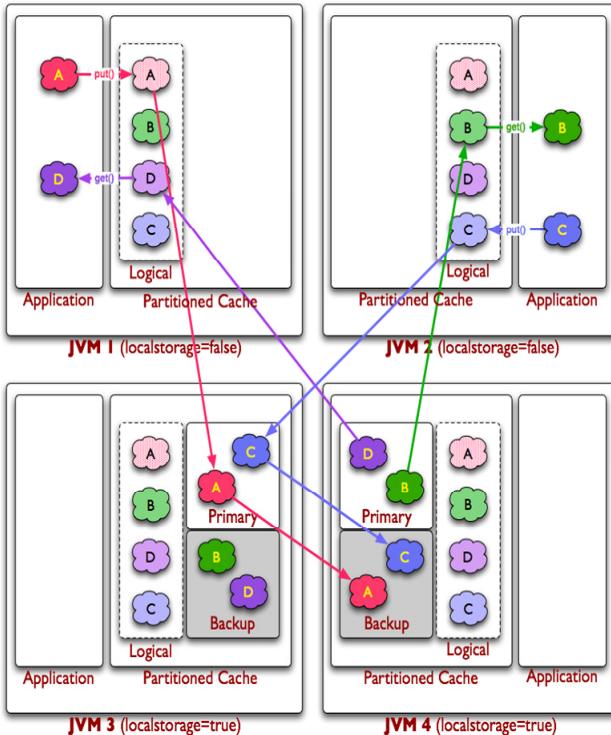
All this is done automatically, with no human intervention or command. When a component is taken out of the system or added to the system, Coherence automatically reconfigures itself. During this period, there is a brief performance lag, especially for queries and writes, because the partitions are being rebalanced.

However, this is neither an interruption to service nor is the application going to hang. The application slows down for a few seconds, but the service does not get interrupted. Also, all these tasks are performed in parallel. Rebalancing is done in parallel or with multiple threads. This happens very quickly. There is:

- No loss of inflight operations while rebalancing
- No requirement to shut down a cluster to:
 - Recover from member failure
 - Add new members
 - Add named caches
- No network exceptions to catch during rebalancing

Dynamic rebalancing allows for scale-out on demand.

Storage Disabled Partitioned Cache



- Storage-disabled nodes are members of the cluster that hold no partitions.
- They do not cause rebalancing.
- Rebalancing increases work for the other members (and network traffic).
- Disable storage for certain nodes to avoid rebalancing.
- Storage-disabled nodes are still full members of the cluster.

ORACLE®

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

Storage Disabled Partitioned Cache

The distributed cache service also allows certain Coherence nodes to be configured to store partitions, and others to be configured to *not* store partitions. The name of this setting is "local storage enabled". Coherence nodes that are configured with the "local storage enabled" option provide the cache storage and backup storage for the distributed cache. Regardless of this setting, all Coherence nodes have the same exact view of the data, due to location transparency.

The previous examples showed four storage JVMs. In this particular topology, the JVMs in use are still partitioned, but JVM 1 and JVM 2 are client-only or nonstorage JVMs. These JVMs are client-only and do not store any data partitions.

How Do You Make a JVM Client-Only?

On the command line, pass the `localstorage=false` parameter. When the JVM starts up, the node is still connected to the grid using the TCMP protocol. So, the JVM is still a full member of the cluster. However, the JVM is not going to cause a partition rebalancing event.

Storage Disabled Partitioned Cache (continued)

Temporary members create a lot of traffic in the cluster as they cause redistribution of partitions both when they enter and exit the cluster. If a client is going to do some processing and then disconnect, avoid partition rebalancing by making that client storage-disabled. An application's needs will determine which processes should and should not be storage-disabled. Storage-disabled clients are useful for members that do not have the capacity to hold a lot of data, like Web servers, and members that tend to come and go frequently.

Configuring a Partitioned Cache

Example:

```

<cache-schemes>
  <distributed-scheme>
    <scheme-name>MyPartitionedScheme</scheme-name>
    <service-name>DistributedCache</service-name>

    <backing-map-scheme>
      <local-scheme>
        <scheme-ref>MyLocalCachingScheme</scheme-ref>
      </local-scheme>
    </backing-map-scheme>

    <autostart>true</autostart>
  </distributed-scheme>

  <local-scheme>
    <scheme-name>MyLocalCachingScheme</scheme-name>
  </local-scheme>
</cache-schemes>
</cache-config>

```

WARNING: "Distributed" is an old term kept around for backwards compatibility. "Partitioned" is the more accurate way to refer to this scheme type, but the actual configuration is called distributed. Keep this in mind for the rest of the course!!

coherence-cache-config.xml



ORACLE

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

Configuring a Partitioned Cache

Here is a simple example of a partitioned cache configuration. To conserve space, only the `<caching-scheme>` section is shown.

To declare a partitioned cache:

- Use the `<distributed-scheme>` element to indicate the scheme is for a partitioned cache.
Note: See the slide above for a warning note!
- Just as with the local cache scheme, a scheme name is specified that is used by a `<cache-mapping>` element to create an instance of this partitioned cache.
- The `<service-name>` element is an optional parameter used to name the service that manages this cache. By default, the partitioned cache service is named `DistributedCache`.
- The `<backing-map-scheme>` element specifies what backing map is used by this scheme to store data on each node. A `LocalCache` scheme is defined using the `<local-scheme>` element. The `<scheme-ref>` element specifies that this replicated cache will inherit (and reuse) the local cache scheme named `MyLocalCachingScheme`.

Defining Partitioned Cache Parameters

Parameter	Description
distributed-scheme	Specifies that this is a partitioned scheme type
service-name	Specifies the service that manages caches created with this scheme
serializer	Specifies the class configuration for a POF serializer. Applies to Invocation Service, Replicated Cache, and Partitioned Cache
partition-count	Specifies the number of partitions that a partitioned cache is split into. Each member running the partitioned cache service and local-storage set to true manages an even number of partitions. Should be a prime number
backup-count	Specifies the number of backup copies for a data entry. Recommended values are 0 or 1. Note that a backup count of 0 is not fault tolerant.



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

Defining Partitioned Cache Parameters

This slide shows some of the most common parameters that can be used with a partitioned cache scheme. Other parameters will be covered as they relate to other features covered in this course. For information about all possible parameters see the *Coherence Developer Guide*.

Quiz

True or False? If a JVM holding a primary partition fails, your application must have the logic to connect to the JVM that contains the backup partition.

- a. True
- b. False



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

Answer: b

Coherence automatically retrieves the data entry from the backup partition and automatically fails over.

Quiz

True or False? The partitioned cache topology provides for excellent scalability.

- a. True
- b. False



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

Answer: a

Quiz

A partitioned cache stores the same objects on:

- a. Every node in the cluster.
- b. Half of the nodes in the cluster.
- c. Two nodes in the cluster.
- d. One node in the cluster and one copy on disk.



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

Answer: c

By default, a partitioned cache stores two copies of data in the cluster, a primary and a backup.

Quiz

True or False? Data in a partitioned cache is stored in serialized form.

- a. True
- b. False



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

Answer: a

Quiz

A partitioned cache is used when: (Select all that apply)

- a. Fault tolerance is needed.
- b. Predictable read and write performance and scalability is needed.
- c. Large data set support is needed.
- d. Application cannot tolerate stale data.



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

Answer: a,b,c,d

Yes, all of the above. This question is almost a complete copy of the “When to use a Partitioned Cache” slide.

Practice.04.03 Overview: Configuring a Partitioned Cache Scheme

This practice covers the following topics:

- Configuring a partitioned cache scheme
- Running some clients that form a Coherence cluster that uses your partitioned cache scheme
- Viewing the results of how and where data is stored in the cluster when using a partitioned cache



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

Practice.04.03 Overview: Configuring a Partitioned Cache Scheme

This practice demonstrates configuring a partitioned cache, running two client applications that serve as Coherence cluster members that use the configuration, and analyzing the output of the clients to see how data is stored and retrieved from partitioned caches in a cluster.

Agenda

Coherence Cache Topologies

- What Happens to Cached Data?
- Cache Configuration Concepts
- **Topologies**
 - Local Cache
 - Replicated Cache
 - Partitioned Cache
 - **Near Cache**
 - Overflow Cache
 - Other Cache Types, Topologies Summary, and Advanced Configurations

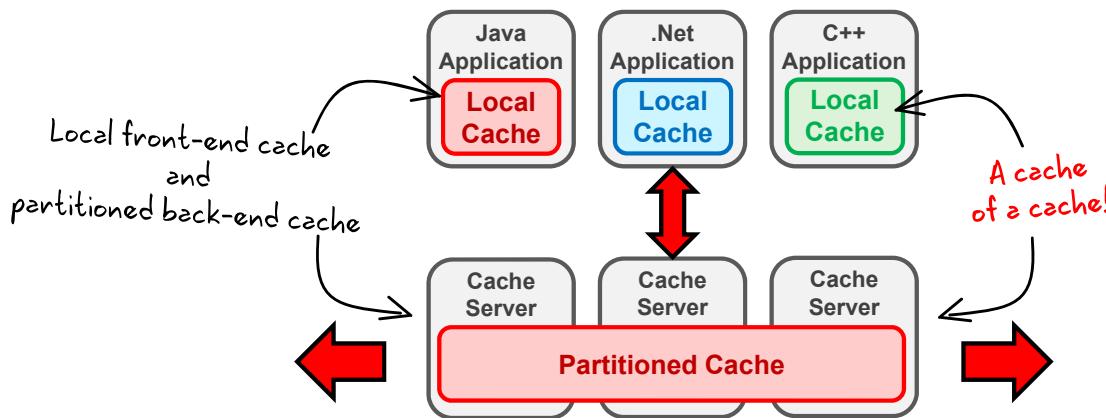
ORACLE

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

What is a Near Cache?

A near cache:

- Is a composition of pluggable front and back schemes
- Increases read performance for a partitioned cache



ORACLE

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

What is a Near Cache?

The near scheme defines a local cache that sits in front of a partitioned cache for very fast read access. The near scheme defines a two-tier cache consisting of a front tier that caches a subset of a back-tier cache. The front tier is generally a fast, size-limited cache, while the back tier is slower, but with much higher capacity. A typical deployment may use a local scheme for the front tier and a distributed scheme for the back tier. The result is that a portion of a large partitioned cache is cached locally in memory, in serialized form, allowing for very fast read access.

Near caches allow specifying an invalidation strategy that is used to keep the front tier of the near cache in sync with the back tier. Depending on that strategy, a near cache is configured to listen to certain events occurring on the back tier and automatically update (or invalidate) the front portion of the near cache.

A near cache provides local cache access to recently or often-used data, backed by a centralized or multilevel cache that is used to load-on-demand for local cache misses. Near caches have configurable levels of cache coherency: from the most basic expiry-based caches and invalidation-based caches to advanced data-versioning caches that can provide guaranteed coherency. The result is a tunable balance between the preservation of local memory resources and the performance benefits of truly local caches.

When To Use a Near Cache?

Use a near cache when:

- Fault tolerance is needed.
- High performance reads are needed.
- Predictable read and write performance and scalability is needed.
- Large data set support is needed.
- Application cannot tolerate stale data.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
Yes	Very fast	Inexpensive (scalable)	Larger than a single heap	Yes



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

When To Use a Near Cache?

The near cache topology uses a front cache for high speed read access by locally caching the data of a higher capacity cache on the back end. As a result, a near cache is capable of attaining the best of all worlds for high performance caching.

Fault tolerance, write performance, scalability, extremely large data set support, and data consistency are achieved by the back tier and front tier invalidation strategy.

High speed read performance is achieved by locally caching a subset of the back tier data, and avoiding any network hops for subsequent reads after the first read.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

When To Not Use a Near Cache?

Do not use a near cache when:

- Fault tolerance is not required.
- The application is write intensive, and the data set size is small to medium sized.
- Application can tolerate stale data.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
Yes	Very fast	Inexpensive (scalable)	Larger than a single heap	Yes



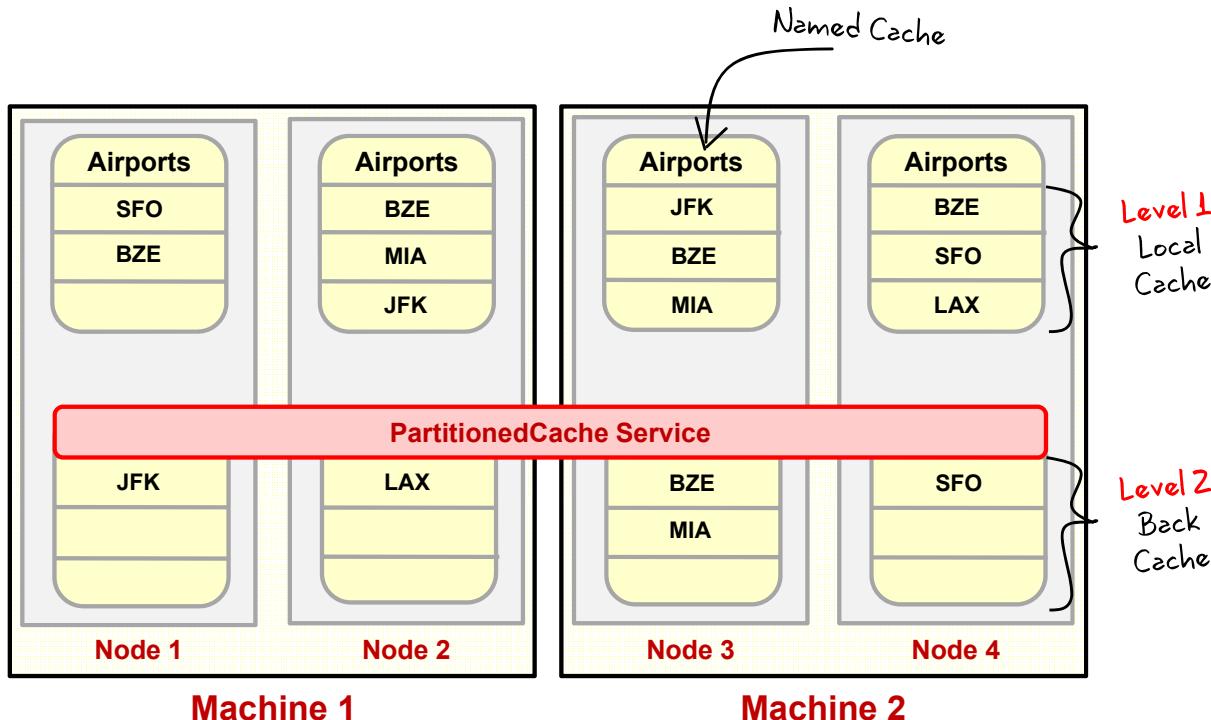
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

When To Not Use a Near Cache?

Because the near cache is designed to be used with other caches, such as the partitioned cache, which is designed for fault tolerance, large data sets, and predictable performance and scalability, it is not the best fit for applications that are smaller in nature and do not have these requirements.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

Visualizing Data in a Near Cache



ORACLE

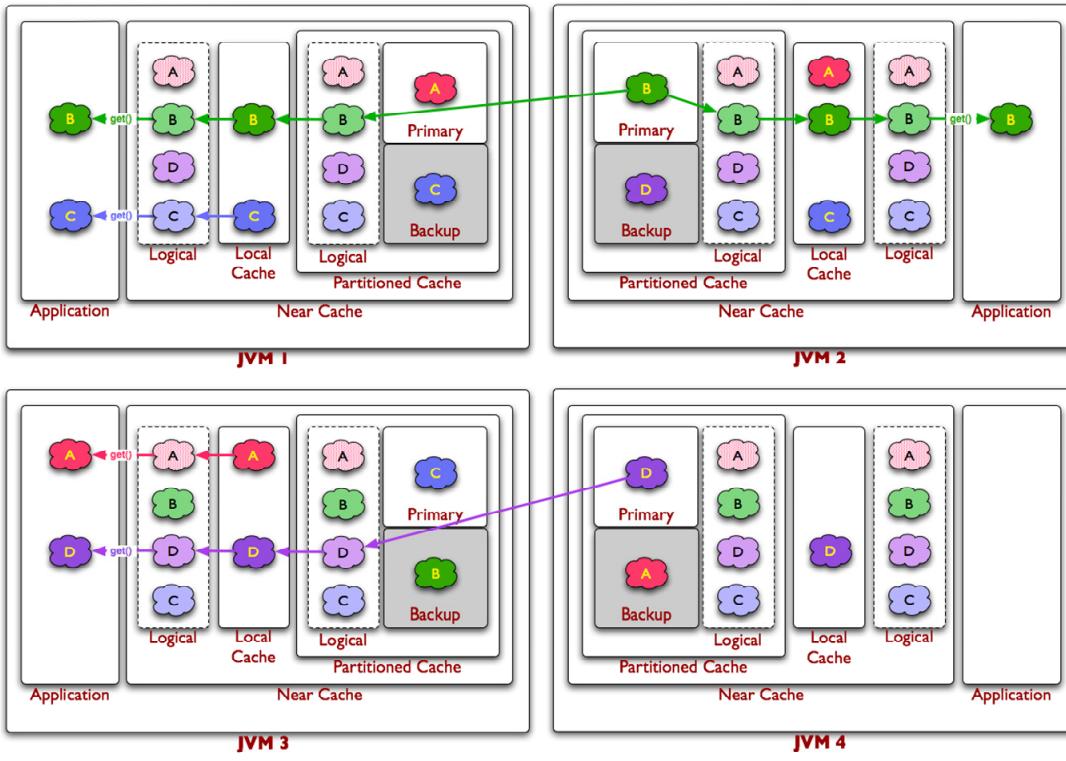
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Visualizing Data in a Near Cache

This diagram shows how the local (near) cache and partitioned cache stores work together to provide a high performance front-end for reads, backed by a scalable large data cache on the back-end.

- The front-end cache is a local cache, and as a result each cache contains different data that is pertinent to the application that is using that data. Each local cache may or may not contain the same objects that are stored in the other local caches. Notice how the BZE airport exists in all local caches, even though it only exists in a single primary copy on the partitioned cache.
- The back-end cache is a partitioned cache, which maintains its data in the same primary and backup stores as seen previously.

Near Cache get Diagram



ORACLE

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

Near Cache get Diagram

A near cache is actually a `NamedCache` implementation that wraps two caches—a front cache (assumed to be "inexpensive" and probably "incomplete") and a back cache (assumed to be "complete" and "correct," but more "expensive")—using a read-through/write-through approach. If the back cache implements the `ObservableCache` interface, the near cache provides four different strategies for invalidating the front cache entries that have been changed by other processes in the back cache.

This diagram shows a local cache in front of a partitioned cache. When a `get()` is performed on an object in a partitioned environment, Coherence goes across the network, brings back the object in binary form, deserializes it, and gives it back to the application. There is latency associated with this—typically 5 milliseconds associated with serialization and network I/O. When a near cache is involved, the `get` causes the object retrieved from the partitioned cache to be put into the local near cache. Subsequent gets retrieve the data directly from the local cache without ever touching the partitioned cache.

