Development-WO Session

Overview

A WOSession provides the encapsulation of all the state that is associated with a user's use of your application. Each WebObjects application has its own subclass of WOSession, which is a significant difference between session tracking in WebObjects and J2EE. In J2EE, HttpSession is not subclassed, and all state is stored as attributes in HttpSession's Map. In WebObjects, sessions are much richer, providing the ability to take advantage of all the benefits of being a full-blown class (typed fields, methods, etc). WOSessions have a timeout time, which by default is configured in the WOApplication configuration. This timeout time determines how long the server will keep the session alive without any active requests. When the session times out, it will be garbage collected.

Accessing the Session

```
Session session = (Session)session();
```

Note that this works only in WOComponent or WODirectAction subclasses. If you want to access the session from elsewhere, pick an answer from this list:

- Your design is wrong; don't access the session from that class.
- Create and maintain an index of which thread is dealing with which session, then you can always find the current session from anywhere.
- Run single-threaded only, and maintain a store of which session is current.
- Pass the session in from the calling component or direct action. But, see the first item.

Just one more word of caution do not set the session as a class variable of WOComponent or WODirectAction, this would create a loop and prevent the garbage collection of old sessions.

Is the above really true? In a WO5 pure-Java application, circular references like this might create more work for the garbage collector, but shouldn't prevent anything from getting garbage collected. The Java garbage collector is supposed to handle circular references just fine. - JonathanRochkind

If you like to simplify your code just insert the following method in your component (this will save you some typing):

```
public Session Session() {
  return (Session)this.session();
}
```

We always begin our projects with a WOComponent subclass that we use as a superclass for all the rest of our components. We include this method as well as a few other useful utility methods. - JoshuaMarker

As I recall the warnings about storing references to Session were specific to inner classes of Session. Storing references to Session inside another component is perfectly ok, although an accessor method is better. - BrianMarquis

I still think even this is not a problem in WO5, although it probably is in WO4.5. JRE 1.3.1 is supposed to garbage collect circular references just fine, whether involving an inner class or not. But maybe the JRE doesn't do what it's supposed to? - JonathanRochkind

Session Tracking

The first time a session is requested for a request, WebObjects ensures that all subsequent requests from the same user will use the same WOSession instance. WebObjects achieves this session tracking with one of two techniques: URL rewriting, cookies, or query strings.

With URL rewriting, once the session is initialized, generated URLs will include the session's ID in the URL path. For instance, if you use a WOHyperlink, the generated URL will look something like "http://hostname.com/cgi-bin/WebObjects/AppName.woa/1/wo/EMVnGH1g8VLOW2TRMm3Ptg/6.0.19.15.1.3.1". The "EMVnGH1g8VLOW2TRMm3Ptg" in this example is the uniquely generated session ID. All requests that come in with this session ID in the URL will be attached to the same WOSession instance.

Cookies work in a similar fashion, except that instead of rewriting URLs to pass the session ID, the session ID is passed in a cookie called "wosid".

Lastly, you can pass a query string attribute named "wosid" that contains the session ID. This is often used when calling DirectActions that need access to the user's session.

Performance Implications

Session creation and persistence must be carefully monitored in large deployments due to the memory usage required to keep the session alive. It is generally recommended to minimize session creation whenever possible. When using "normal" WebObjects techniques, this can be difficult. The default behavior of WOHyperlink, for instance, when calling an action on your WOComponents is to cause sessions to be generated. The typical method for avoiding session creation is to implement your application using DirectActions. DirectActions do not require a page cache to be created for page state preservation, and thus can be used in a completely stateless way. The downside of using only DirectActions is that you lose the benefits of a more stateful approach. For instance, when using DirectActions only, you will find that you have to manually wire up your state objects by interpreting query string variables.

You can check for the existence of a session in a DirectAction by calling the existingSession() method. If a session exists, it will be returned. If a session has not yet been created, then this method will return null.

Adjusting your session timeout can have a large impact on performance. The longer a session timeout is, the longer the session will consume memory. Under heavy load, "dead" sessions can build up, so the timeout time should be tuned to be as low as possible without negatively affecting your user's experience with your application.

Hanging Sessions

There are certain cases in WebObjects where thrown exceptions can cause a session to not be checked back into the WOSessionStore, which will deadlock the session. One of the more common causes of this is an exception thrown from a DirectAction that uses a session. If you use Project Wonder's ERXApplication, ERXSession, etc, it provides a prevention mechanism for this problem. However, if you are using a default WebObjects installation without Project Wonder, you will need to deal with this problem yourself. You should not let your DirectActions throw an exception in this case.

Checking for Existing Session

If you are in a DirectAction method when trying to do this, check the following WOAction method (WOAction is WODirectAction's superclass):

```
public WOSession existingSession()
```

Restores the session based on the request. If the request did not have a session ID or the session ID referred to a non-existent session, then this method returns null. To determine if a session failed to restore, check the request's session ID to see if it non-null and if so, call this method to check its result.

Returns:
a WOSession object which represents periods during which access to a WebObjects application and its resources is granted to a particular client

If you're not in a DirectAction method, look at WOContext's method:

```
public boolean hasSession()
```

Returns true if a session exists for the receiving context, false otherwise.