Using Java CompletionStage in Asynchronous Programming

Douglas Surber
Oracle Database JDBC Architect
Database Server Technologies
October 25, 2018
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, timing, and pricing of any features or functionality described for Oracle’s products may change and remains at the sole discretion of Oracle Corporation.
Introduction to CompletionStage

What are java.util.concurrent.CompletionStage and java.util.concurrent.CompletableFuture?
CompletableFuture: 
The Promises of Java 
[DEV5375]

Venkat Subramaniam 
Wednesday, October 24
Hands-on Lab: The Asynchronous Java Database Access Driver
[HOL4799]

Douglas Surber
Wednesday, October 24
public interface CompletionStage<T>

• A stage of a possibly asynchronous computation, that performs an action or computes a value when another CompletionStage completes. A stage completes upon termination of its computation, but this may in turn trigger other dependent stages.

• stage.thenApply(x -> square(x))
  .thenAccept(x -> System.out.print(x))
  .thenRun(() -> System.out.println());
java.util.concurrent.CompletableFuture

public class CompletableFuture<T> implements Future<T>, CompletionStage<T>

• Both a CompletionStage and a Future
• A Future that may be explicitly completed (setting its value and status), and may be used as a CompletionStage, supporting dependent functions and actions that trigger upon its completion.
• CompletableFuture future = ...;
  future.complete(value);
  future.get();
Example

supplyAsync(Supplier supplier), thenApply(Function function)

CompletionStage task = CompleteableFuture.supplyAsync(() -> 10);
CompletionStage squareTask = task.thenApply( v -> v * v );
supplyAsync(Supplier supplier)
java.util.concurrent.CompletableFuture

public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier)
Returns a new CompletableFuture that is asynchronously completed by a task running in the ForkJoinPool.commonPool() with the value obtained by calling the given Supplier.

Type Parameters:  U - the function's return type

Parameters:   supplier - a function returning the value to be used to complete the returned CompletableFuture

Returns: the new CompletableFuture
thenApply(\(Function<? super T,? extends U> fn\))

```
java.util.concurrent.CompletionStage
```

\(<U> CompletionStage<U> thenApply(Function<? super T,? extends U> fn)\)

Returns a new CompletionStage that, when this stage completes normally, is executed with this stage's result as the argument to the supplied function.

**Type Parameters:** U - the function's return type

**Parameters:** fn - the function to use to compute the value of the returned CompletionStage

**Returns:** the new CompletionStage
Example

CompletionStage task = CompletableFuture.supplyAsync(() -> 10);
CompletionStage squareTask = task.thenApply(v -> v * v);
Example Execution

```
thenApply

squareTask
v -> v * v

thenApply

squareTask
v -> v * v

( () -> 10 )
```
Introduction to AoJ

What are AoJ and ADBA?
Asynchronous Database Access (ADBA)
Proposed Java Standard

• What: Java standard database access API that never blocks user threads
• Who: Developed by the JDBC Community, JDBC Expert Group and Oracle
• When: Targeted for a near future release, Java 14 perhaps
• Why: Async apps have better scalability
  – Fewer threads means less thread scheduling, less thread contention
  – Database access is slow so blocked threads leave resources idle for a long time
• http://hg.openjdk.java.net/jdk/sandbox/file/JDK-8188051-branch/src/jdk.incubator.adba/share/classes
ADBA Example

Select some items from a table

```java
public CompletionStage<List<Item>> itemsForAnswer(DataSource ds, int answer) {
    String sql = "select id, name, answer from tab where answer = :target";
    try (Session session = ds.getSession()) {
        return session.<List<Item>>rowOperation(sql)
            .set("target", answer, AdbaType.NUMERIC)
            .collect(Collectors.mapping(
                row -> new Item(row.at("id").get(Integer.class),
                                 row.at("name").get(String.class),
                                 row.at("answer").get(Integer.class)),
                Collectors.toList()))
            .submit()
            .getCompletionStage();
    }
}
```
ADBA over JDBC (AoJ)

Open Source implementation of ADBA using any JDBC as a backend

```java
DataSource ds = DataSourceFactory
    .newFactory("com.oracle.adbaoverjdbc.DataSourceFactory")
    .builder()
    .url("jdbc:derby:/myDB")
    .username("scott")
    .password("tiger")
    .build();
```

- https://github.com/oracle/oracle-db-examples/tree/master/java/AoJ
Using CompletionStage
public CompletionStage<List<Item>> itemsForAnswer(DataSource ds, int answer) {
    String sql = "select id, name, answer from tab where answer = :target";
    try (Session session = ds.getSession()) {
        return session.<List<Item>>rowOperation(sql)
            .set("target", answer, AdbaType.NUMERIC)
            .collect(Collectors.mapping(
                row -> new Item(row.at("id").get(Integer.class),
                               row.at("name").get(String.class),
                               row.at("answer").get(Integer.class)),
                Collectors.toList()))
            .submit()
            .getCompletionStage();
    }
}
submit()
com.oracle.adbaoverjdbc.Operation

public Submission<T> submit() {
    if (isImmutable()) {
        throw new IllegalStateException("TODO");
    }
    immutable();
    return group.submit(this);
}
submit(Operation op)
com.oracle.adbaoverjdbc.OperationGroup