Near Cache Concurrency Options

Local cache concurrency options:

- Seppuku: Event-based "kill yourself" invalidation
- Standard expiry: LFU, LRU, hybrid, custom

No messaging system required for invalidation:

- Built into infrastructure
- High performance

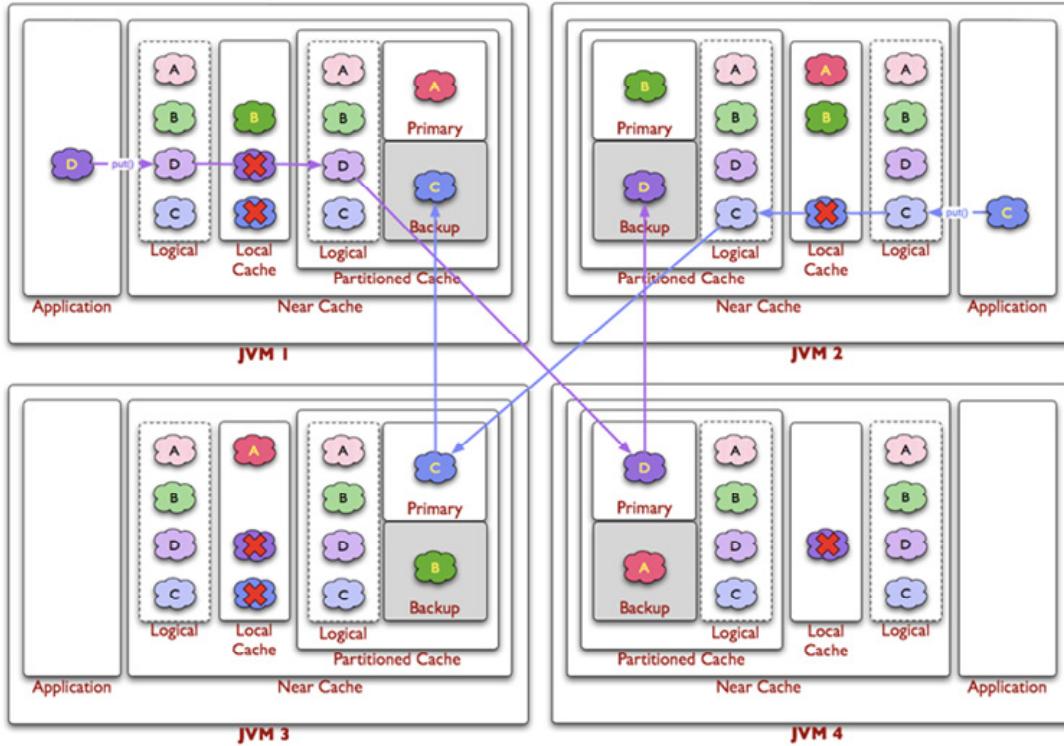


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Near Cache Concurrency Options

The near cache is a local cache, so it supports the same expiry options that a local cache supports. It also uses a Coherence event listener to receive notification when the data it has cached has been updated in the underlying cache so it can be invalidated locally.

Near Cache put Diagram



ORACLE

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

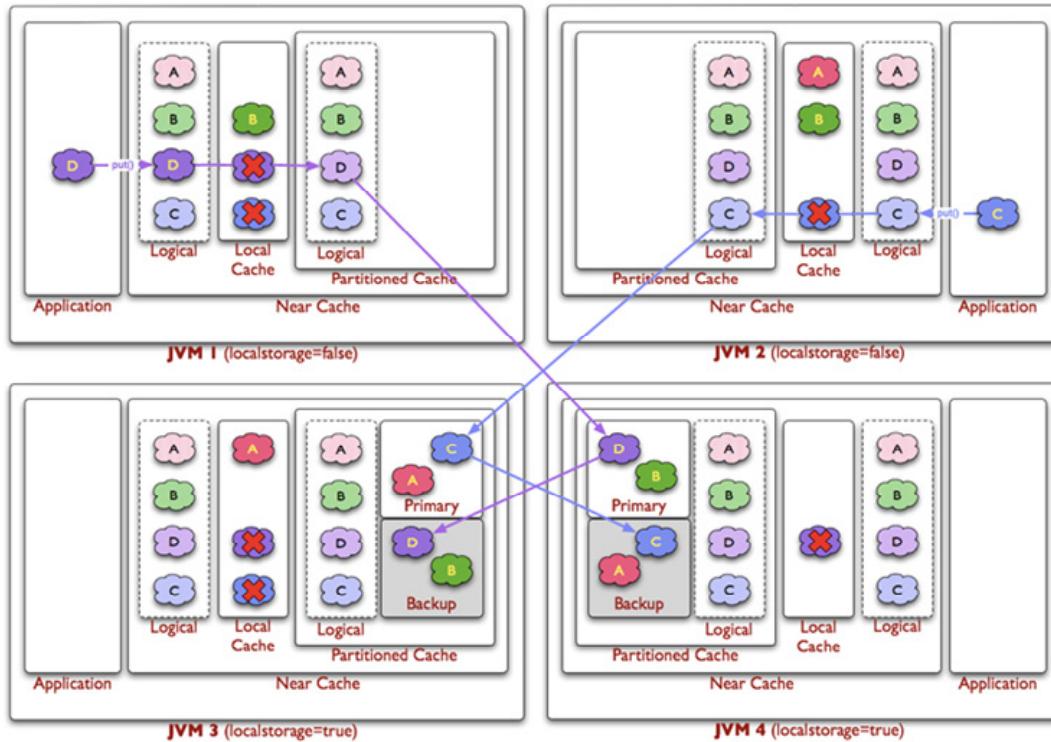
Near Cache put Diagram

Here is the same diagram showing a `put()` example. When a `put()` is performed with object D, Coherence places the object in the partitioned cache in JVM 4. The next time object D is requested in any JVM, the near cache will not have the object, and object D is retrieved from the partitioned cache, stored in the local cache, and returned to the calling application. Coherence automatically creates the near cache, sizes it, and expires its objects.

For example, object C is in the local cache on JVM 1, and object C is updated in the partitioned cache. Coherence automatically invalidates the object on JVM 1. The next time object C is requested, it is retrieved from the partitioned cache, and the near cache is loaded with its new value.

The combination of a local cache in front of a partitioned cache is very powerful. It provides very fast read access and the underlying resiliency of the partitioned cache.

Near Cache put Diagram with Local Storage Disabled



ORACLE

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

Near Cache put Diagram with Local Storage Disabled

This diagram shows a near cache cluster with local storage disabled for JVM 1 and JVM 2. All primary and backup objects are stored on JVM 3 and JVM 4, while the local near cache exists on all nodes. Note that disabling local storage is a setting only for partitioned caches.

Near Cache Invalidation Options

- An object in a near cache can be automatically invalidated when an object is updated in the backing partitioned cache.
- Invalidation options are set with the `<invalidation-strategy>` element in `<near-scheme>`. Values are:
 - **None**: Near cache objects are not invalidated if the object is updated in the backing distributed cache.
 - **Present**: Listen only to backing cache update events for objects that are in the cache.
 - **All**: Listen to all events.
 - **Auto** (default): Switch between All and Present, based on cache statistics.

WARNING: According to engineering, this was the goal, but this setting actually sets to **All** only.



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

Near Cache Invalidation Options

The Listen `None` strategy instructs the cache not to listen for invalidation events at all. This is the best choice for raw performance and scalability when business requirements permit the use of data that may not be absolutely current. Freshness of data can be guaranteed by the use of a sufficiently brief eviction policy for the front cache.

The Listen `Present` strategy instructs the near cache to listen to the back cache events that are related only to the items currently present in the front cache. This strategy works best when each instance of a front cache contains a distinct subset of data relative to the other front cache instances (for example, sticky data access patterns).

The Listen `All` strategy instructs the near cache to listen to all back cache events. This strategy is optimal for read-heavy, tiered, access patterns where there is significant overlap between the different instances of the front caches.

The Listen `Auto` strategy instructs the near cache to switch automatically between the Listen `Present` and Listen `All` strategies based on the cache statistics.

Note: See slide warning!

Configuring a Near Cache

Example:

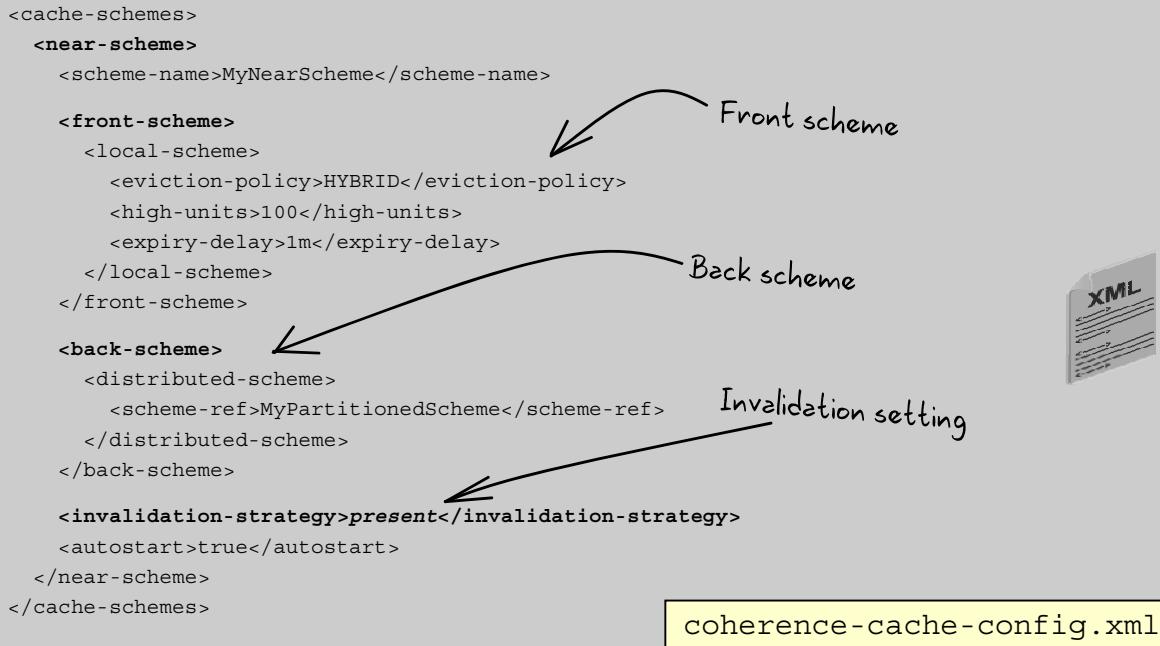
```

<cache-schemes>
  <near-scheme>
    <scheme-name>MyNearScheme</scheme-name>

    <front-scheme>
      <local-scheme>
        <eviction-policy>HYBRID</eviction-policy>
        <high-units>100</high-units>
        <expiry-delay>1m</expiry-delay>
      </local-scheme>
    </front-scheme>

    <back-scheme>
      <distributed-scheme>
        <scheme-ref>MyPartitionedScheme</scheme-ref>
      </distributed-scheme>
    </back-scheme>

    <invalidation-strategy>present</invalidation-strategy>
    <autostart>true</autostart>
  </near-scheme>
</cache-schemes>
```



coherence-cache-config.xml

ORACLE

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

Configuring a Near Cache

Here is a simple example of a near cache configuration. To conserve space, only the `<caching-scheme>` section is shown.

To declare a near cache:

- The `<near-scheme>` element indicates the scheme is for a near cache.
- Just as with the local cache scheme, a scheme name is specified that is used by a `<cache-mapping>` element to create an instance of this near cache.
- The `<front-scheme>` element specifies what caching scheme is used for the local front tier cache.
- The `<back-scheme>` element specifies what caching scheme is used for the back tier cache.
- The `<invalidation-strategy>` element specifies the invalidation technique that Coherence will use to handle updates to cached objects.

Defining Near Cache Parameters

Parameter	Description
near-scheme	Specifies that this is a near scheme type
front-scheme	Specifies the cache-scheme to use in creating the front-tier cache
back-scheme	Specifies the cache-scheme to use in creating the back-tier cache
invalidation-strategy	Specifies the strategy used to keep the front-tier in sync with the back-tier



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

Defining Near Cache Parameters

This slide shows some of the most common parameters that can be used with a near cache scheme. For information about all possible parameters see the *Coherence Developer Guide*.

Quiz

What is the act of combining two or more caching schemes called?

- a. Two-tier caching
- b. Near cache
- c. Backing map
- d. Scheme composition



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

Answer: d

Quiz

True or False? A near cache can consist of a local cache as the front cache, backed by a partitioned cache.

- a. True
- b. False



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

Answer: a

This is how Coherence is often configured.

Quiz

What is the primary purpose of the near cache?

- a. Faster reads for slower topologies
- b. Faster writes for slower topologies
- c. Intelligent data synchronization between cluster members
- d. Extra storage for when the backing cache is full



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

Answer: a

A near cache is a closer subset of cached data, or essentially a cache of a cache that provides much faster reads in an application that reuses the same cached data frequently.

Quiz

How does a near cache synchronize its front-tier data with its back-tier data?

- a. Event-based updates
- b. Event-based invalidation
- c. Message-based updates
- d. Message-based invalidation



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

Answer: b

A near cache updates its front tier with changes to the back tier by using event-based invalidation. The front-tier cache receives a notification that the data has changed in the back tier and the data invalidates itself and is deleted.

Practice.04.04 Overview: Configuring a Near Cache Scheme

This practice covers the following topics:

- Configuring a near cache scheme
- Running some clients that form a Coherence cluster that uses your near cache scheme
- Viewing the results of how and where data is stored in the cluster when using a near cache



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

Practice.04.04 Overview: Configuring a Near Cache Scheme

This practice demonstrates configuring a near cache, running two client applications that serve as Coherence cluster members that use the configuration, and analyzing the output of the clients to see how data is stored and retrieved from near caches in a cluster.

Agenda

Coherence Cache Topologies

- What Happens to Cached Data?
- Cache Configuration Concepts
- **Topologies**
 - Local Cache
 - Replicated Cache
 - Partitioned Cache
 - Near Cache
 - **Overflow Cache**
 - Other Cache Types, Topologies Summary, and Advanced Configurations

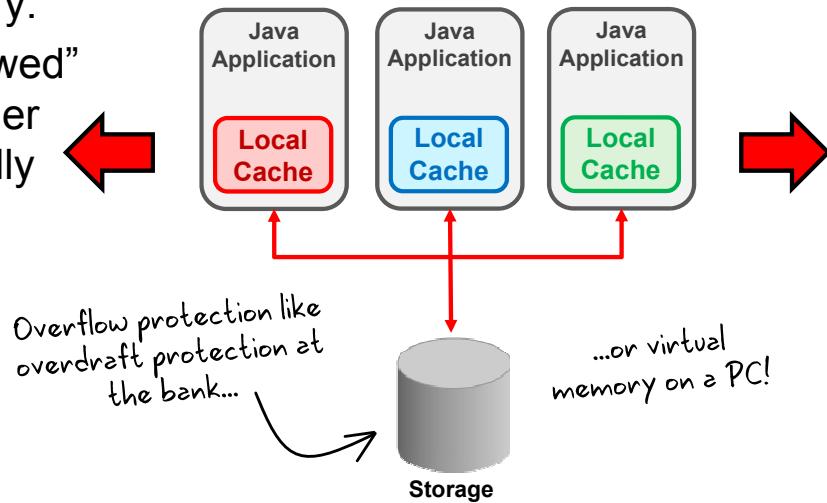
ORACLE

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

What is an Overflow Cache?

An overflow cache:

- Is a composition of pluggable front and back schemes.
- Guards against occasional situations where all data does not fit in memory.
- Stores “overflowed” entries in another scheme, typically external to the JVM.
- Is completely managed by Coherence.



ORACLE

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

What is an Overflow Cache?

An overflow cache is a two-tier cache that assumes a front tier that holds data in memory, with a back tier that provides disk-based storage for temporary occasions when all of the data set cannot fit in the memory of the front-tier cache. Think of an overflow cache like overdraft protection at the bank. It provides the security of knowing you will not run into trouble with your regular balance, but is not something you want to dip into if you can help it. The reason for this is that disk-based storage is much slower than in-memory storage, much like RAM versus virtual memory on a PC.

An overflow cache internally is managed entirely by Coherence, and data is stored in binary form on the disk. It is a bad idea to try to store too much data using the overflow cache, or to try to use it as a regular storage mechanism because, depending on how the cluster is set up, data repartitioning is an expensive operation that is optimized for in-memory caches, but can be very problematic for disk-based caches.

When To Use an Overflow Cache?

Use an overflow cache when the entire data set temporarily does not fit in the front tier in-memory cache.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
No	Slow	Slow	Could be Large (but not meant for large data)	No



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

When To Use an Overflow Cache?

An overflow cache does not really fit in with the characteristics used to gauge the other topologies. However, keep in mind that this cache type has a purpose, and when used properly can provide a nice safety net for data. A good example of an overflow cache is as a front scheme for a near cache. This provides the near cache's front tier a temporary storage area for the local cache should the entire data set not fit in-memory at the time.

The best use of the overflow scheme is to take advantage of the memory available on servers with large memory footprints, simply because large heaps with Java do not behave well. For example, typically a JVM with 4GB is fine, but a JVM with 30GB is not so good. However, a JVM with a 4GB heap and 10gB of overflow is good. Because an overflow cache has two tiers, an in-memory tier and a disk-based tier, its performance rating is slow. The front tier may perform well, as it is most likely an in-memory local cache, but the disk-based back tier is very slow.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

When To Not Use an Overflow Cache?

Do not use an overflow cache to hold large amounts of data for long periods of time.

Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
No	Slow	Slow	Could be Large (but not meant for large data)	No



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

When To Not Use an Overflow Cache?

Because the overflow cache is designed for a specific purpose, it generally cannot be gauged by these characteristics alone, and generally works best when combined with other caching schemes.

See the page titled “Choosing the Right Cache” at the end of this lesson for a summary chart comparing all the cache types.

Configuring an Overflow Cache

Example:

```

<caching-schemes>
  <overflow-scheme>
    <scheme-name>MyOverflowScheme</scheme-name>

    <front-scheme>
      <local-scheme>
        <scheme-ref>MyLocalCachingScheme</scheme-ref>
      </local-scheme>
    </front-scheme>

    <back-scheme>
      <external-scheme>
        <scheme-ref>example-bdb</scheme-ref>
      </external-scheme>
    </back-scheme>
  </overflow-scheme>
</caching-schemes>

```

The diagram shows the `coherence-cache-config.xml` file with handwritten annotations. Arrows point from the text "Front scheme (in-memory)" to the `<local-scheme>` element and from the text "Back scheme (disk-based)" to the `<external-scheme>` element. To the right of the code, there is a small icon of an XML document.

coherence-cache-config.xml

ORACLE

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

Configuring an Overflow Cache

Here is a simple example of an overflow cache configuration. To conserve space, only the `<caching-scheme>` section is shown.

To declare an overflow cache:

- The `<overflow-scheme>` element indicates the scheme is for an overflow cache.
- Just as with the local cache scheme, a scheme name is specified that is used by a `<cache-mapping>` element to create an instance of this overflow cache.
- The `<front-scheme>` element specifies what caching scheme is used for the local front tier cache, should be an in-memory cache.
- The `<back-scheme>` element specifies what caching scheme is used for the back-tier cache, should be a disk-based cache such as the Berkley Database option shown here using the `<external-scheme>` element. Any disk-based external scheme is valid for the back-scheme.

There is an example of `example-bdb` in the default `coherence-cache-config.xml` file.

