
ORACLE®

ORACLE®

Observing Data Changes

Lesson 4

Objectives

After completing this lesson, you should be able to:

- Describe the ObservableMap interface
- Understand how to listen for
  - All events
  - Events that satisfy a filter
  - Events on a particular object key
- Understand the components required for a simple chat program

ObservableMap

- Provides ability to “observe” changes in Cache Entries
  - com.tangosol.util.ObservableMap
- Standard Bean Event Model (Listener Pattern)
  - extends java.util.EventListener
- All NamedCaches implement ObservableMap
- History
  - Originally designed to provide pluggable invalidation and cache pruning (internal use)
  - Now used for reacting to Entry changes

Features : ObservableMap Interface

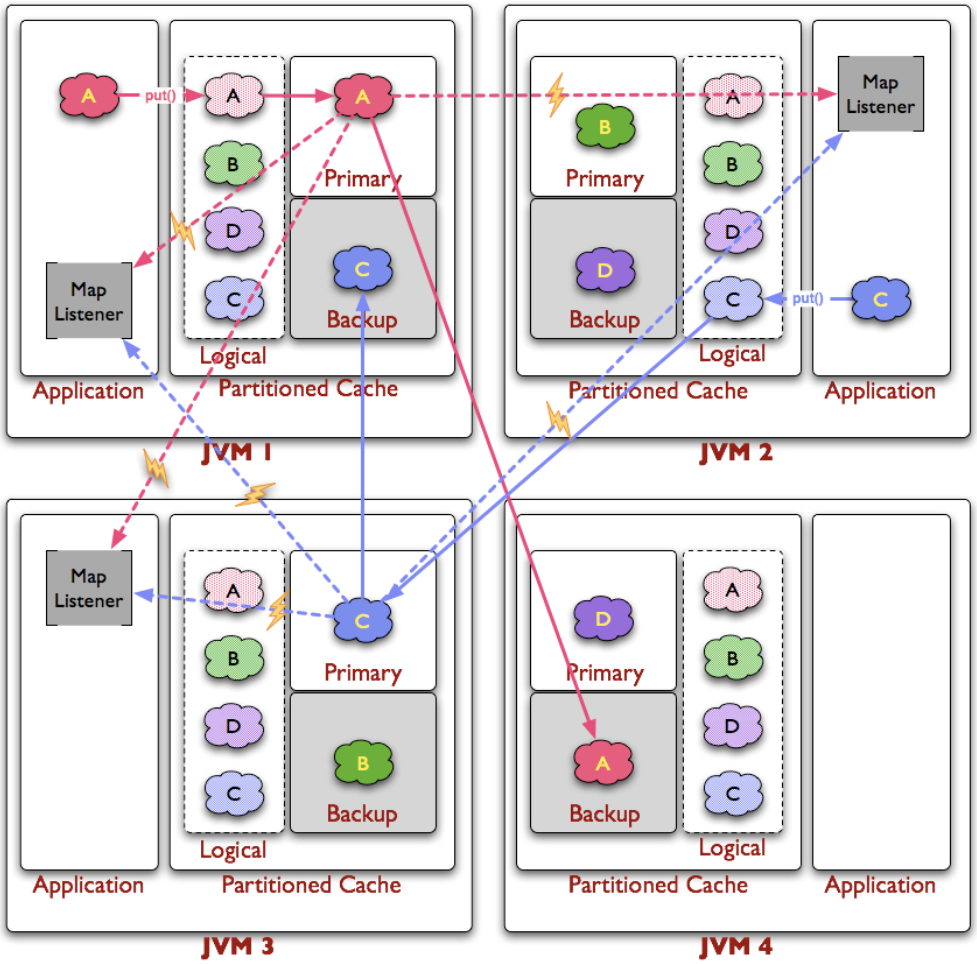
- Real-time filterable (bean) events for entry insert, update, delete
- Filters applied in parallel (in the Grid)
- Filters completely extensible
- A large range of filters out-of-the-box:

```
All, Always, And, Any, Array, Between, Class,
Comparison, ContainsAll, ContainsAny,
Contains, Equals, GreaterEquals, Greater, In,
InKeySet, IsNotNull, IsNull, LessEquals,
Less, Like, Limit, Never, NotEquals, Not, Or,
Present, Xor…
```

