EOF-Using EOF-Concurrency

Using ThreadLocals for Accessing Multiple EOObjectStoreCoordinators

Brendan Duddridge

We're trying to fix some of our editing context issues we've been having and we're thinking of creating an architecture like the following:

Note first... we have WOAllowsConcurrentRequestHandling = true

- Create a ThreadLocal to lazily instantiate an EOEditingContext with it's own new EOObjectStoreCoordinator
- When a direct action request comes in, ask the ThreadLocal for it's editing context in performActionNamed.
- Lock the ThreadLocal's editingContext in a try block.
- Process the request and call generateResponse().
- unlock the ThreadLocal's editing context in the finally block.
- Return the response.

So what we'll end up with is a separate connection to the database for each worker thread. So we could potentially have many connections to the database. As worker threads are finalized, we'll close the connection to the database. We may end up using the JavaPoolingJDBCAdaptor to limit the connections to the database. But this would provide for good multi-threaded concurrent access to our application.

Does anyone foresee any undesired issues with this approach? I've never seen this approach postulated in the WO dev archives. One of my developers suggested this approach and I thought it sounded good, so I'd see what others thought first before implementing it. We'd be happy to share our findings and code if it works out well.

David LaBer

I've worked on an application architected this way, and was not happy. Just so you know where I'm coming from 🙂

A couple of thoughts:

- You will have multiple snapshots - one for each of your EOObjectStoreCoordinators. They will not be synchronized. So, you will loose optimistic locking and will have to either implement some kind of inter-stack notification or ensure fresh data by fetching often (sacrificing caching).
- You will complicate your Application architecture making it harder to maintain, harder to understand, and harder to hand off to other WebObjects developers.
- Do you really require multiple concurrent connections to your database for all of your queries? I can understand doing this for queries that profiling has identified as being bottlenecks. However, I'd probably look at other optimizations if possible (ie: shifting as much load as possible to SQL server).
- Multiple instances (although more costly in hardware) will be more cleaner and more maintainable (even **with** implementing a change notification architecture).

As always these opinions are my own, and are open to discussion 🙂

Chuck Hill

Locking-wise, I think that will be fine. Memory usage is the most likely issue that you will run into. The extra EOF stacks will consume more memory than a single stack implementation. I'm assuming that this is OK for your purposes, that the requests will not be fetching in a lot of data. If they do fetch a lot of data, or you do end up with many worker threads, you may find that you apps runs out of memory rather quickly. You might want to consider some sort of round robin usage of a set of stacks stored at the app level. This will at least somewhat limit memory usage. You may also get more optimistic locking failures at the database like this. If you are running multiple instances (which I recall is the case) you have to handle these anyway so that becomes a non-issue.