Defining Overflow Cache Parameters

Parameter	Description
overflow-scheme	Specifies that this is an overflow scheme type
front-scheme	Specifies the cache-scheme to use in creating the front-tier cache
back-scheme	Specifies the cache-scheme to use in creating the back-tier cache
miss-cache-scheme	Specifies a cache-scheme for maintaining information on cache misses



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

Defining Overflow Cache Parameters

This slide shows some of the most common parameters that can be used with an overflow cache scheme. Miss-caches are described in more detail later. For information about all possible parameters see the *Coherence Developer Guide*. The miss-cache scheme is used to avoid the latency associated with searching the disk-based back tier of an overflow cache by storing the keys of any missed entries locally. When the back tier of the overflow cache is checked for a key that is not present in the back tier, the key will be placed into a miss-cache. When that key is requested again, the miss-cache will be checked for the value first. If it is in the miss-cache, then the back tier will not be checked for the key because Coherence already knows it is not present in the back tier. This avoids the latency associated with slow disk-based I/O.

Agenda

Coherence Cache Topologies

- What Happens to Cached Data?
- Cache Configuration Concepts
- **Topologies**
 - Local Cache
 - Replicated Cache
 - Partitioned Cache
 - Near Cache
 - Overflow Cache
 - **Other Cache Types, Topologies Summary, and Advanced Configurations**



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

Other Cache Types

Coherence also offers other cache types:

Cache Type	Description
Remote Cache	Coherence*Extend client-side configuration scheme that provides information needed to connect to a proxy server on the cluster
Transactional Cache	A transactional cache that allows Coherence to coordinate and join XA transactions, as well as perform multiple actions as a single unit of work. Covered in the transactions lesson
Continuous Query Cache	Cache that provides a live dynamic view of data based on a query. Covered in the queries lesson
Optimistic Cache	Similar to replicated cache, but offers no concurrency control



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

Other Cache Types

Coherence offers many different types of caches for different purposes. This slide describes a few of them. It is also possible to write custom cache types.

Choosing the Right Cache

Topology Type	Fault Tolerance	Read Performance	Write Performance	Data Size	Data Consistency
Local	No	Very fast	Very fast	All entries may be managed on-heap	No
Replicated	Yes	Very fast	Expensive (not scalable)	All entries may be managed on-heap	Yes
Partitioned	Yes	Fast	Inexpensive (scalable)	Larger than a single heap	Yes
Near	Yes	Very fast	Inexpensive (scalable)	Larger than a single heap	Yes
Overflow	No	Slow	Slow	Could be Large (but not meant for large data)	No

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

Choosing the Right Cache

This slide shows a summary of the different caches discussed in this lesson, showing the criteria for selecting the cache that is right for an application's needs.

Advanced Configurations

Configuration Type	Description
Read-Write Backing Map	Enables integrating a data store of any type as the back end source of data that can populate and persist data used by a Coherence cache. Covered in the integrating with a data source lesson.
Miss-Cache	A local cache that stores missed reads on expensive read-write operations to avoid trying to read them again
Listeners	Powerful mechanism that provides the ability to dynamically respond to data and cluster changes instantly when they occur. One use of listeners is near cache invalidation. Covered in the data events lesson.



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

Advanced Configurations

This slide shows some advanced configuration concepts available with Coherence. Some of these topics are covered in more detail in later lessons. The miss-cache is used to avoid expensive calls to a back end data source that may not be available or in which a particular key may not exist. When a key does not exist in the data source, that key is placed into the miss-cache. Whenever that key is requested again, the miss-cache is checked for that key first. If the key is found in the miss-cache, then Coherence knows it is not available in the back end data source, and avoids making the expensive call.

Summary

In this lesson, you should have learned how to:

- Explain the main topologies of Coherence
- Configure and use Coherence topologies
- Determine when to use each topology



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

Observing Data Grid Events

ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Describe Event concepts, including:
 - Event lifecycle
 - Event handler hooks
 - Event transformation
- Describe, compare and contrast listener types and lifecycle
- Implement and register map listeners
- Filter and transform map listener events
- Implement and register map triggers
- Implement backing map listeners
- Implement and use continuous query caches



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

Agenda

Event Processing

- Concepts

Map Listeners

Map Triggers

Backing Maps and Events

Events and the Continuous Query Cache



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

Event Concepts

Applications can listen for events which include:

- Data changes such as add, update or delete
- Cache changes such as backup creation, cluster rebalancing, and data partitioning

Events are:

- Application Level: Events can be registered locally and processed within the application that registered for them.
- Remote Level: Events can be registered remotely, and processed where data updates occur.
- Implemented via:
 - Event listeners, implementing the `MapListener` interface
 - Map triggers, implementing the `MapTrigger` interface

Beyond the scope of
this course



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

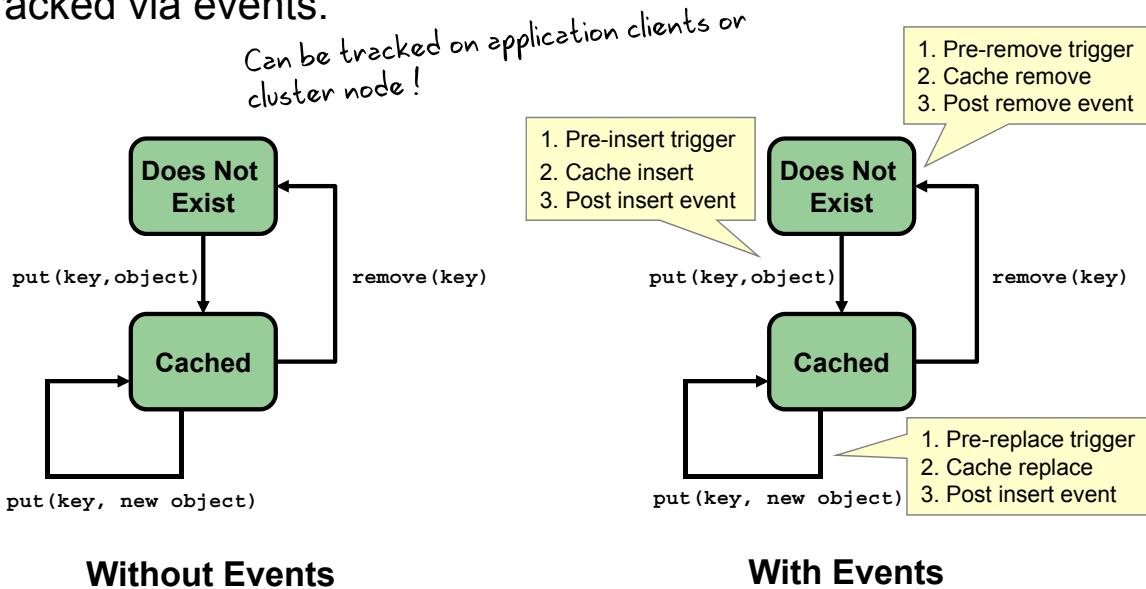
Event Concepts

Coherence applications can take advantage of rich event model which supports events in a variety of ways.

- **Data Changes:** Coherence provides event hooks for data changes such as add, delete and update. These can be pre- or post events, with event hooks being called before or after the event has occurred. Such changes can even be denied.
- **Location:** Events can be signaled to an application, to a handler within a cluster node, or both, or a subset of the cluster.

Simple Object Lifecycle and Events

Object entries in cache have a simple lifecycle which can be tracked via events.



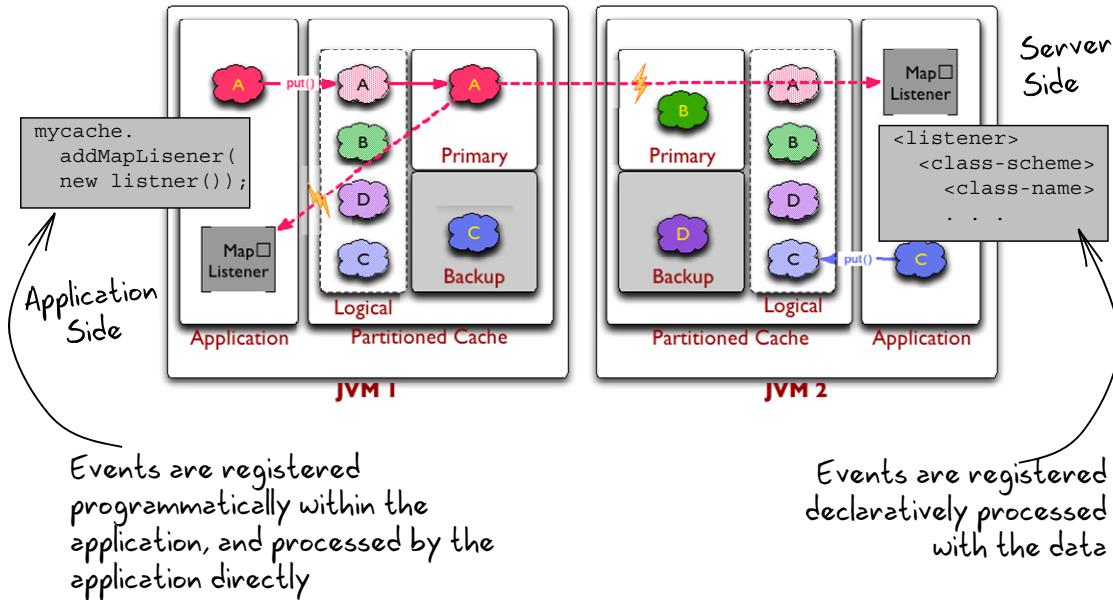
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

ORACLE

Simple Object Lifecycle and Events

Object entries in a Coherence cache have a lifecycle which can be tracked via events. Events can occur both prior to a cache operation, and postoperation. For example, an event can be triggered just before an insert, and after the insert has completed. The Coherence event system supports a number of possible combinations of events. Additionally, events can be local or remote. A local event occurs within the JVM of the application, whereas a remote event happens on a cache cluster member, typically behind the scenes.

Application versus Server Side Events



ORACLE

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

Application versus Server Side Events

Coherence supports both client side and server side events. In either case, an event is generated based on a lifecycle event such as an insert/update/delete map operation. In the case of client side events, all registered listeners are called in the order they are registered, often programmatically. Likewise, on the server side, all listeners registered, typically declaratively, are called in the order they are registered. In either case, both pre- and postoperation events are supported, via the `MapListener` and `MapTrigger` interfaces.

Listener Behavior

- All update events on the grid are guaranteed to be received by a listener, even if a storage JVM fails.
- When a listener is instantiated, it begins to listen for events, *asynchronously*. Listeners do not receive events that occurred before they were instantiated.
- Ordering of events is not guaranteed. For example: `put (1)`, `put (2)`, and `put (3)` may be received by a listener as 2, 1, and 3.

Storage JVM means local-storage = true



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

Listener Behavior

It is important to understand how listeners behave when they are implemented, and how they work in the grid environment.

All update events on the grid are guaranteed to be received by a listener, even if a storage JVM fails. The way that Coherence works with listeners is that when an object is to be inserted into the grid, that object is inserted into the storage JVM that owns the particular object. Therefore, it is this storage JVM that is responsible for sending the event to the client.

What the JVM actually does is it sends the event into a queue. The events are queued up and sent to the client. This queue is backed up along with the data. If a JVM or the server fails in the middle of an update, Coherence automatically processes the update. It sends the update to the new storage JVM, and to the surviving ones that are automatically determined by Coherence. These queues in turn are backed up. The listener is guaranteed to receive the event. Therefore, if an event such as an insert, update, or delete does occur in the grid, the client-side listeners are guaranteed to receive that event at least once.

When a listener is instantiated, it begins to listen for events. However, listeners will not receive events that occur before they begin to listen. They begin to listen when they are instantiated.

Listener Behavior (continued)

Note that the ordering of events is not guaranteed. By default, you do not get the objects at the event listener in the order that they were put.

The distributed data manager of Coherence indicates that a set of objects are to be inserted into the grid in the order 1, 2, and 3. These objects may be stored in separate storage JVMs. Therefore, they do not necessarily arrive at the client in the same order. So a `put(1)`, `put(2)`, and `put(3)` may be received by a listener as 2, 1, and 3.

However, there are ways in which you can ensure the order in which the events are inserted into the grid. Coherence has an equivalent of a sequence in a database, where you can assign an incremental integer to a particular object to see that certain objects get inserted in a certain order. You need to code this implementation in your application.

Listeners versus Triggers

- Map Triggers/Listeners:
 - Can be registered programmatically
 - Can be registered declaratively
 - Directly for listeners
 - Via custom factory classes for triggers
 - Triggers can be used to:
 - Prevent invalid transactions
 - Enforce complex business rules or security
 - Provide transparent event logging and auditing and others
 - Listeners:
 - Can be used to post process data
 - Can be used with filters to selectively handle specific changes
- Backing map listeners are a special kind of listener, and are almost always registered declaratively*
- Backing map listeners can preprocess data*



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

Listeners versus Triggers

Map triggers supplement the standard capabilities of Oracle Coherence to provide a highly customized cache management system. For example, map triggers can be used to prevent invalid transactions, enforce complex security authorizations or complex business rules, provide transparent event logging and auditing, and gather statistics on data modifications. Other possible use for triggers include restricting operations against a cache to those issued during application redeployment time.

For example, assume that you have code that is working with a `NamedCache`, and you want to change an entry's behavior or contents before the entry is inserted into the map. The addition of a map trigger will allow you to make this change, without having to modify all the existing code.

Map triggers could also be used as part of an upgrade process. The addition of a map trigger could prompt inserts to be diverted from one cache into another.

Listeners versus Triggers

Map Triggers:

- Execute *before* data changes occur
- Execute *synchronously* in response to a pending change/delete to the corresponding map entry
- Can be used to validate, modify, or even reject an operation

Conceptually similar to
database triggers

Map Listeners:

- Execute *after* data changes occur
- Execute *asynchronously* in response to an add, delete, or update to the corresponding map entry
- Can be used with filters to selectively register for events

Both:

- Can be registered programmatically or declaratively

Declaratively = in coherence-cache-config.xml



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

Agenda

Event Processing

Map Listeners

- Concepts
- Implementation and interfaces
- The Observable Interface
- Filters and Transforms

Map Triggers

Backing Maps and Events

Events and the Continuous Query Cache



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

Pre-events and MapTriggers

Map Triggers:

- Are developed by implementing MapTrigger
- Support a single process method *before* operation execution

```
package com.tangosol.util;

public interface MapTrigger extends Serializable {
    public abstract void process(Entry entry);
}
public interface MapTrigger.Event extends InvocableMap.Entry {
    public abstract Object getOriginalValue();
    public abstract boolean isOriginalPresent();
}
public interface InvocableMap.Entry extends QueryMap.Entry {

    public abstract Object getKey();
    public abstract Object getValue();
    public abstract Object setValue(Object obj);
    public abstract void setValue(Object obj, boolean flag);
    public abstract void update(ValueUpdater valueupdater, Object obj);
    public abstract boolean isPresent();
    public abstract void remove(boolean flag);
}
```

maptrigger.java



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

Pre-events and Map Triggers

Map triggers differ from map listeners in that they provide a single method which is invoked before the operation occurs. The `process` method of the map trigger interface is provided a single argument representing the object being processed. Two methods are provided on the `Event` interface, `getOriginalValue()` and `isOriginalPresent()`, which allow a developer to return the prior version of the object. In the case of inserts, these values might be missing. The `MapTrigger.Event` class extends `InvocableMap.Entry`, which itself provides a number of useful methods such as `getKey()` and `getValue()`.

The `process` method is used to validate, reject, or modify the pending change in the map. This method is called *before* an operation that intends to change the underlying map content is committed. Implementations of this method can evaluate the pending change by examining the original and the new value and:

- Override the requested change with a different value
- Undo the pending change by resetting the original value
- Remove the entry from the underlying map
- Reject the pending change by throwing a `RuntimeException`
- Do nothing, and allow the pending change to be committed

Post-events and MapListeners

Map Listeners:

- Are developed by implementing MapListener
- Support insert, update, or delete events, *asynchronously after* the operation occurs
- Can extend AbstractMapListener and selectively implement methods

```
package com.tangosol.util;

public interface MapListener extends java.util.EventListener{
    public void entryInserted (MapEvent event);
    public void entryUpdated (MapEvent event);
    public void entryDeleted (MapEvent event);
}
```

maplistener.java



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

Post-events and Map Listeners

Map listeners are developed by implementing/extending either the MapListener or AbstractMapListener. AbstractMapListener allows you to selectively decide which of the three events you are interested in tracking. A map listener is called after the operation occurs.

Each of the methods takes a single argument, a MapEvent, for which a partial list of methods is shown, which describes the specifics of the event which occurred. Using the map event, you can determine the type of event using the getId() method and the old and new values. Depending on the type of event, for example ENTRY_INSERTED, there may be only a partial set of data.

You can register MapListener for all cache events, or those satisfying a filter or a specific key. You register a MapListener using the addMapListener() method. You can also have a server-side event listener, which can be set up in the cache configuration XML file. These listeners run on the server, that is, they run on the JVM where the entry that is being updated is actually located.

Contract for writing a MapListener

MapListeners are implemented by:

1. Extending the MapListener interface
2. Implementing the three required methods:

```
import com.tangosol.util.MapListener;           Required imports

public class SimpleMapListener implements MapListener { Implements...
    public void entryInserted(MapEvent event) {
        System.out.println("\n\n+++entryInserted++++\n");
    }
    public void entryUpdated(MapEvent event) {
        System.out.println("\n\n+++entryUpdated++++\n");
    }
    public void entryDeleted(MapEvent event) {
        System.out.println("\n\n+++entryDeleted++++\n");
    }
}
```

SimpleMapListener.java

Exceptions thrown within MapListeners are wrapped and rethrown to the application thread which registered the listener!

ORACLE

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

Contract for writing a MapListener

The map listener interface is straightforward to implement. Simply include the MapListener import, and implement the required methods.

Selected MapEvent Methods

```

package com.tangosol.util;

public class MapEvent extends EventObject {
    public int getId();
    public Object getKey();
    public Object getOldValue();
    public Object getNewValue();
}

```

Defines:
ENTRY_INSERTED
ENTRY_UPDATED
ENTRY_DELETED

getId()
Return the type of events, inserted, updated or deleted.

getKey()
Return the key associated with the object for which the event occurred

getOldValue()
getNewValue()
Return the prior and new values of the object, respectively.

mapevent.java



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

Selected MapEvent Methods

Some useful MapEvent methods:

- **getNewValue ()**: Returns a new value associated with this event. The new value represents a new value inserted into or updated in a map. It is always null for “delete” notifications. The `getNewValue ()` method that you saw in the preceding code example returns the value in the object that was changed in the grid.
- **getOldValue ()**: Returns an old value associated with this event. The old value represents a value deleted from, or updated in, a map. It is always null for “insert” notifications. If it is a nonlite event, the method gets or returns the old value. In a sales order example, there is an update in the quantity of the number of items being ordered. You retrieve the old value of quantity as 5 and insert the new value of quantity as 10. Therefore, in the update event, you can write code to take action based on that change.
- **getKey ()**: Returns a key associated with this event. It returns the key of the object that was changed.

MapListener Event Contents

Based on the event type, the old and new values of a MapListener event may contain the following values:

Event Type	Old Value	New Value
Insert	null	Inserted object
Update	Previously inserted object	Newly inserted object
Delete	Previously inserted object	null



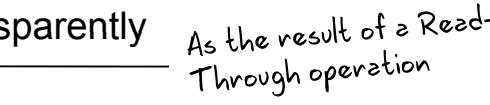
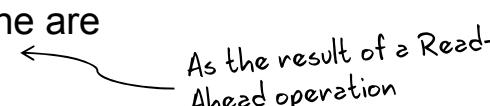
Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

MapListener Event Contents

Map listener events provide both an old and a new value. The values differ depending on the event type.

