Build Highly Scalable Oracle ADF Applications

Eugene Fedorenko
eProseed NL
The Netherlands

Keywords:

ADF, Java Heap, Garbage Collector, Lifespan, Memory scopes

Introduction

A scalable application is able to keep working well and increase its total output under an increased load when resources such as memory and CPU are added.

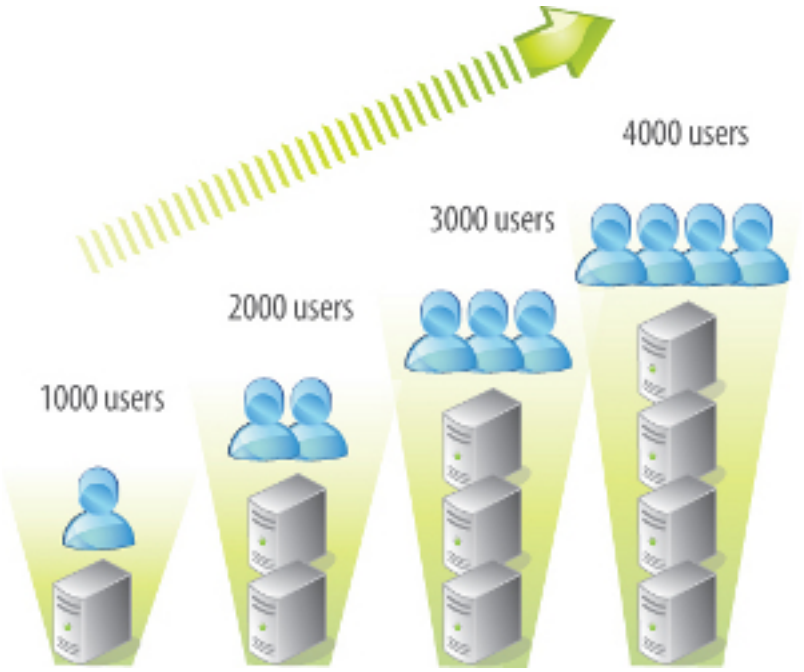

Illustration. 1: Scalable application

When we build real-life OLTP systems with thousands of simultaneously working users the memory footprint could become a bottleneck, eliminating application scalability. It is essentially important to know how to develop a healthy Java EE application and and how to keep objects in memory as long as it is really needed.

Heap

Whenever a Java process starts, it gets some memory from the Operating System. Part of this memory, JVM uses to allocate objects. This is the Heap. The objects, allocated inside the heap, can be large and small. The objects can be referenced by the program and they can have references to each other. Java references can be hard, soft or weak. A hard reference means that we are working with the object right now and we need to keep it alive for a while. A soft reference means that we are working with the object but we can survive without it in case of lack of memory and recreate it later. A weak reference means that it is cool to have the object alive but no problem at all if it has gone. For example entity cache in ADF BC is implemented as a map of weak references, on the other hand any regular reference to an object in our Java code is a hard reference. Actually, there is one more type of references, which is called phantom.

Garbage Collection

Whenever the entire heap space gets full or nearly full, Java Virtual Machine runs some special process in order to cleanup the heap or in other words, to collect the garbage. All modern JVMs implement Mark and Sweep garbage collection model. That means that there are two phases for performing the garbage collection of the whole heap.

During the Mark phase, the collector looks for objects that are still in use and cannot be released. So, objects reachable by hard references, and by soft references are going to be considered alive. All other objects, I mean reachable by weak references and unreachable at all, are going to be considered garbage. If there is lack of free space in the heap, some softly reachable objects can be considered garbage as well according to the LRU algorithm.

During the second phase, Sweep, all gaps between alive objects are going be recorded in a so-called free list and they are going to be available for further allocations. The Garbage collector prevents heap fragmentation performing a compaction of some portion of the heap.

Oracle ADF objects lifespan

If we were working with plain Java code, being just regular Java developer, that would not be that complicated to control the lifespan of objects created within the application. Of course we would have to think about that and keep that in mind but everything would be under our control and we'd be free to manipulate objects as we wish.
But as we are working with Oracle ADF we have to deal with lots of objects created and provided by the framework. Apparently, in order to develop healthy Java applications with Oracle ADF, we need to understand the life cycle of core building blocks of the framework.

