EOF-Using EOF-Context and Database Locking

Contents

Overview

Errors in locking your editing context (shared editing contexts or not) will cause your instance to deadlock. This is one of the most vexing problems.

Do you have to lock & unlock the editing context only when you are using a stateless section (directActions) of your application or do you have to lock & unlock it in state as well? What about the defaultEditingContext, does that need locking & unlocking too?

If your sessions only use the default editing context, you don't worry about locking.

Also, manually locking/unlocking objects is error prone. And the resulting errors are the most difficult to fix since they only occur randomly on heavy load. I really feel that WebObjects should use Java synchronization to ensure safe access without requiring manual coding. But this would probably mean a considerable rework of WebObjects

I used to think along the same lines, but it turns out this would introduce one hell-uv-a-lot of overhead. You'd have a performance situation not unlike the original Java collections (BAD). If you can acquire one lock and then plow through EC and EO API at will until unlock, you really save some cycles. I'm sure there are others more enlightened here that can elaborate or correct me here; but this is my impression.

BTW: I tend to create extra ECs at the Session level and do the locking and unlocking in the Session awake and sleep. The request handler prevents more than one request from getting access to a Session at a time; and you are guaranteed that Session awake and sleep trigger just once per request cycle. That's not true of component awake and sleep calls. You have a greater potential for a deadlock if you do the lock/unlock at the component level:

Transaction starts on one page (lock), then instead of returning that page after action, you create another page (potential lock again, deadlock).

The only tricky thing I've run into is if your app uses Long Running Request Page. That's a case where you get multiple request/responses as viewed by the session over one logical "request". That is, the status page will be reloading, but your background job is busy doing its thing continuously. If it's busy using a EC that is locking and unlocking at the session awake/sleep you wouldn't get very far without complaints. So in these cases, if a user enters such a page AND is using a EC, I disable the automatic session lock/unlock for that EC until the job is over. Or I create a EC used strictly for the background job.

David LeBer

In general you are better to lock EC's in the session (if you've got one) rather than the component.

As that is the usual approach, there are a couple of solutions to implement that for you:

- MultiECLockManager
- Project Wonder's ERXEC

Jonathan Rochkind

Note: One very handy (if I do say so myself) reusable solution for locking ECs you create (not neccesary for session default EC) can be found at: http://WOCode.com/cgi-bin/WebObjects/WOCode.woa/wa/ShareCodeItem?itemId=301

Anjo Krank

One thing to watch out for is that EC locking on the page's awake() and sleep() doesn't really work, because awake may be called more often than sleep. This means that normally you can't use DirectToWeb because it does just that. However, ProjectWonder has an editing context factory, an automatically locking and unlocking EC subclass and a lock manager that will relaese all locks when the application sleeps.

This works nicely with long response pages, provided that you don't use the session's editing context in the long running task.

Locking in a DirectAction

If i call session().defaultEditingContext() from within a DirectAction, I'm currently defensively locking it. Do I need to, or does it follow the same (automatic) locking rules as in a normal WOComponent?

Anjo Krank

You don't need to lock it.

But anyway, doing sth like:

```
public WOActionResults someAction() {
  WOActionResults page = pageWithName("Foo");
  session().defaultEditingContext().lock;
  try {
  ...
  } finally {
      session().defaultEditingContext().unlock;
  }
  return page;
}
```

isn't sufficient, as the appendToResponse phase only occurs **after** the page is returned and faults could still get fired there... so when you create an EC there instead of using the session EC, you still have to lock it in the component.

Chuck Hill

Here is another way to handle this:

```
public WOActionResults someAction() {
  WOComponent page = pageWithName("Foo");
  WOResponse response;
  session().defaultEditingContext().lock;
  try {
  ...
      response = page.generateResponse().
  } finally {
      session().defaultEditingContext().unlock;
  }
  return response;
}
```

The call to generateResponse before the unlock ensures that appendToResponse is called while the EC is locked and that all the faults are fired in a safe state.

Robert Walker

I generally avoid using the session's default editing context within direct actions. A call to session().defaultEditingContext() will create a session, even if you don't actually need one. It can be more efficient to use a new editing context and lock it as necessary, since this will not cause unnecessary sessions to be created.