For example, on an insert there is no old value, since a new value is being added to the cache. Likewise, on a delete there is no new value as the value is being removed.

Synthetic Events

- Events normally reflect changes such as insert, update, or delete on a cache.
- Events can originate within the cache itself. Such events are referred to as *synthetic* events.
- Common synthetic event reasons are:
 - When entries automatically expire from a cache
 - When entries are evicted from a cache because the maximum size of the cache has been reached
 - When entries are transparently added to a cache 
As the result of a Read-Through operation
 - When entries in a cache are transparently updated 
As the result of a Read-Ahead operation

ORACLE

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

Synthetic Events

Events usually reflect the changes being made to a cache. For example, one server is modifying one entry in a cache, while another server is adding several items to a cache, while a third server is removing an item from the same cache, all while fifty threads on each and every server in the cluster is accessing data from the same cache! All the modifying actions will produce events that any server within the cluster can choose to receive. These actions are referred to as *client actions*, and the events as being *dispatched to clients*, even though the "clients" in this case are actually servers. This is a natural concept in a true peer-to-peer architecture, such as a Coherence cluster: Each peer is both a client and a server, both consuming services from its peers and providing services to its peers. In a typical Java Enterprise application, a "peer" is an application server instance that is acting as a container for the application, and the "client" is that part of the application that is directly accessing and modifying the caches and listening to events from the caches.

Synthetic Events (continued)

Some events originate from within a cache itself. There are many examples, but the most common cases are:

- When entries automatically expire from a cache;
- When entries are evicted from a cache because the maximum size of the cache has been reached;
- When entries are transparently added to a cache as the result of a Read-Through operation;
- When entries in a cache are transparently updated as the result of a Read-Ahead or Refresh-Ahead operation
- Repartitioning due to cluster node failure. Repartitioning events are only provided to backing map listeners. .

Determining if an Event is Synthetic

- MapEvents generated for synthetic reasons are derived from CacheEvent.
- CacheEvent includes the `isSynthetic` method which returns true if an event is synthetic.

```
import com.tangosol.util.MapEvent;
import com.tangosol.net.cache.CacheEvent;
. . .
private boolean isSynthetic(MapEvent event) {
    if (event instanceof CacheEvent) {
        return ((CacheEvent) event).isSynthetic();
    }
    return false;
}
. . .
```

Likely a method
In a MapListener



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

Determining if an Event is Synthetic

Synthetic events are generated by the underlying server and derived from CacheEvent. However, to be consistent with the MapListener and MapTrigger interfaces, the event is send as a MapEvent. To determine if an event is synthetic, first check that the class is an instance of CacheEvent, then cast it to CacheEvent and check the `isSynthetic()` method as shown in the example.

Behind the scenes: ObservableMap

ObservableMap:

- Provides the ability to “observe” changes in cache entries
- Implements the JavaBean event model listener pattern
- Is extended by all NamedCaches including LocalCache, OverflowMap, NearCache, and others

```
package com.tangosol.util;

public interface ObservableMap extends Map {    Register for all
    public abstract void addMapListener(MapListener listener);
    .
    .
    public abstract void removeMapListener(MapListener listener);
}

```

Unregister

observablemap.java

 ORACLE

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

Behind the scenes: ObservableMap

ObservableMap and Events

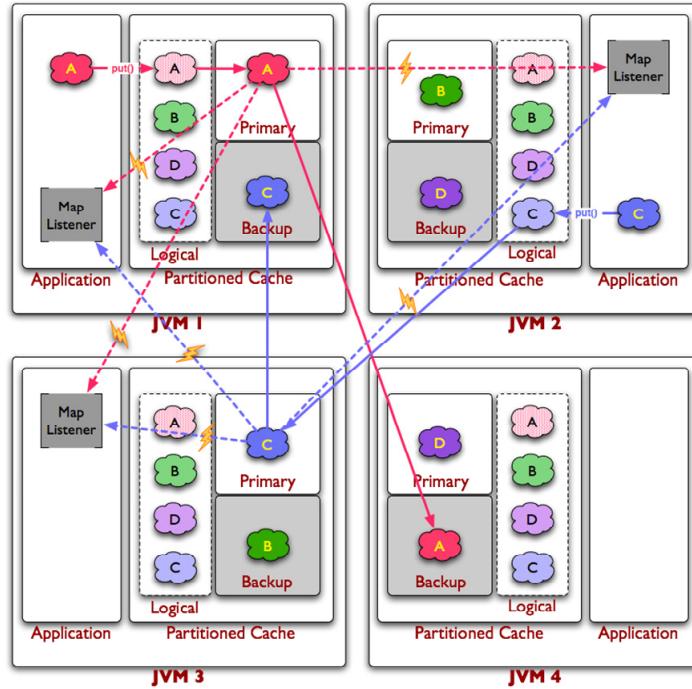
The slide lists some of the ObservableMap interface methods. The methods that are available to create a map listener on the client are:

- `addMapListener (MapListener listener)` adds a standard map listener that receives all events (inserts, updates, and deletes) that occur against the map, along with the key, old value, and new value. The method takes the listener to add as parameter.
- `addMapListener (MapListener listener, Filter filter, boolean fLite)` adds a map listener that receives events based on a filter evaluation.
- `addMapListener (MapListener listener, Object oKey, boolean fLite)` adds a map listener for a specific key. This method includes the listener to add, the key that identifies the entry for which to raise events, and `fLite` set to `TRUE` to indicate that the `MapEvent` objects need not include the `OldValue` and `NewValue` property values to allow optimizations. A Lite event is an event that does not contain the old value of the object. With `entryUpdated`, you can do a `getOldValue()` and `getNewValue()` to prepare the old value and new value of the object. If you do not need that functionality, you pass `TRUE` to the `fLite` attribute. If the event is Lite, it does not have to send the old value back to the client. There is less network traffic and also less CPU involved. If you do not need the old value, you can set the `Lite` attribute to `TRUE`.

ObservableMap and Events (continued)

- `removeMapListener (MapListener listener)` removes a standard map listener that previously signed up for all events.
- `removeMapListener (MapListener listener, Filter filter)` removes a map listener that previously signed up for events based on a filter evaluation.
- `removeMapListener (MapListener listener, Object oKey)` removes a map listener that previously signed up for events about a specific key. This method includes the listener to remove and the key that identifies the entry for which to raise events. You can dynamically add or remove map listeners on the client, whenever you need to.

Observable Map Workings



ORACLE

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

Quiz

Synthetic events are: (Select all that apply)

- a. Generated when an insert/update/delete mutation occurs
- b. Generated when Coherence acts on an cache entry
- c. A sub class of MapEvent
- d. Instances of CacheEvent



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

Answer: b,d

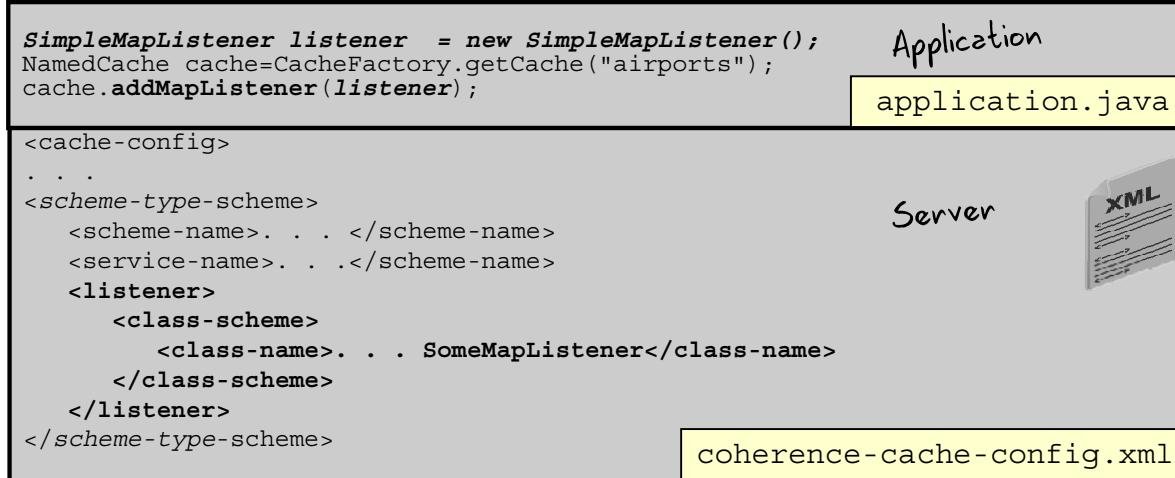
Synthetic events are those which occur when Coherence changes an entry for some reason, such as when repartitioning of an entries occurs due to the addition of new servers.

Registering MapListenerS

Class implementing MapListener must be registered.

Registration can be:

- Programmatic, via ObservableMap.addMapListener()
- Declarative, via coherence-cache-config.xml



ORACLE

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

Registering MapListeners

Map Listeners are registered in one of two ways: programmatically, or declaratively via `coherence-cache-config.xml`.

The primary differences are:

- Programmatically: The map listeners are registered via one of the `addMapListener` methods exposed by the `ObservableCache` interface. Multiple listeners can be registered programmatically.
- Declaratively: A `listener` element is added to a scheme definition, refer to the `cache-config.dtd` for more information on the elements of schemes such as `distributed-scheme` and others. A single listener only can be registered declaratively.

Map Listeners and Exceptions

- MapListeners can be registered programmatically via `addMapListener()`, or declaratively, via a cache configuration.
- Exceptions on MapListeners registered programmatically:
 - Are client side exceptions and do not effect the cluster
 - Do not traverse the network and are only thrown to the registering thread
- Exceptions on MapListeners registered declaratively:
 - Are server side exceptions and fail silently
 - Are wrapped in `RuntimeException` and logged
 - Are not sent across the networks as there is no client
 - Must be carefully coded to not affect the cluster



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

Map Listeners and Exceptions

Map Listeners are written in Java and can raise exceptions.

Most map listeners are client-side, and exceptions that occur there will be raised in the client application. Such exceptions don't traverse the network. These exceptions are usually due to developer code. Such exceptions are similar to an exception being raised in an event handler, for example, in Swing. Such exceptions should not affect the cluster.

Map listeners registered in the configuration must be very carefully considered. Such an exception can affect the cluster. Since there is no client, such an exception is first wrapped in a `RuntimeException`, and then logged. Extreme care should be taken when raising an exception in such cases.

MapListener Registration Variations

Programmatic map listener registration comes in several forms:

1. Simple registration for all events on a cache
2. Register for all events by key
3. Register for events using a Filter

Also supports events 'lite', or event notification w/o content

```
package com.tangosol.util;

public interface ObservableMap extends Map {
    public abstract void addMapListener(MapListener listener);
        Add the specified listener which will be invoked for all changes, for all elements

    public abstract void addMapListener(MapListener maplistener,
                                         Object key, boolean liteFlag);
        Add the specified listener which will be invoked for only events for the given key, the event
        may be also 'lite', that is no 'content'

    public abstract void addMapListener(MapListener listener,
                                         Filter filter, boolean liteFlag);
        Call the event listener based on the contents of the provided filter, either lite or not.
}
```



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

MapListener Registration Variations

Map listeners can be registered using a variety of mechanisms. The slide shows three variations:

- A simple listener which is registered for all changes
- A listener that takes a object as a key only is triggered if the key matches
- A listener that takes a filter to select a set of objects to trigger events on

Additionally several interfaces provide the 'liteFlag', which is covered in later slides.

MapListener Lite

Map Listeners can be registered as *Lite*, meaning that only the key is provided and not the entry

- Normally the old and new parameters of the event are null, but may be passed if no overhead is incurred

```
SimpleMapListener listener = new SimpleMapListener();
NamedCache cache=CacheFactory.getCache("airports");
cache.addMapListener(listener,someKey,true);
```

application.java

```
import com.tangosol.util.MapListener;
public class SimpleMapListener implements MapListener {
    .
    public void entryInserted(MapEvent event) {
        System.out.println("\n\n+++entryInserted++++\n");
    }
}
```

SimpleMapListener.java

*May contain null
old and new
values!*

Lite MapListeners are useful when the underlying values are unimportant. Because they are not sent across the wire, network traffic is minimized



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

MapListener Lite

If you have specified an event as Lite, it means that the old value may or may not be present in these events. If you do not need the old values, you can make it a lite event. This means that the old value is not sent back to the client and you will have less network traffic and fewer overheads involved with the event listener.

The `MapEvent` object that is passed to `MapListener` carries the necessary information about the event that has occurred, including the `source` (`ObservableMap`) that raised the event, the identity (key) that the event is related to, the action against that identity (insert, update, or delete), the old value, and the new value.

To sign up for events, pass an object that implements the `MapListener` interface to one of the `addMapListener` methods on `ObservableMap`.

By default, Coherence provides both old and new values as part of an event. When an application does not require the old and new values to be included in the event, it can indicate that by requesting only “lite” events. When you add a listener, you can request lite events by using one of the two `addMapListener` methods that takes an additional `boolean fLite` parameter.

In some cases, a lite event’s old value and new value may be null. In others, even if you request lite events, the old and new values may be included if there is no additional cost to generate and deliver the event. Therefore, when you request that a `MapListener` receive lite events, it is just a hint to the system that `MapListener` does not need to know the old and new values for the event

MapListener **Implementations**

Several MapListener helper implementations exist:

- `AbstractMapListener`:
 - Has empty implementations for each MapListener signature
 - Simplifies your implementations by overriding default empty implementations
- `MultiplexingMapListener`:
 - Introduces abstract `onMapEvent` method
 - Sets all MapListener methods to `onMapEvent`
 - Simplifies your implementations by overriding `onMapEvent`



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

Anonymous AbstractMapListener Example

Anonymous inner class can be used to register MapListener using AbstractMapListener and overriding only the methods of interest.

```
cache.addMapListener(  
    new AbstractMapListener() {  
        public void entryUpdated(MapEvent mapEvent) {  
            // Implementation  
        }  
    });
```

Override only method of interest

Application.java



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

Examples

The slide shows an example of the AbstractMapListener, which provides dummy overrideable implementations of all required methods. An additional class MultiplexingMapListener, requires one method called onMapEvent () that can handle any event that occurs in the grid. Within the method, you can use the event ID to determine what type of event it is, whether it is an insert, update, or delete.

For example:

```
public static class EventPrinter extends MultiplexingMapListener {  
    public void onMapEvent (MapEvent evt) {  
        out (evt);  
    }  
}
```

Because only one method needs to be implemented to capture all the events, MultiplexingMapListener can also be very useful when creating an inner class to use as a MapListener.

Making Maps Observable

Maps such as `HashMap`, which are not derived from `ObservableMap`, can be made observable using `WrapperObservableMap`

```
import com.tangosol.util.WrapperObservableMap;  
.  
.  
.  
Map hashMap = new java.util.HashMap();  
WrapperObservableMap wom = new WrapperObservableMap(hashMap);  
wom.addMapListener(new SimpleMapListener());  
wom.put ("somekey", "some object");
```

Application.java



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

Making Maps Observable

Coherence provides support for making traditional Java maps observable. Any map which is derived from, or implements `java.util.Map`, can be wrapped using `WrapperObservable` map and then used with the Coherence listener infrastructure.

The example shown uses a traditional Java hash map, wraps it, and then instantiates a listener. On each put or remove operation the listener is then called.

The wrapper works as a write-thru map, providing the functionality required, but adding the listener support.

MapListeners and Filters

Filter instances:

- Can be used with MapListeners to limit event scope
- Must extend Filter and implement evaluate, returning true if the event is to be processed

```
import com.tangosol.util.Filter;
public class DeletedFilter implements Filter, Serializable {
    public boolean evaluate(Object o) {
        MapEvent evt = (MapEvent) o;
        return evt.getId() == MapEvent.ENTRY_DELETED;
    }
}
```

Only return true if a deleted event

DeletedFilter.java

```
SimpleMapListener listener = new SimpleMapListener();
NamedCache cache = CacheFactory.getCache("airports");
cache.addMapListener(listener, new DeletedFilter(), false);
```

Application.java



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

MapListeners and Filters

Filters extend the capability of a listener to select which events should be processed. Custom filters can be easily developed by implementing the Filter interface and providing an implementation of the evaluate method as shown in the slide. Evaluate is passed an event object, which can be used to determine how to process the filter. In the example shown, the ID field of the event is used to determine if a DELETE event is being processed. The filter is then used by passing an instance to the addMapListener method with the appropriate listener.

Provided Filters

Coherence provides a set of filters in the `com.tangosol.util.filter` package:

Filter	Description
MapEventFilter	Evaluates the content of a <code>MapEvent</code> object according to the specified criteria, includes support for event types such as insert, update, and delete
AndFilter	Provides logical AND between two filters
OrFilter	Provides logical OR between two filters
ValueChangeEvent Filter	Evaluates to true only for update events that change the value of an extracted attribute.
ContainsFilter	Filter which tests a <code>Collection</code> or Object array value returned from a method invocation for containment of a given value



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

Provided Filters

Coherence provides a number of filters in the `com.tangosol.util.filter` package, a subset of which is shown on the slide. These filters provide a mechanism for creating very complex filter functionality by providing and, or, xor, list, and a variety of other filters, available for immediate use without any development effort.

Transforming Events

Events:

- Can be transformed before processing if partial, different, or modified data is required by the underlying map listeners
- Are transformed using `MapEventTransformer`

```
package com.tangosol.util;
public interface MapEventTransformer {
    public abstract MapEvent transform(MapEvent mapevent);
}
```

Called before the event is processed!

MapEventTransformer.java

Returning null causes the event to be ignored!



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

Transforming Events

Coherence supports the concept of an event transform. An event transform allows you to manipulate the contents of an event prior to any handler getting it. The entire contents of a event can be changed, or event replaced by a transform, or the event dropped completely if the transform returns null rather than an actual event.

Transformer Example

To implement a MapEventTransformer:

1. Implement the required interface
2. Implement a Serializable interface
3. Return an updated map event representing the transform

```
import com.tangosol.util.MapEventTransformer;
public class DoNothingMapEventTransformer
    implements MapEventTransformer,
              Serializable {

    public MapEvent transform(MapEvent evt) {

        MapEvent result = evt;
        result = new MapEvent(evt.getMap(),
                             evt.getId(),
                             evt.getKey(),
                             evt.getOldValue(), evt.getNewValue());
    }
}                                Return the 'transformed' event
```

DoNothingMapEventTransformer.java



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

Transformer Example

Implementing a MapEventTransformer requires three specific steps:

1. Create a Java class which implements the `com.tangosol.util.MapEventTransformer` interface.
2. Implement a serializable.
3. Implement the transform method to return an instance of an updated MapEvent, in the example shown the event is just a copy of the provided event, but reconstructed for illustration purposes.

The 'Semi-light' event transformer, which deletes values from the event to limit bandwidth, was such a commonly developed transform that it is now available out of the box with Coherence.

Registering a Transform Programmatically

Transforms are registered programmatically using a `MapEventTransformerFilter` instance.

