Evaluating the Cons 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
• Understand the cons 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
• Understand the cons of using the Java completable futures framework
• Again, we evaluate the Java completable futures framework compared with the parallel streams framework

Cons of the Java Completable Futures Framework
void processStream() {
    List<URL> urls = getInput();

    List<Image> images = urls
                    .parallelStream()
                    .filter(not(this::urlCached))
                    .map(this::blockingDownload)
                    .flatMap(this::applyFilters)
                    .collect(toList());

    logResults(images); ...
void processStream() {
    List<URL> urls = getInput();
    List<Image> images = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());
    logResults(images); ...
}

void processStream() {
    List<URL> urls = getInput();
    CompletableFuture<Stream<Image>> resultsFuture = urls
        .stream()
        .map(this::checkUrlCachedAsync)
        .map(this::downloadImageAsync)
        .flatMap(this::applyFiltersAsync)
        .collect(toFuture())
        .thenApply(this::logResults)
        .join(); ...
}

Cons of the Java Completable Futures Framework

- It’s easier to program Java parallel streams than completable futures
- The overall control flow is similar when using the Java streams framework
void processStream() {
    List<URL> urls = getInput();
    List<Image> images = urls
        .parallelStream()
        .filter(not(this::urlCached))
        .map(this::blockingDownload)
        .flatMap(this::applyFilters)
        .collect(toList());

    logResults(images); ...
}

void processStream() {
    List<URL> urls = getInput();
    CompletableFuture<Stream<Image>> resultsFuture = urls
        .stream()
        .map(this::checkUrlCachedAsync)
        .map(this::downloadImageAsync)
        .flatMap(this::applyFiltersAsync)
        .collect(toFuture())
        .thenApply(this::logResults)
        .join(); ...
}

Cons of the Java Completable Futures Framework

- It’s easier to program Java parallel streams than completable futures
- The overall control flow is similar when using the Java streams framework
- However, async behaviors are more complicated than the sync behaviors!
Cons of the Java Completable Futures Framework

- There's a tradeoff between computing performance & programmer productivity when choosing amongst these frameworks.

Printing 4 results for input file 1 from fastest to slowest:
- COMPLETABLE_FUTURES_1 executed in 312 msecs
- COMPLETABLE_FUTURES_2 executed in 335 msecs
- PARALLEL_STREAM executed in 428 msecs
- SEQUENTIAL_STREAM executed in 981 msecs

Printing 4 results for input file 2 from fastest to slowest:
- COMPLETABLE_FUTURES_2 executed in 82 msecs
- COMPLETABLE_FUTURES_1 executed in 83 msecs
- PARALLEL_STREAM executed in 102 msecs
- SEQUENTIAL_STREAM executed in 251 msecs
Printing 4 results for input file 1 from fastest to slowest
COMPLETABLE_FUTURES_1 executed in 312 msecs
COMPLETABLE_FUTURES_2 executed in 335 msecs
PARALLEL_STREAM executed in 428 msecs
SEQUENTIAL_STREAM executed in 981 msecs

Printing 4 results for input file 2 from fastest to slowest
COMPLETABLE_FUTURES_2 executed in 82 msecs
COMPLETABLE_FUTURES_1 executed in 83 msecs
PARALLEL_STREAM executed in 102 msecs
SEQUENTIAL_STREAM executed in 251 msecs

Cons of the Java Completable Futures Framework

- There's a tradeoff between computing performance & programmer productivity when choosing amongst these frameworks, e.g.

  - Completable futures are more efficient & scalable, but are harder to program

In general, asynchrony patterns aren’t well understood by many developers
Cons of the Java Completable Futures Framework

- There's a tradeoff between computing performance & programmer productivity when choosing amongst these frameworks, e.g.
  - Completable futures are more efficient & scalable, but are harder to program.
  - Parallel streams are easier to program, but are less efficient & scalable.

Use sequential streams for initial development & then trivially make them parallel!
Cons of the Java Completable Futures Framework

• As usual, it is essential to know the better practices & patterns to program completable futures effectively!!
End of Evaluating the Cons of the Java Completable Futures Framework