
ORACLE®

ORACLE

Oracle Coherence REST

Dave Felcey
Coherence Product Manager

Agenda

- Overview
- Deployment
- RESTful API
- Configuration
- Data Representation

Overview
REST

- REST stands for “REpresentational State Transfer”
- The REST architectural style was developed in parallel with HTTP/1.1, based on the existing design of HTTP/1.0
- Based on concept of clients and servers
- Requests and responses are built around the transfer of representations of resources
- While one or more requests are outstanding, the client is considered to be in transition – between states
- Scales very well, loosely couples components and interfaces can be general purpose

Overview

Coherence REST

- Provides easy access to Coherence caches and cache entries over HTTP protocol
- Similar to Coherence*Extend, providing data access from outside Coherence cluster
- No specific serialization required - like POF
- Using HTTP data can be marshaled in multiple representation formats, such as JSON and XML
- Allows access to Coherence from languages like Ruby and Python
- Has a dependency on Oracle and third-party libraries: JAXB, JAX-RS (Jersey), Grizzly and Jackson (Apache)

Deployment

Coherence REST

- Coherence RESTful Web Services can be deployed either using embedded HTTP server or using any of standard Servlet containers
- When embedded HTTP server is used communication via proxy server
  - `http-acceptor` element (inside `proxy-scheme`) configured
  - `http-acceptor` specifies connection details etc.
- When external Servlet container is used its like deploying any other web application
  - Package web.xml descriptor, dependency jars and mandatory REST configuration into single WAR file (in `WEB-INF`, `WEB-INF\lib` and `WEB-INF\classes` folders respectively). Specify Jersey application as Servlet

RESTful API

Coherence REST

- **Single object operations** - GET/PUT/DELETE /{cache-name}/{key}
  - Returns/creates/deletes a single object from the cache based on a key.
  - For GET supports partial results (see section on partial object operations details below), for PUT returns 201 (Created) if the object was created, 200 (OK) if it was updated.
  - For GET/DELETE returns 404 (Not Found) if the object with the specified key does not exist and for PUT returns 409 (Conflict) and current object as an entity if optimistic concurrency check fails (see Concurrency Control section below for details).

RESTful API *(continued)*

Coherence REST

- **Multi-object operations** allow users to retrieve, update or delete multiple objects in a single network request, which can significantly reduce the “chattiness” and improve network performance - GET/DELETE /{cacheName}/({key1, key2, …})
  - Returns/deletes a set of objects from the cache based on the specified keys. The ordering of returned objects is undefined.
  - Supports partial results (see section on partial object operations below for details).
  - Always returns 200 (OK), even if there are no objects found or in the result set. For GET an empty result set would be returned if none found. Missing return objects are silently omitted
  - PUT has been implemented intentionally. It would require URL/body ordering and could easily produce tainted data

RESTful API *(continued)*

Coherence REST

- **Partial Object Properties** can also be retrieved and returned as a matrix. In the URL `GET /people/123;p=id,name,address:(country)` the id, name and country field of the Person object with the key 123 has been requested. The Person object is shown below:

```
public class Address {
    public String street;
    public String city;
    public String country;
}


public class Person {
    public Long id;
    public String name;
    public Address address;
}
```

RESTful API *(continued)*

Coherence REST

- **Queries** can be performed by passing Coherence QL expression as a query parameter q, results can be sorted using asort matrix parameter, or limited by specifying start and count matrix parameters - GET /{cacheName};sort={sortOrder};start={start};count={count}?q={query}
  - Query must be a URL-encoded Coherence QL expression
  - Sort order is a comma-separated list of properties to sort on, each can have optional :asc (default) or :desc qualifier, e.g. `sort=name,age:desc`
  - Start and count are optional integer arguments

RESTful API *(continued)*

Coherence REST

- **Aggregations** can be performed on data in a cache using both built-in and custom aggregators
  - `GET /{cacheName}/{aggregator(args, …)}` aggregates all entries in the cache.
  - `GET /{cacheName}/{aggregator(args, …)}?q={query}` aggregates query results.
  - `GET /{cacheName}/({key1, key2, …})/{aggregator(args, …)}` aggregates specified entries.
  - Returns 200 (OK) with aggregation result as entity if aggregation is performed successfully
  - Returns 400/500 if aggregation fails
  - Where URL doesn't contain any parameters or constructor accepting single `ValueExtractor` then default constructor used, e.g. `GET /people/count())`

RESTful API *(continued)*

Coherence REST

- **Processing** can be invoked against built-in and custom entry processors, on one or more objects in the cache.
  - `POST /{cacheName}/{processor(args, …)}` processes all entries in the cache.
  - `POST /{cacheName}/{ processor (args, …)}?q={query}` processes query results.
  - `POST /{cacheName}/({key1, key2, …})/{ processor (args, …)}` processes specified entries.
  - Returns 200 (OK) with processing result as entity if processing is performed successfully. Returns 400/500 if processing fails
  - Coherence REST doesn't assume anything about processor creation. For each entry processor implementation there needs to be ProcessorFactory implementation that that takes string inputs from URL call to instantiate processor instance

RESTful API *(continued)*

Coherence REST

- Coherence REST provides two such factories for `NumberIncrementor` and `NumberMultiplier`
- For example to increment each person's age for increment value 5, one could: `GET /people/increment(age, 5)`

RESTful API *(continued)*

Coherence REST

- **Concurrency Control** is implemented in Coherence REST using optimistic concurrency only, as it maps cleanly to the HTTP protocol.
  - Cached objects must implement the `com.tangosol.util.Versionable` interface to use concurrency controls - `Comparable getVersionIndicator()` and `void incrementVersion()`
  - A version property is returned as an *etag* header when object is requested
  - When the user submits a PUT request to update the object, the *etag* header is sent as well
  - This is used to determine if the object within the cluster has been changed in the meantime.
  - If change detected then update performed and 409 (Conflict) status with the existing entity - so the can reapply and retry

Configuration

Coherence REST

- REST Configuration is made out of few segments: Resource, Aggregator and Processor configurations
  - Resource configuration is the only mandatory segment of Coherence REST configuration, e.g.

```
<resource>
 <cache-name>test-cache</cache-name>
 <key-class>java.lang.Integer</key-class>
 <value-class>com.foo.Person</value-class>
</resource>
```

  - Aggregator configuration is where custom aggregator factories and names are specified
  - Processor configuration is where custom entry procesor factories and names are specified

Data Representation

Coherence REST

- Input and output **Data** in Coherence REST can be either in an XML or JSON format
  - XML. In order to use XML representation, objects stored in the cache will have to have the appropriate JAXB bindings defined. These are created using the `xjc` tool.
  - JSON. In order to use JSON representation, objects stored in the cache will either have to have appropriate *Jackson bindings* defined or JAXB bindings defined. Using Jackson annotations gives user more power on controlling the output JSON format, but in case when both XML and JSON formats are needed, JAXB annotations can be enough for both formats.