To create a transform for a given listener:

1. Instantiate the transform instance
2. Create a `MapEventTransformerFilter` with a null filter argument and the transform
3. Pass the filter to the `addMapListener` method of the cache

```
NamedCache cache= . . .
DoNothingMapEventTransformer transform = . . .;

Filter filter = new MapEventTransformerFilter( null, transform) ;
cache.addMapListener(new SimpleMapListener(), filter,false);
```

Application.java



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

Registering a Transform Programmatically

Registering a transform programmatically is a simple process:

1. Create the transform instance normally, providing whatever is required
2. Create an instance of a `MapEventTransformerFilter`. This is simple a wrapper for the transform. Notice that the filter portion of the constructor is null.
3. Pass the filter to the `addMapListener` method on the cache being monitored. Note that you may use the `lite` argument as `false`, as it makes no sense to transform a event and then not pass it to the listener!

Quiz

True or False? In Map Listeners, transforms are executed before the listener is provided the event, but can only partially transform the event.

- a. True
- b. False



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

Answer: b

Message Listeners transforms are executed before the event is processed, but can completely rewrite the event if desired.

Practice.05.01 Overview: Working with Map Listeners

This practice covers the following topics:

- Creating a simple map listener
- Registering and testing the map listener programmatically
- Registering and testing the map listener declaratively and run a test using a Run Configuration



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

Agenda

Event Processing

Map Listeners

Map Triggers

- Concepts
- Implementation, interfaces and registration

Backing Maps and Events

Events and the Continuous Query Cache



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

Map Triggers - Review

Map Triggers:

- Execute *before* data changes occur
- Execute *synchronously* in response to a pending change/delete to the corresponding map entry
- Are guaranteed to execute *once* per change
- Can be used to validate, modify, or *reject* an operation
- Can be registered programmatically or declaratively
- Must implement `hashCode()` and `equals()` methods

Conceptually similar to
database triggers

```
public class SimpleMapTrigger implements MapTrigger {  
    public void process(Entry entry) { . . . }  
    public boolean equals(Object o) { . . . }  
    public int hashCode() { . . . }  
}
```

SimpleMapTrigger.java



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

MapTrigger Basics

Map triggers are implemented by:

1. Extending the MapTrigger interface and implementing the process method
2. Implementing equals() and hashCode()
3. Registering the trigger

```
import com.tangosol.util.MapTrigger;           Required imports
public class SimpleMapTrigger implements MapTrigger {   Implements...
    @Override
    public void process(Entry entry) {
        System.out.println("Called with '" + entry + "'");
    }
    public boolean equals(Object o) {
        return o != null && o.getClass() == this.getClass();
    }
    public int hashCode() {
        return getClass().getName().hashCode();      Minimal equals and
    }                                              hashCode
}
```

SimpleMapTrigger.java



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

MapTrigger Basics

Maptriggers are simpler to implement than listeners, requiring only a single process method be implemented for user functionality. However, Maptriggers do require hashCode and equals methods for management.

Maptriggers require that you implement hashCode and equals because they are essentially placed in an internal HashSet of Triggers, in which these methods are used to differentiate between instances, especially when adding/removing. This management differs from MapListeners which are managed via references.

Registering MapTriggers Programmatically

Registering a MapTrigger programmatically requires:

- An instance of the class implementing MapTrigger
- A wrapper MapTriggerListener
- Registration of the wrapper using addMapListener(...);

```
NamedCache cache=CacheFactory.getCache("airports");  
  
SimpleMapTrigger trigger = new SimpleMapTrigger();  
  
MapTriggerListener listener = new MapTriggerListener(trigger);  
  
cache.addMapListener(listener);
```

Application.java



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

Registering MapTriggers Programmatically

MapTriggers are registered programmatically in a three-step process:

1. Create an instance of a class that implements a MapTrigger Listener.
2. Create a com.tangosol.util.MapTriggerListener wrapper instance passing the MapTrigger class.
3. Pass the wrapper instance to the addMapListener method of the cache instance.

MapTrigger Example

```

import com.tangosol.util.MapTrigger;
public class ToUpperMapTrigger implements MapTrigger { Implements...

    public void process(Entry entry) {
        System.out.println("\n\nProcess('" + entry.getValue() + "')");
        Object o = entry.getValue();           ← Use entry arg to get
        AirPortJS ap = (AirPortJS)o;          ← current value object
        String upper = ap.getName().toUpperCase(); ← Cast it as required
        ap.setName(upper);                  ← Change it somehow
        System.out.println("Replacing entry with:" + ap);
        entry.setValue(ap);                ← Save the change
    }

    public boolean equals(Object o) {
        return o != null && o.getClass() == this.getClass();
    }

    public int hashCode() {
        return getClass().getName().hashCode();
    }
}

```

ToUpperMapTrigger.java



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

MapTrigger Example

The power of a MapTrigger comes from a combination of when it is called in an object's lifecycle as well as its ability to change an object in the background.

In the example shown, a map trigger is used to replace the value of a field with an upper case value. In this particular case, you use the entry arguments `getValue()` method to return the object, then cast it to the appropriate type. Note that production code would likely check the class type using `instanceof`. Then change one of the values in some way and store it back into the entry.

Registering a MapTrigger Declaratively

MapTriggers:

- Are registered similarly to map listeners
- Require a Factory class to produce an instance
- Factories are registered in coherence-cache-config.xml

```
<cache-config>
  ...
    <distributed-scheme>
      ...
        <listener>
          <class-scheme>
            <class-factory-name>[package.]Factory</class-factory-name>
            <method-name>createTriggerListener</method-name>
          </class-scheme>
        </listener>
      <distributed-scheme>
    ...
</cache-config>
```

coherence-cache-config.xml



ORACLE

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

Registering a MapTrigger Declaratively

MapTriggers are registered almost identically to MapListeners, with the exception of the use of a class factory. In the example shown a class, Factory, returns, via its createTriggerListener method, a MapTrigger Listener.

Additionally such factory methods can take a parameters by following the method name element with an init-params element. The init-params element then contains one or more init-params each a param-type/param-value set. Currently supported types are string, int, long, double and float.

An example of a method which takes a string might be:

```
<method-name>createTriggerListener</method-name>
<init-params>
  <init-param>
    <param-type>string</param-type>
    <param-value>value</param-value>
  </init-param>
</init-params>
```

MapTriggers and Exceptions

Map triggers can be registered programmatically or declaratively, but raise exceptions in the same way in both scenarios. Exceptions:

- Are logged locally on the storage-enabled member in which they occurred
- Are wrapped in a `RuntimeException`
- Are returned to the client which resulted in the trigger



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

MapTriggers and Exceptions

All event processors can raise exceptions, and map triggers are no exception. There are several ways an exception is handled when raised in a map trigger:

- If no client is involved, the exception is logged
- if a client was involved, then the exception is passed to the client who “caused” it.

In all cases the exceptions are wrapped in a `RuntimeException` and the resulting exception returned.

Remember: Throwing an exception in a map trigger method causes the mutation of the object to be discarded.

Quiz

Which are true of Map Triggers? (Select all that apply)

- a. They are asynchronous
- b. They execute after a mutation has occurred
- c. They are synchronous
- d. They execute before an event has occurred
- e. Map triggers can be registered programmatically or declaratively



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

Answer: c,d,e

Practice.05.02 Overview: Working with Map Triggers

This practice covers the following topics:

- Creating a map trigger
- Registering and testing the map trigger declaratively
- Testing using a Run Configuration



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

Agenda

Event Processing

Map Listeners

Map Triggers

Backing Maps and Events

- Concepts
- Developing Backing Map Listeners
- Registering Backing Map Listeners

Events and the Continuous Query Cache



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

Backing Map Listeners and Events

Backing Map Listeners typically execute on all nodes in the cluster. Backing Maps Listeners:

- Are similar to MapListeners
- Differ in that:
 - They execute *synchronously after* an entry is added/updated/delete
 - Are guaranteed to execute *at most once* per change
 - They provide virtually zero latency events
 - They are registered and embedded on the server side *only*
 - Only a single instance occurs per distributed cache
 - Receive event entries based on Binary objects

Data must be deserialized on demand, not automatically



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

Backing Map Listeners and Events

For some advanced use cases, it may be necessary listen to the map behind the service. Replication, partitioning and other approaches to managing data in a distributed environment are all distribution services. The service still has to have something in which to actually *manage* the data, and that *something* is called a "backing map."

Backing maps are configurable. If all the data for a particular cache should be kept in object form on the heap, then use an unlimited and non-expiring LocalCache (or a SafeHashMap if statistics are not required). If only a small number of items should be kept in memory, use a LocalCache. If data are to be read on demand from a database, then use a ReadWriteBackingMap, and in turn give the ReadWriteBackingMap a backing map such as a SafeHashMap or a LocalCache to store its data in.

Backing Map Listener Basics

Backing Map Listeners:

- Are implemented by extending `MapListener`
- Require a constructor taking a `BackingMapContextManager`
- Are configured as a `<listener>` on a `<backing-map-scheme>`

```
import com.tangosol.net.BackingMapManagerContext;
import com.tangosol.util.MapEvent;
import com.tangosol.util.MapListener;

public final class EchoingBackingMapListener implements MapListener {

    private BackingMapManagerContext context;
    public EchoingBackingMapListener (BackingMapManagerContext context) {
        this.context = context;
    }
    .
    .
}
```

EchoingBackingMapLister.java



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

Backing Map Listener Basics

Backing Map Listeners are developed similarly to their `MapListener` counterparts, implementing the same `MapListener` interface. However, that is where the similarities end.

In the partial example shown (remainder to follow), there is public constructor which takes a `BackingMapManagerContext`. This context is used to access converter information so that event values can be converted to and from their internal Coherence format. Remember, backing map listeners receive events like listeners, but the event contents differ, containing the internal Coherence Binary object format. In order to examine, change, or otherwise manipulate the contents of a object, it must be returned to its original object format. The context provides mechanism to support this conversion.

Backing Map Listener Event Conversion

Backing Map Listeners:

- Must implement the same event methods as `MapListeners`
- Will need to use a context to convert from keys and objects internal format to/from original format

```
.
.
protected Object getConvertedKey (MapEvent event) {
    return context.getKeyFromInternalConverter().convert(event.getKey());
}

protected Object getConvertedValue (Object internal) {
    return context.getValueFromInternalConverter().convert(internal);
}

protected Object getOldValue(MapEvent event) {
    return getConvertedValue(event.getOldValue());
}

protected Object geNewValue(MapEvent event) {
    return getConvertedValue (event.getNewValue());
}
.
```

*Commonly developed
Map Listener
methods*



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

Backing Map Listener Event Conversion

Backing map listeners receive events which contain the Coherence internal format of an object. As a result, the events are delivered with little latency, since no conversion is required. However, to be useful, the event contents require conversion to their original values.

The slide shows several convenience methods which use the `BackingMapManagerContext` and two of its conversion methods to convert the internal object format back to its original format.

Selected BackingMapManagerContext Methods

Common BackingMapManagerContext conversion routines:

```
package com.tangosol.net;

public class BackingMapManagerContext extends XmlConfigurable {

    public abstract Converter getKeyToInternalConverter();
    Return a Converter which can convert a key TO Coherence internal format.

    public abstract Converter getKeyFromInternalConverter();
    Return a Converter which can convert a key FROM Coherence internal format.

    public abstract Converter getValueToInternalConverter();
    Return a Converter which can convert a value TO Coherence internal format.

    public abstract Converter getValueFromInternalConverter();
    Return a Converter which can convert a value FROM Coherence internal format.

    ...
}
```

BackingMapManagerContext.java



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

Selected BackingMapManagerContext Interface Methods

Some useful BackingMapManagerContext methods include the following:

- `public abstract Converter getKeyToInternalConverter()`: Return a Converter object which can be used to convert a key object represented in its normal external format to a coherence internal format
- `public abstract Converter getKeyFromInternalConverter()`: Return a Converter object which can be used to convert a key object from Coherence internal format to normal external format
- `public abstract Converter getValueToInternalConverter()`: Return a Converter object which can be used to convert a value object represented in its normal external format to a coherence internal format
- `public abstract Converter getValueFromInternalConverter()`: Return a Converter object which can be used to convert a value object from Coherence internal format to normal external format

Keys and Backing Map Ownership

- Backing maps may not manage a specific key/value pair due to a variety of reasons, including partitioning.
- Backing maps can test keys to determine if they are being managed by a backing map using `isKeyOwned`.

```
package com.tangosol.net;

public class BackingMapManagerContext extends XmlConfigurable {
    . . .
    boolean isKeyOwned(java.lang.Object oKey);
    . . .
}
```

Return `true` if the key is owned, or managed, by the backing map. Note that the key is in internal format.

BackingMapManagerContext.java



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

Keys and Backing Map Ownership

A backing map may only contain a partial set of key/value pairs for a given cache. Partial coverage of a cache can happen for any number of reasons, most likely partitioning. The `IsKeyOwned` method of the backing map context supports testing if a particular key is owned/managed by the backing map associated with the current backing map listener.

`IsKeyOwned()` is used often with repartitioning events. When an entry is moved from one partition to another, the owning partition, for example the partition where the element is being moved to, returns `true`. Backing map listeners can potentially ignore the source partition and process the target side only.

Registering Backing Map Listeners

Backing Map Listeners are registered as part of the `<backing-map-scheme>` element of a `<[type]-scheme>`



```
<cache-config>
  ...
<caching-schemes>
  <....-scheme>
    <schema-name>... </schema-name>
    <service-name>... </service-name>
    ...
    <backing-map-scheme>
      <local-scheme>
        <scheme-ref>... </scheme-ref>
        <listener> ← Backing map schemes support the
                      listener element
          <class-scheme>
            <class-name> ← Fully qualified package not shown
              ... EchoingBackingMapListener</class-name>
            </class-scheme>
        </local-scheme>
      </backing-map-scheme>
    </....-scheme>
</cache-config>
```

coherence-cache-config.xml



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

Registering Backing Map Listeners

Backing Map listeners are registered in a fashion similar to their map listener counterparts, using a `<listener>` element, but as part of the `<backing-map-scheme>` element.

A `<listener>` element is added which specifies a `<class-scheme>`, which itself then specifies the fully qualified path name to the backing map listener within a `<class-name>` element.

Immediately following the `</class-scheme>` is a `<init-params>` element specifying, at a minimum, a backing map manager context.

Note that the backing map manager context uses the `{manager-context}` macro, which is filled in at run-time with an actual value.

Registering Backing Map Listeners

Backing Map context is specified via an `<init-params>` element using the `{manager-context}` macro.

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

- A comment above the code: "Fully qualified package required but removed for clarity".
- An annotation pointing to the `<param-value>{manager-context}</param-value>` line: "Macro! Replaced at run time with an actual object instance."
- A small icon of an XML document is in the top right corner.
- The file name "coherence-cache-config.xml" is in a yellow box at the bottom right.

```
...  
<init-params>  
  <init-param>  
    <param-type>... BackingMapManagerContext</param-type>  
    <param-value>{manager-context}</param-value>  
  </init-param>  
  </init-params>  
</listener>  
</local-scheme>  
</backing-map-scheme>  
</...-scheme>  
</caching-schemes>  
</cache-config>
```



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

Backing Map Listeners and Exceptions

Backing map listeners are always registered declaratively.

Exceptions on backing map listeners:

- Are server side exceptions
- Are wrapped in `RuntimeException` and logged
- Do not traverse the network as there is no client to receive them
- Must be carefully coded to not affect the cluster



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

Backing Map Listeners and Exceptions

Extreme care should be taken when raising exceptions in backing map listeners. Such exceptions will be wrapped as Runtime Exceptions, then logged on the storage enabled member in which they occurred. The exceptions are not sent across the network, as there essentially is no client. Exceptions should not "bring down" the Coherence Service on which the exception occurred.

Backing Map Listener Best Practices

Backing Map Listeners:

- Should never process events on the calling thread, but rather should always add an event to a backing thread queue from processing
- Should never perform costly operation such as accessing the network
- Should never access another cache directly, which can cause deadlock
- Should extend the `AbstractBackingMapListener` class provided in the Coherence incubator



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

Quiz

Which are true of backing map listeners? (Select all that apply)

- a. They are registered declaratively
- b. They execute on a coherence server and not a client
- c. They are executed asynchronously after an entry is mutated in some way
- d. They are executed synchronously after an entry is mutated in some way
- e. Require a context to convert from Binary to original object form



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

Answer: a,b,d,e

Backing map listeners are:

- Rarely registered programmatically, in fact it requires special understanding of Coherence APIs to register a backing map listener programmatically.
- Run on a server, not a client, when registered declaratively
- Are executed synchronously after an event
- Are provided Coherence internal format objects, and as such, must use a `BackingMapManagerContext` to convert values

Practice.05.03 Overview: Working with Backing Map Listeners

This practice covers the following topics:

- Creating a backing map listener
- Registering and testing the backing map listener



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

Agenda

Event Processing

Map Listeners

Map Triggers

Backing Maps and Events

Events and the Continuous Query Cache

- Concepts
- Continuous Query Cache Observation



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

Continuous Query Cache

A Continuous Query Cache, or CQC is:

- Based on an existing cache and a filter
- A local cache that is constantly updated as data changes
- Useful for:
 - Real-time consoles (update a GUI screen immediately if data is updated on the grid)
 - Complex event processing
 - Any application that needs zero latency access to a subset of data in a distributed cache



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

Continuous Query Cache

It would be easy to imagine a cache which is maintained as a result of listener events. Such a cache would be maintained in real time, and always consistent. Coherence provides such a cache in the form of a Continuous Query Cache. Such a cache provides a local, in process view of data that is updated as if a query was constantly run with no latency. Effectively, Coherence provides a feature that combines a query result with a continuous stream of related events in order to maintain up-to-date query results in a real-time fashion.

Coherence implements the Continuous Query functionality by materializing the results of the query into a continuous query cache, and then keeping that cache up-to-date in real time using event listeners on the query.

The benefits of a continuous query cache:

- It is an ideal building block for Complex Event Processing (CEP) systems and event correlation engines.
- It is ideal for situations in which an application repeats a particular query, and would benefit from always having instant access to the up-to-date result of that query.

Continuous Query Cache (continued)

- A continuous query cache is analogous to a materialized view, and is useful for accessing and manipulating the results of a query using the standard NamedCache API, as well as receiving an ongoing stream of events related to that query.
- A continuous query cache can be used like a near cache, because it maintains an up-to-date set of data locally where the data is being used, for example, on a particular server node or on a client desktop. Note that a near cache is invalidation-based, whereas the continuous query cache actually maintains its data in an up-to-date manner.

An example is a trading system desktop in which a trader's open orders and all related information must be maintained in an up-to-date manner at all times.

Continuous query cache is a powerful feature in Coherence. A continuous query cache is similar to the Oracle Database materialized view. In Oracle Database, when you create a materialized view, it copies data from the tables that are involved into the view. Any changes that occur to the database are automatically updated in the view. This essentially allows you to see the changes to a result set. This is a powerful feature in Oracle Database.

There is a similar functionality in Coherence called the continuous query cache. A continuous query cache is a local copy of a cache that is created on the client. You can use filters that limit the size of the cache. Combined with an event listener, the cache can be updated in real time.

For example, on your client, you want to monitor all your sales orders for customers in real time. You create a continuous query cache. When you create a continuous query cache, you also start up an event listener that listens for all events of customers. Coherence automatically creates a local copy of a cache and queries for all objects on the grid of that particular customer. Coherence places a local copy of a cache, copies the objects of that customer into the local cache, so that they can be accessed with fast memory speeds. The local cache automatically listens for any inserts, updates, or deletes on the grid for that particular customer.

