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

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

- Evaluate 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

```java
Task 1
supplyAsync
  (getStartPage())
```

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

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

```java
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

\[
\text{Task 1:} \quad \text{getPage} = \text{supplyAsync} (\text{getPageStartPage()})
\]

\[
\text{Task 2:} \quad /\text{imgNum1} = /\text{page} . \text{thenApplyAsync} (\text{countImages(page)}) . \text{thenApply} (\text{List.size})
\]

\[
\text{Task 3:} \quad /\text{imgNum2} = /\text{page} . \text{thenComposeAsync} (\text{crawlHyperLinks(page)})
\]

\[
\text{Task 4:} \quad /\text{imgNum1}. \text{thenCombine} (/\text{imgNum2}, \text{(imgNum1, imgNum2) \rightarrow Integer.size})
\]
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
8
```
```
//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,\n  (imgNum1, imgNum2) \rightarrow Integer::sum)
```

Spider-Sense...
TINGLING!
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
supplyAsync(getStartPage())
```

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

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

```
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
- 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
Pros of the Java Completable Futures Framework

• No explicit synchronization or threading is required for completable futures

```
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::checkUrlCachedAsync)
flatMap(this::applyFiltersAsync)
collect(toFuture())
thenAccept(this::logResults)
```
There's no substitute for benchmarking, e.g., java-performance.info/jmh!

Pros of the Java Completable Futures Framework

• Completable futures are often more efficient than parallel streams

• Naturally, your mileage may vary..

```java
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

```
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

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 efficiently in parallel.
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 efficiently in parallel.
• However, combining parallel streams & completable futures is overkill..
End of Evaluating the Pros of the Java Completable Futures Framework