The Java ExecutorService Interface

(Part 2)

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

- Recognize the powerful features defined in the Java ExecutorService interface & related interfaces/classes
- Know key methods provided by the Java ExecutorService

```java
// <<Java Interface>>
interface ExecutorService {
    shutdown(): void
    shutdownNow(): List<Runnable>
    isShutdown(): boolean
    isTerminated(): boolean
    awaitTermination(long, TimeUnit): boolean
    submit(Callable<T>): Future<T>
    submit(Runnable,T): Future<T>
    submit(Runnable): Future<?>
    invokeAll(Collection<? extends Callable<T>>): List<Future<T>>
    invokeAny(Collection<? extends Callable<T>>)
    invokeAny(Collection<? extends Callable<T>>,long, TimeUnit)
}
```
Key Methods in the ExecutorService Interface (Part 1)
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit(Callable<T> task);
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
- `execute()` runs one-way tasks that return `void`

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);
...
```

However, this method isn’t very useful/common in practice
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
- submit() runs two-way async tasks that return a value via a future

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...}
```

This method is the most useful/common in practice
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return `void`
- `submit()` runs two-way async tasks that return a value via a future
  - Supports “synchronous future” processing model

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
                      (Callable<T> task);
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
- submit() runs two-way async tasks that return a value via a future
  - Supports “synchronous future” processing model
- `Future.get()` can block until task completes successfully

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit (Callable<T> task);
    ...
}
```
### Key Methods in the ExecutorService Interface

- **ExecutorService can execute individual tasks**
  - `execute()` runs one-way tasks that return `void`
  - `submit()` runs two-way async tasks that return a value via a future
    - Supports “synchronous future” processing model
    - `Future.get()` can block until task completes successfully
    - After which point `get()` returns the task’s result

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit (Callable<T> task);
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way async tasks that return a value via a future
  - `submit()` can also run one-way async tasks that return no value

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    <T> Future<T> submit
        (Runnable task);

    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way async tasks that return a value via a future
  - `submit()` can also run one-way async tasks that return no value
  - It is possible to cancel this computation, however

```java
class ExecutorService {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    <T> Future<T> submit
        (Runnable task);
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks

```java
public interface ExecutorService extends Executor {
    ...
    <T> List<Future<T>> invokeAll(
        Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny(
        Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny(
        Collection<? extends Callable<T>> tasks,
        long timeout, TimeUnit unit)
        ...;
    ...
```
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
- Returns a list of futures when all tasks complete

```java
class ExecutorService extends Executor {
    ...
    <T> List<Future<T>> invokeAll
        (Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends Callable<T>> tasks,
         long timeout, TimeUnit unit)
        ...; ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAll](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAll)
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
  - Returns a list of futures when all tasks complete
  - Return the result of *one* successful completion

```java
public interface ExecutorService extends Executor {

  ...

  <T> List<Future<T>> invokeAll
      (Collection<? extends Callable<T>> tasks) ...;

  <T> T invokeAny
      (Collection<? extends Callable<T>> tasks) ...;

  <T> T invokeAny
      (Collection<? extends Callable<T>> tasks,
       long timeout, TimeUnit unit)
      ...;

Useful for concurrent algorithms that just want the result that completes first
```
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
- Returns a list of futures when all tasks complete
- Return the result of one successful completion
- Cancel uncompleted tasks

```java
public interface ExecutorService
    extends Executor {
    ...
    <T> List<Future<T>> invokeAll
        (Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny(Collection<? extends Callable<T>> tasks,
                    long timeout, TimeUnit unit) ...; ...
```
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
  - Returns a list of futures when all tasks complete
- Return the result of *one* successful completion
- Cancel uncompleted tasks
- Ignore other completed task results

```java
public interface ExecutorService extends Executor {
    ...
    <T> List<Future<T>> invokeAll(
        Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny(
        Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny(Collection<? extends Callable<T>> tasks,
        long timeout, TimeUnit unit) ...;
    ...
}
```
public interface ExecutorService extends Executor {

  ...

  <T> List<Future<T>> invokeAll
      (Collection<? extends Callable<T>> tasks) ...;

  <T> T invokeAny
      (Collection<? extends Callable<T>> tasks) ...;

  <T> T invokeAny
      (Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit)
      ...; ...

• ExecutorService can also execute groups of tasks
• Returns a list of futures when all tasks complete
• Return the result of one successful completion

Don’t modify the collection param while invokeAll() or invokeAny() are running!!!
public interface ExecutorService extends Executor {

...<T> List<Future<T>> invokeAll
(Collection<? extends Callable<T>> tasks) ...;

<T> T invokeAny
(Collection<? extends Callable<T>> tasks) ...;

<T> T invokeAny(Collection<? extends Callable<T>> tasks,
long timeout, TimeUnit unit)
...; ...

These methods block the calling thread until they are finished, which may be non-intuitive.
Key Methods in the ExecutorService Interface (Part 2)
An ExecutorService client can initiate shutdown operations to manage its lifecycle.

