EOF-Using EOF-EOSharedEditingContext

Overview

EOSharedEditingContext, which should more appropriately be named EOReadOnlyEditingContext (or EOReadMostlyEditingContext), provides an editing context that can be used across your entire application for keeping globally shared EO's in memory. While it is possible to make changes to objects in a shared editing context, it is notoriously difficult to do so without creating problems such as a deadlocks. The best type of objects to put in a shared editing context are those that change very infrequently like enumerated type objects, for instance.

Performance Implications

There is a performance hit on your Application when EOSharedEditingContexts are enabled. If your application does not use them, you should call:

```
EOSharedEditingContext.setDefaultSharedEditingContext(null);
```

in your Application constructor. This will disable shared editing contexts completely, and you will no longer see any performance loss.

Art Isbell

If you haven't done so, consider reading Apple's Documentation.

From the doc, it says 'Since shared editing contexts listen for ObjectsChangedInStoreNotifications, the shared editing context updates when it learns that an object was modified'. Does that also mean if the database records changed by other applications, the shared editing context will get the notification too.

No.

If it is not, then how the application can detect something has been changed in external data store?

Only when a change has been made in the same process will a shared editing context and all other editing contexts in that process as well be notified. This is no different from the behavior of all editing contexts.

Does the SharedEditingContext only work with the fetchSpecifications which are pointed by either 'Share all objects' or 'Share objects fetched with' options in Shared Object Inspector OR it can work with any regular fetchSpecifications?

A shared editing context can use any fetch specification programmatically, but those entities designated in an eomodel to "Share all objects" or "Share objects fetched with" will be fetched automatically into the default shared editing context.

After shared EO objects are loaded into sharedEditingContext at Application level, how can we retrieve the data that is already in sharedEditingContext at each page ?

The EOSharedEditingContext JavaDoc states:

"Objects can be fetched into a shared context using objectsWithFetchSpecification and bindObjectsWithFetchSpecification. The latter method makes it easier to access result sets, using objectsByEntityNameAndFetchSpecificationName."

objectsByEntityName() can also be used. Or you can store them in Application instance variables accessed by public Application methods. If you do so, a resource is being shared among sessions, so accessor methods should be synchronized to make access thread-safe.

Is that 'new EOEditingContext().sharedEditingContext()' is same as 'EOSharedEditingContext.defaultSharedEditingContext()'? If it is not, how can we get the same object of EOSharedEditingContext as in the application level, which will be used to get objects in each session or in each page?

The shared editing context of all editing contexts is the default shared editing context. editingContext.sharedEditingContext() and EOSharedEditingContext.defaultSharedEditingContext() will return the same shared editing context unless you have explicitly set an editingContext's shared editing context to a different shared editing context.

Does anybody find any memory issue with using EOSharedEditingContext?

A shared editing context can use more memory if you fetch objects into it that aren't needed by your app. But if you're careful to fetch only those objects likely to be used by your app, the instantaneous memory usage might be less than that when no shared editing context were used because the same objects won't be fetched into multiple editing contexts. The average memory usage over the life of the process might be similar with a shared editing context because shared objects always exist. So I don't see memory usage as a shared object advantage or disadvantage, but I've never done a comparison.

Shared editing context advantages are reduced database access and no need to create local instances of shared objects in other editing contexts.

Only those objects that will never be source objects in relationships with unshared destination objects can be shared. Shared objects should be read-mostly as well because updating them requires a special procedure.

Jonathan Rochkind

[I personally have always been scared of using EOSharedEditingContexts, because there seemed such potential for problems. They are complex in how they work, not entirely well documented, and I suspected there would be Apple bugs.

Robert A Decker

However, one user kindly provided his own guidelines he uses with EOSharedEditingContexts, and which he says results in no problems at all. I include his guidelines here.

I haven't had problems, yet, with EOSharedEditingContext and objects that are changed rarely, as long as I follow some rules:

- NEVER fetch using the shared editing context yourself. Do all access to shared objects through the shared editing context's objectsByEntityNameAndFetchSpecificationName or objectsByEntityName methods, which return dictionaries containing the shared objects being held by the shared editing context. Repeat, DO NOT FETCH using the shared editing context.
- When making changes to objects or deleting, you have to do it in an editing context without a sharedEC. I always make a new ec and set its sharedEc to null, then copy the object I'm going to edit into it using EOUtilities.localInstanceOfObject
- When adding objects you have to tell the sharedEc to refresh its array of objects the type you just added. You use the bindObjectsWithFetchSpecificationName.

