ORACLE®

Coherence and Java 8

Made for Each Other

Aleksandar Seovic
Architect
Oracle Coherence
November 07, 2014

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Note: The speaker notes for this slide include instructions for when to use Safe Harbor Statement slides.

Tip! Remember to remove this text box.

Agenda

1. Coherence API Generification
2. Lambda Support
3. New Default Methods on java.util.Map
4. Stream API Support

Coherence API Generification

- NamedCache is now NamedCache<K, V>
  - Fully generified core NamedCache API
  - EntryProcessor<K, V, R>
  - EntryAggregator<K, V, R>
- PofReader, PofWriter and PofSerializer<T> are generic as well
- Many other public APIs have been generified as well
- **The end result:** cleaner code, less casting, better type safety
- Kudos to Mark and Brian for getting it done

Lambda Support

- Lambdas are a major new feature in Java 8
  - Allow code to be passed in as arguments and returned as a result
  - Can be used instead of anonymous inner classes for many functional interfaces
    - Runnable, Callable, Comparable, etc.
- Many new functional interfaces were added to JDK
  - Function, Predicate, Supplier, Consumer, etc.
- Many legacy Coherence interfaces are de facto functional interfaces
  - ValueExtractor, Filter, EntryProcessor, Converter, etc.

Lambdas in Coherence 12.2.1

- Method references as ValueExtractors
  - `Person::getName` instead of `new ReflectionExtractor(“getName”)`
  - Type-safe, fully refactorable
- You *can* use lambdas as filters

```
cache.entrySet((p) -> p.getName().equals(“Ana”) || p.getAge() >= 18)
```

  - But you *shouldn’t* do that with distributed caches!
- Use the new Filter DSL instead, in order to leverage indexes

```
equal(Person::getName, "Ana")
      .or(greaterEqual(Person::getAge, 18)
```

ORACLE

Lambdas in Coherence 12.2.1

- You can (and should) use lambdas as entry processors

```
private static EntryProcessor<String, Position, Position> Split(int nFactor)
        {
        return (entry) ->
            {
            Position p = entry.getValue();
            p.setQuantity(p.getQuantity() * nFactor);
            p.setPrice(p.getPrice() / nFactor);
            entry.setValue(p);
            return p;
            }
        }
```

Lambdas in Coherence 12.2.1

- Because calling it just feels right

```
positions.invokeAll(
    equal(Position::getSymbol, “APPL”),
    Split(7));
```

- And because you don’t have to
  - Implement serialization code
  - Register EP class in POF config
  - It just works!

Lambda Limitations

- JDK lambdas are somewhat “static”
  - Only the metadata describing lambda is sent across the wire
  - Require the same compiled code in both the client and server class path
- Which makes them somewhat limited in a distributed environment
  - Any changes or introduction of new lambdas on the client require redeployment and restart of both the client and the server
  - Very cumbersome and time consuming for large clusters

Dynamic Lambdas

- Ship not only the metadata, but the actual byte code to execute to the server as well
  - Client-side byte code is parsed and a new lambda class generated from it
  - Server defines a lambda class based on the byte code received from the client and executes it
  - Allows modification of the existing or the introduction of new behavior without the need for redeployment or server restart

Dynamic Lambdas: Mechanics

1. Enable dynamic lambdas within the cluster

```
<class-name>com.tangosol.io.pof.ConfigurablePofContext</class-name>
<init-params>
  <init-param>
    <param-type>string</param-type>
    <param-value>pof-config.xml</param-value>
  </init-param>
  <init-param>
    <param-type>string</param-type>
    <param-value>dynamic</param-value>
  </init-param>
</init-params>
```

Dynamic Lambdas: Mechanics

2. Apply @Dynamic annotation to a functional interface

```
@Dynamic
@FunctionalInterface
public interface DynamicEntryProcessor<K, V, R>
        extends EntryProcessor<K, V, R>
    {
    }
```

Dynamic Lambdas: Mechanics

3. Capture dynamic lambda using a static method

```
private static DynamicEntryProcessor<String, Position, Position> Split(int nFactor)
        {
        return (entry) ->
            {
            Position p = entry.getValue();
            p.setQuantity(p.getQuantity() * nFactor);
            p.setPrice(p.getPrice() / nFactor);
            entry.setValue(p);
            return p;
            }
        }
```

Dynamic Lambdas: Mechanics

4. Invoke as usual

```
positions.invokeAll(
    equal(Position::getSymbol, “APPL”),
    Split(7));
```

New Default Methods on java.util.Map

- Java 8 adds a number of "default" methods to Map interface
  - getOrDefault, computeIfAbsent, merge, replaceAll, etc.
- In theory, default methods shouldn't break anything
  - But in reality they do, in a major way
    - JDK implementation assumes data locality
    - JDK implementation is not thread-safe, let alone cluster-safe
  - So we had to re-implement all of them…
    - Using lambda-based entry processors, of course ;-)

Default Methods: Example

```
positions.replaceAll(equal(Position::getSymbol, “APPL”), Split(7));

private static BiFunction<String, Position, Position> Split(int nFactor)
        {
        return (k, v) ->
            {
            v.setQuantity(p.getQuantity() * nFactor);
            v.setPrice(p.getPrice() / nFactor);
            return v;
            }
        }
```

Stream API Support

- Java 8 introduces Stream API as a way to
  - Aggregate read-only streams of data
  - Replace external with internal iteration
  - Parallelize aggregation
- Similar in purpose to Coherence Aggregator API, but
  - Completely different API
  - JDK implementation is very inefficient in a distributed environment
  - So we had to re-implement it
    - This time using standard Coherence aggregators

Stream API: Examples

- Legacy API

```
Integer count =
 cache.aggregate(filter, new Count());

Long max = cache.aggregate(filter,
          new LongMax(Person::getAge))
```

- Stream API

```
long count = cache.stream(filter)
                  .count();

int max = cache.stream(filter)
            .mapToInt(Person::getAge)
            .max();
```

Stream API: Examples

```
String names = cache.stream()
     .flatMap(p -> p.getChildren().stream())
     .map(Person::getName)
     .filter(s -> s.startsWith(“A”))
     .map(String::toUpperCase)
     .collect(joining(“, “));
```

Stream API: Under the Hood

- Collector-based
  - Supplier: creates a new result container
  - Accumulator: incorporates single data element into a result container
  - Combiner: merges two result containers into one
  - Finisher: performs final transformation of a result container
- New CollectorAggregator
  - Used internally within Stream API implementation
  - New base class for most built-in aggregators
    - AbstractAggregator is deprecated
- Many useful Collector implementations out-of-the-box

Aggregator Improvements

- Aggregators are now stream-based, for performance reasons

```
public R aggregate(Stream<? extends Entry<? extends K, ? extends V>> stream);
```

- Breaking change, but not if you extend AbstractAggregator
  - Although you should probably consider switching to CollectorAggregator
- We are also working on support for
  - Sequential aggregators
  - Aggregator short-circuiting

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Note: The speaker notes for this slide include instructions for when to use Safe Harbor Statement slides.

Tip! Remember to remove this text box.

Hardware and Software

Engineered to Work Together

ORACLE®