```java
public interface ExecutorService extends Executor {
    void shutdown();
    List<Runnable> shutdownNow();
    ...
}
```
An ExecutorService client can initiate shutdown operations to manage its lifecycle.

Perform “orderly shutdown” that completes active tasks.

```java
class ExecutorService extends Executor {
    ...
    void shutdown();
    List<Runnable> shutdownNow();
    ...
}
```
Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
- Perform “orderly shutdown” that completes active tasks
- But ignores new tasks

```
public interface ExecutorService extends Executor {
    ...
    void shutdown();

    List<Runnable> shutdownNow();
    ...
```
Key Methods in the ExecutorService Interface

- An ExecutorService client can initiate shutdown operations to manage its lifecycle
  - Perform “orderly shutdown” that completes active tasks
  - Attempt to cancel active tasks & don’t process waiting tasks

```java
public interface ExecutorService extends Executor {
    ...
    void shutdown();

    List<Runnable> shutdownNow();
    ...
}
```
An ExecutorService client can initiate shutdown operations to manage its lifecycle.

- Perform “orderly shutdown” that completes active tasks.
- Attempt to cancel active tasks & don’t process waiting tasks.
- Activate tasks are cancelled by posting an interrupt request to executor thread(s).

```java
public interface ExecutorService extends Executor {
    ...
    void shutdown();

    List<Runnable> shutdownNow();
    ...
}
```

*Remember that all these Java interrupt requests are “voluntary”!!*

See [docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html](docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html)
An ExecutorService client can initiate shutdown operations to manage its lifecycle

- Perform “orderly shutdown” that completes active tasks
- Attempt to cancel active tasks & don’t process waiting tasks
  - Activate tasks are cancelled by posting an interrupt request to executor thread(s)
- Returns waiting tasks

```java
public interface ExecutorService extends Executor {
  ...
  void shutdown();

  List<Runnable> shutdownNow();
  ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can query status of a shutdown, as well as wait for termination to finish.

```java
public interface ExecutorService extends Executor {
    ...
    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination()
        (long timeout,
         TimeUnit unit) ...
};
```
Key Methods in the ExecutorService Interface

- ExecutorService can query status of a shutdown, as well as wait for termination to finish
- True if Executor shut down

```java
public interface ExecutorService extends Executor {

    ...  
    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination (long timeout, 
                              TimeUnit unit) ...;

```
Key Methods in the ExecutorService Interface

- ExecutorService can query status of a shutdown, as well as wait for termination to finish
  - True if Executor shut down
  - True if all tasks completed after shut down

```java
public interface ExecutorService extends Executor {
    ...
    boolean isShutdown();
    boolean isTerminated();
    boolean awaitTermination(
        long timeout,
        TimeUnit unit) ...;
```
Key Methods in the ExecutorService Interface

- ExecutorService can query status of a shutdown, as well as wait for termination to finish
  - True if Executor shut down
  - True if all tasks completed after shut down
  - Blocks until all tasks complete

```java
public interface ExecutorService extends Executor {
  ...
  boolean isShutdown();
  boolean isTerminated();
  boolean awaitTermination(
    long timeout,
    TimeUnit unit)
  ...
}
```

shutdownNow() may reduce blocking time for awaitTermination()
Key Methods in the ExecutorService Interface

- ExecutorService can query status of a shutdown, as well as wait for termination to finish
  - True if Executor shut down
  - True if all tasks completed after shut down
  - Blocks until all tasks complete

```java
public interface ExecutorService extends Executor {
    ...
    boolean isShutdown();
    boolean isTerminated();
    boolean awaitTermination(
        long timeout, TimeUnit unit) ...
```

`shutdown*()` & `awaitTermination()` provide barrier synchronization

See en.wikipedia.org/wiki/Barrier_(computer_science)
End of Overview of Java ExecutorService (Part 2)