Overview of Java Futures

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

- Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony
- Know how Java futures provide the foundation for completable futures in Java

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html
Learning Objectives in this Part of the Lesson

• Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony

• Know how Java futures provide the foundation for completable futures in Java

• Visualize Java futures in action
Learning Objectives in this Part of the Lesson

• Motivate the need for Java futures by understanding the pros & cons of synchrony & asynchrony

• Know how Java futures provide the foundation for completable futures in Java
  • Visualize Java futures in action
  • Understand a human known use of Java futures
Overview of Java Futures
Overview of Java Futures

- Java 5 added async call support via the Java Future interface

See en.wikipedia.org/wiki/Java_version_history
Overview of Java Futures

- Methods on Java Future can manage a task’s lifecycle after it’s submitted to run asynchronously.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html
Overview of Java Futures

- Methods on Java Future can manage a task’s lifecycle after it’s submitted to run asynchronously, e.g.
  - A future can be tested for completion

```
5.run()
4.take()
3.offer()
submit()
workQueue
callable

Future<V>
- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long, TimeUnit)

if (future.isDone())
...
Overview of Java Futures

• Methods on Java Future can manage a task’s lifecycle after it’s submitted to run asynchronously, e.g.
  • A future can be tested for completion
  • A future be tested for cancellation & cancelled

```
Thread (main thread)
BigFraction
Future
Callable
submit(task)
Callable
submit()
offer()
take()
run()
Future
Callable
if (!future.isCancelled())
future.cancel();
ThreadPoolExecutor
WorkQueue
run()
Fixed WorkerThreads
Callable
Callable
Callable
Callable
Callable
Callable
```
Overview of Java Futures

- Methods on Java Future can manage a task’s lifecycle after it’s submitted to run asynchronously, e.g.
  - A future can be tested for completion
  - A future be tested for cancellation & cancelled
  - A future can retrieve a two-way task’s result

```
Future<V> future = new Future<>();
...
ResultType result = future.get();
```

```
Future.run();
Future.take();
Future.get();
Future.isStop();
Future.isRunning();
Future.isCancelled();
Future.isDone();
```
Overview of Java Futures

- An Java async call returns a future & continues running the computation in the background
Visualizing Java Futures
Visualizing Java Futures

- `ExecutorService.submit()` can initiate an async call in Java

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit)
Visualizing Java Futures

- `ExecutorService.submit()` can initiate an async call in Java
- Create a thread pool

```java
ExecutorService executorService = Executors.newFixedThreadPool(sMAX_THREADS);
```
Visualizing Java Futures

- ExecutorService.submit() can initiate an async call in Java
  - Create a thread pool
  - Submit a task

```java
Callable<BigFraction> task = () -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future = executorService.submit(task);
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#submit)
Visualizing Java Futures

- ExecutorService.submit() can initiate an async call in Java
- Create a thread pool
- Submit a task
- Return a future
- Implemented as a FutureTask

Callable<BigFraction> task = () -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future = executorService.submit(task);

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html
Visualizing Java Futures

- ExecutorService.submit() can initiate an async call in Java
  - Create a thread pool
  - Submit a task
  - Return a future
  - Run computation asynchronously

Callable<BigFraction> task = () -> {
    BigFraction bf1 = new BigFraction(f1);
    BigFraction bf2 = new BigFraction(f2);
    return bf1.multiply(bf2);
};

Future<BigFraction> future = executorService.submit(task);
Visualizing Java Futures

- When the async call completes the future is triggered & the result is available.

See [www.nurkiewicz.com/2013/02/javau...future-basics.html](http://www.nurkiewicz.com/2013/02/javau...future-basics.html)
Visualizing Java Futures

- When the async call completes the future is triggered & the result is available
- get() can block

BigFraction result = future.get();

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get
When the async call completes the future is triggered & the result is available

- `get()` can block
- `get()` can also be (time-)pollled

```java
BigFraction result = future.get(n, SECONDS);
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html#get)
When the async call completes the future is triggered & the result is available

- get() can block
- get() can also be (time-)polled

**OUT OF ORDER**

Computations can complete in a different order than the async calls were made
A Human Known Use of Java Futures
A Human Known Use of Java Futures

- A future is essentially a proxy that represents the result(s) of an async call.

```
Result get_result ()
begin
    // Suspend calling thread until result is available.
    if (result == NULL) then
        thread.wait();
        return result;
    end
end
```

Result obtained only after the computation completes.

Asynchronous computation.

See en.wikipedia.org/wiki/Futures_and_promises
A Human Known Use of Java Futures

- A future is essentially a proxy that represents the result(s) of an async call

```python
Result get_result ()
begin
  ## Suspend calling thread until result is available.
  if (result == NULL) then
    thread.wait ();
  return result;
end
```

Table tent #'s are a human-known-use of futures!
A Human Known Use of Java Futures

- A future is essentially a proxy that represents the result(s) of an async call.

```java
Result get_result ()
begin
  ## Suspend calling thread until result is available.
  if (result == NULL) then
    thread.wait ();
  return result;
end
```

Table tent #’s are a human-known-use of futures!

```
12
```

e.g., McDonald’s vs Wendy’s model of preparing fast food
End of Overview of Java Futures