This is a very powerful functionality for real-time consoles where you update the GUI screen and the data is updated immediately on the grid. This is used in financial services where brokers can see a real-time console on their screens—for example, positions that a broker currently holds in a particular security. When market conditions change or as orders come in, brokers can take decisions based on real-time events. Continuous query cache can also handle complex event processing and if you want near zero latency, the continuous query cache acts as a subset of data in the distributed cache.

When you perform a regular query, Coherence automatically sends that query to the grid and executes it in parallel on the storage JVMs. However, if you execute the same query against the same data set multiple times by using a continuous query cache, that data is copied in local memory (JVM). So your access time is reduced to a few microseconds because the data is in-memory on the client. This dramatically improves the performance of reads.

Continuous query cache can be used for both real-time event monitoring on the client as well as caching to improve read speed.

Continuous Query Cache Basics

A continuous query cache:

- Is based on an existing cache
- Uses a filter to keep the cache up to date

```
import com.tangosol.net.NamedCache;
import com.tangosol.net.cache.ContinuousQueryCache;           Required
import com.tangosol.util.Filter;                                imports
import com.tangosol.util.filter.*;                              

NamedCache flights = CacheFactory.getCache("flights");          Based on a specific cache

Filter filter = new AndFilter(                                     Maintained
    new EqualsFilter("state", "flying"),                         based on a
    new EqualsFilter("departedFrom", "BOS"));                     filter

ContinuousQueryCache inTheAir =                                     CQC created from the
    new ContinuousQueryCache(flights, filter);                   original cache
                                                               and filter
```



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

Continuous Query Cache Basics

Coherence implements continuous query using a combination of its data fabric parallel query capability and its real-time event filtering and streaming. The result is support for thousands of client application instances, such as trading desktops or air traffic controllers.

To create a CQC:

1. Create a normal cache, for example, a set of flights, as shown on the slide.
2. Create a filter which represents the query to be executed.
3. Associate the query with the cache, to create a cache which is constantly up to date.

The result of the continuous query is maintained locally, and optionally, all the corresponding data can be cached locally as well. The Coherence implementation of continuous query is found in the `com.tangosol.net.cache.ContinuousQueryCache` class. This class implements the standard `NamedCache` interface.

Note that changes made to an entry in a CQC are reflected in the parent cache.

Summary

In this lesson, you should have learned how to:

- Describe, compare and contrast listener types and lifecycle
- Implement and register map listeners
- Filter and transform map listener events
- Implement and register map triggers
- Implement backing map listeners
- Understand and use continuous query caches



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

Querying and Aggregating Data in the Cache



ORACLE®

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

Objectives

After completing this lesson, you should be able to:

- Develop Filters to subset cache data, including sorting and aggregation of data objects
- Construct Filters using ValueExtractors
- Develop custom aggregators, which can run in parallel in a Coherence cluster
- Extend objects to support data affinity
- Implement Java clients that can perform dynamic queries using the Coherence Query Language (CohQL)
- Improve filter and query performance using Indexes



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

Agenda

Coherence Queries using Filters

- Filter Concepts and the Query Map Interface
- Accessing Object Attributes
- Query Processing
- Sorting, Paging, and Aggregating
- Improving Performance using Indices

Coherence Query Language



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

Filters and Caches

- Caches are easily accessible via get operations, but *only* using well-known object keys and sets of keys
- Filters can access a cache using identities, object attributes, or combinations via the `QueryMap` interface
- `QueryMap` methods return a Set of objects based on a Filter

```
package com.tangosol.util;

public interface QueryMap extends Map {

    public abstract Set keySet(Filter filter);
    public abstract Set entrySet(Filter filter);
    public abstract Set entrySet(Filter filter, Comparator comparator);
}
```

QueryMap.java



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

Filters and Caches

In prior lessons, you examined how to create and access caches either returning all elements, or a specific element using that element's identity. However, often you need to access cache data based on other attributes of data, where you don't know an object's identity, or access data using both scenarios. The query map interface has two core functions:

- It provides the ability to perform simple queries against a cache. These queries are either for the keys of a set of objects, the entries themselves, or a sorted set of the elements.
- Coherence also provides indexing and parallelization features. Indices are especially useful with partitioned caches. By adding servers to a partitioned cache, you can increase throughput (total queries per second) and reduce latency by reducing the time taken by queries.

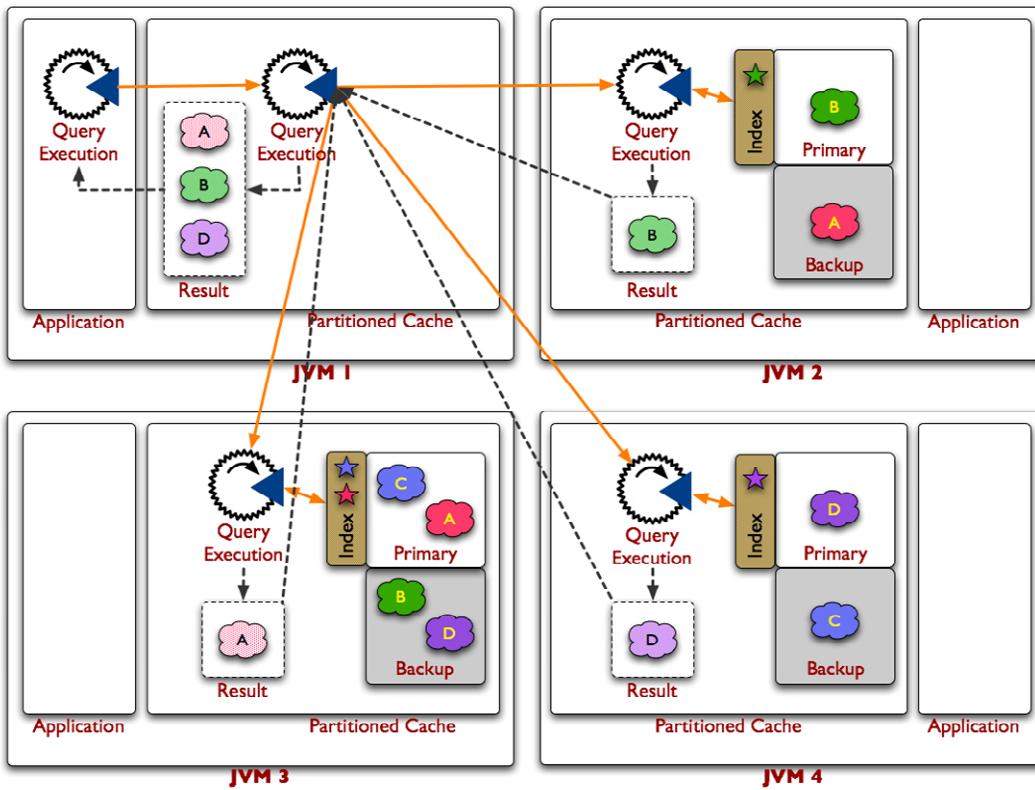
In relational database terms, you can create indexes on a column. In Coherence, indexes are created by attribute. This improves the performance of the query just completed in a relational database.

Filters and Caches (continued)

Methods of the `QueryMap` interface for filtering include:

- The `Set entrySet (Filter filter)` returns a set view of the entries contained in this cache that satisfy the criteria specified in the filter. Use this method to get an entry set. You can pass a filter to that entry set. The filter object represents the criteria that the entries of this cache should satisfy. Similar to the JDBC result set, this method returns a collection of values, which is the result set from the query.
- The `Set entrySet (Filter filter, Comparator comparator)` method returns a set view of the entries contained in this cache that satisfy the criteria specified in the filter. In addition, the comparator parameter performs a comparison function, which imposes a total ordering on a collection of objects. You can pass a filter parameter and a comparator to the `entrySet` method. This comparator parameter can also be custom code that you have written to perform comparisons, for example, if you have specialized data and you have overwritten the `equals()` method in your object. If you have a special need for comparing objects, to determine whether they are equal or whether one object is greater than or less than another, you can pass a comparator parameter to the `entrySet` method.
- The `Set keySet (Filter filter)` method returns a set view of the keys contained in the cache for entries that satisfy the criteria specified in the filter. The `keySet ()` method returns a set of the keys that meet the result of the queries rather than the entries themselves.

Filter Execution



ORACLE

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

Filter Execution

Examine the graphic in the slide and note the following:

- At the top left, observe that you have a client that executes a query.
- Coherence automatically executes this query in parallel on all the JVMs that store the data. In Coherence, you have multiple JVMs spread out across multiple physical servers. In this process, Coherence automatically spans a task on each of these JVMs, automatically does the filtering on the query, gets the results, and sends the result set back to the client.
- Note the word “Index” in the brown box in the graphic. Coherence has a simple built-in optimizer. If there is an index that you created, Coherence uses this index to get better performance from the query.
- The parallelization is done automatically, and you need not specify anything on the client. Coherence automatically optimizes and parallelizes the query. This means that in a typical Coherence deployment, the power of all the CPUs in your physical servers are being used to perform the query.

Developing Filters

Coherence provides a set of out of the box filters, but supports custom filters via the `Filter` interface:

```
package com.tangosol.util;
public interface Filter {
    public abstract boolean evaluate(Object obj);
}

import com.tangosol.util.Filter;
public class InFlight implements Filter, Serializable {
    public boolean evaluate(Object o) {
        Flight flight= (Flight) o;
        return flight.isInFlight();
    }
}

NamedCache flights = ...;
Set inflightFlights = flights.entrySet (new InFlight());
```

The same filter you saw in events!

Filter.java

InFlight.java

Application.java



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

Developing Filters

Filters are nothing more than classes which implement the `com.tangosol.util.Filter` interface. This interface has a single method `evaluate` which returns true in the event the object meets the criteria of the filter.

In the example provided, the `InFlight` filter returns all `Flight` objects which are marked as such. An application then uses the filter against a cache to return objects which meet the filter criteria.

Accessing Object Properties

Filters can access object properties using method names.

```
public class Flight implements Entity<Long>, Comparable, . . . {  
    . . .  
    public boolean isInFlight() { return mbInFlight; }  
    . . .  
}
```

Assuming
this class...

... you can rewrite the prior
example using a stock filter to
access a method as

```
NamedCache flights = ...;  
Set inflightFlights = flights.entrySet (new EqualsFilter("isInFlight", BOOLEAN.TRUE));
```

ORACLE

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

Accessing Object Properties

Filters can access object values via the getter methods as show in the example above.

In this example, the flights in progress are returned using an out-of-the-box filter known as the EqualsFilter. This filter takes a method name, in this case `isInFlight`, and compares its value to the provided one. Any objects which report the value provided for the method specified are added to the set.

Out-of-the-box Filters

Most expressions can be expressed in terms of an object's attributes and traditional logical operations such as AND, OR, NOT, equals, and others.

Filter	Description (<code>com.tangosol.util.filter</code>)
EqualsFilter	Compares a value extracted via a method to a value
AllFilter	Performs the logical and of a set of filters in a filter array
ContainsFilter	Returns <code>true</code> if the value extracted via a method contains the provided Object
IsNullFilter	Returns <code>true</code> if the value extracted via the method is null. See also <code>isNotNull</code> .
LessFilter	Returns the <code>true</code> if the value exacted is less than the provided value. Returns <code>false</code> on null returns.



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

Out-of-the-box Filters

This slide shows a small subset of the supplied filters. The `com.tangosol.util.filter` package contains 34 filters, all of which can be combined using logical expressions to create complex cache filters.

A note on the Container filter: The associated value extractor must return a collection and the second argument is any Object . The collection is tested for containing that passed Object.

Example: Filtering with EqualsFilter

To filter a set of airports based on a specific country:

```
public class AirPort . . . {  
    . . .  
    public boolean getCountry() { . . . }  
    . . .  
}  
  
import com.tangosol.util.filter.EqualsFilter;  
.  
.  
NamedCache airports= ...;  
  
Filter USAirports = new EqualsFilter("getCountry","US");  
Set <Map.Entry> usAirports= airports.entrySet (USAirports );  
  
for (Map.Entry entry: entries) { . . . }
```

Airport.java

Application.java



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

Example: Filtering with EqualsFilter

Using the out of the box `com.tangosol.util.filter.EqualsFilter` to select a specific set of airports:

1. Create a filter using the `getCountry` method exposed by the `AirPort` class. This filter uses the `String` constructor of the `EqualsFilter` object to select "US" airports
2. Using the filter obtain an entry set, note that entry sets return `Map.Entry` elements, and this example qualifies the result.
3. Process the results as required.

Note that the `EqualsFilter` provides constructors for a variety of types other than `String`. See the JavaDoc for a complete set.

Sorting

Result sorting on Filters happens in two ways:

- Natural: Using the object's Comparable implementation
- Specific: Using a provided Comparator

```
package com.tangosol.util;  
  
public interface QueryMap extends Map {  
    . . .  
    public abstract Set entrySet(Filter filter,  
                                Comparator comparator);  
    . . .  
}
```

QueryMap.java



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

Sorting

Objects returned by Filter operations are sorted in one of three specific ways:

- Random, and not Comparable: If an object does not implement the Comparable interface, then the ordering of returns based on filter is often the order that the objects were entered into the cache based on the hash of the key. This is most likely a result of differing cache topologies. Objects can come from more than one node, in which case the order is effectively random.
- Order based on Comparable: If an object does implement the Comparable interface, and no overriding Comparator is provided, then the objects will be returned in what is considered *natural* order. Natural order is the order the objects enforce on themselves.
- Specific order: If a Comparator is provided, regardless of whether the object implements Comparable, then objects are returned based on the order defined by the Comparator itself.

Example: Sorting with a Comparator

```

    ...
public class AirportComparator implements java.util.Comparator {
    public int compare(Object first, Object second) {
        AirPort firstAP = (AirPort) first;
        AirPort secondAP = (AirPort) second;
        if (firstAP.getCountry().equals(secondAP.getCountry())) {
            return firstAP.getCity().compareTo(secondAP.getCity());
        } else {
            return firstAP.getCountry().compareTo(secondAP.getCountry());
        }
    }
}

```

Orders by city within country

airportcomparator.java

```

    ...
NamedCache airports= ...;
...
Filter USAirports = new EqualsFilter("getCountry", "US");

Comparator apc = new AirportComparator();
Set <Map.Entry> usAirports= airports.entrySet (USAirports,apc);
...

```

Application.java

ORACLE

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

Example: Sorting with a Comparator

Writing a Comparator is simple process if you understand the return results. Comparators are defined to provide a complete ordering of elements. The return is one of the values based on the provided objects. It returns:

- A negative value: If the first object is less than the second, return a negative value. The exact value is unimportant, just that it is less than zero.
- Zero: If the first object is equal to the second object, return zero.
- A positive integer: If the first object is greater than the second object, return a value greater than zero. The exact value is unimportant, just that it is more than zero.

Paging and LimitFilters

LimitFilters

- Differ from other filters which are created, used, and discarded
- Take a traditional filter and a number of results to return per page as arguments
- Uses `nextPage()` to return the next N results

```
...
NamedCache airports= ...;
Filter USAirports= new LessEqualsFilter("getCountry", "US");
Filter pagedAirports = new LimitFilter (USAirports, 5);
...
Set <Map.Entry> usAirports = null;
do {
    usAirports= airports.entrySet (pagedAirports, . . .);
    if ( usAirports.isEmpty()) break;
    ...
    pagedAirPorts.nextPage();
} while (true);
```

Application.java



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

Paging and LimitFilters

One filter which requires special notice is the `LimitFilter`. A `LimitFilter` is not really a filter at all, but rather acts as a buffer returning sets of objects based on a limit. When creating a limit filter, another filter is provided, along with a count of objects to return. Then, executed by a `entrySet` operation, the filter defines what elements to return, and the limit filter defines how many to return. Second and subsequent sets of values are returned via the `getNextPage()` method, and then reexecuting the `entrySet` operation.

ValueExtractors

- Most filters that can accept a method name can also accept a ValueExtractor.
- ValueExtractors provide a mechanism for more generic filters to act on values.
- Generic filters can then be coupled with value extractors to obtain values.

```
package com.tangosol.util;

public interface ValueExtractor {
    public abstract Object extract(Object obj);
    public abstract boolean equals(Object obj);
    public abstract int hashCode();
    public abstract String toString();
}
```

valueextractor.java



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

ValueExtractor~~s~~

Custom filters are reasonably straight forward to write, but depend on the internals of an object. Out of the box filters require no development, but most often require well-defined methods to extract values. ValueExtractors bridge the gap between these two mechanisms by providing a mechanism to encapsulate the return of values from objects, without the need to encode the logic in the filter itself.

There are a number of out of the box value extractors that can be used in a variety of generic cases. These extractors will be examined on the next few slides.

Provided ValueExtractors

Select value extractors include:

Extractor	Description (com.tangosol.util.Extractor)
AbstractExtractor	Abstract base for ValueExtractor implementations, which provides common functionality to allow extending extractors to be used as a value Comparator.
ReflectionExtractor	The reflection extractor is used to extract a single attribute and in prior examples where an attribute name was provided this extractor used automatically.
IdentityExtractor	The identity extractor doesn't actually extract a value, but rather returns the target object itself.
ChainedExtractor	The chained extractor takes an array of extractors, and executes them left to right, proving the return value from the first to the second, and so on.



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

Provided ValueExtractors

This slide provides a partial list of out of the box value extractors. Students are directed to the com.tangosol.util.Extractor package for a complete list.

Value Extractors and Dot notation

- Often its necessary to access a property of a property.
- Chained extractors can be used to access nested properties as required.

```
ValueExtractor ve  
    = new ChainedExtractor ("getFlight.getFlightId" );
```

- Most value extractors constructors will accept a Dot notation string value to access nested properties.

```
NamedCache customers= ...;  
Filter filter = new EqualsFilter("getFlight.getFlightId",10);
```



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

Practice.06.01 Overview: Filtering and Sorting Data

This practice covers the following topics:

- Developing and using a comparator for sorting instance
- Developing Filters to subset cache data



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

Aggregating Results

- Aggregation is the process of applying a process or function to a set of values to retrieve a single result, such as a sum or average.
- Filters can be used to aggregate by iterating over all values, but force the return of all data to a single process which may result in:
 - High network traffic due
 - Single threaded bottle neck in the processing application
- Aggregators allow the process to be executed on each node.

```
public abstract class AbstractAggregator
    implements InvocableMap.ParallelAwareAggregator {
    .
    .
}
```

AbstractAggregator.java

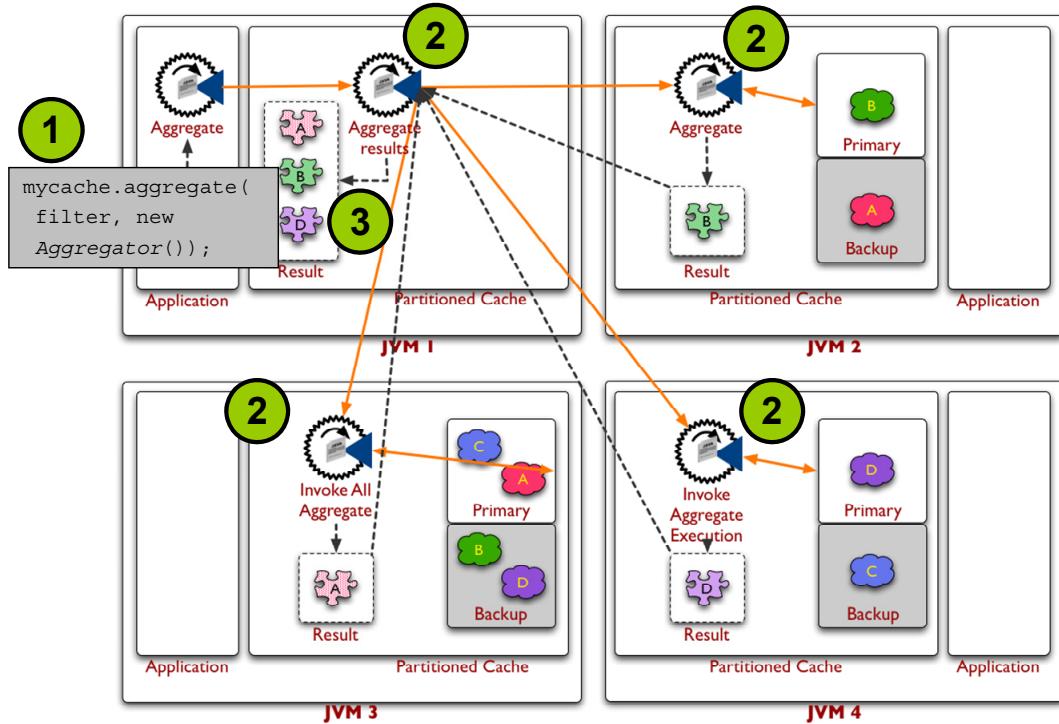


Copyright © 2010, 2012, Oracle and/or its affiliates. All rights reserved.