Tracking EOEditingContext Locks

Project WOnder's ERXEC provides extensive logging facilities for tracking EOEditingContext locks. If you are not interested in using Project WOnder, there are several other ways:

You can display the stack trace of editingcontext locking warnings with the Launch Argument:

```
  -NSDebugLevel 2 -NSDebugGroups 18
```

This will give you some information but there is a better way. You need to sub-class EOEditingContext like this:

LockErrorScreamerEditingContext

```
// LockErrorScreamerEditingContext.java
//
// Copyright (c) 2002-2003 Red Shed Software. All rights reserved.
// by Jonathan 'Wolf' Rentzsch (jon at redshed dot net)
// enhanced by Anthony Ingraldi (a.m.ingraldi at larc.nasa.gov)
//
// Thu Mar 28 2002 wolf: Created.
// Thu Apr 04 2002 wolf: Made NSRecursiveLock-aware by Anthony.
// Thu Jun 22 2003 wolf: Made finalizer-aware. Thanks to Chuck Hill.

import com.webobjects.eocontrol.*;
import com.webobjects.foundation.*;
import java.io.StringWriter;
import java.io.PrintWriter;

public class LockErrorScreamerEditingContext extends EOEditingContext {
   private String _nameOfLockingThread = null;
   private NSMutableArray _stackTraces = new NSMutableArray();

   public LockErrorScreamerEditingContext() {
       super();
   }

   public LockErrorScreamerEditingContext(EOObjectStore parent) {
       super(parent);
   }

   public void lock() {
       String nameOfCurrentThread = Thread.currentThread().getName();
       if (_stackTraces.count() == 0) {
           _stackTraces.addObject(_trace());
           _nameOfLockingThread = nameOfCurrentThread;
           //NSLog.err.appendln("+++ Lock number (" +  _stackTraces.count() + ") in " + nameOfCurrentThread);
       } else {
           if (nameOfCurrentThread.equals(_nameOfLockingThread)) {
               _stackTraces.addObject(_trace());
               //NSLog.err.appendln("+++ Lock number (" + _stackTraces.count() + ") in " + nameOfCurrentThread);
           } else {
               NSLog.err.appendln("!!! Attempting to lock editing context from " + nameOfCurrentThread
                                  + " that was previously locked in " + _nameOfLockingThread);
               NSLog.err.appendln("!!! Current stack trace: \n" + _trace());
               NSLog.err.appendln("!!! Stack trace for most recent lock: \n" + _stackTraces.lastObject());
           }
       }
       super.lock();
   }

   public void unlock() {
       super.unlock();
       if (_stackTraces.count() > 0)
           _stackTraces.removeLastObject();
       if (_stackTraces.count() == 0)
           _nameOfLockingThread = null;
       String nameOfCurrentThread = Thread.currentThread().getName();
       //NSLog.err.appendln("--- Unlocked in " +  nameOfCurrentThread + " (" + _stackTraces.count() + "
remaining)");
   }

   public void goodbye() {
       if (_stackTraces.count() != 0) {
           NSLog.err.appendln("!!! editing context being disposed with " + _stackTraces.count() + " locks.");
           NSLog.err.appendln("!!! Most recently locked by: \n"
                              + _stackTraces.lastObject());
       }
   }

   public void dispose() {
       goodbye();
       super.dispose();
   }
```

```
   protected void finalize() throws Throwable {
      try {
          goodbye();
      } finally {
          super.finalize();
      }
   }

   private String _trace() {
       StringWriter stringWriter = new StringWriter();
       PrintWriter printWriter = new PrintWriter(stringWriter);
       (new Throwable()).printStackTrace(printWriter);
       return stringWriter.toString();
   }
}
```

Now you'll have to get the subclass substituted whenever an EOEditingContext is created. Too bad there's no generalized EOEditingContext factory method in WOF, but this will cover default editing contexts:

```
public class Session extends WOSession {
   public Session() {
           super();
           setDefaultEditingContext( new LockErrorScreamerEditingContext() );
   }

   public Session( String sessionID ) {
           super( sessionID );
           setDefaultEditingContext( new LockErrorScreamerEditingContext() );
   }

 ...
}
```

EOObjectStoreCoordinator locking

Most developers do not need to concern themselves with EOObjectStoreCoordinator locking
but for those who are doing more than the usual EOF manipulation, it is advised to lock EOObjectStoreCoordinator in the following situations:

- When using methods of EOObjectStoreCoordinator directly
- When using methods of EOEntity, for example see ERXFetchSpecificationBatchIterator constructor, entity.schemabasedQualifier(...)

Note that locking an EOObjectStoreCoordinator will in turn lock each of its registered
EOCooperatingObjectStores, which are typically instances of the concrete EODatabaseContext
class.

EODatabaseContext locking

Most developers do not need to concern themselves with EODatabaseContext locking
but for those who are doing more than the usual EOF manipulation, it is advised to lock EODatabaseContext in the following situations:

- When using methods of EODatabaseContext directly
- When using EOAdaptor
- When using EOAdaptorChannel
- When using EODatabase
- When using ERXSQLHelper

EOF Core Classes Overview

This is a useful diagram with overview of core classes and how they are related.