JSF lifecycle vs Oracle ADF page lifecycle

The following diagram represents JSF and ADF lifecycles:

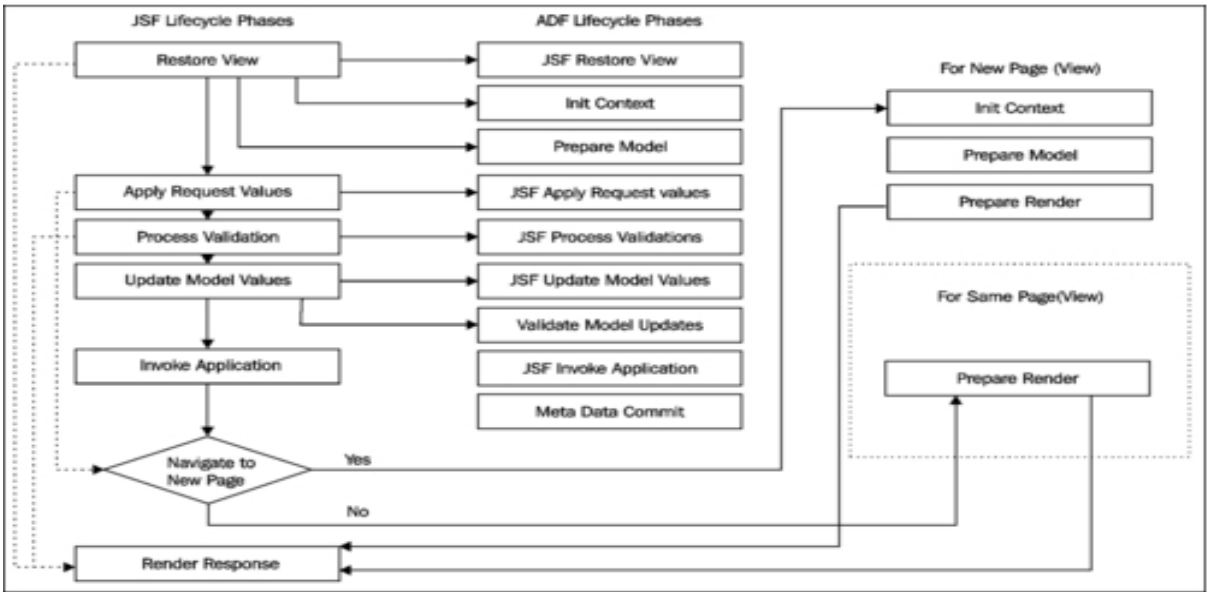

Illustration. 2: JSF and ADF lifecycles.

JSF lifecycle phases are good and simple. But ADF is a bit more than JSF and it provides its own extended version of the lifecycle. Actually ADF extends standard JSF lifecycle implementation class and provides ADF specific phase listener which is notified with before and after phase events. There are two type of the request. The initial request and postback. During the initial request the lifecycle is pretty short. Right after the Restore View phase we are jumping to the render response phase skipping the rest of lifecycle phases.

Bindings in Oracle ADF lifecycle

When a user requests a page from a browser the application server performs some pre-processing of the request with a chain of servlet filters. Among others in that chain there is ADF Binding Filter. When the request reaches ADF Binding Filter filter, it looks for the Binding Context in the current session and if it is not present it creates the Binding Context. The Binding Context as a representation of DataBinding.cpx file contains mapping between pages and pagedef files and it contains a list of data controls used by the application. Having all that the framework investigates what binding containers and data controls are going to participate in the request. The framework finds or creates an instance of each datacontrol participating in the request and invokes its beginRequest method in order to get it ready for action.

Oracle ADF BC lifespan

As the Binding Context is initiated the control is returned to ADF Faces rich client framework that takes over the processing thereafter and starts the page lifecycle. The framework acquires an instance of the application module from the application module pool when it is referenced within the request for the first time. This happens during the render response phase when the data bound UI elements require to read data. When components ask for data, the framework will evaluate the associated

binding EL expression. The EL evaluator used by ADF delegates the call to the binding layer classes which will check out the appropriate application module instance from the pool for use.

