Multi-threaded MASS Library

Table of Contents

Overview

The multi-threaded MASS library currently supports the Places class and implements Places.exchangeAll and Places.callAll. Multiple threads, one for each available processor core will be created automatically on computers running Linux and Mac OS X. If there are problems determining how many threads to create, only one thread will be created. The number of threads can be set explicitly by passing in to MASS.init(String[] args) the String "-c" or "-cores" immediately followed by a String containing the number of desired cores as part of or all of the array args.

The library is also commented to conform with the requirements of the javadoc tool.

Using the Multi-threaded MASS Library

The multi-threaded MASS library currently has three classes that are used by an application. These are MASS, Places, and Place.

The MASS Class

The MASS class currently supports the following methods that are appropriate for use in an application:

```
public static void init( String[] args )
public static void finish( )
public static Places getPlaces( int handle )
```

Other methods, although they may be public, should not be used by an application. They are used by other classes within the MASS package.

The init method is used to initialize the MASS environment and should be called before anything else is done involving the MASS environment. The args array is inspected for a string containing "-c" or a "-cores". If one of these strings is found, the next element in the array is converted to an integer and used as the number of threads that will run in the MASS environment. If no "-c" or "-cores" string is found, if there is a problem converting the next element to an integer, or if args is null, an attempt will be made to figure out how many processor cores are available. The number of processor cores can only be figured out if the computer is running Linux or Mac OS X. If successful at figuring out how many cores are available, init will create an environment with one thread for each core. If not successful at figuring out how many cores are available, init will create an environment with just one thread.

The finish method is used to shut down the MASS environment and should be called when there is no longer a need for the MASS environment. It is important to call finish so that all of the threads (other than the main thread) in the MASS environment will be properly terminated. In a MASS environment with more than one thread, failure to call finish will leave threads running after the main application has finished, causing the JVM to continue running as well.

The getPlaces method may be used to get a Places object that has already been created in the MASS environment. The handle is assigned when the Places object is created. The Places class is discussed below.

The Places Class

The Places class supports the following methods that are appropriate for use in an application:

```
public Places( int handle, String className, Object argument, int... size )
        throws Exception
public int getHandle( )
public int[] size()
public void callAll( int functionId )
public void callAll( int functionId, Object argument )
public Object[] callAll( int functionId, Object[] arguments )
public void exchangeAll( int handle, int functionId,  Vector<int[]> destinations )
```

The Places constructor has four parameters. The first, handle, is an integer that must be distinct from other handles chosen for other Places objects. If handle is not distinct, an exception will be thrown when the constructor is used. The parameter className is the name of the class that extends the Place class (discussed below), which will be used to populate this Places object. The parameter argument is passed in to the constructor for each object of type className. The parameter size is a series of integers that will be used as the dimensions of the Places object. If any of the elements of size is less than 1, an exception will be thrown.

The getHandle method returns the handle associated with the Places object.

The size method returns the size array used as the dimensions for the Places object.

The callAll method is used to perform a function call on each Place object in a Places object. Each of the callAll methods has a parameter functionId, which is the number corresponding to the method that will be called for each className object in the Places object. This method will be called through the callMethod method of the className object. (The set-up for the className object is discussed below with the Place class.) The version of the callAll method that takes the parameter argument will pass in argument on each call to the className object's callMethod method. The version of the callAll method that takes arguments (Object array) will pass in, on the call to each className object's callMethod method, the array element that corresponds to each className object's position in the Places object. This means that the arguments array must have as many elements as there are Place objects in the Places object. The returned array is populated in a corresponding manner, with the returned element from each call placed at the same index as the element from arguments that was passed in. Index positions in the parameter array and the returned array correspond to positions in the Places object as follows. The position at index 0 of the parameter and return arrays corresponds to the position in the Places object with dimension coordinates of all 0's. The last element in the return and parameter arrays corresponds to the position in the Places object with dimension coordinates of the highest index values for each dimension. The return and parameter arrays are traversed in sequence, while the Places object is traversed by incrementing the last element of each Place object's index and carrying towards element 0 of each Place object's index (see the Place class below for an explanation of the index for each Place object). For example, in a Places object with a size of { 3, 2, 4 }, the parameter and return arrays' element 0 corresponds to the Place object at coordinates { 0, 0, 0 }, 1 to { 0, 0, 1 }, 2 to { 0, 0, 2 }, 3 to { 0, 0, 3 }, 4 to { 0, 1, 0 } (carry towards element 0; i.e., leftwards), 5 to { 0, 1, 1 }, and so on until element 23 corresponds to { 2, 1, 3 } (the highest index values).

