Advanced Java CompletableFuture

Features: Introducing Factory Methods

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand advanced features of completable futures, e.g.
- Factory methods initiate async computations

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
Factory Methods Initiate Async Computations
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations

See [en.wikipedia.org/wiki/Factory_method_pattern](en.wikipedia.org/wiki/Factory_method_pattern)
Four factory methods initiate async computations

These computations may or may not return a value

```java
class CompletableFuture<T> {
    CompletableFuture()  
    cancel(boolean): boolean  
    isCancelled(): boolean  
    isDone(): boolean  
    get()  
    get(long, TimeUnit)  
    join()  
    complete(T): boolean

    supplyAsync(Supplier<T>): CompletableFuture<T>  
    supplyAsync(Supplier<T>, Executor): CompletableFuture<T>  
    runAsync(Runnable): CompletableFuture<Void>  
    runAsync(Runnable, Executor): CompletableFuture<Void>

    completedFuture(T): CompletableFuture<T>  
    thenApply(Function<T>): CompletableFuture<T>  
    thenAccept(Consumer<T>, super T): CompletableFuture<Void>

    whenComplete(BiConsumer<T, ?>): CompletableFuture<Void>

    allOf(CompletableFuture[]): CompletableFuture<T>  
    anyOf(CompletableFuture[]): CompletableFuture<Object>
}
```
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
- `supplyAsync()` allows two-way calls via a supplier

<table>
<thead>
<tr>
<th>Methods</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>supply Async</td>
<td>Supplier</td>
<td>Completable Future with result of Supplier</td>
<td>Asynchronously run supplier in common fork/join pool</td>
</tr>
<tr>
<td>supply Async</td>
<td>Supplier, Executor</td>
<td>Completable Future with result of Supplier</td>
<td>Asynchronously run supplier in given executor context</td>
</tr>
</tbody>
</table>

See [docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html](https://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html)
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
- supplyAsync() allows two-way calls via a supplier
- Can be passed params

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704"

CompletableFuture<BigFraction> future = CompletableFuture
    .supplyAsync(() -> {
        BigFraction bf1 = new BigFraction(f1);
        BigFraction bf2 = new BigFraction(f2);

        return bf1.multiply(bf2);
    });

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
- `supplyAsync()` allows two-way calls via a supplier
- Can be passed params

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";
CompletableFuture<BigFraction> future = CompletableFuture.supplyAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
});
```

See javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
  - supplyAsync() allows two-way calls via a supplier
    - Can be passed params
  - Returns a value

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";

CompletableFuture<BigFraction> future = CompletableFuture
    .supplyAsync(() -> {
        BigFraction bf1 = new BigFraction(f1);
        BigFraction bf2 = new BigFraction(f2);

        return bf1.multiply(bf2);
    });
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex8)
## Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
  - `supplyAsync()` allows two-way calls via a supplier
  - `runAsync()` enables one-way calls via a runnable

### Methods Params Returns Behavior

<table>
<thead>
<tr>
<th>Method</th>
<th>Params</th>
<th>Returns</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>runAsync()</code></td>
<td><code>Runnable</code></td>
<td><code>CompletableFuture with result of Void</code></td>
<td>Asynchronously run runnable in common fork/join pool</td>
</tr>
<tr>
<td><code>runAsync()</code></td>
<td><code>Runnable, Executor</code></td>
<td><code>CompletableFuture with result of Void</code></td>
<td>Asynchronously run runnable in given executor context</td>
</tr>
</tbody>
</table>

See [docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
  - `supplyAsync()` allows two-way calls via a supplier
  - `runAsync()` enables one-way calls via a runnable
- Can be passed params

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";

CompletableFuture<Void> future = CompletableFuture.runAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    System.out.println(bf1.multiply(bf2).toMixedString());
});
```

Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
  - These computations may or may not return a value
    - supplyAsync() allows two-way calls via a supplier
    - runAsync() enables one-way calls via a runnable
    - Can be passed params
    - Returns no value

```java
String f1 = "62675744/15668936";
String f2 = "609136/913704";

CompletableFuture<Void> future = CompletableFuture.runAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);

    System.out.println(bf1.multiply(bf2).toString());
});
```

"Void" is not a value!

See [www.baeldung.com/java-void-type](http://www.baeldung.com/java-void-type)
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
  - supplyAsync() allows two-way calls via a supplier
  - runAsync() enables one-way calls via a runnable
- Can be passed params
- Returns no value

```
String f1 = "62675744/15668936";
String f2 = "609136/913704";
CompletableFuture<Void> future = CompletableFuture.runAsync(() -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    System.out.println(bf1.multiply(bf2).toMixedString());
});
```

Any output must therefore come from "side-effects"

See en.wikipedia.org/wiki/Side_effect_(computer_science)
Factory Methods Initiate Async Computations

• Four factory methods initiate async computations
  • These computations may or may not return a value
    • supplyAsync() allows two-way calls via a supplier
    • runAsync() enables one-way calls via a runnable

supplyAsync() is more commonly used than runAsync() in practice
• Four factory methods initiate async computations
  • These computations may or may not return a value
• Async functionality runs in a thread pool

Help make programs more *elastic* by leveraging a pool of worker threads
Four factory methods initiate async computations

- These computations may or may not return a value
- Async functionality runs in a thread pool

By default, the common fork-join pool is used

See [dzone.com/articles/common-fork-join-pool-and-streams](dzone.com/articles/common-fork-join-pool-and-streams)
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
- Async functionality runs in a thread pool

However, a pre- or user-defined thread pool can also be given

See www.baeldung.com/thread-pool-java-and-guava
Factory Methods Initiate Async Computations

- Four factory methods initiate async computations
- These computations may or may not return a value
- Async functionality runs in a thread pool
- In contrast, Java parallel streams use the common fork-join pool

Parallel Streams
- filter(not(this::urlCached))
- map(this::downloadImage)
- map(this::applyFilters)
- reduce(Stream::concat) ...
- collect(toList())

Common ForkJoinPool

See lesson on “Java Parallel Stream Internals: Parallel Processing via the Common Fork-Join Pool”
End of Advanced Java CompletableFuture Features: Introducing Factory Methods