View Objects lifespan

For a data bound web page, the execution of a view object happens when it is referenced for the first time in page lifecycle. Once the query is executed, the framework will populate the default row set of the view object with the rows in the result set returned by the JDBC call. While doing so, if the view object is backed up by entity objects, the framework will generate appropriate entity instances to back up the view rows. The newly created entity objects will be added to the appropriate entity cache based on the entity definition type. At runtime, each attribute in the view row will actually be referring to the underlying entity object instance while reading or writing attribute values. A view object can have multiple row sets for managing a collection of rows from a query. Every row set in a view object is backed up by a query collection object that stores the result set returned by the JDBC APIs. All query collections belonging to a view object in a regular application module are cached as weak references. A query collection is referenced by one or more row sets in a view object instance if they share the same values of bind variables. All the secondary row sets in view objects are also weakly referenced by the parent view object. However, a view object gives special treatment to the default row set. The default row set resulted from the query execution is strongly referenced by the parent view object. So, the default row set will survive the garbage collection cycle and stay in memory along with the parent view object instance till the owning application module is recycled.

Entity Objects lifespan

Entity object cache implementation stores unmodified rows in a weakly referenced hash map (java.util.WeakHashMap). When you insert, modify, or delete rows, they will be transferred to a strongly held list which will survive the garbage collection cycle. All weakly referred entity rows in the cache will be cleaned up in memory pressure, keeping dirty rows alive.

Managed Beans scopes

Managed beans are Java classes used for implementing business logic or data model for your JSF page. The scope specified for a managed bean decides how long it should stay in memory.

The **backingBean** scope exists for the duration of a request. The backingBean scope is limited to the containing task flow or UI component. If you have multiple instances of the same task flow or declarative components in a page, each instance will get its own backingBean scope memory area and they are not visible to each other.

The **request** scope exists for the duration of a request. Unlike the backingBean scope, the request scope is shared across task flows and components in a page.

The **view** scope is longer than the request scope, and shorter than the session scope. View-scoped memory variables or managed beans exist until the user finishes interaction with the current view.

The **pageFlow** scope exists across activities for the duration of a task flow. Each task flow has its own pageFlow scoped memory area.

The **session** scope exists for the duration of a user session (HttpSession). It starts when the user starts accessing the application and ends when the session times out, or when the application invalidates the session object.

The **none** scope is the shortest one. A managed bean in this scope is available for garbage collection once it is not referenced by other objects. The none scoped beans live to serve other beans.

Task Flow Activation

ADF Task flows play the key role at ADF controller layer. Basically, from UI perspective, an ADF application is a bunch of task flows. Bounded task flows that are built with page fragments are displayed on a page by using ADF region component. ADF region component refers to a task flow executable binding which actually provides the content for the region. The task flow binding has activation property that determines when the content should be loaded. There are three available values of the activation property.
So, when it is Immediate, the region is going to be loaded on initial display of the page. This option is good if the task flow is exposed to the user once page is rendered. But if the region is displayed in a popup or inside a tabbed pane, then we would like to postpone the loading of the region content. And we can consider two options for that: deferred and conditional. Deferred is the default option in Jdeveloper 12c and it actually defers the activation of a region until it is shown to the user. However, it is available only for facelets document type. So if we are old fashioned and work with good old jspx documents, then conditional option would work. This option activates the region if the EL expression specified for the active attribute evaluates to true, for example, the tab containing the region is open.

UI Tree lifespan

On initial page request the framework creates UI tree at the very beginning of the request on Restore View phase, it adds components to the tree and saves it in the FacesContext object at the end of the request on Render Response phase. The UI tree is going to exist until a page refresh or a redirect to the same or another view. After that the tree will be available for garbage collection. However, if any single component in a tree is hard referenced from a managed bean with a scope longer then request, then GC won't be able to clean the entire tree until the managed bean has gone.

Contact address:

Eugene Fedorenko
eProseed NL
Rembrandtlaan 24e
3723 BJ, Bilthoven

Phone: +31650420681
Blog: adfpractice-fedor.blogspot.com
Twitter: @fisbudo
Email eugene.fedorenko@eproseed.com
Internet: http://www.eproseed.com