Module java.base
Package java.util.concurrent

Interface Future<V>

Type Parameters:

V - The result type returned by this Future's `get` method

All Known Subinterfaces:

RunnableFuture<V>, RunnableScheduledFuture<V>, ScheduledFuture<V>

All Known Implementing Classes:

CompletableFuture, CountedCompleter, ForkJoinTask, FutureTask, RecursiveAction, RecursiveTask, SwingWorker

---

```
public interface Future<V>
```

A `Future` represents the result of an asynchronous computation. Methods are provided to check if the computation is complete, to wait for its completion, and to retrieve the result of the computation. The result can only be retrieved using method `get` when the computation has completed, blocking if necessary until it is ready. Cancellation is performed by the `cancel` method. Additional methods are provided to determine if the task completed normally or was cancelled. Once a computation has completed, the computation cannot be cancelled. If you would like to use a `Future` for the sake of cancellability but not provide a usable result, you can declare types of the form `Future<?>` and return `null` as a result of the underlying task.

Sample Usage (Note that the following classes are all made-up.)

```
interface ArchiveSearcher { String search(String target); }
class App {
  ExecutorService executor = ...;
  ArchiveSearcher searcher = ...;
  void showSearch(String target) throws InterruptedException {
    Callable<String> task = () -> searcher.search(target);
    Future<String> future = executor.submit(task);
    displayOtherThings(); // do other things while searching
    try {
      displayText(future.get()); // use future
    } catch (ExecutionException ex) { cleanup(); return; }
  }
}
```

The FutureTask class is an implementation of `Future` that implements `Runnable`, and so may be executed by an `Executor`. For example, the above construction with `submit` could be replaced by:

```
FutureTask<String> future = new FutureTask<>(task);
executor.execute(future);
```

Memory consistency effects: Actions taken by the asynchronous computation *happen-before* actions following the corresponding `Future.get()` in another thread.

Since:

1.5

See Also:

`FutureTask`, `Executor`

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method	Description
boolean	**cancel**(boolean mayInterruptIfRunnin	Attempts to cancel execution of this task.
V	**get**()	Waits if necessary for the computation to complete, and then retrieves its result.
V	**get**(long timeout, **TimeUnit** unit)	Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available.
boolean	**isCancelled**()	Returns `true` if this task was cancelled before it completed normally.
boolean	**isDone**()	Returns `true` if this task completed.

Method Details

cancel

```
boolean cancel(boolean mayInterruptIfRunning)
```

Attempts to cancel execution of this task. This method has no effect if the task is already completed or cancelled, or could not be cancelled for some other reason. Otherwise, if this task has not started when `cancel` is called, this task should never run. If the task has already started, then the `mayInterruptIfRunning` parameter determines whether the thread executing this task (when known by the implementation) is interrupted in an attempt to stop the task.

The return value from this method does not necessarily indicate whether the task is now cancelled; use `isCancelled()`.

Parameters:

`mayInterruptIfRunning` - `true` if the thread executing this task should be interrupted (if the thread is known to the implementation); otherwise, in-progress tasks are allowed to complete

Returns:

`false` if the task could not be cancelled, typically because it has already completed; `true` otherwise. If two or more threads cause a task to be cancelled, then at least one of them returns `true`. Implementations may provide stronger guarantees.

isCancelled

```
boolean isCancelled()
```

Returns `true` if this task was cancelled before it completed normally.

Returns:

`true` if this task was cancelled before it completed

isDone

```
boolean isDone()
```

Returns `true` if this task completed. Completion may be due to normal termination, an exception, or cancellation -- in all of these cases, this method will return `true`.

Returns:

`true` if this task completed

get

```
V get()
throws InterruptedException,
       ExecutionException
```

Waits if necessary for the computation to complete, and then retrieves its result.

Returns:

the computed result

Throws:

`CancellationException` - if the computation was cancelled

`ExecutionException` - if the computation threw an exception

`InterruptedException` - if the current thread was interrupted while waiting

get

```
V get(long timeout,
      TimeUnit unit)
throws InterruptedException,
       ExecutionException,
       TimeoutException
```

Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available.

Parameters:

`timeout` - the maximum time to wait

`unit` - the time unit of the timeout argument

Returns:

the computed result

Throws:

`CancellationException` - if the computation was cancelled

`ExecutionException` - if the computation threw an exception

`InterruptedException` - if the current thread was interrupted while waiting

`TimeoutException` - if the wait timed out