Again, do not fetch using the shared ec. Using this simple rule I haven't had any problems with objects that are changed rarely while the app is running. I've deleted and updated up to thousands of objects in separate threads using these rules without problems.

Robert A Decker

and on how the objects get in the shared ec in the first place, rob writes:

For each of my shared entities, I add a fetch spec named 'FetchAll' and in the EOModeler inspector I set that fetch spec to be the fetch that holds the shared instances. (click on entity, look at inspector third button, click on 'Shared objects fetched with' radio button for the FetchAll fetch specification).

You can keep your other fetch specs on those shared entities arountd, but you shouldn't really use them like you normally would - instead, only use them on doing in-memory fetching and sorting of the NSArrays you get back from the shared entity.

Then, in Application I have two methods:

```
public static void refreshSharedEntity(String entityName) throws
MarvinGeneralException {
   Application.refreshSharedEntityForFetchSpecificationName(entityName,
"FetchAll");
}

public static void refreshSharedEntityForFetchSpecificationName(String
entityName, String fetchSpecName)
        throws MarvinGeneralException {
  EOModelGroup modelGroup = EOModelGroup.defaultGroup();
  EOSharedEditingContext sharedEC =
  EOSharedEditingContext.defaultSharedEditingContext();
  EOFetchSpecification fs =
modelGroup.fetchSpecificationNamed(fetchSpecName, entityName);

  if (fs == null) {
    throw new MarvinGeneralException(Application.class.getName() +
".refreshSharedEntityForFetchSpecificationName",
            entityName + "." + fetchSpecName + " doesn't exist");
  } else {
    sharedEC.bindObjectsWithFetchSpecification(fs, fetchSpecName);
  }
}
```

I use these methods for updating the shared editing context for a specific entity.

As an example for an entity, I have the methods:

```
public static NSArray enclosureMasters() {
  NSDictionary objectsByEntityName =
(NSDictionary)EOSharedEditingContext.defaultSharedEditingContext()
          .objectsByEntityNameAndFetchSpecificationName();
  NSDictionary objectsForEnclosureMaster = (NSDictionary)
objectsByEntityName.objectForKey("EnclosureMaster");
  NSArray enclosureMasters = (NSArray)
objectsForEnclosureMaster.objectForKey("FetchAll");
  return enclosureMasters;
}

public static NSArray enclosureMasters(String fetchSpecName, NSDictionary
bindings) {
  // filter in memory against all enclosure masters
  // EOQualifier.filteredArrayWithQualifier( NSArray objects, EOQualifier
aQualifier)

  if (bindings == null) bindings = new NSDictionary();
  EOFetchSpecification fetchSpec =
EOFetchSpecification.fetchSpecificationNamed(fetchSpecName,
"EnclosureMaster");
  EOQualifier boundQualifier =
fetchSpec.qualifier().qualifierWithBindings(bindings,
          fetchSpec.requiresAllQualifierBindingVariables());
  if (boundQualifier == null) return
  EnclosureMaster.enclosureMasters(); // return all -
             // otherwise the next call will give us an empty array
  NSArray results =
EOQualifier.filteredArrayWithQualifier(EnclosureMaster.enclosureMasters(),
boundQualifier);
  // we should now use the fetchSpec's sort orderings to sort the results
  NSArray sortedResults = results;
  NSArray sortOrderings = fetchSpec.sortOrderings();
  if (sortOrderings != null)
    sortedResults = EOSortOrdering.sortedArrayUsingKeyOrderArray(results,
sortOrderings);
  return sortedResults;
}
```

These two methods let me fetch all objects of a certain type and, even better, let me use my other fetch specs on these objects.

All other static methods I would put on EnclosureMaster will go through these two methods. This way I never do any fetches myself and instead just use the objects that are already in memory and held by eosharededitingcontext.

Also, when the app launches it will automaticalyl fetch the shared objects using the FetchAll fetch spec that you set earlier.

When you addd a new object you have to reload by calling, for example,

```
Application.refreshSharedEntity("EnclosureMaster";);
```

When you edit or delete a shared entity the SharedEditing context will take care of updating the objects it's holding.