Aggregating Results

Aggregating results is a common operation in most database applications. Typical aggregations include sums, averages, minimums, maximums, oldest, and so on. Performing aggregations with Filters is a simple process, but loses some of the distributed benefits of Coherence by pushing all data to a single process. EntryAggregators solve this problem by allowing developers to push processing to the nodes containing the data and then return a final result. With aggregators, each node calculates a partial result, and then aggregates the partial result into a single final result.

Understanding Aggregation Execution



ORACLE

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

Understanding Aggregation Execution

In theory, aggregators can run on any node in the cluster. Each cluster node then processes a portion of the aggregation, and returns its values back for final aggregation.

The aggregation process is where each cluster node processes its portion of the aggregation. For example, if a set of flights was being distributed across all members of the cluster, each cluster member would have some portion of the total flights. The local node would then aggregate its portion of flights and send its partial result back to the requestor. The original requestor then aggregates all the partial results.

Each time a call is made to process a result, it can be in one of two modes: final or intermediate. When an intermediate call is made, the aggregation code simply aggregates the next value provided to it. If the call is final, then the aggregator is actually processing individual aggregation rollup results.

Developing Aggregators

Aggregators are typically developed against `AbstractAggregator`, which supports parallel processing and provides three methods which must be implemented.

```
public abstract class AbstractAggregator
    implements InvocableMap.ParallelAwareAggregator {
    public AbstractAggregator(ValueExtractor valueExtractor) { }

    Should call super(valueExtractor) to store the provided value extractor.

    protected abstract void init(boolean isFinal);
    Initializes the aggregation class, here is where context variables initialized.

    protected abstract void process(Object value, boolean isFinal);
    Called to perform processing.

    protected abstract Object finalizeResult(boolean isFinal);
    Called to return the final result
}
```

AbstractAggregator.java



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

Developing Aggregators

Technically, aggregators are developed by extending the `com.tangosol.util.InvocableMap.EntryAggregator` interface. However, in practice, they implement the `AbstractAggregator` class, which provides a set of common code, which can be used to streamline the development processing.

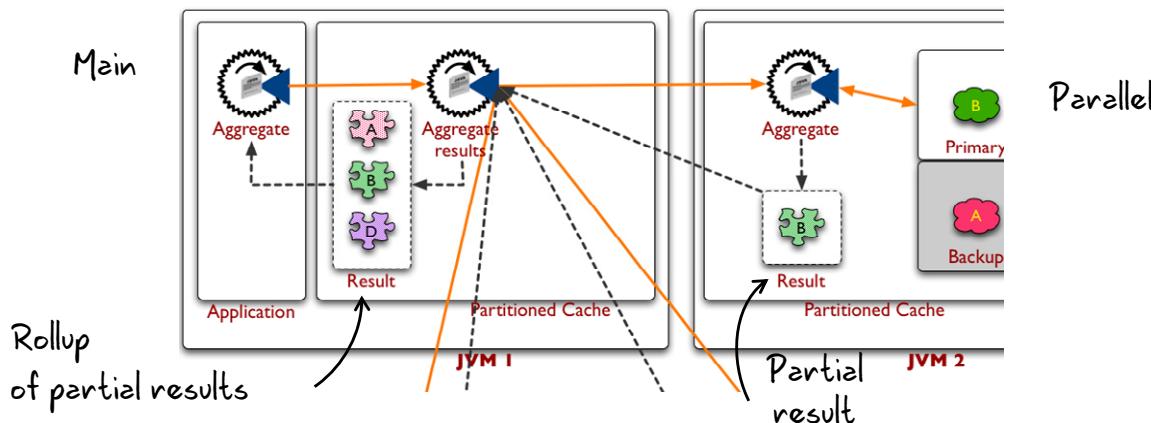
Aggregators developed this way must implement three methods and a constructor:

- Constructor: each aggregator must implement a constructor which `AbstractAggregator(ValueExtractor valueExtractor)` mimics by calling `super`
- `init`: Initializes and temporary variables
- `process`: Performs any required processing, such as keeping running totals, and so on
- `finalResult`: Returns the end result of the processing.

Main versus Parallel Execution

Aggregators run as either main or parallel.

- Main aggregators: Responsible for summarizing partial results
- Parallel aggregators: Responsible for creating partial results



ORACLE

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

Main versus Parallel Execution

Aggregators run in two specific ways: main and parallel. Understanding the distinction is crucial to correct implementation.,

In main mode, the aggregator is started via an application. The main aggregator may aggregate results, acting in parallel, if on a storage enabled node with content, or as the diagram shows, acts to roll up partial results to provide the final aggregation.

In parallel mode, the aggregator takes a set of values, based on a Filter, and a value extractor, and creates intermediate results.

Since the same class is used in both scenarios, a mechanism must be provided to determine which mode to execute. The mechanism for this is a simple flag passed to each of the three aggregator methods, `isFinal`, which tells the method which set of semantics to execute.

Working with `isFinal`

Method	Semantics	Object Argument/return
<code>init</code>	Has a single purpose, initialize the aggregator.	NA
<code>process (false)</code>	the method should execute to construct a partial value, such as adding to a total	Object contents of type returned by value extractor passed by constructor
<code>process (true)</code>	The method should perform a roll up of partial values	Object is of the value internally defined by the aggregator
<code>finalizeResult (false)</code>	The method should return the final result of a partial aggregation	Returns a roll up partial result. The type is often a internal use only private class.
<code>finalizeResult (true)</code>	Return the result of an entire aggregation	Returns a final result. The type being appropriate for the aggregation such as a Double etc.



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

Working with `isFinal`

Understanding how to interpret the `isFinal` flag is crucial to the development of aggregators.

The `isFinal` flag dictates how each method should function.

- `init`: The state, except in some specialized cases, is ignored and the aggregator should simply initialize to an appropriate value of zero.
- `process (false)`: This is the normal case, the aggregator should simply total results as required. In this case, the object passed to the aggregator is defined by the `ValueExtractor` passed to the constructor.
- `process (true)`: In this case, there are no more results to roll up, and the aggregator is being used to perform rollup processing. The value passed is an intermediate rollup result, which can be anything as defined by the aggregator.
- `finalizeResult (false)`: Returns a partial result as dictated by the aggregator
- `finalResult (true)`: Returns the end result of aggregating a set of partial results.

Example Aggregator

```
import com.tangosol.util.ValueExtractor;
import com.tangosol.util.aggregator.AbstractAggregator;
public class StringAggregator extends AbstractAggregator {

    transient StringBuffer results;
    public StringAggregator() { super(); }

    public StringAggregator(ValueExtractor extractor) {
        super(extractor);
    }

    protected void init(boolean isFinal) {
        results = new StringBuffer();
    }

    ...
}
```

StringAggregator.java



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

Example Aggregator

This slide shows the constructor and initialization methods of a string aggregator. The string aggregator takes a string field from an entity, which must be returned by the value extractor passed to the constructor, and compiles a list of values.

Example Aggregator

```
protected void process(Object object, boolean isFinal) {  
    if (object == null) return;  
    if (!isFinal) {  
        if (results.length() == 0) {  
            results.append((String) object);  
        } else {  
            results.append(",");  
            results.append((String) object);  
        }  
    } else { // isFinal  
        // basically the same thing  
        // but with an entire set of values  
        if (results.length() > 0) {  
            results.append(",");  
        }  
        results.append((String) object);  
    }  
}  
protected Object finalizeResult(boolean isFinal) {  
    return results.length() > 0 ? results.toString() : null;  
}
```

StringAggregator.java



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

Example Aggregator (continued)

The process method runs in either main or parallel mode as indicated by the `isFinal` method. Because of the simplicity of this aggregator, the behavior is essentially the same. Finally when the parallel aggregations are complete, the result if any, is returned.

Using Aggregators

- Aggregators are started against a cache using one of the `InvocableMap.aggregate()` methods, which `NamedMap` extends.

```
package com.tangosol.util;
public interface InvocableMap extends Map {
    ...
    public abstract Object aggregate(Collection collection,
                                     EntryAggregator entryaggregator);
    public abstract Object aggregate(Filter filter,
                                     EntryAggregator entryaggregator);
    ...
}

NamedCache airports= ...;

Filter le = new EqualsFilter("getCountry", "US");

Object result = airports.aggregate(le,
        new StringAggregator(new ReflectionExtractor("getCity")));
```



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

Using Aggregators

Aggregators are used by executing one of the aggregate methods against the `InvocableMap` interface. Named cache, as well as other caches, extends this interface so the methods are available for use.

The two forms are similar, one executing against an existing collection, and one against a filter. Each takes a class which extends the aggregator interface, and requires a value extractor.

In the example shown, the reflection extractor is used to access the city field of the `AirPort` object. Then aggregator returns a string which is a collection of all the cities names based on the provided filter, which selects only US airports.

Out-Of-the-Box Aggregators

A selection of out-of-the-box aggregators:

Aggregator	Description (com.tangosol.util.aggregator)
DoubleSum	Sums up numeric values extracted from a set of entries in a Map
Count	Calculate the number of entries
LongMin	Calculate the minimum of a set of entries
LongMax	Calculate the maximum of a set of entries
LongSum	Calculate the sum of a set of entries
DistinctValues	Return the set of unique values extracted from a set of entries in a Map



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

Out-Of-the-Box Aggregators

This slide is a partial list of provided aggregators. Students are directed to the extended by com.tangosol.util.aggregator package for a complete list.

Practice.06.02 Overview: Developing and using Aggregators

This practice covers the following topics:

- Developing a simple aggregator
- Instantiating and using an aggregator



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

Filters and Indexes

- Each application may suggest the same set of indexes when they start.
- There is no downside to suggesting an index regardless of whether another application has already suggested the same index.
- `addIndex()` is a no-op for existing indexes.
- Indexes are maintained by Cache Entry Owners. That is, for partitioned topology, the primary partitions maintain their own indexes.
- A sorted index affects the performance of filters.



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

Filters and Indexes

As discussed earlier, you can search for cache entries that meet specific criteria. You can index any queryable attribute with the `addIndex()` method. This method serves only as a hint to the cache implementation. This can be ignored by the cache if the indexes are not supported or if the desired index already exists.

Each application using Coherence may suggest the same set of indexes when it starts. However, there is no downside to executing an `addIndex()` method because it is a “no operation” (no-op) if the index already exists.

For example, in a partitioned topology, the primary partitions maintain their own indexes. Coherence knows when indexes are “on” for the attributes, and will act accordingly. Indexing requires the ability to extract attributes on each partitioned cache node. For dedicated CacheServer instances, this usually implies that the application classes must be installed in the CacheServer `CLASSPATH` environment variable.

For local and replicated caches, queries are evaluated locally against unindexed data. For partitioned caches, queries are performed in parallel across the cluster, using indexes if available. Coherence has a built-in Cost-Based Optimizer (CBO) to prioritize the usage of indexes. A sorted index determines how well filters perform, and can potentially speed up

Filters and Indexes (continued)

filters that require iteration over the index, such as `LessFilter` or `GreaterFilter`. In order to obtain results that are sorted, the `entrySet(Filter, Comparator)` should be used.

The boolean parameter in `addIndex` indicates that the index itself is sorted, and not the results of a query. A sorted index will benefit certain filters that require iterating over the contents of the index.

With partitioned caches, queries are indexed and parallelized. To query against `NamedCache`, all objects should implement a common interface (or base class). Any field of an object can be queried. Indexes are optional, and used to increase performance. With a replicated cache, queries are performed locally, and do not use indexes.

To add an index to `NamedCache`, you first need a `valueextractor`. A `valueextractor` accepts as input a value object and returns an attribute of that object. Indexes can be added blindly (duplicate indexes are ignored.). Indexes can be added at any time, before or after inserting data into the cache.

Index: Example with Ordering

- Suggest an index for trades based on their stock symbol.
- Queries that use this index are sorted (such as an ORDER BY) and use natural ordering (therefore the null).

```
NamedCache trades = . . .

trades.addIndex(
    new ReflectionExtractor("getSymbol"),
        true, /*sort */
        null);

Filter allA = new LikeFilter("getSymbol", "A%");

Set stockORCL = trades.entrySet(allA);
```



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

Index: Example with Ordering

The slide shows an example of adding an index. To add an index:

1. Obtain a handle on NamedCache.
2. Before you execute a query, you need to add an index.
3. Create a new ReflectionExtractor on the getSymbol() method. The addIndex() method indexes the result set of this method. This is similar to indexing the symbol column in relational databases. Pass a Boolean value of true to sort the result set.

This example does not include an optional comparator. If you do not pass a comparator, Coherence sorts based on what Java thinks is the natural order. For a numeric column, it sorts in the ascending order, that is, smallest to highest number. Depending on the language that you use with the string, it sorts based on the natural ordering of that particular string. So, it is alphabetical order in this case.

Now that you have created the index, you execute a query that includes the entrySet method. In this case, because you are using LikeFilter, the method will query all symbols that begin with the letter "A."

Index: Example with Ordering (Continued)

Observe that there are two things that happen that are different from the earlier slides:

- Coherence uses the index that you specified. The query execution here is much faster than query execution when there is no index defined.
- The result gets sorted because the index itself is sorted.

Index: Example without Ordering

- Suggest an index for trades based on their market.
- No specified order

```
trades.addIndex(  
    new ReflectionExtractor("getMarket") ,  
    false, /* do not sort */  
    null) ;
```



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

Index: Example without Ordering

When Coherence gives the result set back to the client, it is sorted by alphabetical order.

If you use the `addIndex()` method in your application, and the index already exists, it is essentially a hint to the optimizer. By calling `addindex`, you tell Coherence to use the index if it exists. If it does not exist, you instruct Coherence to create and use the index.

In this example, you perform the following:

- You add an index on the `getmarket()` method.
- Because you do not plan to sort the result set, you set the Boolean parameter to `false`.
- You set the optional comparator parameter to `null`.

Quiz

True or False? Indexes improve the performance of queries.

- a. True
- b. False



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

Answers: a

Quiz

True or False? Calling `addIndex()` on an attribute that already has an index throws `IndexExistsException`.

- a. True
- b. False



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

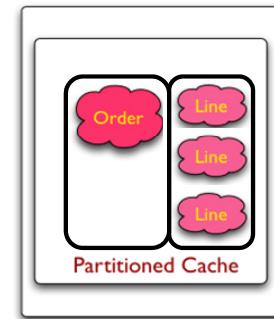
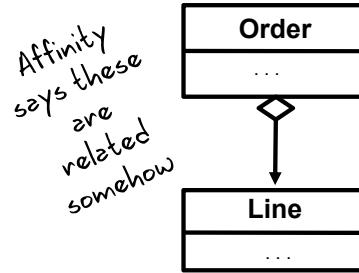
Answer: b

False, adding the same index twice is a no-op and has no side effects.

Data Affinity

- Data affinity describes the concept of relating one element or piece of data to another
- Coherence data affinity extends the concept to support grouping similar items together in partitions within a cache service
- Examples of data affinity include account (transactions, order), line items and similar relationships

Affinity says these are related somehow



ORACLE®

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

Data Affinity

Data has relationships. For example, orders have lines, bank accounts have customers, flights have passengers. All of these relationships are loosely grouped into the concept of data affinity. Coherence takes the concept one step further and describes how related objects can expose their relationships in such a way as to keep related objects together within the same cache service and on a specific node in the cluster.

Associated data may span multiple caches as long as they are managed by the same cache service. For example, order objects can expose their associated line items and the two sets kept colocated.

The benefits of data affinity are two-fold:

- Only a single cache node is required to manage queries and transactions against a set of related items.
- All concurrency operations can be managed locally, avoiding the need for clustered synchronization.

Specifying Data Affinity

- Affinity is defined in terms of the parent-child relationship based on the keys of the related elements.
- Coherence uses object identity as a way to relate objects via:
 - KeyAssociation: Child classes can implement the KeyAssociation interface and return their associated key
 - KeyAssociator: An external class which relates objects and is configured via cache configuration



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

Specifying Data Affinity

Data affinity is specified either by a child object itself or via an associate. It is a matter of design and convenience which method is implemented, however the KeyAssociation interface requires access to an objects sources, whereas the KeyAssociator interface does not.

Implementing KeyAssociations

- Key association is implemented against the key of an object rather than the object itself.

```
package com.tangosol.net.cache;
public interface KeyAssociation {
    public abstract Object getAssociatedKey();
}
```

- Object keys must be complex objects and implement the KeyAssociation interface.

```
public class Booking implements Entity<Booking.Id>, Comparable, Serializable {
    public static class Id implements Serializable, KeyAssociation {
```



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

Implementing KeyAssociations

Child objects must support a more robust key structure in order to support KeyAssociations. In the example provided, the `Booking` object has a defined key of `Booking.Id`.

The key can no longer be primitives and their wrappers, but must be an object with data members representing both the booking ID and its parent key. If there were many bookings for an existing customer, then each would have a different booking key, but the same customer ID.

Booking.Id Example

```

public class Booking implements Entity<Booking.Id>,
    Comparable, Serializable {
    public static class Id implements Serializable, KeyAssociation {
        public Id() { }
        public Id(long customerId, long bookingId) {
            this.customerId = customerId;
            this.bookingId = bookingId;
        }
        public Object getAssociatedKey() { return customerId; }
        public Long getCustomerId() { return customerId; }
        public Long getBookingId() { return bookingId; }
        public int hashCode() { . . . }
        public boolean equals(Object obj) { . . . }
        private Long customerId;
        private Long bookingId;
    }
}

```

Booking class would have
a getId method which resembles:
public Id getId(){
 return id;
}

1

2

3

ORACLE

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

Booking.Id Example

Based on the numbers in the slide:

1. In your fictional airline domain, you have customers that book one or more flights. In this scenario, there is a 1:M relationship between customers and bookings. You model this by creating a `Booking.Id` class within the `Booking` object. Note that the use of an inner class is a convenience and not a requirement. You could have created a class outside the `Booking` object and achieved the same results.
2. The booking ID class then implements the `KeyAssociation` interface.
3. The `get` returns, via `getAssociatedKey` the customer ID, which is the key to the parent class.