The exchangeAll method is used to populate the inMessages array of each Place object in a Places object with the results of method calls to other Place objects in the same Places object. The value of each Place object's outMessage instance variable is passed in as the parameter to the function call to each of the other Place objects. The other Place objects are determined by offsets, which are passed in as the

exchangeAll parameter destinations. Each element of destinations is an offset, which is an array with as many elements as there are dimensions in the Places object. As exchangeAll traverses the Places object, it arrives at each Place object and, using that Place as a base point, calculates the destination to which each offset array leads. Then a call is made to the Place object at each destination. The call is made through the destination Place object's callMethod method, with the exchangeAll parameter functionId used as the callMethod parameter functionId and the value of outMessage from the Place at the base point as the callMethod parameter argument. The return values from the call to each destination Place object's callMethod method are place in the inMesssages array of the Place object at the base point. The return values are placed into the inMessages array in the same order in which the corresponding offsets are found in the exchangeAll parameter destinations. If an offset leads to a destination that is outside the bounds of the Places object, null is placed in the corresponding position in the inMessages array.

The Place Class

The Place class is used only as a superclass. A subclass of Place is used when creating a Places object. When a Places class is instantiated, it is populated with objects of the subclass of Place, which is indicated by the Places constructor's parameter className. As the objects of type className are created, the Places constructor's parameter arguments is passed in to the constructor for the objects of type className. In addition, the Place instance variable size is filled with the size of the Places object in which it is created, and the Place instance variable index is filled with the coordinates corresponding to that particular Place object's unique location within the Places object. Each Place object's instance variables inMessages and outMessage (use described above with exchangeAll for Places class) are set to null when the Place is created.

The Place subclass that is created is expected to override the callMethod method of Place. The signature for this method is as follows:

```
public Object callMethod( int functionId, Object argument )
```

The values that can be used for functionId must somehow be made available to the application writer.

How the Multi-threaded MASS Library Works

Internally, five classes are used in the MASS environment. These classes are MASS, Mthread, Places, the nested class Places.Iterator, and Place.

The MASS Class and the Mthread Class

At the heart of the MASS environment is the MASS class. Through the init method, this class creates Mthreads, which extend the Thread class, and which are used to make the MASS environment a multi-threaded environment when more than one thread is called for. The Mthread contains very little code. It has one method, the run

method, which contains an endless while loop. The execution path through the loop is stopped and started by wait and notify calls on the MASS instance variable STATUS[], which has only one element. There are four values for STATUS[0]. These are as follows:

```
public static final int STATUS_READY = 0
public static final int STATUS_TERMINATE = 1
public static final int STATUS_CALLALL = 2
public static final int STATUS_EXCHANGE_ALL = 3
```

STATUS_READY is the initial value for STATUS[0] and is also the value to which STATUS[0] returns after performing an operation. STATUS_TERMINATE is used to close the MASS environment, which happens when a call is made to MASS.finish(). STATUS_CALLALL and STATUS_EXCHANGE_ALL are used when calls are made to Places.callAll and Places.exchangeAll, respectively.

Mthreads wait on STATUS whenever its value is STATUS_READY. When STATUS.notify is called, each Mthread checks the value of STATUS[0] and may make a call to a method in the MASS class.

Other general variables used by the MASS class are the following:

```
private static int[] threadsRunning = new int[1]
private static Vector<Thread> threads
private static Hashtable<Integer, Places> handles
private volatile static boolean INITIALIZED = false
private volatile static int barrierCounter = 0
```

The variable threadsRunning is used to keep track of the number of threads currently running. It is incremented in MASS.init and decremented in the MASS.recordThreadExit method. It is also used in the MASS.finish method.

The Vector threads is used to keep references to all of the threads in use in the MASS environment, including the main thread. It is populated in MASS.init and used in the MASS.getRange, MASS.getPosition, and MASS.barrier methods.

The Hashtable handles holds the handles of all the Place objects that have been created in the MASS environment. It is used in the MASS.init, MASS.getPlaces, and MASS.addPlaces methods.

The variable INITIALIZED is used to inidicate whether or not the MASS environment has been initialized or not. It is used in several methods of the MASS class. It is needed to prevent the MASS environment from being initialized more than one and to prevent Places objects from being created before the MASS environment has been initizlized. It is also used to prevent attempts by Places objects to perform operations in the MASS environment after the MASS environment has been closed.

This is necessary because Places objects are not destroyed when the MASS environment is closed.

The variable barrierCounter is used in the MASS.barrier method to keep track of how many threads have entered it.

