Evaluating the Pros of the Java Completable Futures Framework

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 the pros of using the Java completable futures framework
Learning Objectives in this Part of the Lesson

- Understand the pros of using the Java completable futures framework
- We evaluate the Java completable futures framework compared with the parallel streams framework

Pros of the Java Completable Futures Framework
Pros of the Java Completable Futures Framework

• Greatly simplifies programming of asynchronous operations

Task 1
```
Task 1
getPage
 supplyAsync (getStartPage())
```

Task 2
```
Task 2
getPage
 .thenApplyAsync (countImages(page))
 .thenApply (List::size)
```

Task 3
```
Task 3
getPage
 .thenComposeAsync (crawlHyperLinks (page))
```

Task 4
```
Task 4
getPage
 .thenCombine (/getPage,
 (imgNum1, imgNum2) -> Integer::sum)
```
Pros of the Java Completable Futures Framework

- Greatly simplifies programming of asynchronous operations
- Supports dependent actions that trigger upon completion of async operations

```java
Task 1
news/api\ = supplyAsync
   (getStartPage())

Task 2
\imgNum1\ = \page\ .thenApplyAsync
   (countImages(page))
   .thenApply(List::size)

Task 3
\imgNum2\ = \page\ .thenComposeAsync
   (crawlHyperLinks(page))

Task 4
\imgNum1\ .thenCombine(/imgNum2,\
   (imgNum1, imgNum2) -> Integer::sum)
```
Pros of the Java Completable Futures Framework

• Greatly simplifies programming of asynchronous operations

• Supports dependent actions that trigger upon completion of async operations

• Async operations can be forked, chained, & joined in a relatively intuitive way

```
Task 1
```

```
Task 2
```

```
Task 3
```

```
Task 4
```

```
8 /page\ = supplyAsync
    (getStartPage())
```

```
1 /imgNum1\ = /page\ 8
    .thenApplyAsync
    (countImages(page))
    .thenApply(List::size)
```

```
26 /imgNum2\ = /page\ 8
    .thenComposeAsync
    (crawlHyperLinks
     (page))
```

```
1 /imgNum1\ .thenCombine(/imgNum2,\
    (imgNum1, imgNum2) -> Integer::sum)
```

```
26
```

```
Spider-Sense...
TINGLING!
```

```
7
```
Pros of the Java Completable Futures Framework

• Greatly simplifies programming of asynchronous operations

• Supports dependent actions that trigger upon completion of async operations
  • Async operations can be forked, chained, & joined in a relatively intuitive way

• Enables async programs to appear like sync programs

```java
BigFraction unreduced = BigFraction
    .valueOf(new BigInteger
        ("846122553600669882"),
        new BigInteger
        ("188027234133482196"),
        false); // Don’t reduce!

Supplier<BigFraction> reduce = () ->
    BigFraction.reduce(unreduced);

CompletableFuture
    .supplyAsync(reduce)
    .thenApply(BigFraction
        ::toMixedString)
    .thenAccept(System.out
        ::println);
```
Pros of the Java Completable Futures Framework

- Greatly simplifies programming of asynchronous operations
- Supports dependent actions that trigger upon completion of async operations
- Async operations run in parallel in a thread pool

```
Task 1
8
$page\ = \ supplyAsync
\(\text{getPage}\())

Task 2
1
/imgNum1\ = /page\ 8
 .thenApplyAsync
   \(\text{getPage}\)
   \(\text{countImages}\) (page))
 .thenApply \(\text{List::size}\)

Task 3
26
/imgNum2\ = /page\ 8
 .thenComposeAsync
   \(\text{getPage}\)
   \(\text{crawlHyperLinks}\) (page))

Task 4
1
/imgNum1\ .thenCombine(/imgNum2\,
   (imgNum1, imgNum2) -> Integer::sum)
```
Pros of the Java Completable Futures Framework

- Greatly simplifies programming of asynchronous operations
- Supports dependent actions that trigger upon completion of async operations
- Async operations run in parallel in a thread pool
- Either a (common) fork-join pool or various types of pre- or user-defined thread pools

```java
Task 1
/page\ = supplyAsync
  (getStartPage())

Task 2
/imgNum1\ = /page\8
  .thenApplyAsync
    (countImages(page))
  .thenApply(List::size)

Task 3
/imgNum2\ = /page\8
  .thenComposeAsync
    (crawlHyperLinks
     (page))

Task 4
/imgNum1\ .thenCombine(/imgNum2\,
                        (imgNum1, imgNum2) -> Integer::sum)
```

10
Pros of the Java Completable Futures Framework

• No explicit synchronization or threading is required for completable futures

```java
Completable Futures
map(this::checkUrlCachedAsync)
map(this::downloadImageAsync)
flatMap(this::applyFiltersAsync)
collect(toFuture())
thenAccept(this::logResults)
```
Pros of the Java Completable Futures Framework

- No explicit synchronization or threading is required for completable futures
- Java libraries handle locking needed to protect shared mutable state

See docs.oracle.com/javase/tutorial/essential/concurrency/collections.html
Pros of the Java Completable Futures Framework

- Completable futures are often more efficient than parallel streams

```java
CompletableFutures
map(this::downloadImageAsync)
flatMap(this::applyFiltersAsync)
collect(toFuture())
thenAccept(this::logResults)
```
Pros of the Java Completable Futures Framework

• Completable futures are often more efficient than parallel streams
• Naturally, your mileage may vary...

There’s no substitute for benchmarking, e.g., java-performance.info/jmh!
Pros of the Java Completable Futures Framework

- Combining sequential streams & completable futures is often a win

```java
map(this::checkUrlCachedAsync)
.flatMap(this::applyFiltersAsync)
.collect(toFuture())
.thenAccept(this::logResults)
```
Pros of the Java Completable Futures Framework

• Combining sequential streams & completable futures is often a win

• Streams guide the overall flow of control...

```
map(this::checkUrlCachedAsync)
map(this::downloadImageAsync)
flatMap(this::applyFiltersAsync)
collect(toFuture())
thenAccept(this::logResults)
```
Pros of the Java Completable Futures Framework

- Combining sequential streams & completable futures is often a win
  - Streams guide the overall flow of control... completable futures perform async operations in parallel

```
map(this::checkUrlCachedAsync)
map(this::downloadImageAsync)
flatMap(this::applyFiltersAsync)
collect(toFuture())
thenAccept(this::logResults)
```
Pros of the Java Completable Futures Framework

• Combining sequential streams & completable futures is often a win
  • Streams guide the overall flow of control... completable futures perform async operations in parallel
  • However, combining parallel streams & completable futures may be overkill.
End of Evaluating the Pros of the Java Completable Futures Framework