Submission<S> submit(Operation<S> op) {
    memberTail =
        op.attachCompletionHandler(op.follows(memberTail, getExecutor()));
    return Submission.submit(this::cancel, memberTail);
}
attachCompletionHandler
com.oracle.adbaoverjdbc.OperationGroup

final CompletionStage<T>
attachCompletionHandler(CompletionStage<T> result) {
    return result.handle((r, t) -> {
        Throwable ex = unwrapException(t);
        checkAbort(ex);
        if (t == null)
            return handleResult(r);
        else
            throw handleException(ex);
    });
}
handle(BiFunction<?, Throwable, ?> fn)
java.util.concurrent.CompletionStage

<U> CompletionStage<U> handle(BiFunction<? super T, Throwable, ? extends U> fn)

Returns a new CompletionStage that, when this stage completes either normally or exceptionally, is executed with this stage's result and exception as arguments to the supplied function. When this stage is complete, the given function is invoked with the result (or null if none) and the exception (or null if none) of this stage as arguments, and the function's result is used to complete the returned stage.

**Type Parameters:** U - the function's return type

**Parameters:** fn - the function to use to compute the value of the returned CompletionStage

**Returns:** the new CompletionStage
attachCompletionHandler

memberTail = follows(...).handle( (r, t) -> { ... } )

Result of previous op

follows(...)

handle

(r, t) -> {...

Result of previous op

memberTail
(r, t) -> { ...
com.oracle.adbaoverjdbc.OperationGroup

final CompletionStage<T>
attachCompletionHandler(CompletionStage<T> result) {
    return result.handle((r, t) -> {
        Throwable ex = unwrapException(t);
        checkAbort(ex);
        if (t == null)
            return handleResult(r);
        else
            throw handleException(ex);
    });
}
submit(Operation op)
com.oracle.adbaoverjdbc.OperationGroup

Submission<S> submit(Operation<S> op) {
    memberTail =
    op.attachCompletionHandler(op.follows(memberTail, getExecutor()));
    return Submission.submit(this::cancel, memberTail);
}
follows(CompletionStage<?> predecessor, Executor executor) {
    predecessor = attachFutureParameters(predecessor);
    return predecessor.
        .thenRunAsync(this::executeQuery, executor)
        .thenCompose(this::moreRows);
}
thenRunAsync(Runnable action)

java.util.concurrent.CompletionStage

CompletionStage<Void> thenRunAsync(Runnable action)

Returns a new CompletionStage that, when this stage completes normally, executes the given action using this stage's default asynchronous execution facility.

Parameters: action - the action to perform before completing the returned CompletionStage

Returns: the new CompletionStage
follows(...)

predecessor.thenRunAsync(this::executeQuery, executor)
follows(CompletionStage<?> predecessor, Executor executor) {
    predecessor = attachFutureParameters(predecessor);
    return predecessor
        .thenRunAsync(this::executeQuery, executor)
        .thenCompose(this::moreRows);
}
thenCompose(Function<?, CompletionStage>, Executor executor)

java.util.concurrent.CompletionStage

<U> CompletionStage<U> thenCompose(Function<? super T,? extends CompletionStage<U>> fn)

Returns a new CompletionStage that is completed with the same value as the CompletionStage returned by the given function. When this stage completes normally, the given function is invoked with this stage's result as the argument, returning another CompletionStage. When that stage completes normally, the CompletionStage returned by this method is completed with the same value.

Type Parameters: U - the type of the returned CompletionStage's result

Parameters: fn - the function to use to compute another CompletionStage

Returns: the new CompletionStage
follows(...)  
thenCompose(this::moreRows)
submit(Operation op, Executor executor)

com.oracle.adbaoverjdbc.OperationGroup
Executing RowOperation

Previous Operation

\[
\text{Result of previous op \rightarrow thenRunAsync \rightarrow executeQuery \rightarrow thenCompose \rightarrow moreRows \rightarrow handle\rightarrow (r,t) \rightarrow \{\ldots\}}
\]
Executing `RowOperation` executeQuery

Result of previous op thenRunAsync executeQuery thenCompose `moreRows` handle `(r,t) -> {...}`
moreRows(Object x)
com.oracle.adbaoverjdbc.RowOperation

protected CompletionStage<T> moreRows(Object x) {
    checkCanceled();
    if (rowsRemain) {
        return CompletableFuture.runAsync(this::handleFetchRows, getExecutor())
            .thenCompose(this::moreRows);
    }
    else {
        return CompletableFuture.supplyAsync(this::completeQuery, getExecutor());
    }
}
runAsync(Runnable action, Executor executor)
java.util.concurrent.CompletionStage

public static CompletableFuture<Void> runAsync(Runnable runnable, Executor executor)

Returns a new CompletableFuture that is asynchronously completed by a task running in the given executor after it runs the given action.