- Events may be synchronous*

```
trades.addMapListener(
  new StockEventFilter("ORCL"),
  new MyMapListener(…));
```

Features : Observable Interface



ObservableMap Methods

- void addMapListener(MapListener listener)
- void addMapListener(MapListener listener, Filter filter, boolean fLite)
- void addMapListener(MapListener listener,Object oKey, boolean fLite)
- void removeMapListener(MapListener listener)
- void removeMapListener(MapListener listener, Filter filter)
- void removeMapListener(MapListener listener, Object oKey)

MapListener Interface

- Register MapListeners for...
  - All cache events, those satisfying a Filter or a specific key
  - Lite == network optimization. (reduce event payload)
- MapListener Interface implementations...
  - Handlers for Insert, Update and Deleted Events
- MapEvent Class captures event information
  - Id (event type), Entry Key, Old Value, New Value
  - Lite means old and new values may not be present in event

Example

- Regsiter a new MapListener on a Named Cache

```
namedCache.addMapListener(new MapListener() {
 public void entryDeleted(MapEvent mapEvent) {
   //TODO... handle deletion event
 }
 public void entryInserted(MapEvent mapEvent) {
   //TODO... handle inserted event
 }
 public void entryUpdated(MapEvent mapEvent) {
   //TODO... handle updated event
 }
});
```

MapListener Implementations

- Several MapListener helper implementations
- AbstractMapListener Class
  - Empty implementations for each MapListener signature
  - Simplify your implementations by overriding default empty implementations
- MultiplexingMapListener Class
  - Introduces abstract onMapEvent method
  - All MapListener methods set to onMapEvent
  - Simplify your implementations by overriding onMapEvent

Examples

- Allows you to just override the method required

```
namedCache.addMapListener(new AbstractMapListener() {
  //other MapListener methods implemented in super-class
  public void entryUpdated(MapEvent mapEvent) {
    //TODO... handle just the updated event
   }
});
```

- Handle all events

```
namedCache.addMapListener(new MultiplexingMapListener() {
  public void onMapEvent(MapEvent mapEvent) {
    //TODO... handle all event (use Id to determine type)
   }
});
```

Useful MapEvent methods

- getNewValue()
  - Return a new value associated with this event.
- getOldValue()
  - Return an old value associated with this event.
- getKey()
  - Return a key associated with this event.

Full Example

```
// listen for insert events on Person
// This can be done in an easier way by using a new AbstractMapListener()
// and then overriding only the method you want to
//
person.addMapListener(new MapListener()
{
  public void entryDeleted(MapEvent mapEvent) {
    // ignore
  }
  public void entryInserted(MapEvent mapEvent) {
     Person p = (Person)mapEvent.getNewValue();
     System.out.println("New person added: " + p.getFirstname() +
                        " " + p.getSurname());
  }
  public void entryUpdated(MapEvent mapEvent) {
     // ignore
  }
}
);
```

Simple Chat Program

- What would you need to do for this?

- Create a Message object to store the chat messages
- Create a ChatClient class that will do the following:
  - Get a persons name
  - Setup a new MapListener to receive messages that are posted in the chat. Ensure that you don't get messages posted by yourself.
  - Loop and read a message from the command line and post this to the 'messages' named cache. (Exit when the user types exit)
- Run multiple copies of this and observe the behaviour
- Some extra features to add if you have time:
- Add a facility to list all users in the chat. (You will need a second named cache)
- Send a private message to a named individual.
- Many more…

Summary

In this lesson, you should have learned how to:

- Describe the ObservableMap inteterface
- Understand how to listen for
  - All events
  - Events that satisfy a filter
  - Events on a particular object key
- Understand the components required for a simple chat program

Lab 6

- Lab 6
  - Observing data changes. Use the person class and identify certain events.
- Bonus
  - Write a clusterable, resilient, yet simple chat program.


ORACLE®