Closing the MASS Environment

The MASS environment closes when a call is made to MASS.finish. When this is done, the main thread goes into MASS.finish and changes STATUS[0] to STATUS_TERMINATE and waits on threadsRunning. Upon finding a value of STATUS_TERMINATE in STATUS[0], each Mthread breaks out of its endless while loop, makes a call to MASS.recordThreadExit, and exits. In MASS.recordThreadExit, threadsRunning is decremented as each Mthread passes through. The last Mthread makes a call to threadsRunning.notify on its way out, which causes the main thread to wake up in MASS.finish. The main thread then sets INITIALIZED to false and exits. Setting INITIALIZED to false allows for the MASS environment to be initialized again, but it does not destroy any Places objects created in the MASS application.

ExchangeAll and CallAll

Calls to Places.exchangeAll and Places.callAll result in calls to MASS.ea_setup and MASS.ca_setup, respectively. There is one method named ea_setup and two named ca_setup. These methods are used to initialize variables in the MASS class that will be accessed by all the threads that will be used in the operation. The variables (and methods) used for exchangeAll and callAll are prefixed with "ea_" and "ca_", respectively. After the variables have been initialized, STATUS[0] is set to either STATUS_CALLALL or STATUS_EXCHANGE_ALL. This will cause the Mthreads to call either MASS.ea_exchangeAll or MASS.ca_callALL. The main thread makes the same call after returning from MASS.ea_setup or MASS.ca_setup.

Keeping the Threads Together

After each thread finishes its task in MASS.ea_exchangeAll or MASS.ca_callAll, it will call MASS.barrier. MASS.barrier is used to make sure all the threads have finished and do not move forward before STATUS[0] is reset to STATUS_READY. If an Mthread were to return to its while loop before STATUS[0] is reset to STATUS_READY, it would find STATUS[0] to still be set on STATUS_EXCHANGE_ALL or STATUS_CALLALL and then attempt to re-enter MASS.ea_exchangeAll or MASS.ca_callAll. The MASS variable barrierCount is used to count the threads as they enter MASS.barrier. If the main thread were to return to the main application code before the other threads had finished, it might initiate another MASS operation, opening the unwanted possibility of different threads performing different operations on the same Places object at the same time.

Dividing Work Among Multiple Threads

When threads enter MASS.ea_exchangeAll or MASS.ca_callAll, a call is made to MASS.getThreadPosition and then to MASS.getRange. MASS.getThreadPosition determines which position the current thread holds among however many threads

are operating in the MASS environment. MASS.getRange then uses this position to determine the index numbers of the first and last elements of a Places object over which the thread will operate. MASS.getRange returns an array of two numbers, corresponding to the first and last index numbers. These index numbers are then passed in to Places.iterator, which returns a Places.Iterator object that is used to traverse a section of the Places object.

The Places Class, the Places.Iterator Class, and the Places Class

The most important parts of the Places class are a one-dimensional Place array named holder and the integer array size. The variable holder contains all the Place objects that make up the Places object. The variable size contains the dimensions of the Places object. From the perspective of a MASS application, the Places class resembles a multidimensional array. Internally, however, it can be viewed as a one-dimensional array with some information and methods for converting index values back and forth between a linear format and a multi-dimensional format. To simplify operations, the instance variable length holds the number of elements in the Places object.

To make access to sections of the Places class easy, the Places class has a nested class called Iterator that implements the Iterator interface. This class is accessed through the two Places.iterator methods. One version of this method takes no parameters and produces an Iterator for traversing the entire Places object. The other takes two parameters, one for a starting index and a second for a final index, and produces an Iterator for traversing a portion of the Places object.

The Place class, always being introduced to the Places constructor by the name of a subclass, is created through reflection. Instance variables named klas and ctor hold the Class and Constructor objects, respectively, for the Place subclass. The variables klas and ctor are are only used in the Places constructor.

Converting Index Values

The following two methods are used to convert index values between array (multi-dimensional) format and linear format:

```
public static int indexArr2Num (int[] index, int[] size)
        throws ArrayIndexOutOfBoundsException
public static int[] indexNum2Arr( int index, int[] size )
        throws ArrayIndexOutOfBoundsException
```

As static methods, these methods can be viewed as utility methods.

Getting and Setting Place Objects

For getting and setting Place objects in the Places object, the following two methods are used:

```
public Place get( int index )
private void set ( int index, Place place )
```

```
    throws ArrayIndexOutOfBoundsException
```

The get method is public and is used by the MASS class. The set method is private, however, and is used only by the Places constructor.