StackContinuations

Patch name: continuation.patch

Background

This experimental implementation of continuation is an extension to the old version of callcc.patch (http://hg.openjdk.java.net/mlvm/mlvm/hotspot/rev/99e6c3830f6d). It is not based on the latest version of callcc.patch.

It

- implements a delimited continuation
- supports save/restore of compiled frames
- supports inter-thread save/restore
- is currently Linux/x86 and Linux/amd64 only
- is reasonably stable

This implementation was used in the work described by the JVM Language Summit 2010 talk titled "Continuation In Servers" (http://wiki.jvmlangsummit.com/Continuations_in_Servers) where the continuation feature is applied to increase the scalability of a synchronous I/O-based servers.

API

The current API looks like the following Javadoc style documentation.

sun/misc/Continuation.java

```
/**
 * The Continuation class. The API design is still in progress.
 */
class Continuation {
    /**
     * Marks the beginning of a new 'scope' in preparation for stack
     * save/resume.  Executes the given Runnable.
     *
     * @param data any user defined data to be passed from this call
     *             site to the point where {@link #resume} is called
     *             for convenience.
     * @return the Continuation object after the scope was saved
     *         into a Continuation object or null if it wasn't and
     *         simply returned
     */
    public static Object enter(Runnable r, Object data);

    /**
     * Copies the stack frames in the current scope, and stores them
     * in this object.  This method must be called in an enclosing
     * scope. Calling this method causes the stack frames in the
     * scope to suspend (including the current frame) and the enter
     * call at the entry of the current scope to return.
     *
     * @return the parameter passed to the resume call when the saved stack
     *         frames are resumed in the future.
     */
    public Object save();

    /**
     * Reactivates the stack frames saved in this object on the
     * current thread.  Overwrites the stack frames in the current
     * scope with the saved stack frames.  This method must be
     * called in an enclosing scope. Calling this method causes the
     * suspended save call to resume from the point where it was
     * suspended.
     *
     * @param rv the value to be returned from the resumed save call site.
     */
    public void resume(Object rv);
}
```

Basic operations

There are the three following methods in the Continuation API.

enter()

It executes a given Runnable. This marks the entry of a scope. In other words, a enter call indicates the bottom of a scope. A scope (of stack frames) is the unit of stack frames that the save method saves and that the resume method resumes. The two methods must be called inside a scope.

save()

Calling this method causes the stack frames in the current scope to be copied off of the current thread into a continuation object. At the same time, the current thread returns out of the scope (the enter call) without returning from the rest of the call chains in the scope.

resume()

Calling this method with a continuation object causes the current thread to abandon the call chains in the current scope and overwrite them with the ones from the saved scope. The current thread then resumes the execution of the saved call chains (scope) as if it is right after the save call at the time the scope was saved. A continuation is resumable only once.

Basic examples

A simple save example

An example of saving a continuation looks like the following:

```
 1  void test() {
 2    final Continuation c =
          new Continuation();            // First, create the continuation to store stack frames into.
 3    Runnable s = new Runnable() {      // Defines the scope for a save. This Runnable will executed in line 9.
 4      public void run() {
 5        c.save();                      // Saves the one-frame scope with the stack frame for the run() call
into c.
 6        System.out.println(
              "Hello JVM Continuation"); // Prints a message.
 7      }
 8    };
 9    Continuation.enter(s, null);       // Enter the scope. The above run() executes.
                                         // Line 5 causes the enter() to return the run() without executing
line 6.
10  }
```

The above program does not do much. It does not print the string in line 6. It only saves the continuation.

A simple resume example

An example of resuming a continuation looks like the following:

```
 1  void test(Continuation c) {          // Suppose a continuation was saved somewhere else and is passed here.
 2    Runnable s = new Runnable() {      // Defines the scope for a resume. This Runnable will executed in line
7.
 3      public void run() {
 4        c.resume(null);                // Resumes the given continuation.
                                         // From this point on, the current thread executes the saved
continuation.
                                         // Note the rest of the run() is unreachable.
 5      }
 6    };
 7    Continuation.enter(s, null);       // Enter the scope. The above run() executes.
                                         // When the saved continuation finishes/returns, control comes back
here.
 8  }
```

A simple save-and-resume example

A basic example of saving and resuming a continuation looks like the following:

```
 1  void test() {
 2    final Continuation c =
          new Continuation();           // First, create the continuation to store stack frames into.
 3    Runnable s = new Runnable() {     // Defines a scope for a save. This Runnable will be executed in line
9.
 4      public void run() {
 5        c.save();                     // Saves the one-frame scope with the stack frame for the run() call
into c.
 6        System.out.println(
              "Hello JVM Continuation"); // Prints a message.
 7      }
 8    };
 9    Continuation.enter(s, null);      // Enters the scope. The above run() executes.
                                        // Line 5 causes the enter() to return without the run() executing
line 6.
10    Runnable r = new Runnable() {     // Defines a scope for a resume. This Runnable will be executed in
line 16.
12      public void run() {
13        c.resume(null)                // Resumes the continuation (the stack frame of run()) from line 5
                                        // and the execution at line 13 is thrown away.
                                        // From this point on, the current thread executes from line 6 above
                                        // and prints "Hello JVM Continuation".
14      }
15    };
16    Continuation.enter(r, null);      // After line 6 was executed, it returns back here as if this enter()
method
                                        // had executed Runnable s instead of r.
17  }
```

Advanced examples

See the continuation test files bundled with the patch. Or, go to http://cr.openjdk.java.net/~hiroshi/webrevs/continuation-jdk/.

Limitations

It does not support monitors held or native stack frames inside a continuation scope at the time of a save call (if there is one, an exception will be thrown).
It does not currently work with compressed oops in the 64 bit build.

Work in progress