**Parameters:**
- `runnable` - the action to run before completing the returned CompletableFuture
- `executor` - the executor to use for asynchronous execution

**Returns:** the new CompletableFuture
moreRows(...)

runAsync(this::handleFetchRows).thenCompose(this::moreRows)
Executing RowOperation

Before `moreRows(Object x)`

Result of previous op → `executeQuery` → `thenRunAsync` → `thenCompose` → `moreRows` → handle → `(r,t) -> {...}`
Executing RowOperation

```java
runAsync(this::handleFetchRows).thenCompose(this::moreRows)
```

Result of previous op

executeQuery

thenCompose

handleFetchRows

runAsync

thenCompose

thenRunAsync

moreRows

thenCompose

(r,t) -> {...}

handle

moreRows

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.
Executing `moreRows`

\[
\text{runAsync}(\text{this}::\text{handleFetchRows}).\text{thenCompose}(\text{this}::\text{moreRows})
\]
Executing moreRows

runAsync(this::handleFetchRows).thenCompose(this::moreRows)
Executing moreRows

runAsync(this::handleFetchRows)
Executing moreRows

`.thenCompose(this::moreRows)`
moreRows(Object x)
com.oracle.adbaoverjdbc.RowOperation

protected CompletionStage<T> moreRows(Object x) {
    checkCanceled();
    if (rowsRemain) {
        return CompletableFuture.runAsync(this::handleFetchRows, getExecutor()).thenCompose(this::moreRows, getExecutor());
    } else {
        return CompletableFuture.supplyAsync(this::completeQuery, getExecutor());
    }
}
Executing `moreRows` when no more rows

```
.supplyAsync(this::completeQuery)
```

Result of previous op

```
executeQuery
```

```
thenCompose
```

```
runAsync
```

```
handleFetchRows
```

```
thenCompose
```

```
runAsync
```

```
handleFetchRows
```

```
thenCompose
```

```
mroeRows
```

```
mroeRows
```

```
mroeRows
```

```
mroeRows
```

```
handle
```

```
Third
```

```
(r,t) -> {...}
```

```
completeQuery
```

```
supplyAsync
```
Executing `completeQuery`.

```java
supplyAsync(this::completeQuery)
```

Diagram:
- `executeQuery`
  - `thenCompose`
    - `runAsync`
      - `handleFetchRows`
    - `thenCompose`
      - `runAsync`
      - `handleFetchRows`
  - `thenCompose`
    - `runAsync`
      - `handleFetchRows`
  - `thenRunAsync`
  - `Result of previous op`
- `moreRows`
  - `thenCompose`
    - `runAsync`
      - `handleFetchRows`
    - `thenCompose`
      - `runAsync`
      - `handleFetchRows`
  - `thenCompose`
    - `runAsync`
      - `handleFetchRows`
- `handle`
- `(r, t) -> {...}`

Third
Query complete

Result of completeQuery propagates back

executeQuery

thenCompose

moreRows

runAsync

handleFetchRows

thenCompose

moreRows

runAsync

handleFetchRows

thenCompose

moreRows

thenRunAsync

supplyAsync

Third

(r,t) -> {...}

Second

handle

First

Result of previous op
Executing completionHandler

```java
.dispatchEvent(
    completionHandler: { (r, t) -> { ... } }
)
```
Summary
follows(CompletionStage<?> predecessor, Executor executor) {
    predecessor = attachFutureParameters(predecessor);
    return predecessor.
        .thenRunAsync(this::executeQuery, executor)
        .thenCompose(this::moreRows);
}
follows(CompletionStage<?> predecessor, Executor executor)
moreRows(Object x)
com.oracle.adbaoverjdbc.RowOperation

protected CompletionStage<T> moreRows(Object x) {
    checkCanceled();
    if (rowsRemain) {
        return CompletableFuture.runAsync(this::handleFetchRows, getExecutor())
            .thenCompose(this::moreRows, getExecutor());
    }
    if {
        return CompletableFuture.supplyAsync(this::completeQuery, getExecutor());
    }
}
moreRows(Object x)
attachCompletionHandler
com.oracle.adbaoverjdbc.OperationGroup

final CompletionStage<T>
attachCompletionHandler(CompletionStage<T> result) {
    return result.handle((r, t) -> {
        Throwable ex = unwrapException(t);
        checkAbort(ex);
        if (t == null)
            return handleResult(r);
        else
            throw handleError(ex);
    });
}
Executing completionHandler

\[\text{handle}( \ (r, t) \rightarrow \{\ldots\} )\]
Methods used to implement the example code

CompletionStage

<U> CompletionStage<U> handle (BiFunction<? super T, Throwable, ? extends U> fn)
<U> CompletionStage<U> thenApply(Function<? super T,? extends U> fn)
<U> CompletionStage<U> thenCompose (Function<? super T,? extends CompletionStage<U>> fn)
CompletionStage<Void> thenRunAsync (Runnable action, Executor executor)

CompletableFuture

public static CompletableFuture<Void> runAsync (Runnable runnable, Executor executor)
public static <U> CompletableFuture<U> supplyAsync (Supplier<U> supplier)