# Abhi On Java





Monday, November 20, 2006

## Java 5 Concurrency: Callable and Future

Till Java 1.4, threads could be implemented by either implementing Runnable or extending Thread. This was quite simple, but had a serious limitation - They have a run method that cannot return values. In order to side-step this, most programmers use side-effects (writing to a file etc.) to mimic returning values to the invoker of the thread. Java 5 introduces the Callable interface, that allows users to return values from a thread. This post describes the Callable and Future interfaces and shows an example of how to use these to interfaces.

**Jump to Sample Code**

```
public interface Callable<V> {
V call() throws Exception;
}
```

The call() method is the entry point into a Callable object, and it's return type is the type parameter set in the Callable object. To implement Callable with no return value, use Callable<void>. Also, note that the **call() method throws a checked exception**, as compared to the run() method in Runnable which does not throw any exception. The *Executors class contains utility methods to convert from other common forms to Callable classes*. However, Callable cannot be used in place of a Runnable. *Callable objects have to be invoked by ExecutorService*. The Executor framework provides the Future interface to allow handling the cancellation and returns of a Callable object.
A Future represents the result of an asynchronous computation.

```
public interface Future {

//Attempts to cancel execution of this task.
boolean cancel(boolean mayInterruptIfRunning);

boolean isCancelled();

boolean isDone();

// Waits if necessary for the computation to complete,
//  and then retrieves its result.
V get() throws InterruptedException, ExecutionException;

// Waits if necessary for at most the given time for the computation
```

The result can be retrieved using method get() when the computation has completed, **blocking if necessary until it is ready**. *If you would like to use a Future for the sake of cancellation but not provide a usable result, you can declare types of the form Future<?> and return null as a result of the underlying task.* The following example demonstrates the use of Callable and future. The first CallableImpl class implements the Callable interface, and returns an integer that is sent to it's constructor. The CallableTester class invokes the CallableImpl through an executor.

```
public class CallableImpl implements Callable<Integer> {

private int myName;
CallableImpl(int i){
  myName = i;
}

public Integer call() {
  for(int i = 0; i < 10; i++) {
      System.out.println("Thread : " + getMyName() + " I is : " + i);
  }
  return new Integer(getMyName());

}
```

Java Search

Custom Search

Categories

ajax aop authentication batch charts code quality concurre hibernate how-to ide Java E javascript JAX-WS messaging ne patterns pdf persiste security servlet3 spring tips web services weblogic xml

Blog Archive

► 2017 (35)
► 2012 (1)
► 2011 (1)
► 2010 (2)
► 2009 (4)
► 2008 (12)
► 2007 (30)
▼ 2006 (138)
► December (26)
▼ November (30)
Struts: Paging and Sorting Displaytag
Pagination with DisplayTa
Google co-op search
Merge PDF files with iTex
PDF generation with iText
Scheduling with Quartz
Java 5 Concurrency
Java 5 Concurrency: Selec
Java 5 Concurrency: Selec Synchronizers
Java 5 Concurrency: Sync
Java 5 Concurrency: Conc
Java 5 Concurrency: Reac
Java 5 Concurrency: Lock
Java 5 Concurrency: Calla
Java 5 Executors: Threadl
Java 5: New features in C
Java: Handling Interrupts
Configuring LDAP in Webl
Weblogic Security
Weblogic: SSO with Windo
Opensourcing Java
Java EE application on Mu processor based ...
Struts 2: Action Redirects
Oracle: Transparent Data

CallableImpl.java

```
public class CallableTester {

public static void main(String[] args) {
  Callable<Integer> callable = new CallableImpl(2);

  ExecutorService executor = new ScheduledThreadPoolExecutor(5);
  Future<Integer> future = executor.submit(callable);

  try {
      System.out.println("Future value: " + future.get());
  } catch (Exception e) {
      e.printStackTrace();
  }
}
```

CallableTester.java

ExecutorService extends Executor to provides method to manage thread termination and methods that can produce a Future for tracking progress of one or more asynchronous tasks. The method submit extends Executor.execute(java.lang.Runnable) to create and return a Future. Methods invokeAny and invokeAll perform the most commonly useful forms of bulk execution, executing a collection of tasks and then waiting for at least one, or all, to complete. For an overview of Executors, visit Java 5 Executors.

Posted by Abhilash Vuyyuru at 9:19 AM

Labels: concurrency, example/sample code

## 2 comments:

**jasonspiro** Wednesday, March 21, 2007 at 7:15:00 PM EDT

Thank you for this useful blog post. One thing: you wrote "To implement Callable with no return value, use Callable<void>". I think you probably wanted to write

Callable<Void>

instead. That's Void with a capital letter V.

Reply

**Brian** Saturday, February 16, 2008 at 11:23:00 AM EST

Hi Abhi,

I love you blog. I have a question about shutting down redundant threads. Based on your blogs i have a main class that creates new threads like so..
..
...
ExecutorService executorService = Executors.newCachedThreadPool();
...
...
void startNewThread(){
if(condition){

//create a new thread
Callable callableHorseRaceThread = new HorseRaceThreadCallable(currentMarketID, currentExchangeID, sessionToken);

//start the thread
executorService.submit(callableHorseRaceThread);

}
}

This starNewThread() method is periodiocally called on a Timer. If the condition has changed..e.g a new race is available, it start a new thread. The thread that has been created is itself repetitively calling methods on a timer. e.g reading info from the internet.

How do I gracefully shutdown the created threads(Callable callableHorseRaceThread) when certains conditions have changed?

The Java 5 for each loop
Struts 2: Validation
Using Sitemesh with Strut
The Struts 2.0 Controller
Weblogic Split Directory E
Celebrate Eclipse's 5th Bi

- October (10)
- September (5)
- August (10)
- July (11)
- June (2)
- May (5)
- April (10)
- March (14)
- February (7)
- January (8)

2005 (8)

- Java: Using Immutable Class Concurrent Programming - 2
- Executing Single Java Sourc One Command - 2/14/2018
- 3 JVM Languages Java Devs in 2018 - 2/14/2018
- The Future of Open Source Is Bright... - 2/14/2018
- CDI (Part 3): Events and Obs Coupling and High Cohesion

- Article: Building a CI System Modules and Vert.x Microservices - 2/9/2018
- Article: JPA 2.2 Brings Some Anticipated Changes - 1/25/
- Article: InfoQ's Top Software Stories, Videos and Podcast 2017 - 12/30/2017
- Article: InfoQ's Top Software Stories, Videos and Podcast 2017 - 12/30/2017
- Article: Getting Started with in SpringBoot - 12/29/2017

Search This Blog

Report Abuse