Implementing a KeyAssociator

An example key associator might be:

```
import com.tangosol.net.PartitionedService;
import com.tangosol.net.partition.KeyAssociator;
public class BookingAssociator implements KeyAssociator {

    private transient PartitionedService service;
    public void init(PartitionedService service) {
        this.service = service;
    }
    public Object getAssociatedKey(Object key) {
        if (key instanceof Booking.Id)
            return ((Booking.Id)key).getCustomerId();
        else
            return key;
    }
}
```



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

Implementing a KeyAssociator

KeyAssociator Implementations define the association in a separate key associator class and are registered declaratively.

Key Associators are implemented by creating a class that implements the KeyAssociator interface. This interface has two required methods:

- **init:** Called when the key associator is created, and is passed the PartitionedService instance, which houses the key associator. In the example, the service is stored but otherwise unused.
- **getAssociatedKey:** Returns the associated key for a given key object. In the example shown, the key's type is examined and then the associated is key returned appropriately. Note that a key associator might return any number of associations, and might include a chain of `instance of` statements.
If no association is supported by the object type, the `getAssociatedKey` method should return the original object.

Note that the PartitionedService has a number of useful methods for returning information about the partition, returning the originating key, and others.

Registering KeyAssociators

Example:

```
<cache-config>
. . .
<distributed-scheme>
    <scheme-name>. . . </scheme-name>
    <service-name>. . .</scheme-name>
. . .
<key-associator>
    <class-name>. . . KeyAssociatorclass</class-name>
</key-associator>
</distributed-scheme>
. . .
</cache-config>
```



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

Registering KeyAssociators

KeyAssociator classes are registered declaratively as a subelement of the `<distributed-scheme>` as shown in the slide.

Quiz

The KeyAssociation interface is used to support data affinity.
What does the getAssociatedKey method return?

- a. The parent key associated with the child object
- b. The child key associated with the parent object
- c. Either the parent or child key, depending on calling parameters



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

Answer: a

The getAssociatedKey method returns the parent key associated with the child object.

Quiz

True or False? When extending objects to support data affinity, both the parent and child objects need to be extended.

- a. True
- b. False



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

Answer: b

False, only the child object needs changes when implementing with KeyAssociation interface.

Agenda

Coherence Queries using Filters

Coherence Query Language

- What is the Coherence Query Language?
- CohQL Interfaces and statements
- CohQL and the command line



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

What is CohQL?

Coherence Query Language, or CohQL:

- Is a light-weight syntax, in the tradition of SQL
- Is used to perform operations on a Coherence cache
- Can be used either programmatically or from the command-line
- Includes:
 - QueryHelper API: supporting programmatic construction of filters using an SQL-like syntax
 - QueryPlus: a command line tool for cache access

```
SELECT * FROM "orders" WHERE 40 > price
```



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

What is CohQL?

Introduced in Coherence 3.6, Coherence Query Language, or CohQL, defines statements to manage data against a cache. CohQL enables you to write clear, concise query statements, which are translated to Filter instances and executed against a cache.

CohQL defines a data model based on the cache objects and their embedded objects, which can be traversed using dot notation (.). CohQL can traverse the schema given to the entities and their fields, both of which are mapped to an underlying cache.

CohQL defines operators and expressions based on the data model using a SQL-like syntax.

CohQL Statements

A CohQL statement can be:

- A SELECT statement, which might include various clauses such as FROM, WHERE, and GROUP BY
- An INSERT, UPDATE, DELETE, CREATE, BACKUP, or RESTORE statement

No order by, but can use comparators!

BNF Grammar (Backus-Naur Form):

```
select_clause ::= SELECT select_expression {, select_expression}*
update_statement ::= update_clause [where_clause]
delete_statement ::= delete_clause [where_clause]
create_clause ::= [CREATE | ENSURE] CACHE "cache_name"
```

Examples:

```
SELECT * FROM Publisher WHERE name = "Macmillan"
```

```
UPDATE "employees" SET salary = 1000, vacation = 200 WHERE grade > 7
```

```
DELETE FROM "employee" WHERE bar.writer IS NOT "David"
```

```
CREATE CACHE "employee"
```



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

CohQL statements

CohQL query language supports the same four types of statements most SQL dialects support: INSERT, SELECT, DELETE, and UPDATE. Additionally, CohQL supports the ability to create caches directly in CohQL as well as backup, restore and otherwise manage caches.

As the slide shows, basic CohQL statements are very similar to their SQL counterparts in both form and function.

A noted difference is the absence of a JOIN clause. CohQL is *not* SQL and does not support joining two caches together. Additionally CohQL does not support ORDER BY.

Working with Queries

The `QueryHelper` factory class provides five methods for creating queries and one for creating a value extractor

```
package com.tangosol.util;
public class QueryHelper {

    public static Filter createFilter(String s);
        Return a Filter from a CohQL statement. For example, state='CA'

    public static Filter createFilter(String s, Object aBindings[]);
        Return a Filter from a CohQL statement using the provided replacement binding
        variables, each binding represented by a numbered parameter.

    public static Filter createFilter(String s, Map mapBindings);
        Return a Filter from a CohQL statement using the provided map binding variables,
        each representing a name=value pair.

    public static Filter createFilter(String s, Object aBindings[],
                                    Map mapBindings);
        Return a Filter from a CohQL statement using the provided array and map of bindings.
        The array represents numbered bindings and the map named bindings.

    public static ValueExtractor createExtractor(String s);
}

}
```



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

Working with Queries

The Query helper factory class provides a set of five methods, four of which are dedicated to creating Filter representations of the provided CohQL statement.

Of particular note are the binding variables. Coherence supports both numeric and named parameter bindings in CohQL statements. Variables of type `Object []` are arrays of bindings using zero based variable replacements of the form `?#`. Variables of type `Map` are named variable replacements of the form `:name`. Each of which will be discussed in more detail in later slides. An important point, `?#` variables are 1-based, but `Object []` is zero-based, as such element 0 maps to `?1`.

Note that each of these methods can throw a `FilterBuildingException`.

General SELECT Syntax

- Select statements are composed of:

```
SELECT (properties* aggregators* | * | alias)
FROM "cache-name" [AS] alias
[WHERE conditional-expression] [GROUP [BY] properties+]
```

- The only required portion of a select is SELECT...FROM...

```
SELECT order FROM "orders"
```

When using QueryHelper

'*SELECT * FROM "cachename" WHERE*'
is implied



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

General SELECT Syntax

Select statements follow a simple core syntax. At a minimum, there is `SELECT` followed by a `FROM <entity> AS <alias>`. Only the select and from portions are required.

Additionally, a `WHERE` clause can be used to refine results, and `GROUP BY` can be used to group sets of returned records, often to perform operations on the set, such averages and so on.

The asterisk (*) character represents the full object instead of subparts. It is not required to prefix a path with the cache-name. The `FROM` part of the `SELECT` statement targets the cache that forms the domain over which the query should draw its results. The `cachename` is the name of an existing cache.

Path Expressions

Relations between objects can be traversed using a Java-like syntax known as a *path expression*.

Path expressions:

- Reference entities using alias or identification variables
- Reference properties using dot (.) followed by property name
- Can reference properties in an embedded field directly
- Are backed by `ReflectionExtractors` that are prepended with `get` and upper casing the first letter of the identifier

```
SELECT order
  FROM Orders AS order
 WHERE order.shipped = FALSE and order.address.city="Boston"
```



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

Path Expressions

Path expressions are a simple dot-separated (.) notation for accessing properties of entities. In its simplest form, a property is referenced by its owner alias, a dot (.), and the property name or accessor name without the “get”. For example, in the slide, the `Order` entity has a property `shipped`. The property can be used in a `WHERE` clause via its alias and name. For example, in Java, you might have:

```
Order order = ...
if ( order.getShipped() == FALSE) { // do something };
```

which in a `WHERE` clause translates into:

```
WHERE order.shipped = FALSE
```

Additionally, path expressions can nest. For example, assuming `Orders` have `getAddress()` and `Addresses` have `getPostalCode()`, you might write:

```
WHERE order.address.postalCode = '01805'
```

Note: The path expressions that cannot translate are style Java Bean methods. However, you can replace a access name with a method such as `isSupported()`, which will be used directly and without conversion.

Filtering with WHERE

A WHERE clause typically has some number of conditions, which can subset a SELECT.

- Value comparison using operators

`<, <=, =, >, >=, ><`

Boolean values and entity expression use only = and <>.

- Like comparison

`LIKE "string"`

Supports string wildcards _ and %

- Null comparison

`IS [NOT] NULL`

Is the variable [not] null?

- Contains

`path-expression CONTAINS "literal"`

Does the expression contain the literal?

- In comparison

`path-expression IN (<list of values>)`

Is value in set?



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

Filtering with WHERE

Operator precedence within the WHERE clause is as follows:

1. Path operator (.)
2. Unary + and -
3. Multiplication (*) and division (/)
4. Addition (+) and subtraction (-)
5. Comparison operators: =, >, >=, <, <=, <>, [NOT] BETWEEN, [NOT] LIKE, [NOT] IN, IS [NOT] NULL, CONTAINS [ALL | ANY]
6. Logical operators (AND, OR, NOT)

The WHERE clause supports only arithmetic at the language level.

The BETWEEN operator can be used in conditional expressions to determine whether the result of an expression falls within an inclusive range of values. Numeric, or string expressions can be evaluated in this way. The form is: BETWEEN *lower* AND *upper*.

Note that CohQL does not support the ORDER BY clause of traditional SQL.

Filtering with WHERE (continued)

The `LIKE` operator can use the `_` (single character) and `%` (zero, one, or more characters) wild-cards.

The `IN` operator can be used to check whether a single-valued path-expression is a member of a collection. The collection is defined as an inline-list or expressed as a bind variable. The syntax of an inline-list is:

```
" (" literal* ") "
```

`CONTAINS [ALL | ANY]` are very useful operators because Coherence data models typically use denormalized data. The `CONTAINS` operator can be used to determine if a many-valued path-expression contains a given value. For example:

```
e.citys CONTAINS "Boston"
```

The `ALL` and `ANY` forms of `CONTAINS` take a inline-list or bind-variable with the same syntax as the `IN` operator.

QueryFilter Example

- Select all employees who live in Massachusetts, but work in another state;

```
String where = "homeAddress.state = 'MA' and " +
               "workAddress.state != 'MA'";
Filter filter = QueryHelper.createFilter(where);
```

- Which is equivalent to:

```
Filter filter =
AndFilter(
    EqualsFilter(
        new ChainedExtractor("getHomeAddress.getState", "MA"),
    NotEqualsFilter(
        new ChainedExtractor("getWorkAddress.getState"), " MA" ));
```

WARNING: Use queries sparingly! They do not scale well, and can cause considerable network traffic!



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

QueryFilter Example

Caution: Queries can cause considerable network traffic across the entire cluster. It is suggested that developers first use a partition filter to reduce load, and then perform a query.

key() and value() Pseudo functions

- Developers may store data within the key of an object rather than duplicate the data inside the object itself.
- The `key()` pseudofunction can be used to reference an object's key and extract data from it. For example:

```
SELECT key().firstName, key().lastName  
  FROM "ContactInfoCache"  
 WHERE homeAddress.state IS 'MA' AND  
       workAddress.state != "MA"
```

- The `value()` pseudofunction represents the complete contents of a object and can be used as:

```
SELECT key(), value() FROM employees
```



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

key() and value() Pseudofunctions

The `key()` and `value()` pseudofunctions reflect the fact that CohQL is not SQL and that Coherence is not a Database. These functions exist because you are dealing with caches which reflect the Map interface. Because you are dealing with caches, you need a way to handle the abstractions of keys and their stored values in a concrete way. The `value()` pseudofunction is in fact an `IdentityExtractor`, and the `key()` function is actually a wrapper around `KeyExtractor`.

Aggregating in SELECT

- Supports aggregation via a variant of the `SELECT` syntax
- CohQL supports aggregation functions:
 - COUNT
 - AVG
 - MIN
 - MAX
 - SUM
- Example:

```
SELECT supplier,SUM(amount),AVG(price)
      FROM "orders"
        GROUP BY supplier
```



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

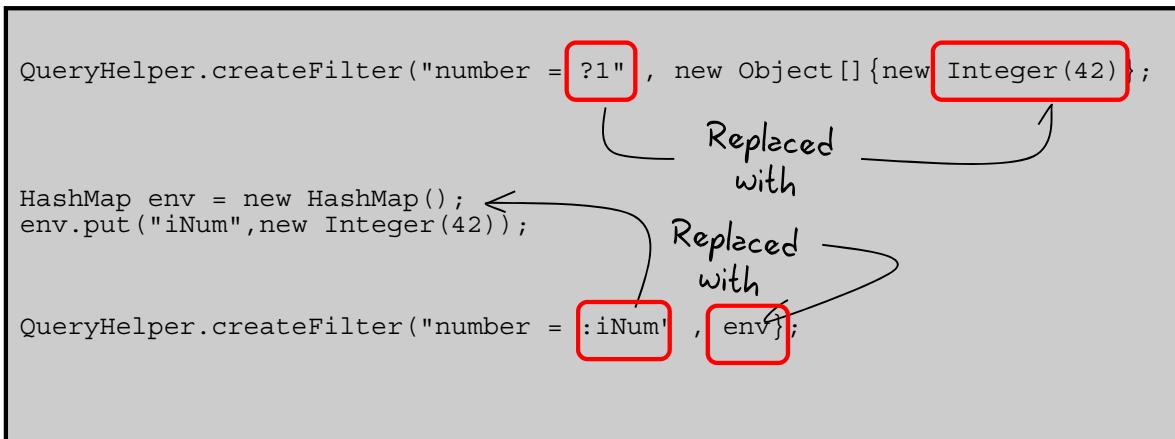
Aggregating in SELECT

An aggregate query is a variation on the `SELECT` query. Use an aggregate query when you want to group results and apply aggregate functions to obtain summary information about the results. A query is considered an aggregate query if it uses an aggregate function or possesses a `GROUP BY` clause. The most typical form of an aggregate query involves the use of one or more grouping expressions followed by aggregate functions in the `SELECT` clause, paired with the same lead grouping expressions in a `GROUP BY` clause.

Parameters

Parameters are identified in queries:

- By name: ':' + *name*, for example, ':po_Id'
- By position: '?#“, for example, '?1'



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

Using Parameters

Parameters are defined in one of two ways:

- **By position:** Positional parameters are specified using parameter names using the ?1 syntax. Each parameter is prefixed with '?' and an increasing integer value, and then uses the parameter in the associated CohQL query. Remember, positional variables are 1-based, but the Object [] is zero-based!
- **By name:** Parameters can be specified by name as well by using the colon (:) followed by a case-sensitive name. The Map variants of createFilter are then used to specify a value at run time.

Quiz

(Select all that apply) Path expressions:

- a. Use a dot (.) notation to reference objects using alias
- b. Reference properties using dot (.) followed by property name *only*
- c. Reference properties using dot (.) followed by accessor name *only*
- d. Reference properties using dot (.) followed by accessor name or property name
- e. Cannot reference properties in an embedded field directly



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

Answers: a,c

Managing Cache Lifecycle

Caches can be created, deleted, backed up, and restored via CohQL.

- To create a cache

```
CREATE CACHE "cache-name"
```

- To use an existing cache in a CohQL script

```
ENSURE CACHE "cache-name"
```

- To backup a cache

```
BACKUP CACHE "cache-name" [TO] [FILE] "file-name"
```

- To restore a cache

```
RESTORE CACHE "cache-name" [ FROM ] [ FILE ] "filename"
```

- To delete a cache

```
DROP CACHE "cache-name"
```



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

UPDATE Syntax

- UPDATE statements are composed of:

```
UPDATE "cache-name" [[AS] alias]
SET update-statement {, update-statement}*  
[ WHERE conditional-expression ]
```

- The only required portion of a select is UPDATE ...FROM...

```
UPDATE "employees"
      SET salary = 1000, vacation = 200
          WHERE grade > 7
```



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

UPDATE Syntax

Each update statement consists of a path expression, assignment operator (=), and an expression. The expression choices for the assignment statement are restricted. The right side of the assignment must resolve to a literal, a bind-variable, a static method, or a new Java-constructor with only literals or bind-variables. The UPDATE statement also supports the use of aliases.

INSERT Syntax

- CohQL fully supports INSERT into a cache, based on a key:
- Insert keys must be either:
 - Provided via the INSERT statement
 - generated object must support getKey ()
- Insert statements are composed of:

```
INSERT INTO "cache-name"
[ KEY (literal | new java-constructor | static method) ]
VALUE (literal | new java-constructor | static method)
```

- Example:

```
insert into "employee" key "writer" value "David"
```



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

DELETE Syntax

- CohQL supports cache object removal via DELETE
- Delete syntax:

```
DELETE FROM "cache-name" [[AS] alias]  
[WHERE conditional-expression]
```

- Delete fully supports SELECT ... WHERE clause syntax and aliases



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

Index Management

CohQL supports the creation and removal of indexes.

- To create or ensure the use of an index:

```
[ CREATE | ENSURE ] INDEX  
[ON] "cache-name" (value-extractor-list)
```

- To drop an existing index:

```
DROP INDEX [ON] "cache-name" (value-extractor-list)
```

- The value extractor list:
 - Is a comma-separated list
 - Uses path expressions to create ValueExtractors.
 - Creates a MultiExtractor, if a list of path expressions is specified



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

Starting QueryPlus

To start an instance of Oracle Coherence CohQL tool:

1. Open a command prompt to %COHERENCE_HOME%/bin.
2. Ensure that `java_home` is set appropriately.
3. Run `query. [cmd|sh]`.

```
C:\oracle\coherence\bin>query.cmd
** Starting storage disabled console **
java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Server VM (build 16.3-b01, mixed mode)

Coherence Command Line Tool
jline library cannot be loaded, so you cannot use the arrow
keys for line editing and history.

CohQL>
```

Warning: QueryPlus is a development tool and
not intended for production use!



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

Starting QueryPlus

The CohQL command-line tool provides a nonprogrammatic way to interact with caches by allowing statements to be issued from the command line. The tool can be run using the `com.tangosol.coherence.dslquery.QueryPlus` class or, for convenience, a startup script is available to run the tool and is located in the `COHERENCE_HOME/bin/` directory. The script is available for both Windows (`query.cmd`) and Unix (`query.sh`).

QueryPlus Arguments

- QueryPlus supports the following arguments:

Argument	Description
-t	Enable trace mode debugging output
-c	Exit after executing the command line arguments
-s	Run in silent mode, often used with I/O redirection
-e	Run the command-line tool in extended language mode. This mode allows object literals in update and insert commands.
-l <i>statement</i>	Execute the given statement. Statements must be enclosed in single or double quotes. Any number of -l arguments can be used together
-f <i>filename</i>	Process the statements in the given file. Statements must be separated by semi-colons (;). Any number of -f arguments can be used.



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

Summary

In this lesson, you should have learned how to:

- Develop Filters to subset cache data, including sorting and aggregation of data objects
- Construct Filters using ValueExtractors
- Develop custom aggregators, which can run in parallel in a Coherence cluster
- Extend objects to support data affinity
- Implement Java clients that can perform dynamic queries using the Coherence Query Language (CohQL)
- Improve filter and query performance using Indexes



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